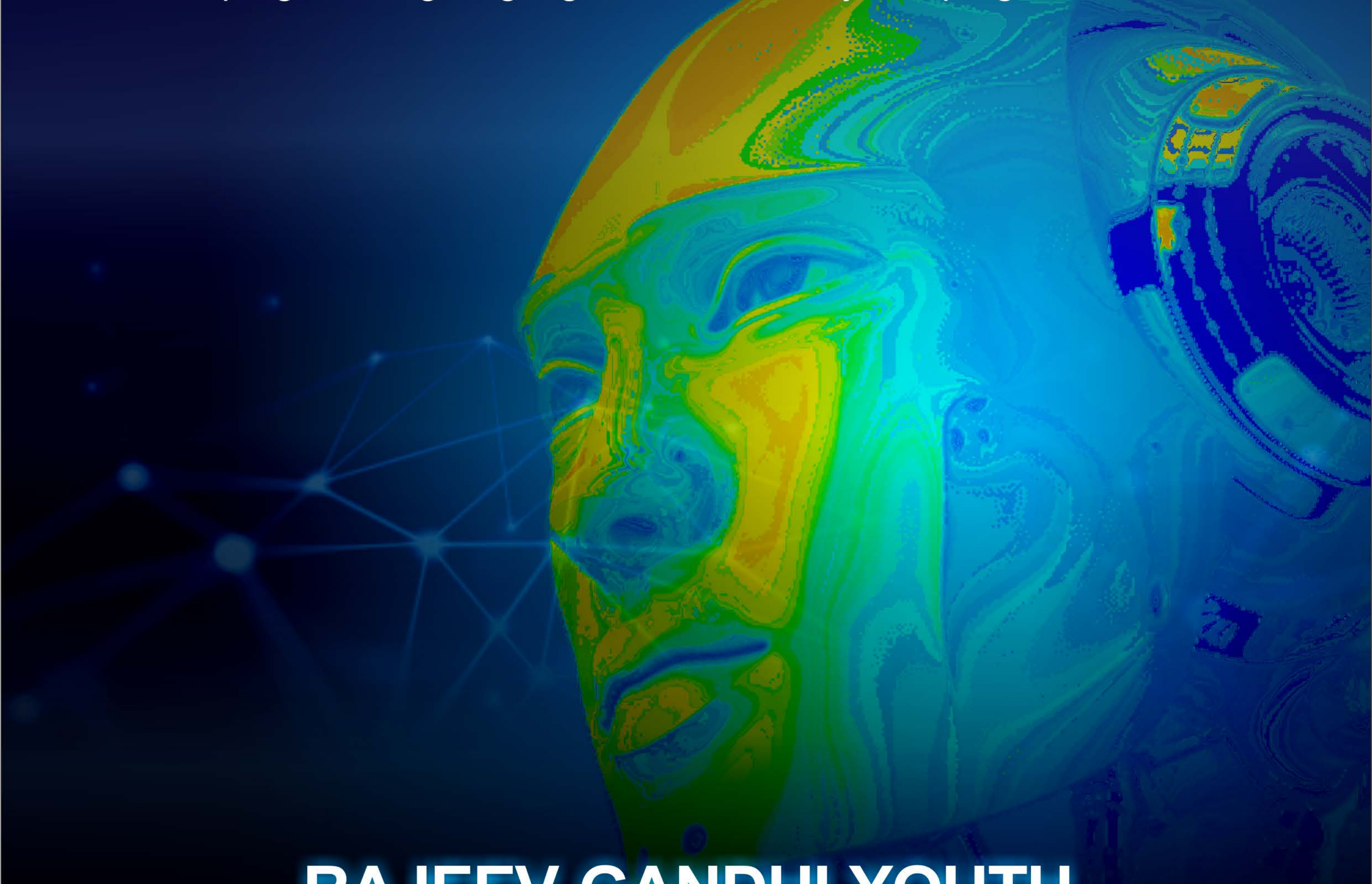
PYTHON

MACHINE LEARNING

This tutorial will reward you for every minute you put into it.
Soon, you'll know one of the world's most powerful, popular
programming languages. You'll be a Python programmer.



RAJEEV GANDHI YOUTH
COMPUTER SAKSHARTA MISSION

Python Machine Learning

A Crash Course for Beginners to Understand Machine learning, Artificial Intelligence, Neural Networks, and Deep Learning with Scikit-Learn, TensorFlow, and Keras.

by

Josh Hugh Learning

Josh Hugh Learning

Table of Contents

Introduction

***Congratulations on purchasing Python Machine Learning, and thank you for doing so*.**

The following chapters will discuss a lot of the different parts that we need to know when it is time to start working with the Python language and getting it to work for some of your own machine learning needs. There are many companies that want to work with machine learning in order to help them learn more about their company, their competition, their industry, and their customers. When we collect the right data and combine it with the right machine learning algorithms, we will be able to make this work for our needs

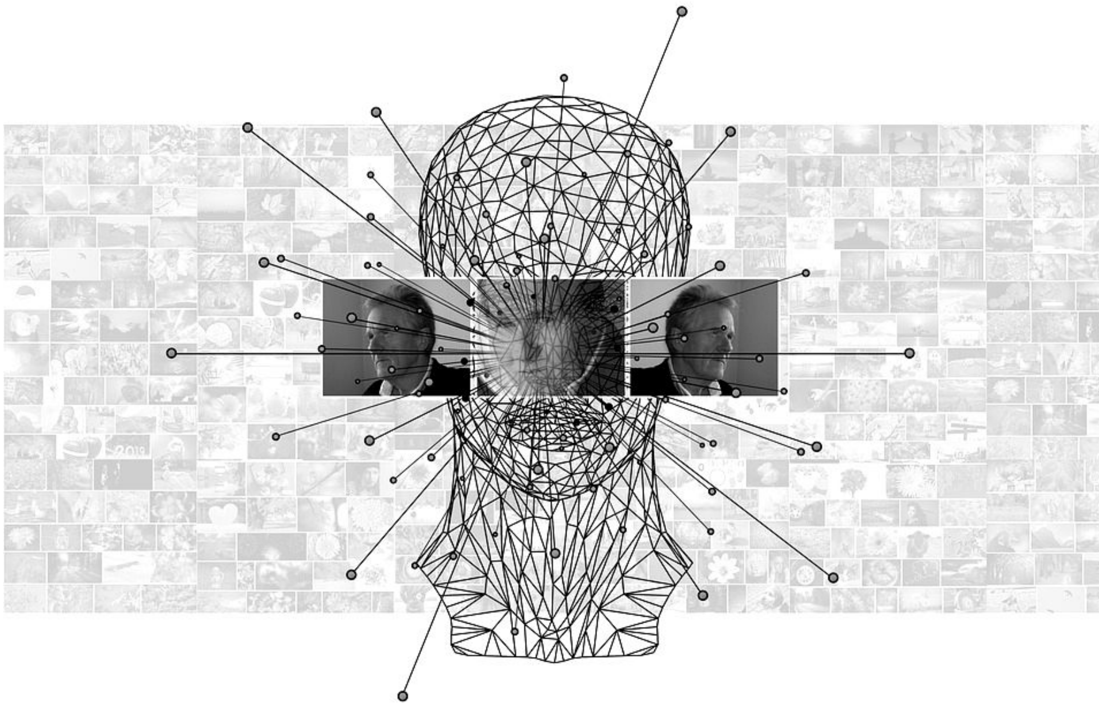
Sometimes, getting started with machine learning is hard, and knowing how to get your own program set up and ready to go will be important. The hardest part is figuring out the algorithms that we are going to spend some time working on along the way. There are really quite a few machine learning algorithms that you are able to work with, and picking the right one often will depend on the different processes that you want to do, the questions that you want the data to answer for you, and even the kind of data that you are trying to work with.

We are going to look at some of the basics that come with the process of machine learning and how to pick out the kind of data that we are able to work with as well. Then we will spend the rest of this guidebook looking at some of the different algorithms that we want to handle in this kind of language, with the help of Python. These will ensure that we are able to take over make sure that our data is handled and that we are actually able to see results with the work that we need to do.

There are many types of algorithms that we are able to explore. Some of the options that we are going to explore in this guidebook will include regressions, linear classification, non-linear, and more. In each of these categories, we are going to spend our time looking at how we can get started with this process, and the types of algorithms that fit into each one, and more. When you are done with this guidebook, you will know what you need about some of the most common machine learning algorithms and how to use them for your own data analysis.

There is so much that we are able to do with the Python language, and learning how to use it to pick out the right machine learning can be important. When you are ready to get started with Python machine learning, make sure to check out this guidebook to help you get started.

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, and please enjoy it!



Chapter 1: The Basics of Machine Learning

The first topic that we need to spend some time working on in this guidebook is information on machine learning and what we are able to do with it. This is a huge word in the technology and business world, but many people are not certain about what this all means, and how they are able to work with machine learning to reach some of their own goals along the way.

To start with, we need to take a look at what machine learning is all about and why it is gaining so much popularity in our world today. Machine learning is basically an application of artificial intelligence that is going to provide our systems with the ability to automatically learn and improve from experience, without being programmed on everything that they should be doing. Machine learning focuses on the creation and improvement computer programs that can access data and then use this data to learn.

It all starts with observations, or even data, such as instructions, direct experiences, and examples, in order to look for patterns in data and make better decisions in the future based on the example that we provide. The primary aim is to allow these computers a way to learn without any assistance or intervention from humans automatically, and then you can see that the computer will be able to adjust their actions accordingly to work with this as well.

There are a lot of applications that go with machine learning, and we are going to spend time in this guidebook looking at a lot of the different algorithms and more that you are able to do with machine learning. When you get all of this working together, you will see some amazing results and really see the true potential that comes with machine learning.

There are a lot of different things that you are able to use in machine learning. Any time that you aren't sure how the end result is going to turn up, or you aren't sure what the input of the other person could be, you will find that machine learning can help you get through some of these problems. If you want the computer to be able to go through a long list of options and find patterns or find the right result, then machine learning is going to work the best for you.

Some of the other things that machine learning can help out with include:

1. Voice recognition
2. Facial recognition
3. Search engines. The machine learning program is going to start learning from the answers that the individual provides, or

the queries, and will start to give better answers near the top as time goes on.

4. Recommendations after shopping
5. Going through large amounts of data about finances and customers and making accurate predictions about what the company should do to increase profits and happy customers along the way.

These are just a few of the examples of when you would want to start utilizing a program that needs to be able to act on its own. Many of the traditional programs that you are going to learn how to use as a beginner are going to be much simpler than this. They will tell the computer exactly what it should do in a given situation. This works great for a lot of programs, but for things like artificial intelligence, it is not going to be enough.

In addition, you will find that this machine learning is going to be a really good thing to use when it comes to handling data analysis, which is what some of the algorithms that we will discuss in this guidebook are used for in most cases. There are many algorithms that happen with this, but knowing how to use them and how they fit in with not only machine learning but also data science is going to be important.

Data analysis is going to be really important when it comes to your business and how competitive you can be in the future. You will find that with the right algorithms, and the information that we are going to go through in this guidebook with those algorithms, you will be able to handle some of the different business problems you have, complete your data analysis, and finally gain a good understanding of what all that big data you have been collecting is all about.

The Benefits of Machine Learning

There are actually quite a few benefits that we are going to see when it comes to working with machine learning on a regular basis. This is most likely one of the major reasons why so many companies want to jump on board and see what this is all about. Depending on the kinds of questions that you are looking to answer about your business and more, you will be able to find an application of machine learning in no time.

Machine learning is going to simplify some of the steps that come with product marketing and can assist when you want to make accurate forecasts of sales. Machine learning is going to be able to do this in more than one manner. For example, you will be able to get through a massive amount of data from as many sources as you want. There is likely to be a lot of information in there to help you modify and review all of your marketing strategies until you get the most effective one. You will also find that machine learning can help with rapid analysis, prediction and processing, and it is good at interpreting the past behaviors of your customers.

All of these come together to help you quite a bit, you will be able to use this unlimited amount of information in order to learn more about the customer, figure out what they are looking for in your business, and learn the best way to reach them in the marketing that you do. Considering marketing is an important part of the success of any business, you can easily see why so many companies want to be able to use this for themselves as well.

Machine learning can also help to facilitate accurate diagnoses and predictions in the medical field. This kind of learning is going to help doctors to identify their high-risk patients, make good diagnoses, and give the best medicines that are possible in each case. These are going to be based, for the most part, on available sets of data on patient records that remain anonymous, as well as the symptoms that these patients were experiencing at the time. This can help doctors and other medical professionals become more efficient at the jobs they are doing for us.

When it is time to really work on data entry, but the work is going to take too long to accomplish manually, machine learning is able to step in and help make this happen easier. Data duplication and inaccuracy are going to be big issues for companies who would like to automate the process of data entry. Machine learning can help work with taking those data entry tasks and getting the work done in no time.

Machine learning is also going to have a big impact on the finance sector. Some of the most common benefits of machine learning when it comes to the financial world will include loan underwriting, algorithmic trading, and fraud detection. In addition, this kind of learning is going to help us with continual data assessments to detect and then analyze any of the anomalies that happen in the financial world, which is going to really help to improve the amount of precision that we can find in our models and rules financially.

We will also see that machine learning is able to help with detecting spam. This was actually one of the earliest problems that machine learning was able to come in and help with. Spam filters are able to make up new rules, using neural networks, in order to eliminate spam mail and keep your inbox as clean as possible. The neural network is able to learn how to recognize phishing messages as well as other junk mail when it evaluates the rules that are found across an ever-growing network of computers.

The manufacturing industry is even able to benefit from some of the things that we see with machine learning. Manufacturing firms need to have corrective and preventative maintenance practices in place. However, these are going to be inefficient and costly in many cases. This is where machine learning can step in to help, and it is going to be a great tool in creating a highly efficient predictive maintenance plan that keeps the business up and running and doing well. In fact, when the company follows these plans, it is going to minimize the chances of failures that are not expected to happen, which will reduce unnecessary preventive maintenance activities.

Machine learning is also going to help with better customer segmentation and accurate lifetime value prediction. These are going to be some of the biggest challenges that marketers are going to face on a daily basis. Marketing and sales units are going to have an enormous amount of data sourced from many channels, but accurate predictions are only going to be found when we look at machine learning.

Some of the best marketers out there right now know that they should use machine learning to eliminate some of the guesswork that comes with their marketing efforts. For example, when they use the data representing the patterns of behavior for their users during a trial period, they are going to be able to help their company make predictions on how likely it is to get conversions to a paid trial and figure out if this paid trial is worth their time or not.

And finally, we are able to look at how machine learning is going to be the right option for recommending products and more to customers. This is one of the best ways for a company to cross-sell and up sell to their customers and can be really useful for customers as well. If you have ever gone onto a website and had something like “customers like you bought these products” or something similar, then you have seen machine learning at work in this way.

The models of machine learning are going to analyze the purchase history that they see with the customer, and based on that, they are able to identify the products that the company has that the customer may be interested in. The algorithm is a good one to help us find the hidden patterns among the items and then will group similar products into clusters. This is going to be a good example of unsupervised learning, which we are going to talk about in a moment.

This kind of model is helpful to businesses because it ensures they are able to provide the best product recommendations back to their customers, which is a great way to motivate customers to make another purchase. In this manner, unsupervised machine learning is going to help us to make a really strong recommendation system for the company and can increase the amount they are going to see in profits along the way.

As we can see, there are a lot of benefits that come with working in machine learning, and companies across all industries out there are going to be able to see some of the benefits. Some of the tasks that come with this is making sure that you collect the right kind of data, and that you take your time to pick out a good algorithm that can actually sort through your data and will help you to really hear the predictions and more that you need .

Supervised Machine Learning

Now there are going to be three types of machine learning that we are able to work with when it comes to the algorithm types. We are going to spend some time looking at each one and how it is meant to work overall. Let us look that the supervised form of machine learning. These can apply what has been learned prior and then putting that towards new data, with the help of examples that are labeled in order to predict whether an event is likely to happen in the future or not.

Beginning from the analysis on a known set of data, the algorithm that you choose here is going to be able to produce for us a function to make predictions about the values we are given. The system, when it is working well, is going to be able to provide targets for any new input after you do enough training on it. The learning algorithm is going to compare the output that it gives with the intended and correct output, then it is able to find out any of the errors that are there modify the models in the right manner along the way.

Along with the same kind of idea, but combining some of the work that we will talk about with unsupervised learning later on, includes semi-supervised machine learning algorithms. It is going to work with labeled and unlabeled data to help with the training. In most cases, we are going to see just a small amount of data that is labeled as being used, and then a large amount of data that is unlabeled that is being used. This is because working with labeled data can be expensive, even when it is efficient, and being able to work with this kind of data is going to be hard to handle, and you will need to add in the unlabeled data to get things done.

The systems that are going to work with this kind of algorithm are going to be higher in the amount of accuracy that they will see with their results. In many cases, this kind of learning is going to be chosen any time that the labeled data that we are working with requires skills and relevant resources in order to either train or learn from it. Otherwise, you will find that acquiring the unlabeled resources and data that you need won’t require additional work to get it all done.

Unsupervised Machine Learning

Now that we have had a chance to take a look at what the supervised machine learning algorithms are able to do, it is time to take a look at what we are able to do with unsupervised machine learning algorithms. These are going to be the ones that we use any time that the information we have is used to train the algorithm, and it is not going to be labeled or classified. This means that the algorithm, and the system or machine it is on, will need to do the learning on their own, without examples and labeled data to help it make more sense.

Unsupervised learning studies show a system is able to infer a function to describe one of the hidden structures from the unlabeled data. The system doesn’t figure out the right output with this one, it is going to explore the data and then draw inferences from the sets of data.

With this one, we are going to use a lot of data that doesn’t have a label on it or any information as to the right answer, and then we are able to send it right through the algorithm and let the system learn along the way. This takes more time, and you may end up with some more runs of training and testing before you are done, but it can be one of the best ways to get some strong systems in place to help with your machine learning.

Reinforcement Machine Learning

This is going to be the method of learning that is going to interact with the environment around it by producing actions, and then discovering the rewards or the errors as it goes on. You can compare this one to the idea of trial and error along the way. The trial and error are going to add to the search and delayed reward and are going to be some of the most relevant characteristics of this kind of learning.

When we work with reinforcement machine learning, we are going to find that it allows the software agents and the machine to automatically, on their own, determine the ideal behavior that they should take to maximize the performance that we are seeing. This is something that we are going to call the reinforcement signal.

When we are looking at reinforcement machine learning, there are going to be a lot of similarities to how the computer learns compared to how a human can learn along the way. This method is set up to help us really be able to work with trial and error, and the computer will be able to use this idea to figure out the right course of action to help them be successful. There is so much that we are able to do when it comes to machine learning, and figuring out these different parts, and how to make them work is a challenge that many data scientists are going to have to deal with on a regular basis. When you are ready to explore more about machine learning, and some of the cool things that you as a programmer can do with this language, make sure to read on through below and see all of the different choices in algorithms and more that are available.

Chapter 2: Learning the Data sets of Python

When it comes to working with machine learning and the Python language, there is nothing better than working with data. The more data that you are able to gather and clean, the easier it is to work with some of the algorithms that come with this process. You will find that Python is going to provide us with many algorithms, but we first need to be able to organize the data and get it set up to go through the algorithms for training and testing, in order to see the results that we would like.

With this in mind, we need to take some time to explore the different types of data that we are able to use. We have to look at some of the differences that come up with unstructured and structured data when to use each one, and how we can use these types of data in order to help us train and test some of our Python machine learning algorithms.

Structured Data Sets

The first type of data that we need to spend time working with is structured data. Traditionally we would just have this kind of data in the past, which was harder to get but was easy to work with. Companies would look for some of the structured data that they need, and then make some of the business decisions and more that they need to move forward.

This kind of data is going to be any data that has been organized well and is then going to fit into a formatted repository for us to use. Usually, this is going to be data that is stored in a database so that the elements can be used for more effective processing and analysis.

We may be able to find this kind of data when we are going through other databases to help with the information, or when we get the results of a survey. This one is much easier to work with because it is already organized, and it is going to fit into the algorithm that you want to work with without you have to worry about missing values, duplicates, outliers, or anything else like this. It is also a much more expensive method of working with data, which can make it harder to work with overall as well.

This is why many companies have to make a balancing act over how much-structured data and how much-unstructured data they want to work with. The structured data can make the work easier and will ensure that the algorithm is going to work better, but it is harder to collect, there is less of it, and it is more expensive. The unstructured data is sometimes hard to work with and takes time to clean and organize, but there are endless amounts of it, it can still be used to handle your machine learning algorithms, and it is a lot less expensive to gather up and use.

Unstructured Data Sets

The second type of data that we need to take a look at is the unstructured data. This is basically going to represent any of the data that doesn't provide us with a recognizable structure to it. It is going to be raw and unorganized, and there may not be any rhyme or reason to what you are seeing. Unstructured data is often going to be called loosely structured data in some cases, where the sources of data may have some kind of structure, but not all of the data in that set will end up following the same structure, so you will still have some work to handle to make them work for your needs.

For those businesses that are going to center around the customer, the data that is found in this kind of form can be examined and there is so much that we are able to get out of it, such as using it to enhance the relationship marketing and the customer relationship management that happens as well. The development of unstructured data, as time goes on, is likely to keep growing because more and more businesses are looking to gather this information, and it can be gathered and created in no time at all.

Unstructured data is going to refer to any data that is able to follow a form that is less ordered than items like a database, table, spreadsheets, and other ordered sets of data. In fact, the term data set is going to be a good way to look at this because it is going to be associated with data that is neat and doesn't have any extra content. We are basically working with a lot of data that is not necessarily organized and can be hard to work with without some help organizing.

There are a ton of instances where we are going to see this kind of data. We may see it in documents, social media posts, medical records, books, collaboration software, instant messages, presentations, and Word documents, to name a few. We are able to work with some non-textual unstructured data, and we will see that this can include video files JPEG images and even some MP3 audio files as well.

Most of the data that you are going to work with over time will rely on the idea of unstructured data. There is so much of this kind of data out there to work with, and it is often easier to find and less expensive compared to some of the structured data that we talked about above. Being prepared to handle some of this unstructured data and make sure that it is prepared and ready to go with some of your machine learning algorithms.

How to Manage the Missing Data

We also need to spend some time working with the missing data that comes in. When we are gathering all of that data from all of those different sources, it is likely that at least some of that data is going to come in missing. Whether this is just one part of the data, or there are a lot of values that are missing for entry, we need to know how we can manage these missing data points.

If we tried to push some of these missing data points through the chosen algorithm, it would not end up going all that well. The algorithm may or may not be able to handle some of the issues with the missing data and even if the algorithm is able to handle the missing values, there could be issues with it skewing the results. This is why it is important to choose which method you would like to use when it is time to manage that missing data.

The method you choose will depend on the type and amount of missing data. If you just have a few points that are missing, then it is probably fine to erase those points and not worry about them at all. This can be the easiest method to work with because you will be able to get them gone in no time. However, for the most part, it is important to keep all of the data that you have, and filling them in is a better way to manage the data.

There are a few ways that you are able to fill in the missing data. Usually, going with the average or the mean of the rest of the data, is going to be a good way to start. This ensures that you are still able to use the data that is missing, while not losing out on some of the important parts that you need with that entry as well. Find the standard that you want to use, and then fill in those missing parts so that the data can work better with the algorithm that we are using.

In addition to the missing data, we need to spend some time learning how to manage the outliers and duplicate content. Both of these, if they are not taken care of, is going to skew the results that you get. It is important to figure out the best way to handle both of these before you move on.

To start, we have the outliers. If you have big outliers that are random but really high or really low compared to the rest of the values, you will find that it is going to mess with your results, and those results are not going to be as accurate as you would like. If this is what happens with your data, then it is probably best to just delete the outlier. It is just something that is not that important, and removing it will ensure that you are able to handle the data in an accurate manner.

Now, there are some situations where the outliers are going to be important, as well. If you are looking at some of the outliers, and it looks like there are a number of outliers that are going to fit into one cluster or group, then this may be a sign that we need to move on to looking at these and using the outliers. If you can see that a significant number of outliers are in this group, rather than just one or two random outliers, then this could be a good sign that there is a new option to work with for reaching customers, marketing, the new product you want to release and more. It never hurts to take a look at these outliers, but for many situations, you will want to delete these.

In addition, we need to focus on the duplicates. Many times we will want to go through and delete the duplicates so that the answers don't end up causing any issues with the results that we have. If you have ten of the same person, with all of the same information for them in your set of data, it is going to skew your results.

If this happens a few times, the issue is going to get even worse overall. For the most part, we want to go through and delete these enough so that we just end up with no duplicates or at least a minimal amount of them.

Splitting Your Data

One thing that we will need to work on when it comes to our data is figuring out how to split it up. There is some work that we have to do in order to handle some of the data that we need before we can go through and add them to the algorithms that we want to use. For example, we need to go through a process of training and to test our algorithms to make sure they will work the way that we want. This means that we need to split up the data that we have into the training data and the testing data.

These two sets are important to making sure our algorithms are going to work properly. Having them set up and using these sets in the proper manner will help us to get the best results when it comes to working in machine learning. The rules are pretty simple with this, though, so you will be able to get started without any problems along the way.

For example, we need to make sure that the data we are using is high quality to start with. If you do not have enough data or the data is not high in quality, then your algorithm is going to get trained improperly, and will not work the way that you want. Always be careful about the kind of data that you are using in this process

Next, we need to make sure that we are splitting up the data properly. We should have a group for testing and a group for training. Your training set should be much larger to ensure that you are properly training the data that you have and that the algorithm will get a good dose of the examples that you present and what you want it to do.

Training and Testing Your Data

As we go through some of the processes with working on our data and these algorithms, we have to make sure that we are training and testing all of the algorithms first. You can't just write a few lines of code and then put in your data, hoping to get a good prediction to pop out. You need to take the time to train and test the data through that algorithm, to ensure that the accuracy is there, and to make sure that the algorithm is going to be ready for you to work with.

The first step to this is going to be the training of your data. You have to make sure that you are spending a good deal of time training your data so that it knows the right way to behave. Out of the splitting of the data that we did before; you want to have about 75 to 85 percent of your data be in the training set. This ensures that you have enough data there that will help you to really train the algorithm and gives it plenty of time to learn along the way as well.

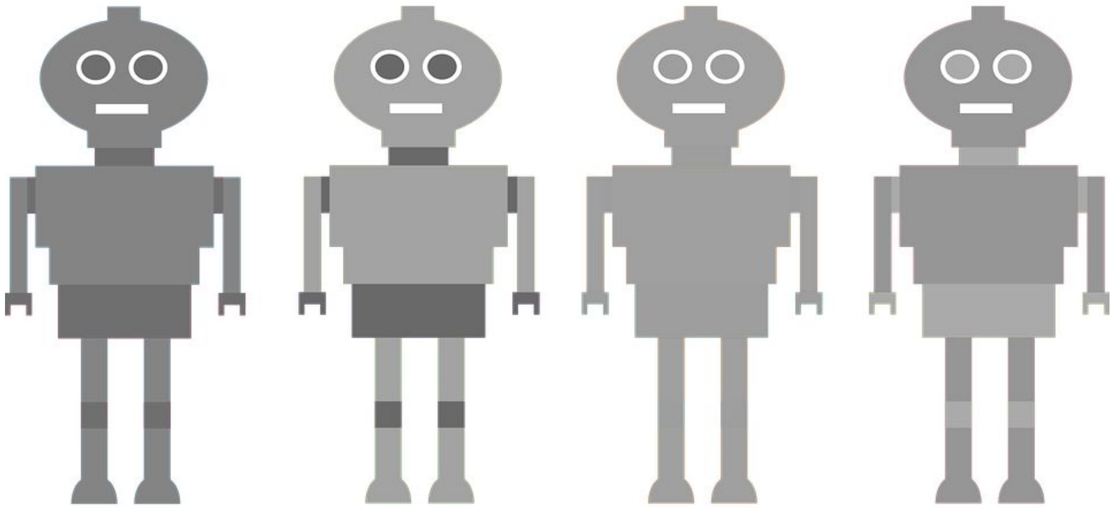
Then you can feed all of that training data through your algorithm and let it have some time to form those connections and learn what it is supposed to do. From there, you will then need to test the data that you are working with, as well. This will be the rest of the data that you are working with. You can feed this through the algorithm, and wait to see how much accuracy comes back.

Keep in mind with this one that most of the time; these algorithms are going to be able to learn by experience. This means that while they may not have as high accuracy as you would like in the beginning, they will get better. In fact, you may have to go through and do the training and testing phases a few times in order to increase the accuracy enough that you will use the algorithm to make predictions.

You want to get the accuracy as high as possible. However, if you are noticing that the accuracy tends to be lower, and is going below 50 percent, or is not improving as you do some iterations of the training and testing phases, then this is a bad sign. It shows us that you either are not using enough data in your training for the algorithm to properly learn, or you are using bad data that is confusing the algorithm.

This is why we do the training and testing phases. It helps us to catch some of the problems that may happen with this data and will allow us time to make the necessary changes to the data and algorithm before we rely on the future of our company using badly trained algorithms. We can make the adjustments and run the phases again until the accuracy goes up, and we know that we can rely on that data again.

Working with data is going to be a very big part of working with the machine learning projects that we want to handle, we need to be able to learn how to distinguish the different types of data, how to handle the missing data and the outliers, and how to split up the data so that we are able to properly train and test the algorithms that we want to use. When we are able to work with this, we are going to see some great results through our machine learning, and we will then be able to use these predictions and insights to help improve our business.



Chapter 3: Supervised Learning with Regressions

We spent a bit of time in the first chapter looking at what supervised learning is going to be all about, but we need to spend some time looking at the different algorithms that we are able to work with when it comes to this kind of supervised learning. We are going to start out here with some looks at how to work with supervised learning on regression problems, but then we will move on to those that we are able to do with classification problems later on.

Remember that the supervised learning that we will use here is going to be the kind of learning that provides the algorithm with a lot of examples. The input is going to include the corresponding output so that the machine and the system are then able to take a look at the information and learn what the right answers are. This may seem like it is cheating a bit, but the system is able to learn from those examples and then use that information on some of the unseen and new data that it gets later on.

We can find that this is an effective and quick method of working with machine learning, and it can get our algorithms written out pretty quickly. That is why supervised machine learning is going to be used on a regular basis on these kinds of projects. Some of the different options that you are able to use when it comes to supervised learning with regression problems will include:

The Linear Regression

We now need to take a look at what a linear regression is all about. These models are going to show us, or predict the relationship that will show up between two factors or variables. The factor that we are predicting in this model will be the dependent variable. Then the factors that we are using in order to predict the value of the dependent variable will be known as the independent variable.

Good data is not always going to tell us the full story. The regression analysis is going to be used in research as it is able to establish the correlation between variables. But the correlation is not always going to be the same as causation. Even a line that comes up in a simple linear regression that fits well with the points of data may not be able to say something definitive when it is time to look at the cause and effect relationship that is there.

In a simple linear regression like this one, each of the observations that we have will consist of two values. One value is going to be for the dependent variable, and then the other will be the independent variable. In this model, we are going to work with a straight line that will approximate the relationship between these two.

Multiple regression analysis, though, is when we are going to take at least two, and sometimes more, independent variables, and we will use these in a regression analysis. When this happens, the model is no longer going to be a simple linear one for us to work with.

The linear regression is going to have a number of practical uses along the way. Most applications that come with this are going to fall into one of the following broad categories. The first one is to predict or forecast or for error reduction. This can be used to help with a predictive model when it is time to work with an observed set of data values, and the response that comes. After we are able to create this model, if there are some additional values that are collected without the right response to it, the fitted model that we can use is still able to make a prediction for this.

If we have a goal to use this to help explain variation in the response variable that can be attributed to the variation in the explanatory variable, then this kind of analysis is going to be used to quantify the strength that we are able to see between the response and the explanatory variable.

Often we are going to be able to fit the linear regression with the approach of the least squared, but there are other options to work with based on what you are hoping to get out of the process. The least-squares approach can be used to help fit some models that are not always linear. What this means is that the terms of the linear model and least-squares are linked to one another closely; they are not going to be synonymous with one another.

The Cost Function

A cost function is going to be a mathematical formula that we are able to use to help us chart how something is going to change, especially when we look at production expenses at different output levels. The cost function is able to estimate the total cost that we see in production, given the quantity of the product or service that we are producing.

The management of your company is able to use this kind of model in order to run different production scenarios and to help predict what the total cost would be to produce your product, based on the level of output that you are using. The cost function is going to have its own formula to get things done, and this is going to be $C(x) = FC + V(x)$. Ci is going to be the total cost of production the FC is going to be the total costs that are fixed, V is the variable cost, and then x is going to be the number of units.

Understanding the cost unction of a company is going to be helpful in a lot of different scenarios, but especially when it comes to the process of budgeting because it is going to help your management to understand the cost behavior that we are able to see with a product. This is important to help us anticipate the costs that could be incurred in the next operating period at a planned level of activity. It will also allow the management to evaluate how efficient they were with the production process when the operating period is all done.

We can take a look at how to work with this one as well. Let's say that we are going to work with a toy manufacturer and they have asked to have a cost study to make sure they can improve the budget forecasts for the next year. They pay rent that is $300 a month right now, and their electricity is going to come out to $30. Each toy is going to require $5 in plastic and then $2 in cloth.

With this in mind, we are going to figure out how much it is going to cost for the company to manufacture 1200 toys that year, and then compare it to how much it will cost them to manufacture 1500 toys for the year.

The first thing that we need to do to make this work is to figure out which costs are going to be considered fixed, and which ones are the variable costs. The fixed costs are basically going to be any that are incurred, regardless of how much we are manufacturing the toys, and then the variable will be the ones that we have to pay per unit of production. What this means is that the electricity and the rent are going to be fixed, and then the cloth and the plastic are going to be variable costs.

Let's start out with the steps that we would take in order to produce the 12,000 toys a year. This is going to get us the following equation (keep in mind that the fixed cost here is going to be 330 multiplied by 12 so that we can figure out how much the rent and the utilities will be for the whole year.

C (1200) = \$3,960 + 1200(5 + 2)
C (1200) = \$12,360

But then we are able to take a look at how much it would take in order to do the same thing with 1500 toys. This one is going to use the formula below to help get it done:

C (1500) = \$3,960 + 1500(5 + 2)
C (1500) = \$14,460

The fixed costs in this on are going to stay the same, no matter how much output we are going to produce. This is why the cost per unit is going to go down or decrease when we make more units. The rent and the utilities will stay the same regardless of how many units we are trying to produce and sell, so usually working with a larger output here is going to give us more in profits for charging the same amount on the products.

Using Weight Training with Gradient Descent

One of the iterative optimization algorithms that we are going to be able to use when we want to find the minimum of a convex function is going to be the gradient descent. This one is going to be based on ideas of calculus, and it is going to really rely on the properties that happen with the first derivative in order to find out in what direction, and even in what magnitude, the coefficients of our function need to be modified along the way. This gradient descent is going to be used when we have some parameters that we are not able to calculate in an analytical manner, and we need to search for it with an optimization algorithm.

Imagine a large container we would use to eat off of, or a big container that we are able to store some fruit in. For our purposes here, the bowl is going to be the cost function or f. A random part on the container is the cost of the current values of your coefficients. We will see that the bottom then is going to be the cost of the coefficients that have the best set and the minimum of the function.

The goal, when using this process, is to try out more than one value for the coefficients, and then evaluate their cost. This will then allow you to go through and select out new coefficients that you can use, ones that have a slightly lower or better cost than the one you were looking at. If you are able to go through and repeat this process enough times, it is going to help us reach the bottom of the container, and then we will know the values of the coefficients that will give us that minimum cost.

There are a few different types of gradient descents that we are able to work with here. The first one is going to be the batch gradient descent for machine learning. The goal of your supervised machine learning is going to be to estimate a target function that is able to map out the input data over to the output variables. This is going to describe all of the regression and classification problems. This is a good look at what the batch gradient descent is all about. This is going to actually be one of the most common forms of gradient descent that we will see in machine learning.

But then we are going to move on to the stochastic gradient descent that is there. These algorithms are going to be slow when you want to run them on some really large sets of data, because one iteration of this kind of algorithm requires that you have a prediction for each instance in training, it can take you a very long time to do this when you have instances that number in the millions.

In these kinds of situations, you can change how you work with the gradient descent and use the stochastic gradient descent. The procedure of a regular descent is going to run, but the update that we see on the coefficients is going to be performed on each instance of training, rather than at the end of the batch of instances. The first step for this is going to require that the order of our set of data for training is going to be random. By mixing up the order that we are doing with these coefficients, we are able to help harness the random walk and make sure that we don’t get stuck or distracted.

The updated procedure that we are able to work with this one is going to be the same as the regular gradient descent, but it will not sum out the cost over all of the training patterns. Instead, it is going to be calculated for one training pattern. The learning is going to be faster with this option when we focus on large sets of data.

Polynomial Regression

And finally, we need to take a look at something that is known as the polynomial regression. When we are working with statistics, this kind of regression is going to be one of the analyses of regression that we can work with that will be able to check out the relationship between the dependent and the independent variable and is going to model this relationship as the nth degree polynomial in x. This is going to fit us into a nonlinear relationship between the value that we see with x and the corresponding conditional mean of y.

There are a lot of times when we will use this kind of regression, especially when we want to work with something like the growth rate of tissues, the distribution that we are able to find with carbon isotopes in some of the lake sediments that we see, and the progression of disease epidemics.

Although this regression is going to take some of our nonlinear models and has the data fit it, it is going to be more of a statistical estimation problem. It is going to be linear with the idea that the regression function is going to be linear in some of the unknown parameters that we have with the estimated data. For this reason, it is going to be considered one of the cases of multiple linear regressions.

The independent variables that show up are going to result from the polynomial expansion of the baseline variables, and they are going to be known as higher-degree terms. Such variables can be used when we are doing settings of classification.

There would be times that we would be working with regression problems when it comes to working with machine learning. Adding in some of these regression algorithms can help you to sort through the data that you have in a more efficient manner, and will ensure that you are able to get your data sorted through and find out the predictions and insights that you are looking for as well. Some of the other times when we would want to work with the polynomial regression will include:

1. When the researcher thinks that there are some relationships that will fit on a curved line. Clearly, these types of cases are going to show us a term that is polynomial.
2. When we want to do an inspection of the residuals, if we try to fit a linear model to a data that is curved, then the scatter plot of residuals on the predictor is going to have patches of many positive residuals in the manner. If this does happen, then we can see that this kind of situation is not going to be appropriate for the needs that we have.
3. An assumption in the usual multiple linear regression analysis that all of our variables that should be independent are

actually this way. in this kind of model, we will find that this is an assumption that is not going to be satisfied at all.

Basically, we will find that the biggest goal of this kind of analysis of regression is that we want to model the expected value that is going to show up in our dependent variables. We would do this in terms of the value of our independent variable that is going to be x. This will help us to get some of the work that we need to be done when it comes to this kind of regression as well.

Chapter 4: Regularization

To start here, we need to look at some of the foundations of overfitting. Let's assume that you are looking to make some predictions on the price movement of a stock in the future. We then decide to go through and gather up some of the historical daily prices of the stock, maybe going back over the past ten days or so, and then plot the stock price on a scatter plot as we would need. You would then want to go through and capture some of the information about the movements of the stock price. You are then able to assess and gather data for 16 features that you would like to follow because you know the stock price is going to be dependent on them. These are going to include:

1. The competition of the company.
2. The sentiment of the investors
3. The Foreign Exchange Rates
4. The interest rates
5. Inflation rate
6. The future contracts of the company.
7. The current contracts of the company
8. Information on the management of the company
9. The state of the M&A of the company
10. The current and the size of the futures contract of the company.
11. The dividends that the company is able to provide.

12. Any future announcements that the company may release.

13. The profits that the company is making.

14. The earnings of the company.

15. How the industry as a whole is performing at the time.

Once we have been able to gather, clean, scale, and transform the data, it is time to split it out into training and test sets of data. You will need to go through and feed the training data into the model that you chose for machine learning in order to get it trained. After you have had some time to train the algorithms or the models, you can then go through and test out the accuracy that happens with the model by passing through the set of test data.

The goal with this is actually to go through and chart out the prices. You should find that the actual prices of the stocks are going to be random. However, the predicted price of your stock is going to fall into a smooth curve. It has not gone through and fits itself too closely with the training set that you have, and this helps us to work with the generalization of the unseen data better.

Different Types of Fitting with Predicted Prices

We may want to make sure that we want to assume that the plot actual versus the predicted stock prices and we are going to then come up with a few different types of charts along the way:

1. **Straight Line to Show Predicted Price**

When we have a chart that shows the predicted price in a straight line, this shows us that the algorithm has gone through and has come up with a really strong pre-conception about the data. This is usually a sign that there is a high bias in the information and will show us something known as underfitting. These are not good models to use when you would like to predict new data and should be thrown out in most cases.

2. **A Very Strong Closely Fitted Line**

This one is an example of the other possible extreme. It may look like it is doing a really good job helping us to predict the price of the stock. However, this is going to be something that is known as overfitting. This is also going to be seen as a high-variance because it has learned the training data in a manner that is so accurate that it will not be able to generalize the information well. This makes it hard to go through and make some predictions on the new and unseen data that is there. These models are also not going to be good when you want to use them to make predictions on the new data.

If we go through with this model and feed it some new data, then you will find that the accuracy of those predictions is going to be really poor. It is also going to be a sign that we are not providing the model with enough data for training. Overfitting is when the model is going to over train itself on the data that you used for this purpose. This could be because we have too many features showing up in the data or because the algorithm has not had time to go through enough data. It is going to happen when the difference that shows up between the predicted values and the actual values is close to 0 .

How to Detect Overfitting

Now that we have taken a look at why this overfitting is such a bad thing, it is important for us to go through and figure out when overfitting is going to occur and then figure out how to fit it. The models that you are working with that have been overfitting on the training data will be the ones that are not able to generalize well to the new examples. These are not going to be very good at predicting some of the data that is not seen yet.

This means that when you are trying to add new data to the mix, then you are going to end up with an algorithm that is not doing its job very well. This implies that the model is going to be extremely accurate during training, but when it is time to make predictions on data that it has not seen before, the results are going to be poor overall.

If the measure of accuracy, such as mean error squared, ends up being quite a bit lower when you are working with training the model, and then you see that the accuracy starts to deteriorate on the set of data that you are using for testing, then this is a good sign that overfitting is happening with the data and you may need to supply it with different data, or at least more data, in order to increase the accuracy again.

Often the best way for us to go through with this and figure out whether or not there is overfitting with the data that we want to use, is to chart out the results on a graph. This may seem like we are getting ahead of ourselves, but these visuals will really help us to see some of the complex relationships that are going to show up on our data, and they can tell us almost instantly whether there is an issue with overfitting going on.

When you are working with a particular algorithm, and you are worried about the issue of overfitting, you simply need to go through and plot out the graph. If there is a straight line that shows up on the graph, and all of the points are right on the line, or at least touching the line that is there, then this is a bad sign that overfitting is going on. It is time to go back through and check on the data that you are using, or maybe just do some more training with a wider variety of data, in order to fix this kind of problem.

How Can I Fix Overfitting?

The good news is that there are a few steps that we are able to work in order to help fix some of the issues that come with overfitting. First, we are able to randomly remove some of the features that we are putting into the algorithm, and then use this to help us assess the accuracy of the algorithm in a more iterative manner. However, this can be effective, but the process is slow and can be really tedious. There are going to be four common methods that we are able to use in order to reduce some of the overfittings that we see. Some of these include:

1. **Reduce the features:** The most obvious out of the options that we are able to use is to reduce some of the features. You are able to compute the correlation matrix of our features, and then we can reduce some of the features that happen to be the most highly correlated with one another.

2. **Model selection algorithms:** Another method that we are able to use is going to be the model selection algorithms. These are the algorithms that have the power to choose the features that have the greatest importance and keeps those around, while limiting some of the others that don’t seem to affect the data as much The biggest problem that we are going to see with this one is that it is possible to lose out on some valuable information at times.

3. **Feed-in more data:** We can also take a look at feeding in more data to the model. Sometimes this is all that we need in order to handle some of the issues that come with overfitting. You should aim in training a set to feed in enough data to the models so that you are able to train, test, and validate the model thoroughly. For example, you should do about 60 percent of your data to help train the model, 20 percent to test the data, and then 20 to help validate the model that you are working with.

We need to explore the idea of regularization a bit more. The aim of this is to help keep all of the features, but then impose a constraint on the magnitude of the coefficients that you are able to get. This is often seen as the preferred method because you do not have to lose out on any of your features because you are busy penalizing them like some of the other methods. When the constraints are applied to the parameters, then the model is going to end up not overfitting as much because it can produce a smooth function.

The parameters that we work within regularization, which are going to be known as the penalty factors, are going to be able to introduce which controls the parameters and will ensure that the model is not going to over train itself on any of the training data that you are working with. We will also find that these parameters are going to be at smaller values to help eliminate the issue of overfitting. When the coefficients work with larger values, then the regularization parameters are going to penalize some of the optimization functions that are there.

While we are on regularization, we should look at the two most common techniques that we are able to work with on this. The first one is going to be Lasso. This is going to be a tool for feature selection, and it is going to be able to help us eliminate any of the features that not important to what we are doing. It can also add in a penalty, which is going to be the absolute of the magnitude that we will see with the coefficient.

What this is going to do is ensure that the features we are working with are not going to end up applying some high weights to the prediction that comes with our algorithm. The result of this is that some of the weights are going to turn into zero. This means that the data of some of our features are not going to be seen as important at all, and they will not be used in the algorithm that we have at all.

And the second technique that works here is going to be a ridge. This one is a bit different but can still have a lot of the features and strengths that we need. With ridge, we are going to add in a penalty, which is going to be the square of the magnitude of the coefficients. As a result of this, you will find that some of the weights that we have are going to end up being close to 0. This is a good way to smooth out some of the effects that we will see on the features as well.

Overfitting our data is something that can be a big issue when we are working with machine learning. We want to get accurate information out of what we are doing in this process, and if the algorithm ends up overfitting, then it is not guessing the data very well. It may do well with the training data that we are working with, but it is not going to do all that well when it comes to taking on new data, and that is when you really need this algorithm to work its best.

Following some of the techniques that we have in this chapter, and learning how this overfitting occurs in the first place, is going to be one of the important first steps that you can follow in order to make sure that this issue doesn’t happen - the more that you are able to prevent this from happening, the more accurate and efficient your models will end up being in machine learning. When we can keep underfitting and overfitting from happening with some of the data that we have, we are going to get amazing results, and our models will work in the manner that we want.

Chapter 5: Supervised Learning with Classification

Supervised machine learning is going to be one of the algorithms that you will use a lot in machine learning because there are a lot of applications. This is a good and effective method of teaching your machine on algorithms and how you would like it to behave. This is because this method is going to show the algorithm all of the examples, with their corresponding answers, right from the beginning, making sure that the algorithm is able to learn the right way faster than before.

This is why there are going to be so many different types of supervised machine learning models and algorithms that we are able to work with. They may take a bit more time in the beginning, but when we use classification and some of the other tools that are out there to help us get it all done, we will find that it is easier to train and test out our models and get some good results in the process. Some of the different supervised machine learning algorithms that we are able to focus on with classification will include:

Logistic Regression

The next algorithm on the list that we need to take a look at is going to be the logistic regression. These are going to be able to help us out with a lot of different problems that we want the data to solve, and if we are able to use it in the right manner, we are going to be able to see some amazing results in the process. As time passed, it started to be used for applications in the social sciences. Logistic regression, though, no matter how we decide to work with it, is going to be used when the target, or our dependent variable, is categorical.

This means that we may use it for a few different situations, such as when we would like to predict whether or not an email that comes to us is spam, or whether or not a tumor is malignant.

To help us see how this goes, we can start with a scenario where we would like to determine whether or not an email that we see is spam or not. If we use linear regression for this instance, we would need to set up a basis wherewith to base our classification with.

From this example alone, it is easy to see that the linear regression is going to fail a bit when it comes to some of the classification problems. Linear regression is not going to be bounded, and this is why we need to work with logistic regression for some of our problems. With this one, the value is going to range from 0 to 1, and nothing in between.

Now, we may see that there are a few different types of logistic regression that we are able to work with. The three main types that we are able to focus our attention on here are going to include:

1. The binary logistic regression: This is going to be a response that is categorical and has only two outcomes possible. When we are looking at emails, for example, it is going to tell us whether the specific email is spam or not.

2. Multinomial logistic regression: This is when there are three or more categories that show up without any order. For example, we may see this one when predicting which food is preferred more such as Vegan, Non-Vegan, and Vegan.
3. Ordinal logistic regression: This is when there are at least three categories, bust sometimes more, to the ordering. For example, we could have a movie rating that goes from one to five.

To help make it easier to predict which class our data is going to belong to, we are going to set a threshold in the beginning. Based on what this threshold is about, the obtained estimated probability is going to be classified into classes. Going back to the idea of the spam earlier, we could have our predicted value be at or above 0.5.

When an email reaches this threshold, then the email is going to be seen as spam. If it does not, then it is not seen as spam.

The decision boundary that we are able to work with is going to be seen as non-linear or linear. If preferred the Polynomial order can be changed to get to a more varied boundary if we would prefer. This would give us the variation that we would need.

When we work with the logistical regression, we will find that there are a lot of the other parts we have talked about in this guidebook so far that they are going to show up in the code. This is because there are often times when we need to combine together more than one option when it comes to working with these algorithms. A good way to see some of this is to look at an example of the coding that is needed to work on the logistical regression, and we can see that below:

```
def weightInitialization(n_features):
w = np.zeros((1,n_features))
b = 0
return w,bdef sigmoid_activation(result):
final_result = 1/(1+np.exp(-result))
return final_result
def model_optimize(w, b, X, Y):
m = X.shape[0]

#Prediction
final_result = sigmoid_activation(np.dot(w,X.T)+b)
Y_T = Y.T
cost = (-1/m)*(np.sum((Y_T*np.log(final_result)) + ((1-Y_T)*(np.log(1-
final_result)))))
#

#Gradient calculation
```

```
dw = (1/m)*(np.dot(X.T, (final_result-Y.T).T))
db = (1/m)*(np.sum(final_result-Y.T))

grads = {"dw": dw, "db": db}

return grads, costdef model_predict(w, b, X, Y, learning_rate,
no_iterations):
costs = []
for i in range(no_iterations):
#
grads, cost = model_optimize(w,b,X,Y)
#
dw = grads["dw"]
db = grads["db"]
#weight update
w = w - (learning_rate * (dw.T))
b = b - (learning_rate * db)
#

if (i % 100 == 0):
costs.append(cost)
#print("Cost after %i iteration is %f" %(i, cost))

#final parameters
coeff = {"w": w, "b": b}
gradient = {"dw": dw, "db": db}

return coeff, gradient, costsdef predict(final_pred, m):
```

```
y_pred = np.zeros((1,m))
for i in range(final_pred.shape[1]):
if final_pred[0][i] > 0.5:
y_pred[0][i] = 1
return y_pred
```

Many times the logistical regression is going to be a better choice to go with compared to the linear regression. This is because this will allow us to catch some of the instances that are going to be missed, like what is going to happen with the linear regression.

Multiclass Classification

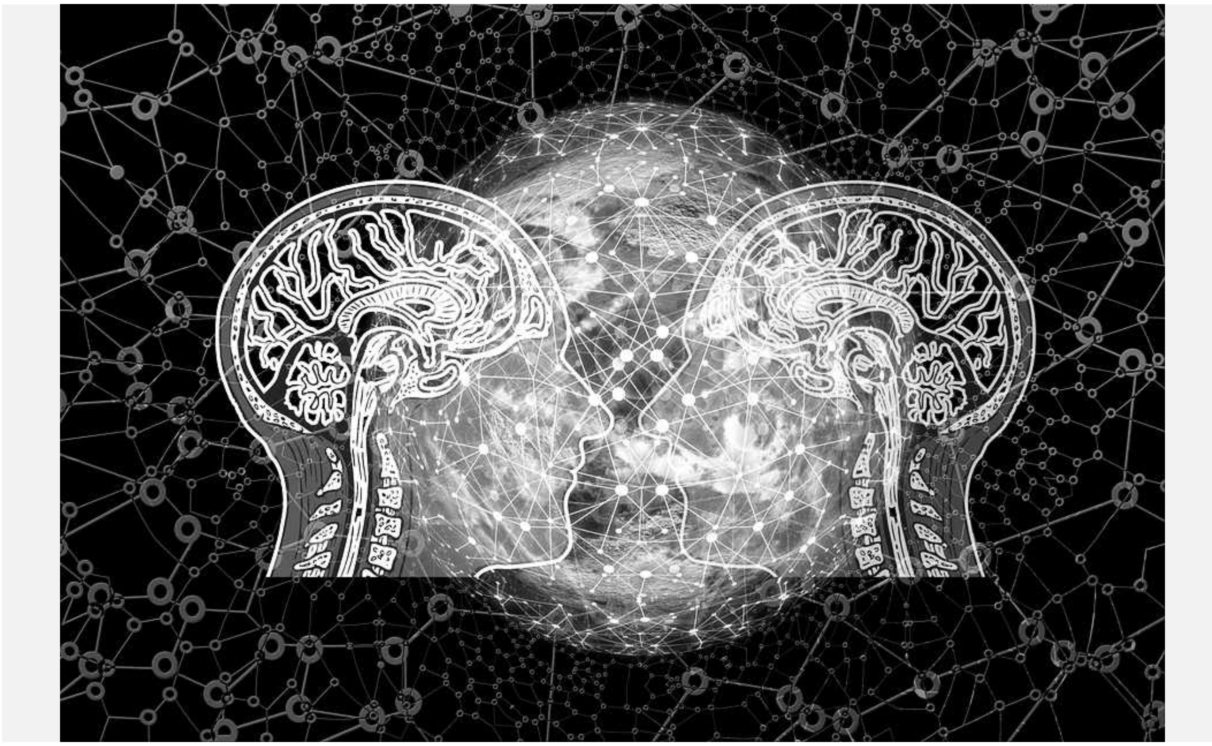
While we are here, we also need to take a look at some of the benefits of working with the multiclass classification. Classification problems are often going to come with many classes, and there is going to be an imbalanced kind of dataset that will present a different challenge compared to what we see with some of the classification problems. Sometimes the skewed distribution is going to make some of the other algorithms with machine learning less effective, especially when it comes to predicting minority class examples.

We will find that with a multiclass classification problem, you are going to be handling a task of classification that has three or more classes to work with. This means that we could do something like classifying a set of images of fruits, which may be things like pears, apples, and oranges, and some other fruits if you would like to add these in as well.

This kind of classification is going to make some assumptions in order to make sure that things are going to happen. For example, it will make one assumption that each of the samples is going to be assigned to one and no more than one label. For example, fruit can be either a pear or an apple, but it is not possible for this fruit to be both at the same time.

While some of the classification algorithms that are out there are naturally going to be set up to permit the use of more than two of these classes, others are going to be binary algorithms instead, and these can also be turned into multinomial classifiers with a lot of different strategies along the way. One thing to remember with this one though is that we should not confuse this kind of classification should not be confused with the idea of multi-label classification, where the multiple labels are to be predicted for each instance.

There are many times when we are going to work with the classification problems, especially when it comes to handling things with supervised machine learning. These can make it easier to split up some of the different algorithms that you have and will ensure that you are able to see what classes are there, and how to understand some of the data that you have available.



Chapter 6: Non-linear Classification Models

There are a lot of things that we are able to do when it is time to work with some of the classification problems that we have along the way. These are really useful when it is time to work through some of the data that we have, and they can often be one of the best ways that we are able to learn about the data, see which groups the data falls into, and so much more. Some of the other classification models that you are able to work with, the ones that do not fit in with the linear classification models, will include some of the following:

K-Nearest Neighbor

The first option that we are going to look at when it comes to working on the non-linear classification models will include the K-Nearest Neighbor or the KNN algorithm. This is going to be an example of a supervised machine learning algorithm, so we will need to have some labeled data in place as well.

There are a few benefits that you will see when it is time to work with the KNN algorithm. When we are working with the algorithm, it is helpful for us to cut down the noise that may be in the set of data. Depending on the data that we are working with, you may find that the noise is going to be really loud, and making sure the noise is gone going to ensure that we are able to handle the work as well and get more accurate results in the process.

There are many algorithms that we are able to work with when it comes to working with machine learning. This makes it hard to know why you would want to work with this kind of algorithm over some of the others. The benefits of working with the KNN algorithm and why you would want to choose it over some of the other options include:

1. It can work well with problems, even if they are considered multi-class.

2. You are able to apply this algorithm to both problems that are regressive and those that are classification.

3. There aren't any assumptions that come up with the data. This ensures that you get the information that you want, rather than having any assumptions in the place, causing some issues.

4. It is an easy algorithm to work with. It is easy to understand, especially if you are brand new to the machine learning process.

However, there are more options for algorithms that you are able to work with because the KNN algorithm isn't going to be perfect in each and every situation that you go to. Some of the negatives that come with using the KNN algorithm include:

1. It is going to be computationally and memory intensive expensive. If you don't have the right system and the right amount of space to work with, it is going to make it more difficult to see the results that you want from this algorithm.

2. If there are a lot of independent variables that you are going to work with, you will find that the KNN algorithm is going to struggle.

3. The KNN algorithm isn't going to work that well if you have any rare event, or skewed, target variables.

4. Sensitive to the scale of data.

For any of the problems that we are going to work with, you will find that having a smaller value of k is going to give us more variance in any of the predictions that we are working with. In addition, when you set it so that k is at a bigger value, it is possible that there is going to be more bias in the model as you work on it too.

While you are working with this one, though, there may be times when you will need to go through and create some dummy variables. This is going to make it easier to figure out the categorical variables that will show up in this algorithm. This is different than the regressions that we will look for though because you can work with creating the k dummies rather than just the k-1.

With this in mind, we need to take a look at the best way to handle finding these k values in the first place. This is often done with the use of cross-validation. It is going to be important to use this process in order to estimate what the error of validation will be. To make this happen, we will need to hold out a subset of the training set from the process of building up the model.

Cross-validation is going to involve us going through and dividing up our training data randomly. We are going to work with a 10 fold validation, so that means we would want to divide up the training sets that we have into 10 groups. We want to keep them as close to the same in size as possible as we go through the dividing. From this, 90 percent of our data is going to be the kind that we use to train our model. The other ten percent or so will be used to help validate the model that we are working with and to test whether or not it is working.

The misclassification rate that we need to focus on for this one is going to be computed when we look at the ten percent that you saved back for the validation. This procedure is going to need to go through and repeat itself ten times because of how we are doing all of this. Each of the groups of observations that we run into is going to be seen as validation, and then you can test it as well.

Decision Trees and Random Forests

Often, the decision tree and the random forest are going to work together. These are going to be efficient tools of data that will help you to take two of the choices that you would like to work with, especially when the choices are very different, and then will use this information in order to help you pick out which decision is the best for your needs so that you can grow your business and more.

When you are presented with more than one option, and they all look like they are good options to work with, the decision tree is going to be a good option to choose along the way. These will help you to take some of these choices and then see what the possible outcomes may be with these, making it easier to figure out what is the best course of action to take.

Now, you will find that there are a few different ways that you are able to work with these decision trees. Many of those who are working with machine learning will use it if either of their variables is categorical, and one is random. However, there are times when you will need to use these decision trees with some of the classification problems that you have. To ensure that you are picking out and creating your decision tree well, then you need to make sure that you take all of the sets of data that you have and then split them up to be in two or more sets, with some similar data in each one. You can

then sort this out with the help of independent variables because it will help you to set it up the way that the decision tree needs.
Sometimes the decision tree is not to be what we need, and we will find that it is better to have more than one decision tree to get the work that we want. This is when the decision tree is going to be turned over to a random forest. These are popular to work with because they allow you to look at many possible, decisions that you want to make, and come up with the one that you would like to work with. So, the best way to think about these random forests is that they are going to be a bunch of different decision trees that are going to work together.

There are going to be many applications of using the random forest. This is because the random forest is perfect most of the time, it is going to do a better job of providing you with some insights and predictions than some of the other algorithms. Some of the ways that you are able to use these forests and make sure that they will benefit you include:

- When you are working on your own training sets, you will find that all of the objects that are inside a set will be generated randomly, and it can be replaced if your random tree things that this is necessary and better for your needs.
- If there are M input variable amounts, then m<M is going to be specified from the beginning, and it will be held as a constant. The reason that this is so important because it means that

each tree that you have is randomly picked from their own variable using M.

- The goal of each of your random trees will be to find the split that is the best for the variable m.
- As the tree grows, all of these trees are going to keep getting as big as they possibly can. Remember that these random trees are not going to prune themselves.
- The forest that is created from a random tree can be great because it is much better at predicting certain outcomes. It is able to do this for you because it will take all prediction from each of the trees that you create and then will be able to select the average for regression or the consensus that you get during classification.

Random forests are a good tool that a programmer is able to use when they would like to make sure that they add in some data science to the machine learning that you are doing, and there are going to be many benefits. But any time that you are looking for an easy way to look through some of the options that are available for your work, and you want help making some smart decisions, then the decision trees and random forests will be the best option for you to choose.

Working with Support Vector Machines

We can also spend some time working with the support vector machines, or SVM. These are going to be there to help us take each set of the data and then plot them so that they will show up on one n-dimensional of N. N is going to be the number of features that you would like to work with all of this. You will then be able to take the value of the features and work to translate this over to the value that you will need for your chosen coordinates. The job that you are able to do when it is time to reach this point is to figure out where your hyperplane will fall because this is going to be the part that will show you what differences are there between the classes that show up.

Here you may notice that it is possible that more than one support vector is going to show up. The good news is that many of these are obviously not going to be important, and they are just going to be the coordinates of the individual observations that you are going to see here. Then you are able to work with the SVM to turn into your frontier, the part that is able to separate these parts into classes, and then there will be the line and the hyperplane, which are the two parts that we need to focus on the most.

Up to this point, some of the work that we are looking at will seem a bit confusing. But there are a few steps that we are able to follow in order to really find out how to sort this data and use the SVM for our needs. First, we need to look for our own hyperplane. One thing that you will notice is that this algorithm is going to bring out more than one hyperplane that we can focus on. This is a challenge for beginners because you want to make sure that the hyperplane you pick is going to be the best one for sorting through the data and making it work for your needs.

The good thing to remember here is that even if you do have a few options when it comes to hyperplanes, there are still going to be some easy steps that we are able to use to help us pick out the right one. The specific steps that you are able to use when trying to figure out the hyperplane for your SVM will include:

- We are going to start out with three hyperplanes that we will call 1, 2, and 3. Then we are going to spend time figuring out which hyperplane is right so that we can classify the star and the circle.

- The good news is there is a pretty simple rule that you can follow so that it becomes easier to identify which hyperplane is the right one. The hyperplane that you want to go with will be the one that segregates your classes the best.

- That one was easy to work with, but in the next one, our hyperplanes of 1, 2, and 3 are all going through the classes, and they segregate them in a manner that is similar. For example, all of the lines or these hyperplanes are going to run parallel with each other. From here you may find that it is hard to pick which hyperplane is the right one.

- For the issue that is above, we will need to use what is known as the margin. This is basically the distance that occurs between the hyperplane and the nearest data point from either of the two classes. Then you will be able to get some numbers that can help you out. These numbers may be closer together, but they will point out which hyperplane is going to be the best.

The Neural Networks

We would be working with the Scikit-Learn library in this process of machine learning, and one that can handle a lot of really things for machine learning will be the neural networks. These are used quite a bit because they will work similar to the human brain, picking up on different patterns and more, and forming stronger connections each time that something is correct with its predictions.

When we are working with these neural networks, we will find that there are often a lot of layers, and each of these layers is going to be spending time to see whether there are some patterns there are not. If the network is able to find that new pattern, then they will go on through to the next layer. And this process will continue until there are no more patterns for the process to find, and until we are done and the neural network is able to make some predictions as well.

There are a few things that will happen at this point, based on how the program works. If the algorithm went through the process above and was able to sort through all of the different layers, it will then make a prediction. If that prediction is right, the neurons in the system will turn out stronger than ever. This is because the program has used artificial intelligence in order to make some strong associations between the patterns and the object. The more times that the system can come back with the right answer, the more efficient it will be when you turn it on and use it again.

Now, this may seem a little bit farfetched, but a closer examination of these neural networks will help us to see how they work together and why they are so important. For our example, let’s say that your goal is to create a program that is able to take a picture that you input into it, and then, by looking at that picture and going through the layers, the program is able to recognize that the image in that picture is that of a car.

If the program has been set up in the proper manner, it is going to make the right prediction that there is a car in the picture. The program is able to come up with this prediction based on some of the features that it already knows belongs to the car, including the color, the number on the license plate, the placement of the doors, the headlights, and more.

With this one, we need to make sure to remember there is the potential for many layers to show up, but the good news is that the more layers we are able to go through with our images, the more accurate the predictions are going to be overall. If your neural network can make some accurate predictions, then it is going to be able to learn this lesson and will hold onto it along the way and will get faster and more efficient at making the predictions later on.

The neat thing that happens when we are working with these neural networks is that they are able to remember some of the work that they have done in the past. So, if you present the neural network with a picture of a car, and it makes the prediction that the image in that picture is a car, it will remember this information later, similar to what the human mind can do.

Then, if you present it with a picture of a car, especially if this new image is similar to the one that you showed to the algorithm earlier, it is going to remember what it learned before. The algorithm will get through the various layers of the image really quickly and can give a prediction of a car in much less time than before. And this process continues on, with the neural networks getting better at predictions the more times that it is able to go through the information and try out its skills. Just think about all of the ways that we would be able to work with this kind of technology, and this algorithm, to get some of our machine learning algorithms done and taken care of.

Chapter 7: Validation and Optimization Techniques

Now that we have taken a look at a few of the different algorithms that go with machine learning, it is time for us to take a look at some of the ways that we can make these algorithms a little bit better. We would be looking at the validation of the algorithm to make sure that it is working the way that we want, and then, we will focus on how to optimize the techniques that we are working on so that we get the best predictions and insights that we are able to get out of those algorithms. So, let’s dive in and see what we are able to do with some of these techniques to make them work for our needs.

Cross-Validation Techniques

The first validation technique that we need to work with is known as the cross-validation technique. We are going to work on our machine learning algorithms here, and at the same time, we are going to take our set of data and divide it into three parts. We are going to have the set for training, the set for validation, and the set for testing.

The training set is the first one that we will look at. This is the one that we are going to use to help train the model. We will want to put about 60 percent of the data that we have available to work on training the model to make sure that it is ready to go.

Then we are going to work with the data set that handles the validation. Once we have been able to select out a model that can perform well with the training set, it is time to run the model with our validation set. This is going to be a small subset of the data, and it is usually going to range from 10 to 20 percent of the data that you have. This set is going to help us with these models because it is going to give us an evaluation, without bias, of the fitness of the model. If the error on the data set for validation increases, then it is possible that we are working with a model that overfits.
And finally, we have the test data set. This is going to be new data that has never been used in training at all. This is going to be a bit smaller, but it is going to contain about 5 to 20 percent of the set of data that we have, and it is meant to help us test out the model

evaluation that we are working on to see whether it is accurate or not.

In some cases, there is going to be training and a test set, and the programmer is not going to work with any validation set. There are some issues with this one, though. Due to the sample variability between the test set and the training, the model is going to provide us with a better prediction on the data that we train but will fail to generalize on the test data. This can make us deal with a low error rate during training, but a high rate of an error on the testing phase of this process.

When we go through and split out the set of data that we have into training, test, and validation set, we are going to work with just a subset of data, and then we will know when it is possible to train on fewer observations of the model are not going to perform well, and then we will see that it is going to give us an overestimated test error rate.

To help us solve both of these issues, we are going to work with cross-validation. This is technique involves partitioning the data so that it all goes into subsets. This allows us to train the data on one of the subsets, and then we will use the other one to help us to evaluate the performance of the model that we are working with as well.

To help us out here and to make sure that we reduce how much variability shows up in our data, we may go through and perform many rounds of this cross-validation, but we are going to do this with different subsets of the same data. We can then combine the validation results form these rounds in order to come up with a good estimate of the predictive performance that we are going to be able to get from that model. The cross-validation then is going to provide us with an estimate of the performance of the model that is more accurate than just training once and then assuming it is all going to work.

With this in mind, there are going to be a few different techniques that we are able to see with cross-validation, and these are going to include:

1. Leave one out cross-validation or LOOCV: In this one, we are going to take our set of data and divide it into two pairs to work on. In the first part, we are going to have a single observation, which is going to be the test data. And then, in the second one, we are going to have all of the other observations that come in our set of data, and these will form up our training data.

    a. There are a few advantages to working with this one. First, we are going to find that there is far less bias because we are going to use all of the set of data for training compared to some of the validation set approach where we are only working with part of the data to help with training.

    b. There isn't going to be any randomness in the training or the test data because we will perform this many times, and it will still give us the same results.

c. There are some disadvantages that come with this one as well. For example, MSE is going to vary as the test data is going to work with just one single observation. This sometimes adds some variability to work. If the data point that you work with ends up being an outlier, then you will find that the variability is going to be much higher.

d. The execution of this model is going to be more expensive than some other options because the model has to be fitted n times rather than just once or twice.

2. K Fold cross-validation: This is going to be a technique of cross-validation that is going to take the set of data and randomly divide it into k groups or folds that are similar in size. The first fold that you have is going to be used for testing, and then the model is going to be trained on k-1 folds. The process is going to be repeated K amount of times, and each time that you do, this will have a different group of the data that you will use for validation.

   a. There are a few advantages that come with this one. First, the computation time is going to be reduced as we go through the process 10 times, or less, depending on what value you give to k.

- **b.** This one is also going to have a reduced bias, so you can rely on the information that you have more.

- **c.** Every point of data gets to be tested just once and is used in training the k-1 times.
- **d.** The variance of the resulting estimate is going to be reduced the number of times that k increases.

- **e.** There are some disadvantages of k fold or the 10-old cross-validation. The training algorithm, compared to some of the other options, is going to be computationally intensive because the algorithm has to start over again and rerun from scratch k times to be effective.

3. Then we can work with the stratified cross-validation. This is a technique where we rearrange the data in a manner that each fold is going to be a proper representation of the set of data it is going to force the process so that each fold has to have at least m instances of each class. This type of approach is going to ensure that one class of data will not be over-represented, especially when the variable you are using as the target is not balanced well.

- **a.** For example, we may work on a binary classification problem where we would like to predict if a person on the Titanic was a survivor or not. We are going to have two classes here; the passenger either survives or doesn’t survive. We will then ensure that each fold is going to have a percentage of passengers who survived, and another percentage of the passengers who did not make it.

4. The time-series cross-validation: Splitting up the time series that you have in a random manner is not going to help out as much because the time-related data is going to get all messed up. If we are working on predicting the prices of the stocks and then we randomly split up the data, this is just going to make things difficult. This is why we would want to work with a time series cross-validation. In this one, each day is going to be a test data, and then we would consider the data that we had from the day before as part of our training set.

    a. We can start by training out the model with a minimum number of observations, and then we will use the data for the next day to help test the data. And we keep moving through this set of data. This will ensure that we are able to consider the time-series aspect that comes with this prediction.

Hyperparameter Optimization

One thing that we need to spend a bit of time looking at is the idea of hyperparameters. These are properties that are specific to the model that we are working with, ones that are going to be fixed even before we have a chance to train or test the data that we have with the model.

We are able to see one of these examples when we are working with a random forest. The hyperparameter is going to include the number of decision trees that we are able to find in our forest to start with. When working with the neural network, there is going to be a learning rate, the number of layers that are hidden, the number of units that we would like to see come with each layer, and a variety of other parameters along the way.

When we bring up the topic of hyperparameter tuning, we are talking about nothing outside of searching for the right set of hyperparameters in order to achieve the high precision and accuracy that we want. When we optimize these hyperparameters, it is going to end up being one of the trickiest and often one of the hardest parts of building a model up with machine learning.

The main aim that a programmer is going to have when it comes to tuning their hyperparameters is to find the sweet spot. This sweet spot in the parameters of the model is important because it ensures that we are able to get the best performance on our project as possible. There are going to be a few techniques that we can use for the parameter tuning, but we are going to focus on the grid search and the random search in the next section because these are the most widely-used options for parameter optimizing.

Grid and Random Search

The final thing that we are going to focus on in this chapter is the idea of the grid search versus the random search. This will help us to figure out which of the two is going to be better for the work that we want to accomplish. Before we look too much into this though, we need to review the hyperparameter optimization that we talked about earlier, because this is going to be important to some of the work that we are trying to do in this section.

First, we are going to take a look at grid searching. This is where we are going to try every combination of a present list of values of the hyperparameters, and then we are going to do an evaluation of the model with each of these combinations. The pattern that we will follow on this one is going to be similar to what we are able to see with a grid because each of the values is going to be placed into the matrix. Each set of parameters can then be taken into consideration, and we will note the accuracy. Once all of the combinations are evaluated, the model that has the set of parameters that provides us with the most accurate overall is considered the best one to work with.

While this is still a pretty straightforward option to work with, one of the biggest issues that we are going to face with it is when it comes to dimensionality; it is going to suffer when the number of these parameters starts to grow. With as few as four parameters in place, the problem can almost be impractical because the number of evaluations that we need to try to work on with this strategy is going to increase. And when we add in more of these parameters, the dimensionality is just going to make the problem worse.

There are times when we are going to use the grid search, but keep in mind that there are times when it is going to take too long and be too complex. This is when we will work with a random search. This is going to be a technique where some of the random combinations of the hyperparameter are going to be used to help us find the best solution for the model that we have built.

In many cases, this search is going to go through the information and will try out some combinations that are random for the range of values. To help optimize this random search, the function is going to be evaluated at some number of random configurations of the parameter space, as well.

The chances of finding an optimal parameter that you can use are going to be quite a bit higher with the random search because the pattern is going to be rained on the optimized parameters without needing to know any aliases. Random search is going to work the best when we have lower-dimensional data since the time that is taken to find the right set for this is going to be less when you have less iteration to work with.

In many cases, the random search is going to be the best technique here, especially when we have fewer dimensions to work with. There are going to be many practical and theoretical concerns when evaluating these strategies. The strategy that is best for your particular problem, though, is going to be one that finds the best value for the fastest and with the fewest function evaluations and it is possible that this is going to vary one problem to the next.

While it is less common in machine learning than the grid search, this random search is going to show us that we are able to get equal, and sometimes better, values compared to the grid search within fewer evaluations of the functions for some of the problems that we try to work with. You have to decide which method you think is the best for the kind of project that you want to work with at the time.

Chapter 8: Unsupervised Machine Learning with Clustering

Unsupervised machine learning is going to be able to help us out with a variety of problems as we handle some of our algorithms. There are times when we need to go through and sort some of the data we have, and we want to be able to make the machine do the work. Being able to handle clustering is a great way to work with unsupervised machine learning because this ensures that we are able to really see where some of our data points lie and can show us some of the hidden insights and predictions and patterns that are there, many of which we did not know about ahead of time. Some of the unsupervised machine learning options that you can do along with clustering will include:

K-Means Clustering

The first type of unsupervised machine learning that works with clustering is going to be the K-means clustering. This clustering is a good way to take care of all the different data points that we have, and see where they are going to be grouped together. You can choose how many groups of clusters you would like. If you are working with separating your customers into genders, then you may only have two clusters. But when you are working with the ages of the customers or even the geographic regions of the customers, then you may end up with five or more clusters.

The idea that comes with this one is that any of the data points that are in the same cluster are going to be closely related to one another. They are not going to have a lot of similarities to the other points that are in the other clusters that you have. This is important because it allows us to see where all of the points of data are going to be placed and will ensure that we are going to see the best results with this in no time at all.

One place where you may see this data clustering happening is when we are working with data mining. This data mining will really work with the clustering if it is more exploratory in nature. You can also work with clustering in other fields based on what we are trying to find out, such as with pattern recognition, lots of machine learning, image analysis, and computer graphics.

The K-Means clustering algorithm is going to form some clusters in your data based on how similar the data values will be. You can then go through and specify what you would like the value of K to be. The value of K is basically going to be the number of clusters that you would like to separate your data out into. The algorithm will be able to help you from here by selecting the center point for your clusters so that the data points fit in.

Then there are going to be three steps that the algorithm will need to go through including:

1. You will want to start with the Euclidian distance between each data instance and the centroids for all of the clusters.

2. Assign the instances of data to the cluster of centroid with the nearest distance possible.

3. Calculate the new centroid values, depending on the mean values of the coordinates of the data instances from the corresponding cluster.

To work with this kind of process, we have to make sure that we can go through and figure out how many clusters we would like to have in the first place. This helps to tell the algorithm where to place all of your data points, and when you print off the visual that goes with this, you will find that it can really help you to see where the data is going to fall, and how all of the different points are meant to go with one another as well. You may even be able to look at this to find a new cluster, and figure out a new market or a new customer base to organize with as well.

There are a lot of different things that we are able to do when it comes to working with the K-means clustering algorithm, but one of the things that we are going to spend some time looking at here is information and the codes that we need to focus on in order to figure out and add in the soft k-means to our code.

Now that we know a bit about the k-means algorithm in general, and we know some of the different ways that we are able to make this work for our needs, it is time to actually take some of these skills and use some Python code in order to make this algorithm work in machine learning. And implementing the soft k-means and the code that we will have below is one of the best ways to make this happen.

To get started with this process, we need to make sure that we start out with some of the standard imports and libraries that are needed, and that we have the utility functions in place as well. This is important because it is going to help us to get something similar to the Euclidean distance, and the cost function going together. The syntax of Python code that we are able to use with this one will include the following:

```
import numpy as np
import matplotlib.pyplot as plt


def d(u, v):
    diff = u - v
    return diff.dot(diff)


def cost(X, R, M):
```

```
    cost = 0
    for k in xrange(len(M)):
        for n in xrange(len(X)):
            cost += R[n,k]*d(M[k], X[n])
    return cost
```

After this part, we are going to take the time to define your function so that it is able to run the k-means algorithm before plotting the result. This is going to end up with a scatter plot where the color will represent how much of the membership is inside of a particular cluster. We would do that with the following code.

```
def plot_k_means(X, K, max_iter=20, beta=1.0):
    N, D = X.shape
    M = np.zeros((K, D))
    R = np.ones((N, K)) / K

    # initialize M to random
    for k in xrange(K):
        M[k] = X[np.random.choice(N)]

    grid_width = 5
    grid_height = max_iter / grid_width
    random_colors = np.random.random((K, 3))
    plt.figure()
```

```
    costs = np.zeros(max_iter)
    for i in xrange(max_iter):
        # moved the plot inside the for loop
        colors = R.dot(random_colors)
        plt.subplot(grid_width, grid_height, i+1)
        plt.scatter(X[:,0], X[:,1], c=colors)

        # step 1: determine assignments / resposibilities
        # is this inefficient?
        for k in xrange(K):
            for n in xrange(N):
                R[n,k] = np.exp(-beta*d(M[k], X[n])) / np.sum( np.exp(-
beta*d(M[j], X[n])) for j in xrange(K) )

        # step 2: recalculate means
        for k in xrange(K):
            M[k] = R[:,k].dot(X) / R[:,k].sum()

        costs[i] = cost(X, R, M)
        if i > 0:
            if np.abs(costs[i] - costs[i-1]) < 10e-5:
                break

   plt.show()

def main():
    # assume 3 means
    D = 2 # so we can visualize it more easily
```

```
    s = 4 # separation so we can control how far apart the means are
    mu1 = np.array([0, 0])
    mu2 = np.array([s, s])
    mu3 = np.array([0, s])

    N = 900 # number of samples
    X = np.zeros((N, D))
    X[:300, :] = np.random.randn(300, D) + mu1
    X[300:600, :] = np.random.randn(300, D) + mu2
    X[600:, :] = np.random.randn(300, D) + mu3

    # what does it look like without clustering?
    plt.scatter(X[:,0], X[:,1])
    plt.show()

    K = 3 # luckily, we already know this
    plot_k_means(X, K)

    # K = 5 # what happens if we choose a "bad" K?
    # plot_k_means(X, K, max_iter=30)

    # K = 5 # what happens if we change beta?
    # plot_k_means(X, K, max_iter=30, beta=0.3)


if __name__ == '__main__':
    main()
```

Hierarchal Clustering

Along the same idea is the K-Means clustering, we also need to take a look at a method that is known as the hierarchal clustering. In statistics and data mining, this is going to be a method of analyzing clusters, where we are going to work to build up a hierarchy of clusters. There are going to be a few strategies that we are able to use to handle this kind of clustering, but often they are going to fall into one of two types, including:

1. ***Divisive**: This is going to be the top-down approach. This will include how all of the observations that you are using will start out in one cluster, and then these will be split up and move down through the hierarchy until you reach the end.*

2. **Agglomerative**: This one is going to be the opposite. You will start out with each observation falling in its own cluster, and then you are able to merge together pairs of the clusters as you go up the hierarchy that you are working with.

For the most part, you will be able to determine the splits and the merges in a greedier manner. The results of this are usually going to be presented with a dendrogram.

To help us determine which ones are going to be combined or split up, we need to be able to look for and measure out the dissimilarity between the sets of observations that we are working with. The good news is that with most methods of this kind of clustering, we are going to be able to make this happen with an appropriate metric, which is a measure of distance between observations and pairs, and a linkage criterion that is going to let us know the dissimilarity of sets as a function of the pairwise distances of observations in the sets we are working with.

DBSCAN

DBSCAN is stands for Density-based spatial clustering of applications with noise. This is going to be a pretty well-known data clustering algorithm that is going to be used in things like data mining and machine learning to help us to move our data into clusters so that we are able to read through it and understand better. Based on a set of points, this algorithm is going to be able to group together points that are close to one another based on some kind of distance measurement and a minimum point amount. It is also going to mark out some of the outliers that we have that are found in some of the lower-density regions to help us see where these outliers are.

To keep it simple, we will find that this kind of algorithm is going to come with 2 parameters that we need to know. These are going to include:

1. Eps: This is going to tell us how close the points need to be to one another before they can be seen as part of the cluster. However, if the distance between the two is either low or even equal, these would be considered as neighbors.
2. minPoints: This is going to be the minimum number of points that are needed to form a region that is dense. So if we set this parameter to 5, then we need to have at least five of these points in order to form a dense region.

Then we are able to move on to doing parameter estimation. This is going to be something that we need to focus on for every kind of task in data mining. To choose the right parameters, we have to understand how they are used and then have at least a basic previous knowledge about the set of data that we are going to work with.

For the eps from above, if the value that you choose is too small, then you will end up with a lot of your data not being clustered. It is going to be considered outliers because it won’t be able to provide the points. On another point, if the value that was chosen is too high, clusters will merge, and the majority of objects are going to fall into the same cluster. This means that we need to choose the eps based on the distance of the set of data, but in general, going with a smaller value for this is going to be preferable.

We can also work with the minPoints that we talked about before. As a rule, the minimum of this can be derived when we take the data set as minPoints greater than or equal to D + 1. Larger values are often going to be better for the sets of data that have a lot of noise and will form more significant clusters. The minimum value for the minPoints must be three, but larger the set of data, the larger the value that should be chosen.

There are a lot of reasons that we are able to use the DBSCAN algorithm for our needs. This algorithm is going to be a good one to use to find associations and structures in data that might be hard to get manually, but that is still useful and relevant to help you predict trends and find the patterns that you want. Clustering methods are going to be used in a lot of industries, and you will be able to use the DBSCAN to handle a lot of this as well along the way.

Any time that your business needs to work with an algorithm that can cluster together different points, the DBSCAN algorithm is a good one to use, it is a simple idea that you can reverse and do work in more than one method at the same time, and it can really help you to see which points belong to each cluster in no time as well.

The good news with this one is that this is an algorithm that a lot of programmers already use, which means that you will not need to go through and do the implementation on your own. You are able to use one of the various python packages or libraries in order to handle it. It is also able to work with R, Matlab, and Python. This is a also great way to separate out the data points that you have while making sure that you can get it all set up and ready to go in no time at all. When you are ready to put this to work for your needs, take a look at some of the options of clustering algorithms that are above to help you get started.

Chapter 9: Reduction of Dimensionality

Lastly, let us look at reduction and dimensionality. We are going to spend our time working with both the principal component analysis, and the linear discriminant analysis. We will then compare the two in order to figure out which is the best one to work with, and if we would want to work with each one individually or together. Let's dive in and see what the PCA and LDA are all about.

The Principal Component Analysis

The first option that we need to take a look at here is going to be the Principal Component Analysis or PCA. This is going to be one of those techniques that we are able to use with machine learning that will help us to work with the identification of a smaller number of variables that are uncorrelated, but they are known as the principal components that come from a much larger data set that we are working with.

This technique is going to emphasize on the variation of our data, and then it will capture some of the stronger patterns that are found in the set of data. Simply put, we are going to take some random variables out of our set of data, and then we are going to make sure they are not correlated, outside of being in the same data. But we hope to use these to help us figure out some of the strong patterns and predictions that are found in your set of data as well.

This is an analysis tool that was invented in 1901 by Karl Pearson, and it is going to be used in a number of different types of applications, including exploratory data analysis and predictive models. This analysis is going to be one of the statistical methods, and we will be able to use it in many industries, including computer graphics, neuroscience, face recognition, and image compression to name a few options.

The PCA is going to help us take our data and will make it easier to explore and visualize how this will work and what is inside of that data. It is going to be a pretty simple technique to work with, and it is non-parametric. And when it is used properly, it is going to help us to take out some of the most useful information that we need to form confusing and complex sets of data overall.

This form analysis is also going to focus its attention on the maximum variance amount with the fewest number of principal components as well. This is done to help us learn as much from the data while using as few data points as possible along the way. When we are able only to use a few points of data to get things done, we will find that it is much easier to make some of the predictions that we want, without having to worry about getting confused and lost with a lot of data.

There are a lot of advantages that come with using the PCA, but one of the distinct advantages that come with this is that once the patterns re-found in the data that you are looking for, you will also find support for compressing the data. One will be able to make sure of the PCA to help eliminate the number of variables that you are working with, or when there are going to be too many predictors present in your work compared to how many observations so that you avoid a problem that is known as multicollinearity.

Another thing that you may notice about the PCA is that it is going to be able to relate closely to the canonical correlation analysis, and will even use something known as the orthogonal transformation. The reason that it uses both of these is to help it convert the observations that you are using into a set of values that will then be the principal components.

The number of these principal components that we are going to use in this kind of analysis is going to be either less than or equal to the lesser number of observations that you want to work with as well. The PCA is going to be pretty sensitive when it comes to the relative scaling of the originally used variables.

There are many times when you will want to use this kind of analysis. For example, it is going to be used in any industry that relies on a large set of data, the social sciences, and market research. This technique can also help to provide us with a lower-dimensional picture of some of the data that we originally had. Only a minimal amount of effort is going to be needed when you use this analysis, even when you are trying to reduce all of that data that is confusing and overwhelming into a simplified set of information that you are able to use.

Linear Discriminant Analysis

Now that we know about the PCA, it is time for us to take a look at a Linear Discriminant Analysis or LDA, and how it is going to be used in machine learning in a slightly different manner than the first one that we talked about. In the LDA, we are going to find a well-established technique of machine learning and classification method that is going to be good at predicting the categories that we need the main advantages that we have with this one compared to some of the other classification algorithms is that the model is going to be easy to interpret, and they are good at making predictions as well.

The LDA is going to be used on a regular basis as a dimensionality reduction technique, and this can make it really easy to work with when you want to handle either classification or pattern recognition in some of your programs in machine learning.

The LDA is going to take a set of data cases, which is going to be known as the observations, and will use this as the input. For each of these cases, you will need to make sure that there is a categorical variable because these are responsible for defining the class, and then we need to have at least a few predictor variables, and we are going to see that these are numeric.

Often we are going to be able to take this input data and visualize it as a matrix, with each of the cases being a row, and then each of the variables being in a column. We can think about each of these cases as a point that will show up in the N-dimensional space. N is going to be the number of variables that we are using as predictors. Every point is going to be labeled by its category to make things a little bit easier.

The algorithm that we can use with LDA is going to use this data to help divide up the space of our predictor variables into regions. These regions are a bit more unique, and we are going to label them based on the categories that we can use, and they will have boundaries that are linear, which is where we get the L in our LDA. The model is going to work at predicting the category of a new unseen case, and it can do this according to which region it is going to lie in. The model will be able to predict that all cases that are inside one of these regions that we created are going to belong to the same category. And as long as we trained the algorithm in the proper manner, this is going to hold true.

The linear boundaries are going to happen because we assume that the predictor variables that we are able to get for each category are going to come with the same multivariate Gaussian distribution. This assumption is not always going to be true in practice, it is going to be fairly accurate, and if it is valid like this, then it is possible that the LDA will still be able to perform well and give us the insights and predictions that we need.

In a mathematical manner, this LDA is going to use the input data to help it to derive the necessary coefficients of a scoring function for all of the categories that we need. Each of these functions is able to take as arguments the numeric predictor variables of the case as well. It is then going to scale the variable going to the specific coefficients of that category, as well as the specific output of a score. The LDA model is going to look at the score that we are going to receive from each function, and then we are able to use the highest score to help us allocate the prediction or the case to a category. We are going to call then the scoring functions, which are important when it comes to helping us make predictions, the discriminant functions.

There are many times when we are able to work with the LDA to help various companies see the results that they would like. To start with, we may find that this can be used with the prediction of bankruptcy. This could happen on accounting ratios and some of the other financial variables. This was actually one of the first methods that were applied to help us explain which firms were able to survive and which ones would enter into bankruptcy.

We can also use the LDA for things like facial recognition. In some of the computerized options for facial recognition, each face is going to be represented with the use of many pixel values. The LDA is able to reduce the number of features that are present in the face to a number that is more manageable before we do the classification. Each of the new dimensions that show up will basically be a combination that is linear to the pixel values, which is then going to form a template. The combinations that are done are going to be known as Fisher's faces, while those that are obtained through the PCA that we talked about before will go by the name of eigenfaces.

Marketing can even work with the LDA on occasion. This can be used to go through a large set of data and distinguish some of the different types of customers and products on the basis of surveys and other forms of data that you were able to collect. These can help us to gather up the data after formulating the problem, estimate the discriminant function, plot the results in the end on a map that we can easily look over and understand in the process as well.

The next place where we are able to work with this is in biomedical studies. This can help us to get an assessment of the severity state of one of your patients and can even give a good prognosis of the outcome of the disease. For example, during the retrospective analysis, patients are going to be divided into groups according to how severe the disease is. Then the results of the analysis, from the clinic and the lab, are going to be studied to help us reveal some of the variables that are different in the studied groups.

When we work with these variables, discriminant functions are going to be built that can help us to classify diseases in the future patient into the severe, moderate, and mild form. This is the same kind of principle that can be used in the biology of different biological groups.

And finally, we will see that the LDA is going to be used to help out with the world of earth sciences in some cases. This method is going to be used to help us to separate out some of the zones of alteration that are there. for example, when we have different data from various zones available to us, this analysis is able to find the pattern within the data and can classify it all in an effective manner.

Comparing PCA and LDA

Now that we have had a chance to talk about the PCA and the LDA options, it is time to take a look at these in comparison to one another. Both of these techniques have a lot to bring to the table and understanding how these are meant to work and how we can combine them to get the best results is going to be imperative to some of the work that we can accomplish with them.

Both the PCA and LDA are going to be techniques of a linear transformation. One option that we are going to see here is that the LDA is going to be a supervised method of machine learning, while the PCA is going to be an example of unsupervised machine learning. This is because the PCA is going to ignore some of the class labels that are there.

A good way to look t the PCA is that it is one of the techniques that you can use that will find the directions of the maximal variance. On the other hand, the LDA is going to work to find a feature subspace in the data that is able to maximize the separability of the class.

Remember, in this that LDA is going to make some assumptions about the classes that are normally distributed and the covariance of the equal classes. This can be important based on some of the algorithms and projects that you are trying to work with along the way.

Many times there is going to be a lot of confusion for programmers when it is time to decide if they should use the LDA or PCA options for their applications. This is often because they are not going to understand some of the fundamental differences that happen between the LDA and PCA. Hopefully, with some of the help of the rest of this section, we are able to get a better idea of how these are similar and how they are separate.

Both the LDA and the PCA are going to be used in the pre-processing step when it comes to problems of pattern recognition and machine learning. The outcome that you are trying to get with both the LDA and PCA is that it will reduce the dimensions that are in our set of data with a minimal amount of information lost in the process. This is going to help reduce the costs of computation along the way, it can speed up how long the computation takes, and can really reduce the issues of overfitting because we are able to project our data over to a lower-dimensional space that will be able to describe the data a bit better.

The main difference that we are going to see between these two is that the PCA is an algorithm that is unsupervised because it is going to ignore the labels of the classes while working to maximize the variance that is able to show up in the set of data. The LDA is going to be slightly different as it is a supervised technique because it is going to compute the directions that are most likely to represent the axes that maximize the separation between the various classes as well.

When we are working with the LDA, rather than just finding the eigenvectors that will maximize the variance of the data, we are also going to have some interest in the axes that are able to maximize how much separation is going to show up between more than one classes. This is important because it is going to help us get this separability to the set of the data, which is something that will be ignored in many cases when it comes to the PCA.

Another difference that we are going to see with this one is that with PCA, we are not going to have the assumptions in place that the points of data are distributed in a normal way. But if the points of data come to us from other distributions, then the PCA is only able to approximate their features through the first few moments. This means that it is not going to be the most optimal options to go with unless the data points are being distributed in a normal manner.

Then we can switch it over to looking at the LDA. IN this situation, you are going to assume that the points of data that we are looking at are going to come to us from two separate multivariate normal distributions that have different means, while still having a covariance matrix that is the same. What this does for us is give us a more generalized method out of the LDA compared to what we are able to see with the PCA.

It is also important to figure out when and how we would visualize the plots that are needed with both LDA and PCA. The plots have been generated for these two algorithms with the help of the Scikit-Learn machine learning library, and with the help of the Iris Dataset. This is a good one to work with because it has 150 images of flowers in three classes, and each flower is going to come with 4 features. You would then be able to work with both of the options above in order to help you to figure out which flower, off of some images that you have, fit into each category.

This is going to bring up the question of when you would want to work with the PCA method and when you would want to work with the LDA method. As we have been going through this part of the guidebook, it may seem like the LDA is going to be the best technique to go with most of the time, but usually, this is not going to be the case. Comparisons will show us over time that the PCA method is often going to be able to outperform the LDA, if the number of samples that are in a class is relatively small, such as what we would be able to find in that Iris data set from above.

However, if you are planning on working with a really big set of data that has a lot of classes, the PCA is not going to work as well with this one, and it is important to work with the LDA method instead. This is due to the fact that class separability is going to be an important factor in helping us make sure that we are also reducing the dimensionality.

One final note before we finish off with this idea is that it is possible to work with the PCA and the LDA together. This will allow you to get some of the benefits of both of these options, without having to worry about some of the negatives with them as much. There are many opportunities when we need to use this kind of option, but it can really add to another level of power when it is time to handle some of the data that we have with machine learning.

Conclusion

Thank you for making it through to the end of *Python Machine Learning*, let's hope it was informative and able to provide you with all of the tools you need to achieve your goals whatever they may be.

The next step is to start working with some of the different algorithms that we have in this guidebook. There are many times when working with machine learning and good data analysis will be able to help your company to see some results. But first, you need to take the time to collect the right data and then run it through a properly trained and tested algorithm to help you get the right insights and predictions that you need.

These are just some of the topics that we are going to explore when it comes to machine learning, and one of those is being able to pick out the right algorithm for machine learning, and figuring out how to put data through each one to make it work is going to be hard. There are just so many Python machine learning algorithms out there, and many of them sound great that it can be confusing to know how to make them run the way that you want.

This is why this guidebook spent time exploring the different algorithms, and discussed in-depth information about how these work and what you are able to do with each one. The most of common algorithms like neural networks, random forests and decision trees, clustering, KNN, have been discussed as well. When you are done, you will have a good idea of how to work with machine learning and how to make all of this work on your machine learning project.

There are many times when you may decide to work with data analysis or some of the other parts of machine learning, and knowing which algorithms to choose is going to be imperative to this process.

★★★★★

If you found this book useful in any way, a review on Amazon is always appreciated !

J.H.L

Josh Hugh Learning

1000 Python Examples

Gábor Szabó

This book is for sale at http://leanpub.com/python-examples

This version was published on 2020-05-28

* * * * *

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

* * * * *

Table of Contents

First steps

What is Python?

- A snake.
- A British comedy group called Monty Python.
- A programming languge. The definition of the language: words, punctuation (operators) and grammar (syntax).
- The compiler/interpreter of the Python programming language. (aka. CPython).

> When people say they Python in relation to programming they either mean the Python programming language or they mean the tool that can translate some text (code) written in the Python programming language to the language a computer can actually understand. On MS Windows this is the **python.exe** you need to install. On Linux/Mac it is usally called **python** or **python3**. The generic name of the tool that translates a programming language for the computer is eiter called a compiler or an interpreter. We'll talk about this later on.

What is needed to write a program?

- An **editor** where we can write in a language.

- A **compiler or interpreter** that can translate our text to the language of the computer.

In order to write and run a program you basically need two things. A text editor in which you can write the program and a compiler or interpreter that can translate this program to the computer.

The source (code) of Python

- Python

Python 2 vs. Python 3

- Python 2.x - old, legacy code at companies, answers on the Internet. Retires on January 1, 2020.
- Python 3.x - the one that you should use. (not fully backward compatible) Available since December 3, 2008.

Python has two major lines the version 2.x and the version 3.x. In a nutshell you **should** always use Python 3 if possible.

Unfortunately you can still encounter many companies and many projects in companies that are stuck on Python 2. In such cases you probably will have to write in Python 2.

In addition when you search for solutions on the Internet in many cases you'll encounter solution that were written for Python 2. Luckily in most of the cases it is almost trivial to convert thise small examples to work on Python 3.

You just need to be able to recognize that the code was originally written for Python 2 and you need to be able to make the adjustments.

For this reason, while the majority of these pages cover Python 3, we are going to point out the places where it might be useful to know how Python 2 works.

You are free to skip these parts and come back to them when the need arises.

Installation

- MS Windows
- Linux
- Apple/Mac OSX

We are going to cover how to install Python all 3 major operating systems.

Installation on Linux

- On Linux you usually have Python 2 installed in **/usr/bin/python**
- Python 3 in **/usr/bin/python3**.
- If they are not installed, you can install them with the appropriate **yum** or **apt-get** command of your distribution.
- An alternative is to install Anaconda with Python 3.x

```
$ which python3

$ sudo apt-get install python3
$ sudo yum install python3
```

Installation on Apple Mac OSX

- On Mac OSX you can have Python 2 installed in **/usr/bin/python** and Python 3 installed as **/usr/bin/python3**.
- Homebrew
- An alternative is to install Anaconda with Python 3.x

```
$ which python3

$ brew install python3
```

Installation on MS Windows

- Anaconda with Python 3.x
- Anaconda shell
- Anaconda Jupyter notebook

*

- An alternative is to install from here.

Editors, IDEs

> Basically you can use any text editor to write Python code. The minimum I recommend is to have proper syntax highlighting. IDEs will also provide intellisense, that is,
> in most of the cases they will be able to understand what kind

of objects do you have in your code and will be able to show you the available methods and their parameters. Even better, they provide powerful debuggers.

PyCharm seems to be the most popular IDE. It has a free version called community edition.

Linux

- Emacs
- vi, vim, gvim
- spf13-vim
- Kate
- Gedit
- jEdit

Windows

- Notepad++
- Textpad
- Ultra Edit

Mac

- CotEditor
- TextWrangler
- TextMate
- Type “text editor” in your Apple Store (filter to free)

All platforms

- Sublime Text (commercial)
- Ligth Table

IDEs

- PyCharm community edition
- Visual Code of Microsoft

- Spyder, a scientific environment (included in Anaconda)
- Jupyter with IPython behind the scene.

- IDLE (comes with Python)
- Komodo of ActiveState
- Aptana
- Pyscripter
- PyDev (for Eclipse)
- Wing IDE
- Atom

Documentation

- Google
- Bing
- DuckDuckGo
- official documentation of Python
- Stack Overflow
- Code Maven
- …

Program types

- Desktop application (MS Word, MS Excel, calculator, Firefox, Chrome, …
- Mobile applications - whatever runs on your phone.
- Embedded applications - software in your car or in your shoelace.
- Web applications - they run on the web server and send you HTML that your browser can show.
- **Command Line Applications**
- Scripts and programs are the same for our purposes
- …

Python on the command line

More or less the only thing I do on the command line with python is to check the version number:

```
python -V
python --version
```

You can run some Python code without creating a file, but I don’t rememeber ever needing this. If you insists

```
python -c "print 42"
```

```
python3 -c "print(42)"
```

Type the following to get the details:

```
man python
```

cmdline

First script - hello world

```
print("Hello World")
```

- Create a file called **hello.py** with the above content.
- Open your terminal or the Anaconda Prompt on MS Windows in the directory (folder)
- Change to the directory where you saved the file.
- Run it by typing **python hello.py** or **python3 hello.py**
- The extension is .py - mostly for the editor (but also for modules).
- Parentheses after print() are required in Python 3, but use them even if you are stuck on Python 2.

Examples

- [The examples are on GitHub]()
- You can download them and unzip them.

Comments

> # marks single line comments.
>
> There are no real multi-line comments in Python, but we will see a way to have them anyway.

```
print("hello")

# Comments for other developers

print("world") # more comments
```

```

# print("This is not printed")
```

Variables

```
greeting = "Hello World!"
print(greeting)
```

Exercise: Hello world

Try your environment:

- Make sure you have access to the right version of Python.
- Install Python if needed.
- Check if you have a good editor with syntax highlighting.
- Write a simple script that prints **Hello world.**
- Add some comments to your code.
- Create a variable, assign some text to it and then print out the content of the variable.

What is programming?

- Use some language to tell the computer what to do.
- Like a cooking recepie it has step-by-step instructions.
- Taking a complex problem and dividing it into small steps a computer can do.

What are the programming languages

- A computer CPU is created from transistors, 1 and 0 values. (aka. bits)

- Its language consists of numbers. (e.g 37 means move the content of ax register to bx register)
- English? too complex, too much ambiguity.
- Programming languages are in-beteen.

A written human language

- Words
- Punctuation: - . , ! ?
- Grammar
- …

A programming language

- Built-in words: print, len, type, def, …
- Literal values: numbers, strings
- Operators: + - * = , ; …
- Grammar (syntax)
- User-created words: variables, functions, classes, …

Words and punctuation matter!

- What did you chose? (Correctly: choose, but people will usually understand.)
- Lets do the homework. (Correctly: Let’s, but most people will understand.)
- Let’s eat, grandpa!
- Let’s eat grandpa!
- see more

- Programming languages have a lot less words, but they are very strict on the grammar (syntax).
- A mising comma can break your code.
- A missing space will change the meaning of your code.
- An incorreect word can ruin your day.

Literals, Value Types in Python

```
print( type(23) )         # int
print( type(3.14) )       # float
print( type("hello") )    # str

print( type("23") )       # str
print( type("3.24") )     # str

print( type(None) )       # NoneType
print( type(True) )       # bool
print( type(False) )      # bool

print( type([]) )         # list
print( type({}) )         # dict

print( type(hello) )      #  NameError: name 'hello' is not defined
print("Still running")
```

```
Traceback (most recent call last):
  File "python/examples/basics/types.py", line 15, in <module>
    print( type(hello) )   # str
NameError: name 'hello' is not defined
```

- Strings must be enclosed in quotes.
- Numbers must be NOT enclosed in quotes.

Floating point limitation

```
print(0.1 + 0.2)   # 0.30000000000000004
```

* floating point

Value Types in Numpy

Numpy but also other programming languages might have them.

* int8
* int32
* float32
* float64
* …

Rectangular (numerical operations)

```
width = 23
height = 17
area = width * height
print(area)    # 391
```

Multiply string

```
width = "23"
height = "17"
area = width * height
print(area)
```

```
Traceback (most recent call last):
  File "python/examples/basics/rectangular_strings.py", line 3, in <module>
    area = width * height
TypeError: can't multiply sequence by non-int of type 'str'
```

Add numbers

```
a = 19
b = 23
c = a + b
print(c)    # 42
```

Add strings

```
a = "19"
b = "23"
c = a + b
print(c)    # 1923
```

Exercise: Calculations

- Extend the rectangular_basic.py from above to print both the area and the circumference of the rectangle.
- Write a script that has a variable holding the radius of a circle and prints out the area of the circle and the circumference of the circle.
- Write a script that has two numbers a and b and prints out the results of a+b, a-b, a*b, a/b

Solution: Calculations

```
width = 23
height = 17
area = width * height
print("The area is ", area)    # 391
circumference = 2 * (width + height)
print("The circumference is ", circumference)    # 80
```

```
r = 7
pi = 3.14
```

```
print("The area is ", r * r * pi)            # 153.86
print("The circumference is ", 2 * r * pi)   # 43.96
```

```python
import math

r = 7
print("The area is ", r * r * math.pi)            # 153.9380400258998
print("The circumference is ", 2 * r * math.pi)   # 43.982297150257104
```

```python
a = 3
b = 2

print(a+b)    # 5
print(a-b)    # 1
print(a*b)    # 6
print(a/b)    # 1.5
```

Second steps

Modules

```
import sys

print( sys.executable )                 # /home/gabor/venv3/bin/python
print( sys.platform )                   # linux
print( sys.argv[0] )                    # examples/basics/modules.py
print( sys.version_info.major )         # 3

print( sys.getsizeof( 1 ) )             # 28
print( sys.getsizeof( 42 ) )            # 28
print( sys.getsizeof( 1.0 ) )           # 24

print( sys.getsizeof( "" ) )            # 49
print( sys.getsizeof( "a" ) )           # 50
print( sys.getsizeof( "ab" ) )          # 51
print( sys.getsizeof( "abcdefghij" ) )  # 59
```

A main function

```
def main():
    print("Hello")
    print("World")
```

This won’t run as the main function is declared, but it is never called (invoked).

The main function - called

You could write your code in the main body of your Python file, but using functions
and passing arguments to it will make your code easier to maintain and understand.
Therefore I recommend that you always write every script with a function called "main".

- Function definition starts with the **def** keyword, followed by the name of the new function ("main" in this case), followed by the list of **parameters in parentheses** (nothing in this case).
- The content or body of the function is then **indented** to the right.
- The function definintion ends when the indentation stops.

```
def main():
    print("Hello")
    print("World")

print("before")
main()
print("after")
```

```
before
Hello
World
after
```

- Use a main function to avoid globals and better structure your code.
- Python uses **indentation** for blocks instead of curly braces, and it uses the colon **:** to start a block.

Indentation

- Standard recommendations: 4 spaces on every level.

Conditional main

```
def main():
    print("Hello World")

if __name__ == "__main__":
    main()
```

- We’ll cover this later but in case you’d like, you can include this conditional execution of the main function.

Input - Output I/O

Input

- Keyboard (Standard Input, Command line, GUI)
- Mouse (Touch pad)
- Touch screen
- Files, Filesystem
- Network (e.g. in Web applications)

Output

- Screen
- File
- Network

print in Python 2

print is one of the keywords that changed between Python 2 and Python 3. In Python 2 it does not need parentheses, in Python 3 it is a function and it needs to have parentheses.

```
print "hello"
print "world"
print "Foo", "Bar"
```

```
hello
world
Foo Bar
```

```
print "hello",
print "world"
print "Foo", "Bar",
```

```
hello world
Foo Bar
```

No newline, but a space is added at the end of the output and between values.

```
import sys
sys.stdout.write("hello")
sys.stdout.write("world")
```

```
helloworld
```

write takes exactly one parameter

print in Python 3

```
print("hello")
print("world")
print("Foo", "Bar")
```

```
hello
world
Foo Bar
```

```
print("hello", end=" ")
print("world")
print("Foo", "Bar")
```

```
hello world
Foo Bar
```

end will set the character added at the end of each print statement.

```
print("hello", end="")
print("world")

print("Foo", "Bar", sep="")
print("END")
```

```
helloworld
FooBar
END
```

`sep` will set the character separating values.

print in Python 2 as if it was Python 3

```
from __future__ import print_function
print("hello", end="")
print("world")
```

```
helloworld
```

Exception: SyntaxError: Missing parentheses in call

What if we run some code with **print "hello"** using Python 3?

```
  File "examples/basics/print.py", line 1
    print "hello"
                ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello")?
```

Prompting for user input in Python 2

```python
from __future__ import print_function

def main():
    print("We have a question!")
    name = raw_input('Your name: ')
    print('Hello ' + name + ', how are you?')

main()
```

```
/usr/bin/python2 prompt2.py

We have a question!
Your name: Foo Bar
Hello Foo Bar, how are you?
```

What happens if you run this with Python 3 ?

```
/usr/bin/python3 prompt2.py
```

```
We have a question!
Traceback (most recent call last):
  File "prompt2.py", line 7, in <module>
    main()
  File "prompt2.py", line 4, in main
    name = raw_input('Your name: ')
NameError: name 'raw_input' is not defined
```

Prompting for user input in Python 3

> In Python 3 the **raw_input()** function was replaced by the **input()** function.

```
def main():
    print("We have a question!")
    name = input('Your name: ')
    print('Hello ' + name + ', how are you?')

main()
```

What happens if you run this using Python 2 ?

```
/usr/bin/python2 prompt3.py
```

```
We have a question!
Your name: Foo Bar
Your name: Traceback (most recent call last):
  File "prompt3.py", line 5, in <module>
    main()
  File "prompt3.py", line 2, in main
    name = input('Your name: ')
  File "<string>", line 1
    Foo Bar
          ^
SyntaxError: unexpected EOF while parsing
```

```
We have a question!
Your name: Foo
Your name: Traceback (most recent call last):
  File "prompt3.py", line 5, in <module>
    main()
  File "prompt3.py", line 2, in main
    name = input('Your name: ')
  File "<string>", line 1, in <module>
NameError: name 'Foo' is not defined
```

Python2 input or raw_input?

In Python 2 always use `raw_input()` and never `input()`.

Prompting both Python 2 and Python 3

```
from __future__ import print_function
import sys

def main():
    if sys.version_info.major < 3:
        name = raw_input('Your name: ')
    else:
        name = input('Your name: ')
    print('Hello ' + name + ', how are you?')

main()
```

Add numbers entered by the user (oups)

```
def main():
    a = input('First number: ')
    b = input('Second number: ')
    print(a + b)

main()
```

```
First number: 2
Second number: 3
```

```
23
```

When reading from the command line using `input()`, the resulting value is a string.
Even if you only typed in digits. Therefore the addition operator `+` concatenates the strings.

Add numbers entered by the user (fixed)

```
def main():
    a = input('First number: ')
    b = input('Second number: ')
    print(int(a) + int(b))

main()
```

```
First number: 2
Second number: 3
5
```

In order to convert the string to numbers use the `int()` or the `float()` functions.
Whichever is appropriate in your situation.

How can I check if a string can be converted to a number?

- stdtypes

```
val = input("Type in a number: ")
print(val)
print(val.isdecimal())
print(val.isnumeric())

if val.isdecimal():
```

```
    num = int(val)
    print(num)
```

```
Type in a number: 42
True
True
42
```

- We’ll talk about this later. For now assume that the user enters something that can be converted to a number.
- Use Regular Expressions (regexes) to verify that the input string looks like a number.
- Wrap the code in try-except block to catch any exception raised during the conversion.

Converting string to int

```
a = "23"
print(a)           # 23
print( type(a) )   # <class 'str'>


b = int(a)
print(b)           # 23
print( type(b) )   # <class 'int'>
```

```
a = "42 for life"
print(a)                    # 42 for life
print( type(a) )            # <class 'str'>

b = int(a)
print(b)
print( type(b) )

# Traceback (most recent call last):
#   File "converting_string_to_int.py", line 5, in <module>
#     b = int(a)
```

```
# ValueError: invalid literal for int() with base 10: '42 for life'
```

Converting float to int

```
a = 2.1
print( type(a) )    # <class 'float'>
print(a)            # 2.1

b = int(2.1)
print( type(b) )    # <class 'int'>
print(b)            # 2
```

```
a = "2.1"
print(a)           # 2.1
print( type(a) )   # <class 'str'>

b = int(a)
print(b)
print( type(b) )

# Traceback (most recent call last):
#   File "converting_floating_string_to_int.py", line 5, in <module>
#     b = int(a)
# ValueError: invalid literal for int() with base 10: '2.1'
```

```
a = "2.1"
b = float(a)
c = int(b)
print(c)                    # 2
print( type(a) )            # <class 'str'>
print( type(b) )            # <class 'float'>
print( type(c) )            # <class 'int'>

d = int( float(a) )
print(d)                    # 2
print( type(d) )            # <class 'int'>

print( int( float(2.1) ))   # 2
```

```
print( int( float("2") ))  # 2
print( int( float(2) ))    # 2
```

Conditionals: if

```
def main():
    expected_answer = "42"
    inp = input('What is the answer? ')

    if inp == expected_answer:
        print("Welcome to the cabal!")

main()
```

Conditionals: if - else

```
def main():
    expected_answer = "42"
    inp = input('What is the answer? ')

    if inp == expected_answer:
        print("Welcome to the cabal!")
    else:
        print("Read the Hitchhiker's guide to the galaxy!")

main()
```

Conditionals: if - else (other example)

```
def main():
    a = input('First number: ')
    b = input('Second number: ')

    if int(b) == 0:
        print("Cannot divide by 0")
    else:
        print("Dividing", a, "by",  b)
        print(int(a) / int(b))

```

```

main()
```

Conditionals: else if

```
def main():
    a = input('First number: ')
    b = input('Second number: ')

    if a == b:
        print('They are equal')
    else:
        if int(a) < int(b):
            print(a + ' is smaller than ' + b)
        else:
            print(a + ' is bigger than ' + b)

main()
```

Conditionals: elif

```
def main():
    a = input('First number: ')
    b = input('Second number: ')

    if a == b:
        print('They are equal')
    elif int(a) < int(b):
        print(a + ' is smaller than ' + b)
    else:
        print(a + ' is bigger than ' + b)


main()
```

Ternary operator

```
x = 3
answer = 'positive' if x > 0 else 'negative'
print(answer)   # positive
```

```
x = -3
answer = 'positive' if x > 0 else 'negative'
print(answer)   # negative
```

```
x = 3
if x > 0:
    answer = 'positive'
else:
    answer = 'negative'
print(answer)   # positive

x = -3
if x > 0:
    answer = 'positive'
else:
    answer = 'negative'
print(answer)   # negative
```

Case or Switch in Python

- There is no case or switch statement in Python.

Exercise: Rectangular

- Write a script that will ask for the sides of a rectangular and print out the area.
- Provide error messages if either of the sides is negative.

```
python rect.py
Side: 3
Side: 4
The area is 12
```

Exercise: Calculator

Create a script that accepts 2 numbers and an operator `(+, -, *, /)`, and prints the result of the operation.

```
python calc.py
Operand: 19
Operand: 23
Operator: +
Results: 42
```

Exercise: Standard Input

- In the previous exercises we expected the userinput to come in on the “Standard Input” aka. STDIN.
- If you would like to practice this more, come up with other ideas, try to solve them and tell me about the task. (in person or via e-mail.)
- (e.g. you could start building an interactive role-playing game.)

Solution: Area of rectangular

```
def main():
    #length = 10
    #width = 3

    length = int(input('Length: '))
    width = int(input('Width: '))

    if length <= 0:
        print("length is not positive")
        return

    if width <= 0:
        print("width is not positive")
        return

    area = length * width
    print("The area is ",  area)
```

```python

main()
```

Same in Python 2

```python
from __future__ import print_function

def main():
    #length = 10
    #width = 3

    length = int(raw_input('Length: '))
    width = int(raw_input('Width: '))

    if length <= 0:
        print("length is not positive")
        return

    if width <= 0:
        print("width is not positive")
        return

    area = length * width
    print("The area is ",  area)

main()
```

Solution: Calculator

```python
def main():
    a = float(input("Number: "))
    b = float(input("Number: "))
    op = input("Operator (+-*/): ")

    if op == '+':
        res = a+b
    elif op == '-':
        res = a-b
    elif op == '*':
        res = a*b
    elif op == '/':
        res = a/b
```

```
    else:
        print("Invalid operator: '{}'".format(op))
        return

    print(res)
    return


main()
```

Same in Python 2

```python
from __future__ import print_function

a = float(raw_input("Number: "))
b = float(raw_input("Number: "))
op = raw_input("Operator (+-*/): ")

if op == '+':
    res = a+b
elif op == '-':
    res = a-b
elif op == '*':
    res = a*b
elif op == '/':
    res = a/b
else:
    print("Invalid operator: '{}'".format(op))
    exit()


print(res)
```

Command line arguments

```python
import sys

def main():
    print(sys.argv)
    print(sys.argv[0])
    print(sys.argv[1])
    print(sys.argv[2])
```

```

main()
```

```
$ python examples/basic/cli.py one two
```

```
['examples/basics/cli.py', 'one', 'two']
examples/basics/cli.py
one
two
```

```
$ python examples/basic/cli.py
```

```
['examples/basics/cli.py']
examples/basics/cli.py
Traceback (most recent call last):
  File "examples/basics/cli.py", line 6, in <module>
    print(sys.argv[1])
IndexError: list index out of range
```

Command line arguments - len

```python
import sys

def main():
    print(sys.argv)
    print(len(sys.argv))

main()
```

Command line arguments - exit

```python
import sys

def main():
    if len(sys.argv) != 2:
        exit("Usage: " + sys.argv[0] + " VALUE")
    print("Hello " + sys.argv[1])
```

```

main()
```

```
echo %errorlevel%
echo $?
```

Exercise: Rectangular (argv)

- Change the above script that it will accept the arguments on the command line like this: `python rect.py 2 4`

Exercise: Calculator (argv)

- Create a script that accepts 2 numbers and an operator `(+, -, *, /)`, on the command line and prints the result of the operation.
- `python calc.py 2 + 3`
- `python calc.py 6 / 2`
- `python calc.py 6 * 2`

Solution: Area of rectangular (argv)

```
import sys

def main():
    if len(sys.argv) != 3:
        exit("Needs 2 arguments:  width length")

    width  = int( sys.argv[1] )
    length = int( sys.argv[2] )

    if length <= 0:
        exit("length is not positive")

    if width <= 0:
        exit("width is not positive")
```

```
    area = length * width
    print("The area is ",  area)

main()
```

Solution: Calculator eval

```
def main():
    a = input("Number: ")
    b = input("Number: ")
    op = input("Operator (+-*/): ")

    command = a + op + b
    print(command)
    res = eval(command)
    print(res)

main()
```

```
$ python examples/basics/calculator_eval.py

Number: 2
Number: 3
Operator (+-*/): +
2+3
5
```

Solution: Calculator (argv)

```
import sys


def main():
    if len(sys.argv) < 4:
        exit("Usage: " + sys.argv[0] + " OPERAND OPERATOR OPERAND")

    a = float(sys.argv[1])
    b = float(sys.argv[3])
```

```
    op = sys.argv[2]

    if op == '+':
        res = a + b
    elif op == '-':
        res = a - b
    elif op == '*':
        res = a * b
    elif op == '/':
        res = a / b
    else:
        print("Invalid operator: '{}'".format(op))
        exit()

    print(res)

main()
```

The multiplication probably won't work because the Unix/Linux shell replaces the * by the list of files in your current directory and thus the python script will see a list of files instead of the `*`. This is not your fault as a programmer. It is a user error. The correct way to run the script is `python calc.py 2 '*' 3`.

Compilation vs. Interpretation

Compiled

- Languages: C, C++
- Development cylce: Edit, Compile (link), Run.
- Strong syntax checking during compilation and linking.
- Result: Stand-alone executable code.
- Need to compile to each platform separately. (Windows, Linux, Mac, 32bit vs 64bit).

Interpreted

- Shell, BASIC
- Development cycle: Edit, Run.
- Syntaxt check only during run-time.
- Result: we distribute the source code.
- Needs the right version of the interpreted on every target machine.

Both?

- Java (running on JVM - Java Virtual Machine)
- C# (running on CLR - Common Language Runtime)

Is Python compiled or interpreted?

There are syntax errors that will prevent your Python code from running

```
x = 2
print(x)

if x > 3
```

```
File "examples/other/syntax_error.py", line 4
    if x > 3
           ^
SyntaxError: invalid syntax
```

There are other syntax-like errors that will be only caught during execution

```
x = 2
print(x)
print(y)
y = 13
print(42)
```

```
2
Traceback (most recent call last):
  File "compile.py", line 5, in <module>
    print y
NameError: name 'y' is not defined
```

- Python code is first compiled to bytecode and then interpreted.
- CPython is both the compiler and the interpreter.
- Jython and IronPython are mostly just compiler to JVM and CLR respectively.

Flake8 checking

```
conda install flake8
pip install flake8

flake8 --ignore= compile.py
```

```
compile.py:3:7: F821 undefined name 'y'
compile.py:6:1: W391 blank line at end of file
```

Numbers

Numbers

```
a = 42   # decimal
h = 0xA  # 10 - hex            - staring with 0x
o = 0o11 # 9  - octal          - starting with 0o
         # 011 works in Python 2.x but Python 3.x
         # requires the o that works in
         # (recent versions of) Python 2.x
b = 0b11 # 3  - binary numbers - starting with 0b

r = 2.3

print(a)  # 42
print(h)  # 10
print(o)  #  9
print(b)  #  3
print(r)  #  2.3
```

In Python numbers are stored as decimals, but in the source code you can also use hexadecimal, octal, or binary notations. This is especially useful if the domain you are programming in is using those kinds of numbers.
For example hardware engineers often talk in hexadecimal values.
In that case you won't need to contantly translate between the form used in the current domain and decimal numbers.

Operators for Numbers

```
a = 2
b = 3
c = 2.3

d = a + b
print(d)        # 5
print(a + b)    # 5
print(a + c)    # 4.3
print(b / a)    # 1.5  # see the __future__
print(b // a)   # 1    # floor division
print(a * c)    # 4.6

print(a ** b)   # 8   (power)

print(17 % 3)   # 2   (modulus)

a += 7          # is the same as a = a + 7
print(a)        # 9

# a++           # SyntaxError: invalid syntax
# a--           # SyntaxError: invalid syntax

a += 1
print(a)        # 10
a -= 1
print(a)        # 9
```

There is no autoincrement (++) and autodecrement (–) in Python, because they can be expressed by += 1 and -= 1 respectively.

Integer division and the future

```
from __future__ import print_function

print(3/2)
```

```
$ python divide.py
1

```

```
$ python3 divide.py
1.5
```

```python
from __future__ import print_function
from __future__ import division

print(3/2)      # 1.5
```

If you need to use Python 2, remember that by default division is integer based so 3/2 would return 1.
Importing the 'division' directive from **future** changes this to the behavior that we usually expect 3/2 being 1.5.
This is also the behavior we have in Python 3.
In case you already use Python 3 and would like to get the "old" behavior, that is to get the integer part of the division, you can
always call the "int" function: int(b/a).

Pseudo Random Number

```python
import random

a = random.random()
print(a) # 0.5648261676148922  a value between 0.0 <=   <
1.0
print(random.random())
print(random.random())
```

- [random]()
- [Pseudo random generator]()

Fixed random numbers

```python
import random

random.seed(37)

print(random.random()) # 0.6820045605879779
print(random.random()) # 0.09160260807956389
print(random.random()) # 0.6178163488614024
```

Rolling dice - randrange

```python
import random

print( 1 + int( 6 * random.random() ))

print(random.randrange(1, 7))

# One of the following: 1, 2, 3, 4, 5, 6
```

Random choice

```python
import random

letter = "abcdefghijklmno"
print(random.choice(letters))      # pick one of the letters

fruits = ["Apple", "Banana", "Peach", "Orange", "Durian", "Papaya"]
print(random.choice(fruits))
      # pick one of the fruits
```

built-in method

- A commont mistake. Not calling the method.

```python
import random

rnd = random.random
print(rnd)      # <built-in method random of Random object
```

```
at 0x124b508>


y = rnd()
print(y)        # 0.7740737563564781
```

When you see a string like the above “built-in method …” you can be almost certainly sure that you have forgotten the parentheses
at the end of a method call.

Exception: TypeError: ‘module’ object is not callable

- A commont mistake. Calling the class and not the method.

```python
import random

print("hello")
x = random()
print(x)
```

```
Traceback (most recent call last):
  File "examples/numbers/rnd.py", line 3, in <module>
    x = random()
TypeError: 'module' object is not callable
```

Fixing the previous code

```python
import random

x = random.random()
print(x)
```

```python
from random import random

x = random()
print(x)
```

Exception: AttributeError: module 'random' has no attribute

- A commont mistake. Using the wrong filename.

This works fine:

```python
print("Hello World")
```

This gives an error

```python
import random
print(random.random())
```

```
Traceback (most recent call last):
  File "rnd.py", line 2, in <module>
    print(random.random())
AttributeError: module 'random' has no attribute 'random'
```

> Make sure the names of your files are not the same as the names of any of the python packages.

Exercise: Number guessing game - level 0

Level 0

- Using the random module the computer “thinks” about a whole number between 1 and 20.
- The user has to guess the number. After the user types in the guess the computer tells if this was bigger or smaller than the number it generated, or if was the same.
- The game ends after just one guess.

Level 1-

- Other levels in the next chapter.

Exercise: Fruit salad

- Write a script that will pick 3 fruits from a list of fruits like the one we had in one of the earlier slides. Print the 3 names.
- Could you make sure the 3 fruits are different?

```
fruits = ["Apple", "Banana", "Peach", "Orange", "Durian",
"Papaya"]
```

Solution: Number guessing game - level 0

```
import random

hidden = random.randrange(1, 21)
print("The hidden values is", hidden)

user_input = input("Please enter your guess: ")
print(user_input)

guess = int(user_input)
if guess == hidden:
    print("Hit!")
elif guess < hidden:
    print("Your guess is too low")
else:
    print("Your guess is too high")
```

Solution: Fruit salad

```python
import random

fruits = ["Apple", "Banana", "Peach", "Orange", "Durian", "Papaya"]
salad = random.sample(fruits, 3)
print(salad)
```

Boolean

if statement again

```
x = 2

if x == 2:
    print("it is 2")
else:
    print("it is NOT 2")


if x == 3:
    print("it is 3")
else:
    print("it is NOT 3")

# it is 2
# it is NOT 3
```

True and False

- True and False are real boolean values.

```
x = 2

v = x == 2
print(v)
if v:
    print(v, "is true - who would thought? ")

v = x == 3
print(v)
if v:
    print(v, "is true - who would thought? ")
else:
    print(v, "is false - who would thought? ")
```

```
# True
# True is true - who would thought?
# False
# False is false - who would thought?
```

Boolean

```
x = 23

if x:
    print("23 is true")

y = 0
if y:
    print("0 is true")
else:
    print("0 is false")

# 23 is true
# 0 is false
```

True and False values in Python

- None
- 0
- ”” (empty string)
- False
- []
- {}
- ()

Everything else is true.

```
values = [None, 0, "", False, [], (), {}, "0", True]

for v in values:
    if v:
```

```
        print("True value:  ", v)
    else:
        print("False value: ", v)

# False value:  None
# False value:  0
# False value:
# False value:  False
# False value:  []
# False value:  ()
# False value:  {}
# True value:   0
# True value:   True
```

`None` is like undef or Null or Nill in other languages.

Comparision operators

```
==              equal
!=              not equal

<               less than
<=              less than or equal
>               greater than
>=              greater than or equal
```

```
a = "42"
b = 42

print(a == b)        # False
print(a != b)        # True
print(b == 42.0)     # True

print(None == None)  # True
print(None == False) # False
```

Do NOT Compare different types

```
x = 12
y = 3
print(x > y)  # True

x = "12"
y = "3"
print(x > y)  # False

x = "12"
y = 3
print(x > y)  # True

x = 12
y = "3"
print(x > y)  # False
```

In `Python 2` please be careful and only compare the same types. Otherwise the result will look strange.

```
True
False
True
False
```

In `Python 3`, comparing different types raises exception:

```
True
False
Traceback (most recent call last):
  File "examples/other/compare.py", line 6, in <module>
    print(x > y)     # True
TypeError: '>' not supported between instances of 'str'
and 'int'
```

Boolean operators

- and
- or

- not

```
if COND:
   do something
else:
   do something other

if not COND:
    do something other

if COND1 and COND2:
    do something

if COND1 or COND2:
    do something

if COND1 and not COND2:
    do something
```

Boolean truth tables

```
COND1 and COND2     Result
True      True      True
True      False     False
False     True      False
False     False     False
```

```
COND1 or COND2      Result
True      True      True
True      False     True
False     True      True
False     False     False
```

```
not COND     Result
True         False
False        True
```

Short circuit

```
def check_money():
    return money > 1000000

def check_salary():
    salary += 1
    return salary >= 1000

while True:
    if check_money() or check_salary():
        print("I can live well")
```

Short circuit fixed

```
def check_money():
    return money > 1000000

def check_salary():
    salary += 1
    return salary >= 1000

while True:
    has_good_money = check_money()
    has_good_salary = check_salary()

    if has_good_money or has_good_salary:
        print("I can live well")
```

Incorrect use of conditions

> In your normal speach you could probably say something like "If status_code is 401 or 302, do something.".
> Meaning status_cone can be either 401 or 302.
>
> If you tried to translate this into code directly you would write something like this:

```
if status_code == 401 or 302:
    pass
```

> However this is incorrect. This condition will be always true as this is actually same as if you wrote: `if (status_code == 401) or (302)` so it will compare status_code to 401, and it will separately check if 302 is True, but any number different from 0 is considered to be True so the above expression will always be True.

What you probably meant is this:

```
if status_code == 401 or status_code == 302:
    pass
```

Alternative way:

> An alternative way to achieve the same results would be though probbaly at this point we have not learned the "in" operator, nor lists (comma separated values in square brackets):

```
if status_code in [401, 302]
    pass
```

Exercise: compare numbers

- Ask the user to enter two numbers and tell us which one is bigger.

Exercise: compare strings

- Ask the user to enter two strings
- Then ask the user to select if she wants to compare them based on ASCII or based on their length
- Then tell us which one is bigger.

```
Input a string: (user types string and ENTER)
Input another string: (user types string and ENTER)
How to compare:
1) ASCII
2) Length
(user types 1 or 2 and ENTER)
```

Solution: compare numbers

```
a_in = input("Please type in a string: ")
b_in = input("Please type in another string: ")
print("How to compare:")
print("1) ASCII")
print("2) Length")
how = input()

if how == '1':
    first = a_in > b_in
    second = a_in < b_in
elif how == '2':
    first = len(a_in) > len(b_in)
    second = len(a_in) < len(b_in)

if first:
    print("First number is bigger")
elif second:
    print("First number is smaller")
else:
    print("They are equal")
```

Solution: compare strings

```python
a_in = input("Please type in a string: ")
b_in = input("Please type in another string: ")
print("How to compare:")
print("1) ASCII")
print("2) Length")
how = input()

if how == '1':
    first = a_in > b_in
    second = a_in < b_in
elif how == '2':
    first = len(a_in) > len(b_in)
    second = len(a_in) < len(b_in)

if first:
    print("First number is bigger")
elif second:
    print("First number is smaller")
else:
    print("They are equal")
```

Strings

Single quoted and double quoted strings

In Python, just as in most of the programming languages you must put any free text inside a pair of quote characters. Otherwise Python will try to find meaning in the text.1

These pieces of texts are called “strings”.

In Python you can put string between two single quotes: ‘’ or between two double quotes: “”. Which one does not matter.

```
soup = "Spiced carrot & lentil soup"
salad = 'Ceasar salad'

print(soup)
print(salad)
```

```
Spiced carrot & lentil soup
Ceasar salad
```

Long lines

```
text = "abc" "def"
print(text)

other = "abcdef"
print(other)


```

```
long_string = "one" "two" "three"
print(long_string)

short_rows = "one" \
    "two" \
    "three"
print(short_rows)

long_string = "first row second row third row"
print(long_string)

shorter = "first row \
second row \
third row"
print(shorter)
```

```
abcdef
abcdef
onetwothree
onetwothree
first row second row third row
first row second row third row
```

Triple quoted strings (multiline)

> If you would like to create a string the spreads on multiple lines,
> there is a possibility to put the text between 3 quotes on both sides. Either 2*3 single-quotes*
> *or 2*3 double-quotes.

```
text = """first row
second row
third row"""

print(text)
```

Can spread multiple lines.

```
first row
second row
third row
```

String length (len)

The `len` function returns the length of the string in number of characters.

```
line = "Hello World"
hw = len(line)
print(hw)   # 11

text = """Hello
World"""
print(len(text))   # 12
```

String repetition and concatenation

You might be used to the fact the you can only multiple numbers, but in python you can also “multiply” a string by a number.
It is called repetition. In this example we have a string “Jar “ that we repeat twice.repetition

We can also add two strings to concatenate them together.repetition

I don’t think the repetition operator is used very often, but in one case it could come very handy.

When you are writing some text report and you'd like to add a long line of dashes that would be exactly the same length as your title.

```
name = 2 * 'Jar '
print(name)         # Jar Jar

full_name = name + 'Binks'
print(full_name)    # Jar Jar Binks


title = "We have some title"
print(title)
print('-' * len(title))

# We have some title
# ------------------
```

A character in a string

```
text = "Hello World"

a = text[0]
print(a)       # H

b = text[6]
print(b)       # W
```

String slice (instead of substr)

```
text = "Hello World"

b = text[1:4]
print(b)          # ell

print(text[2:])   # llo World
print(text[:2])   # He

```

```python
start = 1
end = 4
print(text[start:end])  # ell
```

Change a string

In Python strings are “immutable”, meaning you cannot change them. You can replace a whole string in a variable, but you cannot change it.

In the following example we wanted to replace the 3rd character (index 2), and put “Y” in place. This raised an exception

```python
text = "abcd"
print(text)     # abcd

text[2] = 'Y'

print("done")
print(text)
```

```
abcd
Traceback (most recent call last):
  File "string_change.py", line 4, in <module>
    text[2] = 'Y'
TypeError: 'str' object does not support item assignment
```

Replace part of a string

- Strings in Python are **immutable** - they never change.

How to change a string

```
text = "abcd"
print(text)       # abcd

text = text[:2] + 'Y' + text[3:]
print(text)       # abYd
```

String copy

```
text = "abcd"
print(text)      # abcd

text = text + "ef"
print(text)      # abcdef

other = text
print(other)      # abcdef
text = "xyz"
print(text)      # xyz
print(other)      # abcdef
```

When assigning a variable pointing a string, the new variable is pointing to the same string..
If we then assign some other string to either of the variables, then they will point to two different strings.

String functions and methods (len, upper, lower)

```
a = "xYz"
print(len(a))      # 3

b = a.upper()
print(b)           # XYZ
```

```
print(a)            # xYz   - immutable!
print(a.lower())    # xyz
```

- Type **dir("")** in the REPL to get the list of string methods.
- List of built-in functions.
- List of string methods.

index in string

```
text = "The black cat climbed the green tree."
print(text.index("bl"))      # 4
print(text.index("The"))     # 0
print(text.index("dog"))
```

```
4
0
Traceback (most recent call last):
  File "examples/strings/index.py", line 6, in <module>
    print a.index("dog")     # -1
ValueError: substring not found
```

index in string with range

```
text = "The black cat climbed the green tree."
print(text.index("c"))       # 7
print(text.index("c", 8))    # 10

print(text.index("gr", 8))       # 26
print(text.index("gr", 8, 16))
```

```
7
10
26
Traceback (most recent call last):
  File "examples/strings/index2.py", line 8, in <module>
    print a.index("gr", 8, 16)
ValueError: substring not found
```

rindex in string with range

```
text = "The black cat climbed the green tree."
print(text.rindex("c"))          # 14
print(text.rindex("c", 8))       # 14
print(text.rindex("c", 8, 13))   # 10

print(text.rindex("gr", 8))      # 26
print(text.rindex("gr", 8, 16))
```

```
14
14
10
26
Traceback (most recent call last):
  File "examples/strings/rindex.py", line 10, in <module>
    print(a.rindex("gr", 8, 16))
ValueError: substring not found
```

find in string

Alternatively use find and rfind that will return -1 instead of raising an exception.

```
text = "The black cat climbed the green tree."
print(text.find("bl"))      # 4
print(text.find("The"))     # 0
print(text.find("dog"))     # -1

print(text.find("c"))       # 7
print(text.find("c", 8))    # 10

print(text.find("gr", 8))       # 26
print(text.find("gr", 8, 16))   # -1


print(text.rfind("c", 8))    # 14
```

Find all in the string

Later, when we learned loops.

in string

Check if a substring is **in** the string?

```
txt = "hello world"
if "wo" in txt:
    print('found wo')

if "x" in txt:
    print("found x")
else:
    print("NOT found x")
```

```
found wo
NOT found x
```

index if in string

```
sub = "cat"
txt = "The black cat climbed the green tree"

if sub in txt:
    loc = txt.index(sub)
    print(sub + " is at " + str(loc))

sub = "dog"
if sub in txt:
    loc = txt.index(sub)
    print(sub + " is at " + str(loc))

# cat is at 10
```

Encodings: ASCII, Windows-1255, Unicode

- ASCII
- Hebrew Character

- [Windows-1255]()
- [Unicode (UTF-8)]()

raw strings

```
# file_a = "c:\Users\Foobar\readme.txt"
# print(file_a)

# Python2:  eadme.txtFoobar
# Python3:
#   File "examples/strings/raw.py", line 6
#     file_a = "c:\Users\Foobar\readme.txt"
#             ^
# SyntaxError: (unicode error) 'unicodeescape' codec
#    can't decode bytes in position 2-3: truncated \UXXXXXXXX escape


file_b = "c:\\Users\\Foobar\\readme.txt"
print(file_b)  # c:\Users\Foobar\readme.txt

file_c = r"c:\Users\Foobar\readme.txt"
print(file_c)  # c:\Users\Foobar\readme.txt

text = r"text \n \d \s \ and more"
print(text)    # text \n \d \s \ and more
```

Escape sequences are kept intact and not escaped. Used in regexes.

ord

- [ord]()

```
print( ord('a') )    # 97
print( ord('=') )    # 61
print( ord('\r') )   # 13
print( ord('\n') )   # 10
print( ord(' ') )    # 32

```

```
print( ord('á') )    # 225
print( ord('ó') )    # 243
print( ord('א') )    # 1488
```

ord in a file

```python
import sys

filename = sys.argv[1]

with open(filename) as fh:
    content = fh.read()

for c in content:
    print(ord(c))
```

chr - number to character

- chr

```python
for i in range(32, 126):
    print( i, chr(i) )
```

```
32
33 !
34 "
35 #
36 $
37 %
38 &
39 '
40 (
41 )
42 *
43 +
44 ,
45 -
46 .
47 /
48 0
```

```
49 1
50 2
51 3
52 4
53 5
54 6
55 7
56 8
57 9
58 :
59 ;
60 <
61 =
62 >
63 ?
64 @
65 A
66 B
67 C
68 D
69 E
70 F
71 G
72 H
73 I
74 J
75 K
76 L
77 M
78 N
79 O
80 P
81 Q
82 R
83 S
84 T
85 U
86 V
87 W
88 X
89 Y
90 Z
91 [
92 \
93 ]
```

```
94 ^
95 _
96 `
97 a
98 b
99 c
100 d
101 e
102 f
103 g
104 h
105 i
106 j
107 k
108 l
109 m
110 n
111 o
112 p
113 q
114 r
115 s
116 t
117 u
118 v
119 w
120 x
121 y
122 z
123 {
124 |
125 }
```

Exercise: one string in another string

Write script that accepts two strings and tells if one of them can be found in the other and where?

Exercise: to ASCII CLI

Write script that gets a character on the command line and prints out the ascii code of it.

Maybe even:

Write script that gets a string on the command line and prints out the ascii code of each character.

Exercise: from ASCII CLI

Write script that accepts a number on the command line and prints the character represented by that number.

Solution: one string in another string

```python
import sys

if len(sys.argv) != 3:
    exit(f"Usage: {sys.argv[0]} short-STRING long-STRING")

string = sys.argv[1]
text   = sys.argv[2]

if string in text:
    loc = text.index(string)
    print(string, "can be found in ", text, "at", loc)
else:
    print(string, "can NOT be found in ", text)
```

Solution: compare strings

```python
mode = input("Mode of comparision: [length|ascii|")
if mode != "length" and mode != "ascii":
    print("Not good")
    exit()

str1 = input("String 1:")
```

```python
str1 = input("String 2:")

if mode == "length":
    print(len(str1) > len(str2))
elif mode == "ascii":
    print(str1 > str2)
```

Solution: to ASCII CLI

```python
import sys

if len(sys.argv) != 2:
    exit(f"Usage: {sys.argv[0]} CHARACTER")

print( ord( sys.argv[1]) )
```

```python
import sys

if len(sys.argv) != 2:
    exit(f"Usage: {sys.argv[0]} STRING")

for cr in sys.argv[1]:
    print( ord( cr ) )
```

Solution: from ASCII CLI

```python
import sys

if len(sys.argv) != 2:
    exit(f"Usage: {sys.argv[0]} NUMBER")

print( chr( int(sys.argv[1]) ) )
```

Loops

Loops: for-in and while

- **for in** - to iterate over a well defined list of values. (characters, range of numbers, shopping list, etc.)
- **while** - repeate an action till some condition is met. (or stopped being met)

for-in loop on strings

```
txt = 'hello world'
for c in txt:
    print(c)
```

```
h
e
l
l
o
 
w
o
r
l
d
```

for-in loop on list

```
for fruit in ["Apple", "Banana", "Peach", "Orange", "Durian", "Papaya"]:
    print(fruit)
```

```
Apple
Banana
Peach
Orange
Durian
Papaya
```

---

for-in loop on range

```python
for i in range(3, 7):
    print(i)
```

```
3
4
5
6
```

Iterable, iterator

- iterable

for in loop with early end using break

```python
txt = 'hello world'
for c in txt:
    if c == ' ':
        break
    print(c)
```

```
h
e
l
l
o
```

---

for in loop skipping parts using continue

```python
txt = 'hello world'
for c in txt:
    if c == ' ':
        continue
    print(c)
```

```
h
e
l
l
o
w
o
r
l
d
```

for in loop with break and continue

```python
txt = 'hello world'
for cr in txt:
    if cr == ' ':
        continue
    if cr == 'r':
        break
    print(cr)
print('DONE')
```

```
h
e
l
l
o
w
o
DONE
```

while loop

```python
import random

total = 0
while total <= 100:
    print(total)
    total += random.randrange(20)

print("done")
```

```
0
10
22
29
45
54
66
71
77
82
93
done
```

Infinite while loop

```python
import random

total = 0
while total >= 0:
    print(total)
    total += random.randrange(20)

print("done")
```

```
...
1304774
1304779
1304797
^C1304803
Traceback (most recent call last):
  File "while_infinite.py", line 5, in <module>
```

```
    print(total)
KeyboardInterrupt
```

- Don’t do this!
- Make sure there is a proper end-condition. (exit-condition)
- Use Ctrl-C to stop it

While with complex expression

```python
import random

total = 0
while (total < 10000000) and (total % 17 != 1) and (total ** 2 % 23 != 7):
    print(total)
    total += random.randrange(20)

print("done")
```

While with break

```python
import random

total = 0
while total < 10000000:
    print(total)
    total += random.randrange(20)

    if total % 17 == 1:
        break

    if total ** 2 % 23 == 7:
        break

print("done")
```

While True

```
import random

total = 0
while True:
    print(total)
    total += random.randrange(20)

    if total >= 10000000:
        break

    if total % 17 == 1:
        break

    if total ** 2 % 23 == 7:
        break

print("done")
```

Duplicate input call

```
id_str = input("Type in your ID: ")

while len(id_str) != 9:
   id_str = input("Type in your ID")

print("Your ID is " + id_str)
```

Eliminate duplicate input call

```
while True:
   id_str = input("Type in your ID: ")
   if len(id_str) == 9:
       break

print("Your ID is " + id_str)
```

do while loop

There is no `do ... while` in Python but we can write code like this to have similar effect.

```
while True:
    answer = input("What is the meaning of life? ")
    if answer == '42':
        print("Yeeah, that's it!")
        break

print("done")
```

while with many continue calls

```
while True:
   line = get_next_line()

   if last_line:
       break

   if line_is_empty:
      continue

   if line_has_an_hash_at_the_beginning: # #
      continue

   if line_has_two_slashes_at_the_beginning: # //
      continue

   do_the_real_stuff
```

Break out from multi-level loops

Not supported in Python. “If you feel the urge to do that, your code is probably too complex. create functions!”

Exit vs return vs break and continue

- **exit** will stop your program no matter where you call it.

- **return** will return from a function (it will stop the specific function only)
- **break** will stop the current "while" or "for" loop
- **continue** will stop the current iteration of the current "while" or "for" loop

Exercise: Print all the locations in a string

Given a string like "The black cat climbed the green tree.", print out the location of every "c" charcater.

Exercise: Number guessing game

Level 0

- Using the random module the computer "thinks" about a whole number between 1 and 20.
- The user has to guess the number. After the user types in the guess the computer tells if this was bigger or smaller than the number it generated, or if was the same.
- The game ends after just one guess.

Level 1

- The user can guess several times. The game ends when the user guessed the right number.

Level 2

- If the user hits 'x', we leave the game without guessing the number.

Level 3

- If the user presses ‘s’, show the hidden value (cheat)

Level 4

- Soon we’ll have a level in which the hidden value changes after each guess. In oredr to make that mode easier to track and debug, first we would like to have a “debug mode”.
- If the user presses ‘d’ the game gets into “debug mode”: the system starts to show the current number to guess every time, just before asking the user for new input.
- Pressing ‘d’ again turns off debug mode. (It is a toggle each press on “d” changes the value to to the other possible value.)

Level 5

- The ‘m’ button is another toggle. It is called ‘move mode’. When it is ‘on’, the hidden number changes a little bit after every step (+/-2). Pressing ‘m’ again will turn this feature off.

Level 6

- Let the user play several games.
- Pressing ‘n’ will skip this game and start a new one. Generates a new number to guess.

Exercise: MasterMind

Implement the MasterMind game.

The computer “thinks” a number with 4 different digits.
You guess which digits. For every digit that matched both
in value, and in location the computer gives you a `*`. For every
digit that matches in value, but not in space the computer gives

you a +. Try to guess the given number in as few guesses as possible.

```
Computer: 2153
You:      2467  *
You:      2715  *++
```

Exercise: Count unique characters

Given a string on the command line, count how many differnt characters it has.

```
python count_unique.py abcdaaa
4
```

Solution: Print all the locations in a string

```
text = "The black cat climbed the green tree."
start = 0
while True:
    loc = text.find("c", start)
    if loc == -1:
        break
    print(loc)
    start = loc + 1
```

Solution 1 for Number Guessing

```
import random

hidden = random.randrange(1, 201)
while True:
    user_input = input("Please enter your guess[x]: ")
    print(user_input)

    if user_input == 'x':
        print("Sad to see you leaving early")
        exit()
```

```
    guess = int(user_input)
    if guess == hidden:
        print("Hit!")
        break

    if guess < hidden:
        print("Your guess is too low")
    else:
        print("Your guess is too high")
```

Solution for Number Guessing (debug)

```
import random

hidden = random.randrange(1, 201)
debug = False
while True:
    if debug:
        print("Debug: ", hidden)

    user_input = input("Please enter your guess [x|s|d]: ")
    print(user_input)

    if user_input == 'x':
        print("Sad to see you leaving early")
        exit()

    if user_input == 's':
        print("The hidden value is ", hidden)
        continue

    if user_input == 'd':
        debug = not debug
        continue

    guess = int(user_input)
    if guess == hidden:
        print("Hit!")
        break

    if guess < hidden:
```

```python
        print("Your guess is too low")
    else:
        print("Your guess is too high")
```

Solution for Number Guessing (move)

```python
import random

hidden = random.randrange(1, 201)
debug = False
move = False
while True:
    if debug:
        print("Debug: ", hidden)

    if move:
        mv = random.randrange(-2, 3)
        hidden = hidden + mv

    user_input = input("Please enter your guess [x|s|d|m]: ")
    print(user_input)

    if user_input == 'x':
        print("Sad to see you leaving early")
        exit()

    if user_input == 's':
        print("The hidden value is ", hidden)
        continue

    if user_input == 'd':
        debug = not debug
        continue

    if user_input == 'm':
        move = not move
        continue

    guess = int(user_input)
    if guess == hidden:
        print("Hit!")
        break
```

```python

    if guess < hidden:
        print("Your guess is too low")
    else:
        print("Your guess is too high")
```

Solution for Number Guessing (multi-game)

```python
import random

debug = False
move = False
while True:
    print("\nWelcome to another Number Guessing game")
    hidden = random.randrange(1, 201)
    while True:
        if debug:
            print("Debug: ", hidden)

        if move:
            mv = random.randrange(-2, 3)
            hidden = hidden + mv

        user_input = input("Please enter your guess [x|s|d|m|n]: ")
        print(user_input)

        if user_input == 'x':
            print("Sad to see you leaving early")
            exit()

        if user_input == 's':
            print("The hidden value is ", hidden)
            continue

        if user_input == 'd':
            debug = not debug
            continue

        if user_input == 'm':
            move = not move
            continue

```

```python
        if user_input == 'n':
            print("Giving up, eh?")
            break

        guess = int(user_input)
        if guess == hidden:
            print("Hit!")
            break

        if guess < hidden:
            print("Your guess is too low")
        else:
            print("Your guess is too high")
```

Solution: MasterMind

```python
import random

width = 4
USED  = '_'

hidden = random.sample(range(10), width)
# print(hidden)

while True:
    # print(hidden)

    inp = input("your guess ({} digits):".format(width))
    if inp == 'x':
        print("Bye")
        exit()
    if len(inp) != width:
        print("We need exactly {} characters".format(width))
        continue

    guess = list(map(int, inp))
    # print(guess)

    if hidden == guess:
        print("Match!")
        break

```

```python
    my_hidden = hidden[:]
    my_guess  = guess[:]

    result = ''
    for i in range(width):
        if my_hidden[i] == my_guess[i]:
            result  += '*'
            my_hidden[i] = USED
            my_guess[i] = USED
    for i in range(width):
        if my_guess[i] == USED:
            continue
        if my_guess[i] in my_hidden:
            loc = my_hidden.index(my_guess[i])
            my_hidden[loc] = USED
            guess[i] = USED
            result += '+'

    print(''.join(result))
```

Solution: Count unique characters

```python
import sys

s = sys.argv[1]

unique = ''
for c in s:
    if c not in unique:
        unique += c

print(len(unique))
```

```python
import sys

s = sys.argv[1]

print(len(set(s)))
```

MasterMind to debug

Debug the following version of the MasterMind game.

```python
import random


def number_generator():
    y = [0, 0, 0, 0]

    for i in range(0, 4):
        y[i] = random.randrange(0, 10)
        # print(y)
        if i:
            number += str(y[i])
        else:
            number = str(y[i])
    # print(number)
    return number


def user_input():
    x = input("Type in 4 digits number:")
    if len(x) == 4:
        return x
    else:
        print("wrong input")
        user_input()


def string_compare(x, y):
    r = 0
    q = 0
    for i in range(0, 4):
        if x[i] == y[i]:
            r += 1
            continue
        for j in range(0, 4):
            if x[i] == y[j]:
                if i == j:
                    continue
                else:
                    q += 1
                    break
    return r, q

```

```python

def print_result(r):
    print("")
    for i in range(0, r[0]):
        print("*", end="")
    for i in range(0, r[1]):
        print("+", end="")
    print("\n")


def main():
    comp = number_generator()
    result = 0
    while True:
        user = user_input()
        result = string_compare(comp, user)
        print_result(result)
        # print(result)
        if result[0] == 4:
            print("Correct!")
            return


main()
```

PyCharm

PyCharm Intro

- IDE
- Introspection
- Running, Debugging

PyCharm Project

- At the opening create a new project (directory + Python version)
- File/New Project

PyCharm Files

- New file on Mac: Click on the project on the left hand side / Right-Click / New / File; Windows, Linux: Alt-Insert
- PyCharm Python console - see next slide
- Change Python on Mac: PyCharm / Preferences / Project: (name) / Project Interpreter
- Later File/New also starts to work.

PyCharm - run code

- Run/Run
- Set command line parameters
- Set environment variables

- Run/Debug (but set breakpoints before)

PyCharm Python console at the bottom left

```
2 + 3
x = 2
print(x)
def f(x, y):
    return x+y

f(4, 5)
```

Refactoring example (with and without pycharm)

- Change variable name (in scope only)
- Extract method

Formatted printing

format - sprintf

```
age = 42.12
name = 'Foo Bar'

str_concatenate = "The user " + name + " was born " + str(age) + " years ago."
print(str_concatenate)

str_percentage = "The user %s was born %s years ago." % (name, age)
print(str_percentage)

str_format = "The user {} was born {} years ago.".format(name, age)
print(str_format)

str_f_string = f"The user {name} was born {age} years ago."
print(str_f_string)
```

```
The user Foo Bar was born 42.12 years ago.
The user Foo Bar was born 42.12 years ago.
The user Foo Bar was born 42.12 years ago.
The user Foo Bar was born 42.12 years ago.
```

- When using % to print more than one values, put the values in parentheses forming a tuple.
- In version 2.6 and below you need to write etc, as a placeholder of the format method.
- f-string are from Python 3.6

Examples using format - indexing

```
txt = "Foo Bar"
num = 42.12

print("The user {} was born {} years ago.".format(txt, num))
print("The user {0} was born {1} years ago.".format(txt, num))
print("The user {1} was born {0} years ago.".format(num, txt))


print("{0} is {0} and {1} years old.".format(txt, num))
```

```
The user Foo Bar was born 42.12 years ago.
The user Foo Bar was born 42.12 years ago.
The user Foo Bar was born 42.12 years ago.
Foo Bar is Foo Bar and 42.12 years old.
```

Examples using format with names

```
txt = "Foo Bar"
num = 42.12

print("The user {name} was born {age} years ago.".format(name = txt, age = num))
```

```
The user Foo Bar was born 42.12 years ago.
```

Format columns

In this example we use a list of lists that we have not learned yet, but don’t worry about that for now.
Focus on the output of the two print statements.

```
data = [
    ["Foo Bar", 42],
```

```
    ["Bjorg", 12345],
    ["Roza", 7],
    ["Long Name Joe", 3],
    ["Joe", 12345677889],
]

for entry in data:
    print("{} {}".format(entry[0], entry[1]))

print('-' * 16)

for entry in data:
    print("{:<8}|{:>7}".format(entry[0], entry[1]))
```

```
Foo Bar 42
Bjorg 12345
Roza 7
Long Name Joe 3
Joe 12345677889
----------------
Foo Bar |     42
Bjorg   |  12345
Roza    |      7
Long Name Joe|      3
Joe     |12345677889
```

Examples using format - alignment

```
txt = "Some text"

print("'{}'".format(txt))      #  as is:   'Some text'
print("'{:12}'".format(txt))   #  left:    'Some text   '
print("'{:<12}'".format(txt))  #  left:    'Some text   '
print("'{:>12}'".format(txt))  #  right:   '   Some text'
print("'{:^12}'".format(txt))  #  center:  ' Some text  '
```

Format - string

```
name = "Foo Bar"

```

```
print("{:s}".format(name))
print("{}".format(name))
```

```
Foo Bar
Foo Bar
```

Format characters and types

```
x = 42

print("{:b}".format(x)) #  binary:    101010
print("{:c}".format(x)) #  character: *
print("{:d}".format(x)) #  decimal:   42       (default)
print("{:o}".format(x)) #  octal:     52
print("{:x}".format(x)) #  hexa:      2a
print("{:X}".format(x)) #  hexa:      2A
print("{:n}".format(x)) #  number:    42


print("{}".format(x))   # defaults to decimal
```

Format floating point number

```
x = 412.345678901

print("{:e}".format(x))   #  exponent:     4.123457e+02
print("{:E}".format(x))   #  Exponent:     4.123457E+02
print("{:f}".format(x))   #  fixed point:  412.345679
(default precision is 6)
print("{:.2f}".format(x)) #  fixed point:  412.35 (set
precision to 2)
print("{:F}".format(x))   #  same as f.    412.345679
print("{:g}".format(x))   #  generic:      412.346
(default precision is 6)
print("{:G}".format(x))   #  generic:      412.346
print("{:n}".format(x))   #  number:       4412.346


print("{}".format(x))     # defaults to g  412.345678901
```

f-strings (formatted string literals)

Since Python 3.6

```
name = "Foo Bar"
age = 42.12
pi = 3.141592653589793
r = 2

print(f"The user {name} was born {age} years ago.")
print(f"The user {name:10} was born {age} years ago.")
print(f"The user {name:>10} was born {age} years ago.")
print(f"The user {name:>10} was born {age:>10} years ago.")

print(f"PI is '{pi:.3}'.")   # number of digits (defaults n = number)
print(f"PI is '{pi:.3f}'.")  # number of digits after decimal point

print(f"Area is {pi * r ** 2}")
print(f"Area is {pi * r ** 2:.3f}")
```

```
The user Foo Bar was born 42.12 years ago.
The user Foo Bar    was born 42.12 years ago.
The user    Foo Bar was born 42.12 years ago.
The user    Foo Bar was born      42.12 years ago.
PI is '3.14'.
PI is '3.142'.
Area is 12.566370614359172
Area is 12.566
```

printf using old %-syntax

This slides is here only as a historical page. It is recommended to use the **format** method!

```
v = 65
print("<%s>" % v)     # <65>
print("<%10s>" % v)   # <        65>
print("<%-10s>" % v)  # <65        >
```

```
print("<%c>" % v)      # <A>
print("<%d>" % v)      # <65>
print("<%0.5d>" % v)   # <00065>
```

Format braces, bracket, and parentheses

These are just some extreme special cases. Most people won't need to know about them.

To print `{` include `{{`.
To print `}` include `}}`.

```
print("{{{}}}".format(42))    # {42}

print("{{ {} }}".format(42))    # { 42 }

print("[{}] ({})".format(42, 42))    # [42] (42)

print("%{}".format(42))    # %42
```

Anything that is not in curly braces will be formatted as they are.

Examples using format with attributes of objects

This is also a rather strange example, I don't think I'd use it in real code.

```
import sys

print("{0.executable}".format(sys))
print("{system.argv[0]}".format(system = sys))
```

```
/home/gabor/venv3/bin/python
formatted_attributes.py
```

raw f-strings

```
name="foo"
print(r"a\nb {name}")
print(rf"a\nb {name}")
print(fr"a\nb {name}")  # this is better (for vim)
```

```
a\nb {name}
a\nb foo
a\nb foo
```

Lists

Anything can be a lists

- Comma separated values
- In square brackets
- Can be any value, and a mix of values: Integer, Float, Boolean, None, String, List, Dictionary, …
- But usually they are of the same type:
- Distances of astronomical objects
- Chemical Formulas
- Filenames
- Names of devices
- Objects describing attributes of a network device.
- Actions to do on your data.

```
stuff = [42, 3.14, True, None, "Foo Bar", ['another', 'list'], {'a': 'Dictionary', '\
language' : 'Python'}]
print(stuff)
```

```
[42, 3.14, True, None, 'Foo Bar', ['another', 'list'], {'a': 'Dictionary', 'language\
': 'Python'}]
```

Any layout

- Layout is flexible
- Trailing comma is optional. It does not disturb us. Nor Python.

```
more_stuff = [
    42,
    3.14,
    True,
    None,
    "Foo Bar",
    ['another', 'list'],
    {
        'a': 'Dictionary',
        'language' : 'Python',
    },
]
print(more_stuff)
```

```
[42, 3.14, True, None, 'Foo Bar', ['another', 'list'], {'a': 'Dictionary', 'language\
': 'Python'}]
```

Lists

- Access single element: **[index]**
- Access a sublist: **[start:end]**
- Creates a copy of that sublist

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn']

print(planets)   # ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn']
print(len(planets))    # 6

print(planets[0])          # Mercury
print(type(planets[0]))    # <class 'str'>
print(planets[3])          # Mars

print(planets[0:1])        # ['Mercury']
print(type(planets[0:1])) # <class 'list'>
print(planets[0:2])        # ['Mercury', 'Venus']
print(planets[1:3])        # ['Venus', 'Earth']

```

```
print(planets[2:])         # ['Earth', 'Mars', 'Jupiter', 'Saturn']
print(planets[:3])         # ['Mercury', 'Venus', 'Earth']

print(planets[:])          # ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn\
']
```

List slice with steps

- List slice with step: **[start:end:step]**

```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

print(letters[::])        # ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

print(letters[::1])       # ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

print(letters[::2])       # ['a', 'c', 'e', 'g', 'i']

print(letters[1::2])      # ['b', 'd', 'f', 'h', 'j']

print(letters[2:8:2])     # ['c', 'e', 'g']

print(letters[1:20:3])    # ['b', 'e', 'h']
```

Change a List

```
x = ['abc', 'def', 'ghi', 'jkl']
x[0] = 'qqrq'
print(x)    # ['qqrq', 'def', 'ghi', 'jkl']

x[1:3] = ['xyz', 'dod']
print(x)    #  ['qqrq', 'xyz', 'dod', 'jkl']


x[1:3] = ['bla']
print(x)    #  ['qqrq', 'bla', 'jkl']
```

```
x[1:2] = ['elp', 'free']
print(x)     # ['qqrq', 'elp', 'free', 'jkl']


#x[1] = ['elp', 'free']
#print(x)    # ['qqrq', ['elp', 'free'], 'jkl']
```

- Unlike strings, lists are mutable. You can change the content of a list by assigning values to its elements.
- You can use the slice notation to change several elements at once.
- You can even have different number of elements in the slice and in the replacement. This will also change the length of the array.

Change with steps

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
print(numbers)  # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

numbers[1::2] = [0, 0, 0, 0, 0, 0]
print(numbers)  # [1, 0, 3, 0, 5, 0, 7, 0, 9, 0, 11, 0]
```

List assignment and list copy

```
x = ['apple', 'bob', 'cat', 'drone']
y = x
x[0] = 'qqrq'
print(x)    # ['qqrq', 'bob', 'cat', 'drone']
print(y)    # ['qqrq', 'bob', 'cat', 'drone']
```

- There is one list in the memory and two pointers to it.
- If you really want to make a copy the pythonic way is to use the slice syntax.

- It creates a shallow copy.

```
x = ['apple', 'bob', 'cat', 'drone']
y = x[:]

x[0] = 'qqrq'

print(x)     # ['qqrq', 'bob', 'cat', 'drone']
print(y)     # ['apple', 'bob', 'cat', 'drone']
```

Deep copy

```python
from copy import deepcopy

x = ['apple', 'bob', 'cat', 'drone']
y = deepcopy(x)

x[0] = 'qqrq'

print(x)     # ['qqrq', 'bob', 'cat', 'drone']
print(y)     # ['apple', 'bob', 'cat', 'drone']
```

join

```
fields = ['one', 'two and three', 'four', 'five']

together = ':'.join(fields)
print(together) # one:two and three:four:five

mixed = ' -=<> '.join(fields)
print(mixed) # one -=<> two and three -=<> four -=<> five

another = ''.join(fields)
print(another)  # onetwo and threefourfive
```

join list of numbers

```
a = ["x", "2", "y"]
b = ["x", 2, "y"]
```

```
print(":".join(a))    # x:2:y
# print ":".join(b)    # TypeError: sequence item 1:
expected string, int found

# convert elements to string using map
print(":".join( map(str, b) ))        # x:2:y


# convert elements to string using list comprehension
print(":".join( str(x) for x in b ))  # x:2:y
```

split

- Special case: To split a string to its characters: Use the **list()** function.
- Split using more than one splitter: use **re.split**

```
words = "ab:cd:ef".split(':')
print(words)   # ['ab', 'cd', 'ef']

# special case: split by spaces
names = "foo   bar baz".split()
print(names)   # ['foo', 'bar', 'baz']

# special case: split to characters
chars = list("abcd")
print(chars)   # ['a', 'b', 'c', 'd']
```

for loop on lists

```
things = ['apple', 'banana', 'peach', 42]
for var in things:
    print(var)
```

```
apple
banana
peach
42
```

in list

Check if the value is in the list?

```
words = ['apple', 'banana', 'peach', '42']
if 'apple' in words:
    print('found apple')

if 'a' in words:
    print('found a')
else:
    print('NOT found a')

if 42 in words:
    print('found 42')
else:
    print('NOT found 42')

# found apple
# NOT found a
# NOT found 42
```

Where is the element in the list

```
words = ['cat', 'dog', 'snake', 'camel']
print(words.index('snake'))

print(words.index('python'))
```

```
2
Traceback (most recent call last):
  File "examples/lists/index.py", line 6, in <module>
    print(words.index('python'))
ValueError: 'python' is not in list
```

Index improved

```
words = ['cat', 'dog', 'snake', 'camel']

name = 'snake'
```

```
if name in words:
    print(words.index(name))

name = 'python'
if name in words:
    print(words.index(name))
```

[].insert

```
words = ['apple', 'banana', 'cat']
print(words)  # ['apple', 'banana', 'cat']

words.insert(2, 'zebra')
print(words)  # ['apple', 'banana', 'zebra', 'cat']

words.insert(0, 'dog')
print(words)  # ['dog', 'apple', 'banana', 'zebra', 'cat']

# Instead of this, use append (next slide)
words.insert(len(words), 'olifant')
print(words)  # ['dog', 'apple', 'banana', 'zebra', 'cat', 'olifant']
```

[].append

```
names = ['Foo', 'Bar', 'Zorg', 'Bambi']
print(names)  # ['Foo', 'Bar', 'Zorg', 'Bambi']

names.append('Qux')
print(names)  # ['Foo', 'Bar', 'Zorg', 'Bambi', 'Qux']
```

[].remove

```
names = ['Joe', 'Kim', 'Jane', 'Bob', 'Kim']
print(names)                 # ['Joe', 'Kim', 'Jane', 'Bob', 'Kim']

print(names.remove('Kim'))   # None
print(names)                 # ['Joe', 'Jane', 'Bob',
```

```
'Kim']

print(names.remove('George'))
   # Traceback (most recent call last):
   #   File "examples/lists/remove.py", line 9, in <module>
   #     print(names.remove('George'))  # None
   # ValueError: list.remove(x): x not in list
```

> Remove **first** element from a list given by its value.
> Throws an exception if there is no such element in the list.

Remove element by index [].pop

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter']
print(planets)          # ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter']

third = planets.pop(2)
print(third)            # Earth
print(planets)          # ['Mercury', 'Venus', 'Mars', 'Jupiter']

last = planets.pop()
print(last)             # Jupiter
print(planets)          # ['Mercury', 'Venus', 'Mars']

# planets.pop(4)          # IndexError: pop index out of range

jupyter_landers = []
# jupyter_landers.pop()   # IndexError: pop from empty list
```

Remove and return the last element of a list. Throws an exception if the list was empty.

Remove first element of list

To remove an element by its index, use the slice syntax:

```
names = ['foo', 'bar', 'baz', 'moo']

first = names.pop(0)
print(first)    # foo
print(names)    # ['bar', 'baz', 'moo']
```

Remove several elements of list by index

To remove an element by its index, use the slice syntax:

```
names = ['foo', 'bar', 'baz', 'moo', 'qux']

names[2:4] = []
print(names)    # ['foo', 'bar', 'qux']
```

Use list as a queue

```
a_queue = []
print(a_queue)

a_queue.append('Moo')
```

```
print(a_queue)

a_queue.append('Bar')
print(a_queue)

first = a_queue.pop(0)
print(first)
print(a_queue)
```

```
[]
['Moo']
['Moo', 'Bar']
Moo
['Bar']
```

Queue using deque from collections

```
from collections import deque

# items = deque([])
items = deque(['foo', 'bar'])

print(type(items))   # <type 'collections.deque'>
print(items)         # deque(['foo', 'bar'])

items.append('zorg')
print(items)         # deque(['foo', 'bar', 'zorg'])
print(len(items))    # 3

items.append('zorg')
print(items)         # deque(['foo', 'bar', 'zorg', 'zorg'])

nxt = items.popleft()
print(nxt)           # 'foo'
print(items)         # deque(['bar', 'zorg', 'zorg'])

print(len(items))    # 3

if items:
    print("The queue has items")
```

```python
else:
    print("The queue is empty")
```

- .append
- .popleft
- len() number of elements
- if q: to see if it has elements or if it is empty
- dequeue

Fixed size queue

```python
from collections import deque

queue = deque([], maxlen = 3)
print(len(queue))      # 0
print(queue.maxlen)    # 3

queue.append("Foo")
queue.append("Bar")
queue.append("Baz")
print(queue)           # deque(['Foo', 'Bar', 'Baz'], maxlen=3)

queue.append("Zorg")   # Automatically removes the left-most (first) element
print(queue)           # deque(['Bar', 'Baz', 'Zorg'], maxlen=3)
```

List as a stack

```python
stack = []

stack.append("Joe")
print(stack)
stack.append("Jane")
print(stack)
stack.append("Bob")
print(stack)

```

```python
while stack:
    name = stack.pop()
    print(name)
    print(stack)
```

```
['Joe']
['Joe', 'Jane']
['Joe', 'Jane', 'Bob']
Bob
['Joe', 'Jane']
Jane
['Joe']
Joe
[]
```

stack with deque

```python
from collections import deque
stack = deque()

stack.append("Joe")
stack.append("Jane")
stack.append("Bob")

while stack:
    name = stack.pop()
    print(name)

# Bob
# Jane
# Joe
```

Exercies: Queue

The application should manage a queue of people.

- It will prompt the user for a new name by printing **:**, the user can type in a name and press ENTER. The app will add the name to the queue.

- If the user types in “n” then the application will remove the first name from the queue and print it.
- If the user types in “x” then the application will print the list of users who were left in the queue and it will exit.
- If the user types in “s” then the application will show the current number of elements in the queue.

```
: Foo
: Bar
: Moo
: n
  next is Foo
: n
  next is Bar
: Peter
: n
  next is Moo
: n
  next is Peter
: n
  the queue is empty
```

Exercise: Stack

Implement a Reverse Polish Calculator

```
2
3
4
+
*
=
14
```

```
x = eXit, s = Show, [+-*/=]
:23
:19
:7
:8
```

```
:+
:3
:-
:/
:s
[23.0, -0.631578947368421]
:+
:=
22.36842105263158
:s
[]
:x
```

Solution: Queue with list

```python
queue = []

while True:
    inp = input(":")
    inp = inp.rstrip("\n")

    if inp == 'x':
        for name in queue:
           print(name)
        exit()

    if inp == 's':
        print(len(queue))
        continue

    if inp == 'n':
        if len(queue) > 0:
            print("next is {}".format(queue.pop(0)))
        else:
            print("the queue is empty")
        continue

    queue.append(inp)
```

Solution: Queue with deque

```python
from collections import deque

queue = deque()

while True:
    inp = input(":")
    inp = inp.rstrip("\n")

    if inp == 'x':
        for name in queue:
           print(name)
        exit()

    if inp == 's':
        print(len(queue))
        continue

    if inp == 'n':
        if len(queue) > 0:
            print("next is {}".format(queue.popleft()))
        else:
            print("the queue is empty")
        continue

    queue.append(inp)
```

Solution: Reverse Polish calculator (stack) with lists

```python
stack = []

print("x = eXit, s = Show, [+-*/=]")
while True:
    val = input(':')

    if val == 's':
        print(stack)
        continue

    if val == 'x':
        break

```

```
    if val == '+':
        a = stack.pop()
        b = stack.pop()
        stack.append(a+b)
        continue

    if val == '-':
        a = stack.pop()
        b = stack.pop()
        stack.append(a-b)
        continue

    if val == '*':
        a = stack.pop()
        b = stack.pop()
        stack.append(a*b)
        continue

    if val == '/':
        a = stack.pop()
        b = stack.pop()
        stack.append(a/b)
        continue

    if val == '=':
        print(stack.pop())
        continue

    stack.append(float(val))
```

Solution: Reverse Polish calculator (stack) with deque

```
from collections import deque

stack = deque()

while True:
    val = input(':')

    if val == 'x':
        break
```

```

    if val == '+':
        a = stack.pop()
        b = stack.pop()
        stack.append(a+b)
        continue

    if val == '*':
        a = stack.pop()
        b = stack.pop()
        stack.append(a*b)
        continue


    if val == '=':
        print(stack.pop())
        continue

    stack.append(float(val))
```

Debugging Queue

The following implementation has a bug. (Even though the n was supposed to remove the element
and the code seems to mean that it does, we still see two items after we removed the first.)

The question is how to debug this?

```
q = []

while True:
    name=input("your name: ")

    if name=="n":
        print(q.pop(0))

    if name=="x":
        print(q)
        exit()

```

```
    if name=="s":
        print(len(q))
        exit()
    else:
        q.append(name)
        continue
```

```
your name: Foo
your name: Bar
your name: n
Foo
your name: s
2
```

sort

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn']
print(planets)     # ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn']
planets.sort()
print(planets)     # ['Earth', 'Jupiter', 'Mars', 'Mercury', 'Saturn', 'Venus']

planets.sort(reverse=True)
print(planets)     # ['Venus', 'Saturn', 'Mercury', 'Mars', 'Jupiter', 'Earth']
```

sort numbers

```
numbers = [7, 2, -4, 19, 8]
print(numbers)                          # [7, 2, -4, 19, 8]
numbers.sort()
print(numbers)                          # [-4, 2, 7, 8, 19]

numbers.sort(reverse=True)
print(numbers)                          # [19, 9, 7, 2, -4]

numbers.sort(key=abs, reverse=True)
print(numbers)                          # [19, 9, 7, -4, 2]
```

sort mixed

```
mixed = [100, 'foo', 42, 'bar']
print(mixed)
mixed.sort()
print(mixed)
```

In Python 2 puts the numbers first in numerical order and then the strings in ASCII order.

```
[100, 'foo', 42, 'bar']
[42, 100, 'bar', 'foo']
```

In Python 3 it throws an exception.

```
[100, 'foo', 42, 'bar']
Traceback (most recent call last):
  File "examples/lists/sort_mixed.py", line 5, in <module>
    mixed.sort()
TypeError: unorderable types: str() < int()
```

key sort

- Another example to using a **key**.
- To sort the list according to length

```
animals = ['chicken', 'cow', 'snail', 'elephant']
print(animals)

animals.sort()
print(animals)

animals.sort(key=len)
print(animals)

animals.sort(key=len, reverse=True)
print(animals)
```

```
['chicken', 'cow', 'snail', 'elephant']
['chicken', 'cow', 'elephant', 'snail']
['cow', 'snail', 'chicken', 'elephant']
['elephant', 'chicken', 'snail', 'cow']
```

Sort tuples

Sorting tuples or list, or other complex structures

```
students = [
    ('John', 'A', 2),
    ('Zoro', 'C', 1),
    ('Dave', 'B', 3),
]
print(students)
  # [('John', 'A', 2), ('Zoro', 'C', 1), ('Dave', 'B', 3)]

print(sorted(students))
  # [('Dave', 'B', 3), ('John', 'A', 2), ('Zoro', 'C', 1)]
  # sort by the first element of each tuple

print(sorted(students, key=lambda s : s[1]))
  # [('John', 'A', 2), ('Dave', 'B', 3), ('Zoro', 'C', 1)]
  # sort by the 2nd element of the tuples (index 1)

print(sorted(students, key=lambda s : s[2]))
  # [('Zoro', 'C', 1), ('John', 'A', 2), ('Dave', 'B', 3)]
  # sort by the 3rd element of the tuples (index 2)


from operator import itemgetter
print(sorted(students, key=itemgetter(2)))
  # [('Zoro', 'C', 1), ('John', 'A', 2), ('Dave', 'B', 3)]
  # maybe this is more simple than the lambda version
  # and probably faster
```

sort with sorted

```
animals = ['chicken', 'cow', 'snail', 'elephant']
print(animals)          # ['chicken', 'cow', 'snail', 'elephant']

s = sorted(animals)
print(s)                # ['chicken', 'cow', 'elephant', 'snail']
print(animals)          # ['chicken', 'cow', 'snail', 'elephant']

r = sorted(animals, reverse=True, key=len)
print(r)                # ['elephant', 'chicken', 'snail', 'cow']
print(animals)          # ['chicken', 'cow', 'snail', 'elephant']
```

sort vs. sorted

The sort() method will sort a list in-place and return None.
The built-in sorted() function will return the sorted list and leave the original list intact.

key sort with sorted

To sort the list according to length using sorted

```
animals = ['snail', 'cow', 'elephant', 'chicken']
animals_in_abc = sorted(animals)

print(animals)
print(animals_in_abc)

animals_by_length = sorted(animals, key=len)
print(animals_by_length)
```

```
['snail', 'cow', 'elephant', 'chicken']
['chicken', 'cow', 'elephant', 'snail']
```

```
['cow', 'snail', 'chicken', 'elephant']
```

Sorting characters of a string

```
letters = 'axzb'
print(letters)          # 'axzb'
s = sorted(letters)
print(s)                # ['a', 'b', 'x', 'z']
print(letters)          # 'axzb'

r = ''.join(sorted(letters))
print(r)                # abxz
```

range

```
for i in range(11, 18, 2):
    print(i)
# 11
# 13
# 15
# 17

for i in range(5, 7):
    print(i)
# 5
# 6

for i in range(3):
    print(i)
# 0
# 1
# 2
```

Looping over index

```
things = ['abc', 'def', 'ghi', 42]
for var in things:
    print(var)
```

```python
things = ['abc', 'def', 'ghi', 42]
for i in range(len(things)):
    print(i, things[i])

# 0 abc
# 1 def
# 2 ghi
# 3 42
```

Enumerate lists

```python
planets = ['Mercury', 'Venus', 'Earth', 'Mars',
'Jupiter', 'Saturn']
for idx, planet in enumerate(planets):
    print(idx, planet)

print('')
enu = enumerate(planet)
print(enu.__class__.__name__)
print(enu)
```

```
0 Mercury
1 Venus
2 Earth
3 Mars
4 Jupiter
5 Saturn

enumerate
<enumerate object at 0x7f2c2402adc8>
```

List operators

```python
a = ['one', 'two']
b = ['three']

print(a)     # ['one', 'two']
print(a * 2) # ['one', 'two', 'one', 'two']
print(a + b) # ['one', 'two', 'three']
```

List of lists

```python
x = ['abc', 'def']
print(x)      # ['abc', 'def']

y = [x, 'xyz']
print(y)           # [['abc', 'def'], 'xyz']
print(y[0])        #  ['abc', 'def']

print(x[0])        #    abc
print(y[0][0])     #    abc
```

List assignment

List assignment works in “parallel” in Python.

```python
x, y = 1, 2
print(x)       # 1
print(y)       # 2

x, y = y, x
print(x)       # 2
print(y)       # 1
```

```python
x,y = f()   # works if f returns a list of 2 elements
```

It will throw a run-time ValueError exception if the number of values in the returned list is not 2. (Both for fewer and for more return values).

List documentation

- datastructures

tuple

Tuple

- A tuple is a fixed-length immutable list. It cannot change its size or content.
- A tuple is denoted with parentheses: (1,2,3)

```
t = ('a', 'b', 'c')
print(t)   # ('a', 'b', 'c')
```

List

- Elements of a list can be changed via their index or via the list slice notation.
- A list can grow and shrink using **append** and **pop** methods or using the **slice** notation.
- A list is denoted with square brackets: [1, 2, 3]

```
l = ['abc', 'def', 'qqrq']
t = tuple(l)
print(l) # ['abc', 'def', 'qqrq']
print(t) # ('abc', 'def', 'qqrq')
```

Tuples are rarely used. There are certain places where Python or some module require tuple (instead of list) or return a tuple (instead of a list)
and in each place it will be explained. Otherwise you don’t need to use tuples.

e.g. keys of dictinoaries can be tuple (but not lists).

Exercise: color selector menu

- In a script have a list of colors. Write a script that will display a menu (a list of numbers and the corresponding color) and prompts the user for a number. The user needs to type in one of the numbers. That's the selected color.

1. blue
2. green
3. yellow
4. white

- For extra credit make sure the system is user-proof and it won't blow up on various incorrect input values. (e.g Floating point number. Number that is out of range, non-number)
- For more credit allow the user to supply the number of the color on the command line. **python color.py 3**. If that is available, don't prompt.
- For further credit allow the user to provide the name of the color on the command line: **python color.py yellow** Can you handle color names that are not in the expected case (e.g. YelloW)?
- Any more ideas for improvement?

Exercise: count digits

Given a list of numbers `numbers = [1203, 1256, 312456, 98]`, count how many times each digit appears? The output will look like this:

```
0  1
1  3
2  3
3  2
4  1
5  2
```

```
6   2
7   0
8   1
9   1
```

Exercise: Create list

Given a list of strings with words separated by spaces, create a single list of all the words.

```
lines = [
    'grape banana mango',
    'nut orange peach',
    'apple nut banana apple mango',
]

fruits = ['grape', 'banana', 'mango', 'nut', 'orange', 'peach', 'apple', 'nut', 'ban\
ana', 'apple', 'mango']
```

Then create a list of unique values sorted by abc.

```
unique_fruites = ['apple', 'banana', 'grape', 'mango', 'nut', 'orange', 'peach']
```

Exercise: Count words

```
celestial_objects = [
    'Moon', 'Gas', 'Asteroid', 'Dwarf', 'Asteroid', 'Moon', 'Asteroid'
]
```

Expected output:

```
Moon        2
Gas         1
Asteroid    3
Dwarf       1
```

Exercise: Check if number is prime

Write a program that gets a number on the commnad line a prints "True" if the number is a prime
number or "False" if it isn't.

```
python is_prime.py 42
False
python is_prime.py 19
True
```

Exercise: DNA sequencing

- A, C, T, G are called bases or nucleotides
- Given a sequence like **'ACCGXXCXXGTTACTGGGCXTTGT'** (nucleoids mixed up with other elements) return the sequences containing only ACTG orderd by length.
- The above string can be split up to **['ACCG', 'C', 'GTTACTGGGC', 'TTGT']** and then it can be sorted to get the following:
- Expected result: **['GTTACTGGGC', 'ACCG', 'TTGT', 'C']**

Solution: menu

```
colors = ['blue', 'yellow', 'black', 'purple']
for ix in range(len(colors)):
    print("{}) {}".format(ix+1, colors[ix]))

selection = input("Select color: ")
if not selection.isdecimal():
    exit(f"We need a number between 1 and {len(colors)}")

if int(selection) < 1 or int(selection) > len(colors):
```

```
    exit(f"The number must be between 1 and {len(colors)}")

col = int(selection) - 1
print(colors[col])
```

- We would like to show a menu where each number corresponds to one element of the list so this is one of the places where we need to iterate over the indexes of a list.
- `len(colors)` gives us the length of the list (in our case 4)
- `range(len(colors))` is the range of numbers betwwen 0 and 4 (in our case), menaing 0, 1, 2, 3.
- (Sometimes people explicetly write 4 in this solution, but if later we change the list and include another color we'll have to remember updating this number as well. This is error prone and it is very easy to deduct this number from the data we already have. (The list.))
- We start the list from 0, but when we display the menu we would like to show the numbers 1-4 to make it more human friendly. Therefore we show `ix+1` and the color from locations `ix`.
- We ask for input and save it in a variable.

- We use the `isdecimal` method to check if the user typed in a decimal number. We give an error and exit if not.
- Then we check if the users provided a number in the correct range of values. We give an error and exit if not.
- then we convert the value to the correct range of numbers (remember, the user sees and selects numbers between 1-4 and we need them between 0-3).

Solution: count digits

```
numbers = [1203, 1256, 312456, 98]

count = [0] * 10 # same as [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

for num in numbers:
    for char in str(num):
        count[int(char)] += 1

for d in range(0, 10):
    print("{}  {}".format(d, count[d]))
```

First we have to decide where are we going to store the counts. A 10 element long list seems to fit our requirements so if we have 3 0s and 2 8s we would have `[3, 0, 0, 0, 0, 0, 0, 0, 2, 0]`.

- We have a list of numbers.
- We need a place to store the counters. For this we create a variable called counter which is a list of 10 0s. We are going to count the number of times the digit 3 appears in `counters[3]`.
- We iterate over the numbers so `num` is the current number. (e.g. 1203)
- We would like to iterate over the digits in the curreent number now, but if we write `for var in num` we will get an error `TypeError: 'int' object is not iterable` because `num` is a number, but numbers are not iterables, so we we cannot iterate over them. So we need to convert it to a string useing `str`.
- On each iteration `char` will be one character (which in or case we assume that will be a digit, but still stored as a string).
- `int(char)` will convert the string to a number so for example "2" will be converted to 2.
- `count[int(char)]` is going to be `char[2]` if `char` is "2". That's the location in the list where we count how many times the digit 2 appears in our numbers.

* We increment it by one as we have just encountered a new copy of the given digit.
* That finished the data collection.

* The second for-loop iterates over all the “possible digits” that is from 0-9, prints out the digit and the counter in the respective place.

Solution: Create list

```
lines = [
    'grape banana mango',
    'nut orange peach',
    'apple nut banana apple mango',
]

one_line = ' '.join(lines)
print(one_line)
fruits = one_line.split()
print(fruits)

unique_fruits = []
for word in fruits:
    if word not in unique_fruits:
        unique_fruits.append(word)
print(sorted(unique_fruits))


# a simpler way using a set, but we have not learned sets yet.
unique = sorted(set(fruits))
print(unique)
```

Solution: Count words

```
celestial_objects = [
    'Moon', 'Gas', 'Asteroid', 'Dwarf', 'Asteroid', 'Moon', 'Asteroid'
]
```

```
names   = []
counter = []

for name in celestial_objects:
    if name in names:
        idx = names.index(name)
        counter[idx] += 1
    else:
        names.append(name)
        counter.append(1)

for i in range(len(names)):
    print("{:12}   {}".format(names[i], counter[i]))
```

Solution: Check if number is prime

```
import sys

n = int(sys.argv[1])

#print(n)

is_prime = True
for i in range(2, int( n ** 0.5) + 1):
    if n % i == 0:
        is_prime = False
        break

print(is_prime)


# math.sqrt(n) might be clearer than n ** 0.5
```

Solution: DNA sequencing

```
dna = 'ACCGXXCXXGTTACTGGGCXTTGT'
sequences = dna.split('X')
sequences.sort(key=len, reverse=True)

new_seq = []
```

```
for w in sequences:
   if len(w) > 0:
      new_seq.append(w)

print(sequences)
print(new_seq)
```

Solution: DNA sequencing with filter

```
dna = 'ACCGXXCXXGTTACTGGGCXTTGT'
sequences = dna.split('X')
sequences.sort(key=len, reverse=True)

def not_empty(x):
    return len(x) > 0

print(sequences)
sequences = list( filter(not_empty, sequences) )
print(sequences)
```

Solution: DNA sequencing with filter and lambda

```
dna = 'ACCGXXCXXGTTACTGGGCXTTGT'
sequences = dna.split('X')
sequences.sort(key=len, reverse=True)

print(sequences)
sequences = list( filter(lambda x: len(x) > 0, sequences)
)
print(sequences)
```

[].extend

```
names = ['Foo Bar', 'Orgo Morgo']

names.extend(['Joe Doe', 'Jane Doe'])
print(names) # ['Foo Bar', 'Orgo Morgo', 'Joe Doe', 'Jane
Doe']
```

append vs. extend

What is the difference between [].append and [].extend ?
The method **append** adds its parameter as a single element to the list, while **extend** gets a list and adds its content.

```
names = ['Foo Bar', 'Orgo Morgo']
more = ['Joe Doe', 'Jane Doe']
names.extend(more)
print(names)  # ['Foo Bar', 'Orgo Morgo', 'Joe Doe', 'Jane Doe']

names = ['Foo Bar', 'Orgo Morgo']
names.append(more)
print(names) # ['Foo Bar', 'Orgo Morgo', ['Joe Doe', 'Jane Doe']]

names = ['Foo', 'Bar']
names.append('Qux')
print(names)   # ['Foo', 'Bar', 'Qux']

names = ['Foo', 'Bar']
names.extend('Qux')
print(names)   # ['Foo', 'Bar', 'Q', 'u', 'x']
```

split and extend

When collecting data which is received from a string via splitting,
we would like to add the new elements to the existing list:

```
lines = [
    'abc def ghi',
    'hello world',
]

collector = []

for l in lines:
   collector.extend(l.split())
   print(collector)

# ['abc', 'def', 'ghi']
# ['abc', 'def', 'ghi', 'hello', 'world']
```

Files

Open and read file

```
filename = 'examples/files/numbers.txt'

with open(filename, 'r') as fh:
    for line in fh:
        print(line)            # duplicate newlines

# close is called when we leave the 'with'
```

Filename on the command line

```
import sys

def main():
    if len(sys.argv) != 2:
        exit("Usage: " + sys.argv[0] + " FILENAME")
    filename = sys.argv[1]
    with open(filename) as fh:
        print("Working on the file", filename)

main()
```

```
$ python single.py
Usage: single.py FILENAME

$ python single.py numbers.txt
Working on the file numbers.txt
```

Filehandle with and without

```
filename = 'examples/files/numbers.txt'

```

```
fh = open(filename, 'r')
print(fh)       # <open file 'numbers.txt', mode 'r' at 0x107084390>
data = fh.read()
# do something with the data
fh.close()
print(fh)       # <closed file 'numbers.txt', mode 'r' at 0x107084390>



with open(filename, 'r') as fh:
   print(fh)   # <open file 'numbers.txt', mode 'r' at 0x1070840c0>
   data = fh.read()
print(fh)      # <closed file 'numbers.txt', mode 'r' at 0x1070840c0>
```

Filehandle with return

```
import sys

def process_file(filename):
    with open(filename, 'r') as fh:

        for line in fh:
            line = line.rstrip("\n")
            if len(line) > 0:
                if line[0] == '#':
                    return
# some comment

            if len(line) > 1:
                if line[0:2] == '//':
                    return

            print(line)


process_file(sys.argv[0])
```

Read file remove newlines

```
filename = 'examples/files/numbers.txt'

with open(filename, 'r') as fh:
    for line in fh:
        line = line.rstrip("\n")
        print(line)
```

Read all the lines into a list

```
filename = 'examples/files/numbers.txt'

with open(filename, 'r') as fh:
    lines_list = fh.readlines()   # reads all the lines into a list

# print number of lines
print(len(lines_list))

for line in lines_list:
    print(line, end="")
```

Read all the characters into a string (slurp)

```
filename = 'examples/files/numbers.txt'

with open(filename, 'r') as fh:
    lines_str = fh.read()   # reads all the lines into a string

print(len(lines_str))   # number of characters in file

print(lines_str)        # the content of the file
```

read(20) will read 20 bytes.

Not existing file

```
filename = 'examples/files/unicorns.txt'

with open(filename, 'r') as fh:
    lines  = fh.read()
print("still running")

# Traceback (most recent call last):
#   File "examples/files/open_file.py", line 5, in <module>
#     with open(filename, 'r') as fh:
# IOError: [Errno 2] No such file or directory: 'examples/files/unicorns.txt'
```

Open file exception handling

Exception handling

```
filename = 'examples/files/unicorns.txt'

try:
    with open(filename, 'r') as fh:
        lines = fh.read()
except Exception as err:
    print('There was some error in the file operations.')
    print(err)
    print(type(err).__name__)

print('Still running.')
```

Open many files - exception handling

```
import sys


def main():
    for filename in sys.argv[1:]:
        try:
            #do_some_stuff(filename)
            with open(filename) as fh:
```

```
                total = 0
                count = 0
                for line in fh:
                    number = float(line)
                    total += number
                    count += 1
                print("Average: ", total/count)
        except Exception:
            print("trouble with {}".format(filename))

main()
```

```
23
1
192
17
```

```

```

```
python average_from_files.pyt number_per_line.txt empty.txt number_per_line2.txt
```

```
Average:  58.25
trouble with empty.txt
Average:  3.5
```

Writing to file

```
filename = 'data.txt'

with open(filename, 'w') as out:
    out.write('text\n')
```

Append to file

```
filename = 'data.txt'

```

```
with open(filename, 'a') as out:
    out.write('append more text\n')
```

Binary mode

```
filename = 'README'

try:
    with open(filename, 'rb') as fh:
        while True:
            binary_str = fh.read(5000)
            print(len(binary_str))
            if len(binary_str) == 0:
                break
            # do something with the content of the
binary_str
except Exception:
    pass

# 5000
# 5000
# 5000
# 1599
# 0
```

Does file exist? Is it a file?

- os.path.exists
- os.path.isfile
- os.path.isdir

Exercise: count numbers

```
23 345 12345
67 189 23 17
```

1. Given the file **examples/files/numbers.txt** (or a similar file), count how many times each digit appears? The output will look like this. Just different values.
2. Save the results in a file called **report.txt**.

```
0 0
1 3
2 3
3 4
4 2
5 2
6 1
7 2
8 1
9 1
```

Exercise: strip newlines

How to read all the lines of a file into a list
and remove trailing newlines?

Exercise: color selector

Create a file similar to the colors.txt file
and use it as the list of colors in the earlier example where we prompted for a color.

```
blue
yellow
white
green
```

Extend the previous example by letting the user provide the name of the file on the command line:

```
python color.py examples/files/color.txt
```

Exercise: ROT13

Implement ROT13:

- Create a function that given a string return the rot13 of it.
- Create a script that given a file it will replace with the rot13 of it.

How to check if it works properly:

```
txt = "any text"
encrypted = rot13(txt)
decrypted = rot13(encrypted)
assert decrypted == text
```

Exercise: Combine lists

```
Tomato=78
Avocado=23
Pumpkin=100
```

```
Cucumber=17
Avocado=10
Cucumber=10
```

Write a script that takes the two files and combines them adding the values for each vegetable. The expected result is:

```
Avocado=33
Cucumber=27
Pumpkin=100
Tomato=78
```

Solution: count numbers

```
import sys

if len(sys.argv) < 2:
    exit("Need name of file.")

counter = [0] * 10
filename = sys.argv[1]
with open(filename) as fh:
    for line in fh:
        for c in line.rstrip("\n"):
            if c == ' ':
                continue

            c = int(c)
            counter[c] += 1

for i in range(10):
    print("{} {}".format(i, counter[i]))
```

Solution: strip newlines

```
import sys
filename = sys.argv[0]
with open(filename) as fh:
    lines = []
    for line in fh:
        lines.append(line.rstrip("\n"))
    print(lines)
```

Solution: color selector

```
def main():
    try:
        with open('colors.txt') as fh:
            colors = []
            for line in fh:
                colors.append(line.rstrip("\n"))
    except IOError:
        print("Could not open colors.txt")
        exit()

```

```
    for i in range(len(colors)):
        print("{}) {}".format(i, colors[i]))

    c = int(input("Select color: "))
    print(colors[c])

main()
```

Solution: Combine lists

```
a_names = []
a_values = []
with open('examples/files/a.txt') as fh:
    for line in fh:
        k, v = line.rstrip("\n").split("=")
        a_names.append(k)
        a_values.append(int(v))

b_names = []
b_values = []
with open('examples/files/b.txt') as fh:
    for line in fh:
        k, v = line.rstrip("\n").split("=")
        b_names.append(k)
        b_values.append(int(v))

c_names = []
c_values = []

for i in range(len(a_names)):
    if a_names[i] in c_names:
        j = c_names.index(a_names[i])
        c_values[j] += a_values[i]
    else:
        c_names.append( a_names[i] )
        c_values.append( a_values[i] )

for i in range(len(b_names)):
    if b_names[i] in c_names:
        j = c_names.index(b_names[i])
        c_values[j] += b_values[i]
    else:
        c_names.append( b_names[i] )
```

```
        c_values.append( b_values[i] )


with open('out.txt', 'w') as fh:
    for i in range(len(c_names)):
        fh.write("{}={}\n".format(c_names[i], c_values[i]))
```

Read text file

```
filename = 'examples/files/numbers.txt'

with open(filename, 'r') as fh:    # open(filename) would be enough
    for line in fh:
        print(line)             # duplicate newlines
        #print(line, end="")    # eliminte the trailing newline of print
```

Open and read file

In some code you will encounter the following way of opening files.
This was used before “with” was added to the language.
It is not a recommended way of opening a file as you might easily forget
to call “close” and that might cause trouble. For example you might loose data.
Don’t do that.

```
filename = 'examples/files/numbers.txt'

fh = open(filename, 'r')
for line in fh:
```

```
    print(line)             # duplicate newlines
fh.close()
```

Direct access of a line in a file

```
names = ['Foo', 'Bar', 'Baz']
for name in names:
    print(name)
print(names[1])


filename = 'data/README'
with open(filename, 'r') as fh:
    for line in fh:
        print(line)

with open(filename, 'r') as fh:
    print(fh[2])
```

```
Traceback (most recent call last):
  File "examples/files/fh_access.py", line 14, in <module>
    print(fh[2])
TypeError: '_io.TextIOWrapper' object is not subscriptable
```

> This does NOT work because files can only be accessed sequentially.

Example

```
begin test
do something
report
total: 42
more things
```

```
more
another total: 100
more data
```

```python
import sys
import os

#print(sys.argv)
if len(sys.argv) < 2:
    #exit()
    exit(f"Usage: {sys.argv[0]} FILENAME")

# print(sys.argv[0])
# print(sys.argv[1])

#filename = 'sample.txt'

#filename = input("type in filename: ")

filename = sys.argv[1]

#if not os.path.exists(filename):
#    exit(f"File {filename} does not exist")

with open(filename, 'r') as fh:
    for line in fh:
        line = line.rstrip("\n")
        print(line)
        #if "total" in line:
        #    print(line)

```

Dictionary (hash)

What is a dictionary

- Unordered key-value pairs.
- Keys are immutables (numbers, strings, tuples).
- Values can be any object.

When to use dictionaries

- ID to Name mapping.
- Object to Count mapping.
- Name of a feature to value of the feature.
- Name of an attribute to value of the attribute.

Dictionary

```
user = {}
user['name'] = 'Foobar'
print(user)         # {'name': 'Foobar'}

user['email'] = 'foo@bar.com'
print(user)         # {'name': 'Foobar', 'email': 'foo@bar.com'}

the_name = user['name']
print(the_name)     # Foobar

field = 'name'
the_value = user[field]
print(the_value)    # Foobar

user['name'] = 'Edith Piaf'
```

```
print(user)      # {'name': 'Edith Piaf', 'email': 'foo@bar.com'}
```

keys

```
user = {
   'fname': 'Foo',
   'lname': 'Bar',
}

print(user)   # {'lname': 'Bar', 'fname': 'Foo'}

print(user.keys())    # ['lname', 'fname']
```

- Keys are returned in seemingly random order.

Loop over keys

```
user = {
    'fname': 'Foo',
    'lname': 'Bar',
}

for k in user.keys():
    print(k)

# lname
# fname

for k in user.keys():
    print("{} -> {}".format(k, user[k]))

# lname -> Bar
# fname -> Foo
```

Loop using items

```
people = {
    "foo" : "123",
```

```
    "bar" : "456",
    "qux" : "789",
}

for name, uid in people.items():
    print("{} => {}".format(name, uid))
```

```
foo => 123
bar => 456
qux => 789
```

```
user = {
    'fname': 'Foo',
    'lname': 'Bar',
}

for t in user.items():      #  returns tuples
    print("{} -> {}".format(t[0], t[1]))
    #print("{} -> {}".format(*t))

# lname -> Bar
# fname -> Foo
```

values

- Values are returned in the same random order as the keys are.

```
user = {
   'fname': 'Foo',
   'lname': 'Bar',
}

print(user)   # {'lname': 'Bar', 'fname': 'Foo'}

print(user.keys())    # ['lname', 'fname']

print(user.values())  # ['Bar', 'Foo']
```

Not existing key

If we try to fetch the value of a key that does not exist, we get an exception.

```
def main():
    user = {
        'fname': 'Foo',
        'lname': 'Bar',
    }

    print(user['fname'])
    print(user['email'])

main()
```

```
Foo
Traceback (most recent call last):
  File "examples/dictionary/no_such_key.py", line 11, in <module>
    main()
  File "examples/dictionary/no_such_key.py", line 9, in main
    print(user['email'])
KeyError: 'email'
```

Get key

If we use the `get` method, we get `None` if the key does not exist.

```
user = {
    'fname': 'Foo',
    'lname': 'Bar',
    'address': None,
}

print(user.get('fname'))
print(user.get('address'))
print(user.get('email'))

print(user.get('answer', 42))
```

```
Foo
None
None
42
```

`None` will be interpreted as `False`, if checked as a boolean.

Does the key exist?

```
user = {
    'fname': 'Foo',
    'lname': 'Bar',
}

print('fname' in user)   # True
print('email' in user)   # False
print('Foo' in user)     # False

for k in ['fname', 'email', 'lname']:
    if k in user:
        print("{} => {}".format(k, user[k]))

# fname => Foo
# lname => Bar
```

```
True
False
False
fname => Foo
lname => Bar
```

Does the value exist?

```
user = {
   'fname': 'Foo',
   'lname': 'Bar',
}

```

```
print('fname' in user.values())  # False
print('Foo' in user.values())    # True
```

```
False
True
```

Delete key

```
user = {
    'fname': 'Foo',
    'lname': 'Bar',
    'email': 'foo@bar.com',
}

print(user) # {'lname': 'Bar', 'email': 'foo@bar.com',
'fname': 'Foo'}

fname = user['fname']
del user['fname']
print(fname) # Foo
print(user) # {'lname': 'Bar', 'email': 'foo@bar.com'}

lname_was = user.pop('lname')
print(lname_was) # Bar
print(user) # {'email': 'foo@bar.com'}
```

```
{'fname': 'Foo', 'lname': 'Bar', 'email': 'foo@bar.com'}
Foo
{'lname': 'Bar', 'email': 'foo@bar.com'}
Bar
{'email': 'foo@bar.com'}
```

List of dictionaries

```
people = [
    {
        'name'  : 'Foo Bar',
        'email' : 'foo@example.com'
    },
```

```
    {
        'name'     : 'Qux Bar',
        'email'    : 'qux@example.com',
        'address'  : 'Borg, Country',
        'children' : [
            'Alpha',
            'Beta'
        ]
    }
]

print(people)
print(people[0]['name'])
print(people[1]['children'][0])

print(list(map(lambda p: p['name'], people)))
```

```
[{'name': 'Foo Bar', 'email': 'foo@example.com'}, {'name': 'Qux Bar', 'email': 'qux@\
example.com', 'address': 'Borg, Country', 'children': ['Alpha', 'Beta']}]
Foo Bar
Alpha
['Foo Bar', 'Qux Bar']
```

Shared dictionary

```
people = [
    {
       "name" : "Foo",
       "id"   : "1",
    },
    {
       "name" : "Bar",
       "id"   : "2",
    },
    {
       "name" : "Moo",
       "id"   : "3",
    },
]

```

```
by_name = {}
by_id = {}
for p in people:
    by_name[ p['name' ] ] = p
    by_id[ p['id' ] ] = p
print(by_name)
print(by_id)

print(by_name["Foo"])
by_name["Foo"]['email'] = 'foo@weizmann.ac.il'
print(by_name["Foo"])

print(by_id["1"])
```

```
{'Foo': {'name': 'Foo', 'id': '1'}, 'Bar': {'name':
'Bar', 'id': '2'}, 'Moo': {'name\
': 'Moo', 'id': '3'}}
{'1': {'name': 'Foo', 'id': '1'}, '2': {'name': 'Bar',
'id': '2'}, '3': {'name': 'Mo\
o', 'id': '3'}}
{'name': 'Foo', 'id': '1'}
{'name': 'Foo', 'id': '1', 'email': 'foo@weizmann.ac.il'}
{'name': 'Foo', 'id': '1', 'email': 'foo@weizmann.ac.il'}
```

immutable collection: tuple as dictionary key

```
points = {}
p1 = (2, 3)

points[p1] = 'Joe'
points[(17, 5)] = 'Jane'

print(points)
for k in points.keys():
   print(k)
   print(k.__class__.__name__)
   print(points[k])
```

```
{(2, 3): 'Joe', (17, 5): 'Jane'}
(2, 3)
tuple
```

```
Joe
(17, 5)
tuple
Jane
```

immutable numbers: numbers as dictionary key

```
number = {
    23   : "Twenty three",
    17   : "Seventeen",
    3.14 : "Three dot fourteen",
    42   : "The answer",
}

print(number)
print(number[42])
print(number[3.14])
```

```
{23: 'Twenty three', 17: 'Seventeen', 3.14: 'Three dot fourteen', 42: 'The answer'}
The answer
Three dot fourteen
```

Sort dictionary by value

```
scores = {
   'Foo' : 10,
   'Bar' : 34,
   'Miu' : 88,
}

print(scores) # {'Miu': 88, 'Foo': 10, 'Bar': 34}

sorted_names = sorted(scores)
print(sorted_names)  # ['Bar', 'Foo', 'Miu']
for s in sorted_names:
    print("{}  {}".format(s, scores[s]))

# sort the values, but we cannot get the keys back!
```

```
print(sorted(scores.values())) # [10, 34, 88]

print('')

# sort using a lambda expression
sorted_names = sorted(scores, key=lambda x: scores[x])
for k in sorted_names:
    print("{} : {}".format(k, scores[k]))

# Foo : 10
# Bar : 34
# Miu : 88

print('')

# sort the keys according to the values:
sorted_names = sorted(scores, key=scores.__getitem__)
for k in sorted_names:
    print("{} : {}".format(k, scores[k]))

# Foo : 10
# Bar : 34
# Miu : 88
```

Sort dictionary keys by value

```
scores = {
    "Jane"    : 30,
    "Joe"     : 20,
    "George"  : 30,
    "Hellena" : 90,
}

for name in scores.keys():
    print(f"{name:8} {scores[name]}")

print('')
for name in sorted(scores.keys()):
    print(f"{name:8} {scores[name]}")

print('')
for val in sorted(scores.values()):
    print(f"{val:8}")
```

```
print('')
for name in sorted(scores.keys(), key=lambda x: scores[x]):
    print(f"{name:8} {scores[name]}")
```

```
Jane     30
Joe      20
George   30
Hellena  90

George   30
Hellena  90
Jane     30
Joe      20

      20
      30
      30
      90

Joe      20
Jane     30
George   30
Hellena  90
```

Insertion Order is kept

Since Python 3.7

```
d = {}
d['a'] = 1
d['b'] = 2
d['c'] = 3
d['d'] = 4
print(d)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

Change order of keys in dictionary - OrderedDict

```python
from collections import OrderedDict

d = OrderedDict()
d['a'] = 1
d['b'] = 2
d['c'] = 3
d['d'] = 4

print(d)
d.move_to_end('a')

print(d)
d.move_to_end('d', last=False)

print(d)

for key in d.keys():
    print(key)
```

```
OrderedDict([('a', 1), ('b', 2), ('c', 3), ('d', 4)])
OrderedDict([('b', 2), ('c', 3), ('d', 4), ('a', 1)])
OrderedDict([('d', 4), ('b', 2), ('c', 3), ('a', 1)])
d
b
c
a
```

Set order of keys in dictionary - OrderedDict

```python
from collections import OrderedDict

d = {}
d['a'] = 1
d['b'] = 2
d['c'] = 3
d['d'] = 4
print(d)

```

```
planned_order = ('b', 'c', 'd', 'a')
e = OrderedDict(sorted(d.items(), key=lambda x: planned_order.index(x[0])))
print(e)

print('-----')
# Create index to value mapping dictionary from a list of values
planned_order = ('b', 'c', 'd', 'a')
plan = dict(zip(planned_order, range(len(planned_order))))
print(plan)

f = OrderedDict(sorted(d.items(), key=lambda x: plan[x[0]]))
print(f)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
OrderedDict([('b', 2), ('c', 3), ('d', 4), ('a', 1)])
-----
{'b': 0, 'c': 1, 'd': 2, 'a': 3}
OrderedDict([('b', 2), ('c', 3), ('d', 4), ('a', 1)])
```

Exercise: count characters

Given a long text, count how many times each character appears?

```
text = """
This is a very long text.
OK, maybe it is not that long after all.
"""
```

Extra credit: Change the code so it will be able to count characters of a file.

Exercise: count words

Part of the code:

```
words = ['Wombat', 'Rhino', 'Sloth', 'Tarantula',
'Sloth', 'Rhino', 'Sloth']
```

Expected output: (the order is not important)

```
Wombat:1
Rhino:2
Sloth:3
Tarantula:1
```

Exercise: count words from a file

Given a file with words and spaces and newlines only, count how many times each word appears.

```
Lorem ipsum dolor qui ad labor ad labor sint dolor
tempor incididunt ut labor ad do\
lore lorem ad
Ut labor ad dolor lorem qui ad ut labor   ut ad commodo
commodo
Lorem ad dolor in reprehenderit in lorem ut labor ad
dolore eu in labor dolor
sint occaecat ad labor proident sint in in qui labor ad
dolor ad in ad labor
```

- Based on [Lorem Ipsum]

Expected result for the above file:

```
ad             13
commodo         2
dolor           6
dolore          2
eu              1
in              6
incididunt      1
ipsum           1
labor          10
```

```
lorem         5
occaecat      1
proident      1
qui           3
reprehenderit 1
sint          3
tempor        1
ut            5
```

Exercise: Apache log

Every web server logs the visitors and their requests in a log file. The Apache web server has a log file similar to the following file. (Though I have trimmed the lines for the exercise.) Each line is a “hit”, a request from the browser of a visitor.

Each line starts with the IP address of the visitor. e.g. 217.0.22.3.

Given sucha a log file from Apache, report how many hits (line were from each IP address.

```
127.0.0.1 - - [10/Apr/2007:10:39:11] "GET / HTTP/1.1" 500 606 "-"
127.0.0.1 - - [10/Apr/2007:10:39:11] "GET /favicon.ico HTTP/1.1" 200 766 "-"
139.12.0.2 - - [10/Apr/2007:10:40:54] "GET / HTTP/1.1" 500 612 "-"
139.12.0.2 - - [10/Apr/2007:10:40:54] "GET /favicon.ico HTTP/1.1" 200 766 "-"
127.0.0.1 - - [10/Apr/2007:10:53:10] "GET / HTTP/1.1" 500 612 "-"
127.0.0.1 - - [10/Apr/2007:10:54:08] "GET / HTTP/1.0" 200 3700 "-"
127.0.0.1 - - [10/Apr/2007:10:54:08] "GET /style.css HTTP/1.1" 200 614
127.0.0.1 - - [10/Apr/2007:10:54:08] "GET /img/pti-round.jpg HTTP/1.1" 200 17524
127.0.0.1 - - [10/Apr/2007:10:54:21] "GET /unix_sysadmin.html HTTP/1.1" 200 3880
```

```
217.0.22.3 - - [10/Apr/2007:10:54:51] "GET / HTTP/1.1" 200 34 "-"
217.0.22.3 - - [10/Apr/2007:10:54:51] "GET /favicon.ico HTTP/1.1" 200 11514 "-"
217.0.22.3 - - [10/Apr/2007:10:54:53] "GET /cgi/pti.pl HTTP/1.1" 500 617
127.0.0.1 - - [10/Apr/2007:10:54:08] "GET / HTTP/0.9" 200 3700 "-"
217.0.22.3 - - [10/Apr/2007:10:58:27] "GET / HTTP/1.1" 200 3700 "-"
217.0.22.3 - - [10/Apr/2007:10:58:34] "GET /unix.html HTTP/1.1" 200 3880
217.0.22.3 - - [10/Apr/2007:10:58:45] "GET /talks/read.html HTTP/1.1" 404 311
127.0.0.1 - - [10/Apr/2007:10:54:08] "GET /img/pti-round.jpg HTTP/1.1" 200 17524
127.0.0.1 - - [10/Apr/2007:10:54:08] "GET /img/pti-round.jpg HTTP/1.1" 200 17524
127.0.0.1 - - [10/Apr/2007:10:54:21] "GET /unix_sysadmin.html HTTP/1.1" 200 3880
127.0.0.1 - - [10/Apr/2007:10:54:21] "GET /unix_sysadmin.html HTTP/1.1" 200 3880
217.0.22.3 - - [10/Apr/2007:10:54:51] "GET / HTTP/1.1" 200 34 "-"
```

Expected output:

```
127.0.0.1        12
139.12.0.2        2
217.0.22.3        7
```

Exercise: Combine lists again

See the same exercise in the previous chapter.

Exercise: counting DNA bases

Given a sequence like this: “ACTNGTGCTYGATRGTAGCYXGTN”, print out the distribution of the elemnts to get the following result:

```
A 3 - 12.50 %
C 3 - 12.50 %
G 6 - 25.00 %
N 2 -  8.33 %
R 1 -  4.17 %
T 6 - 25.00 %
X 1 -  4.17 %
Y 2 -  8.33 %
```

Exercise: Count Amino Acids

- Each sequence consists of many repetition of the 4 bases represented by the ACTG characters.
- There are 64 codons (sets of 3 bases following each other)
- There are 22 Amino Acids each of them are represented by 3 bases.
- Some of the Amino Acids can be represented in multiple ways. For example Histidine can be encoded by both CAU, CAC)
- We have a DNA sequence
- Count the Amino acids form the sequence. (For our purposes feel free to generate a DNA sequence with a random number generator.

Exercise: List of dictionaries

Given the following file build a list of dictionaries where each dictionary represents one person.
The keys in the dictionary are the names of the columns (fname, lname, born) the values are the respective values from each row.

```
fname,lname,born
Graham,Chapman,8 January 1941
Eric,Idle,29 March 1943
```

```
Terry,Gilliam,22 November 1940
Terry,Jones,1 February 1942
John,Cleese,27 October 1939
Michael,Palin,5 May 1943
```

```
print(people[1]['fname']) # Eric
```

Exercise: Dictinoary of dictionaries

Given the following file build a dictionary of dictionaries where each internal dictionary represents one person.
The keys in the internal dictionaries are the names of the columns (fname, lname, born) the values are the respective values from each row.
In the outer dictinary the keys are the (fname, lname) tuples.

```
fname,lname,born
Graham,Chapman,8 January 1941
Eric,Idle,29 March 1943
Terry,Gilliam,22 November 1940
Terry,Jones,1 February 1942
John,Cleese,27 October 1939
Michael,Palin,5 May 1943
```

```
print(people[('Eric', 'Idle')]['born']) # 29 March 1943
```

Solution: count characters

```
text = """
This is a very long text.
OK, maybe it is not that long after all.
"""

# print(text)
count = {}

for char in text:
```

```
    if char == '\n':
        continue
    if char not in count:
        count[char] = 1
    else:
        count[char] += 1

for key in sorted( count.keys() ):
    print("'{}' {}".format(key, count[key]))
```

- We need to store the counter somewhere. We could use two lists for that, but that would give a complex solution that runs in O(n**2) time.
- Besides, we are in the chapter about dictionaries so probably we better use a dictionary.
- In the `count` dictionary we each key is going to be one of the characters and the respective value will be the number of times it appeared.
- So if out string is “aabx” then we’ll end up with

```
{
    "a": 2,
    "b": 1,
    "x": 1,
}
```

- The `for in` loop on a string will iterate over it character by charter (even if we don’t call our variable `char`.
- We check if the current character is a newline `\n` and if it we call `continue` to skip the rest of the iteration. We don’t want to count newlines.
- Then we check if we have already seen this character. That is, it is already one of the keys in the `count` dictionary. If not yet, then we add it and put 1 as the values. After all we saw one copy of this character. If we have already seen this character

(we get to the `else` part) then we increment the counter for this character.

- We are done now with the data collection.

- In the second loop we go over the keys of the dictionary, that is the characters we have encountered. We sort them in ASCII order.
- Then we print each one of them and the respective value, the number of times the character was found.

Solution: count characters with default dict

```python
from collections import defaultdict

text = """
This is a very long text.
OK, maybe it is not that long after all.
"""

# print(text)
count = defaultdict(int)

for char in text:
    if char == '\n':
        continue
    count[char] += 1

for key in sorted( count.keys() ):
    print("'{}' {}".format(key, count[key]))
```

- The previous solution can be slightly improved by using `defaultdict` from the `collections` module.
- `count = defaultdict(int)` creates an empty dictionary that has the special feature that if you try to use a key that does not exists, it pretends that it exists and that it has a value 0.

- This allows us to remove the condition checking if the character was already seen and just increment the counter. The first time we encounter a charcter the dictionary will pretend that it was already there with value 0 so everying will work out nicely.

Solution: count words

```
words = ['Wombat', 'Rhino', 'Sloth', 'Tarantula', 'Sloth', 'Rhino', 'Sloth']

counter = {}
for word in words:
    if word not in counter:
        counter[word] = 0
    counter[word] += 1

for word in counter:
    print("{}:{}".format(word, counter[word]))
```

```
from collections import Counter

words = ['Wombat', 'Rhino', 'Sloth', 'Tarantula', 'Sloth', 'Rhino', 'Sloth']

cnt = Counter()
for word in words:
    cnt[word] += 1

print(cnt)
for w in cnt.keys():
    print("{}:{}".format(w, cnt[w]))
```

```
from collections import defaultdict

words = ['Wombat', 'Rhino', 'Sloth', 'Tarantula', 'Sloth', 'Rhino', 'Sloth']

dd = defaultdict(lambda : 0)
for word in words:
```

```
    dd[word] += 1

print(dd)
for word in dd.keys():
    print("{}:{}".format(word, dd[word]))
```

Solution: count words in file

```
import sys

filename = 'README'
if len(sys.argv) > 1:
    filename = sys.argv[1]
print(filename)

count = {}

with open(filename) as fh:
    for full_line in fh:
        line = full_line.rstrip('\n')
        line = line.lower()
        for word in line.split():
            if word == '':
                continue
            if word not in count:
                count[word] = 0

            count[word] += 1

for word in sorted(count):
    print("{:13} {:>2}".format(word, count[word]))
```

Solution: Apache log

```
filename = 'examples/apache_access.log'

count = {}

with open(filename) as fh:
    for line in fh:
        space = line.index(' ')
```

```
        ip = line[0:space]
        if ip in count:
            count[ip] += 1
        else:
            count[ip] = 1

for ip in count:
    print("{:16} {:>3}".format(ip, count[ip]))
```

Solution: Combine lists again

```
c = {}
with open('examples/files/a.txt') as fh:
    for line in fh:
        k, v = line.rstrip("\n").split("=")
        if k in c:
            c[k] += int(v)
        else:
            c[k] = int(v)

with open('examples/files/b.txt') as fh:
    for line in fh:
        k, v = line.rstrip("\n").split("=")
        if k in c:
            c[k] += int(v)
        else:
            c[k] = int(v)


with open('out.txt', 'w') as fh:
    for k in sorted(c.keys()):
        fh.write("{}={}\n".format(k, c[k]))
```

Solution: counting DNA bases

```
seq = "ACTNGTGCTYGATRGTAGCYXGTN"
count = {}
for c in seq:
   if c not in count:
       count[c] = 0
   count[c] += 1
```

```
for c in sorted(count.keys()):
    print("{} {} - {:>5.2f} %".format(c, count[c], 100 *
count[c]/len(seq)))

# >5 is the right alignment of 5 places
# .2f is the floating point with 2 digits after the
floating point
```

Solution: Count Amino Acids

Generate random DNA sequence

```
import sys
import random

if len(sys.argv) != 2:
    exit("Need a number")
count = int(sys.argv[1])

dna = []
for _ in range(count):
    dna.append(random.choice(['A', 'C', 'T', 'G']))
print(''.join(dna))
```

```
dna = 'CACCCATGAGATGTCTTAACGCTGCTTTCATTATAGCCG'

aa_by_codon = {
    'ACG' : '?',
    'CAC' : 'Histidin',
    'CAU' : 'Histidin',
    'CCA' : 'Proline',
    'CCG' : 'Proline',
    'GAT' : '?',
    'GTC' : '?',
    'TGA' : '?',
    'TTA' : '?',
    'CTG' : '?',
    'CTT' : '?',
    'TCA' : '?',
    'TAG' : '?',
    #...
```

```
}

count = {}

for i in range(0, len(dna)-2, 3):
    codon = dna[i:i+3]
    #print(codon)
    aa = aa_by_codon[codon]
    if aa not in count:
        count[aa] = 0
    count[aa] += 1

for aa in sorted(count.keys()):
    print("{}  {}".format(aa, count[aa]))
```

Loop over dictionary keys

Looping over the “dictionary” is just like looping over the keys.

```
user = {
    'fname': 'Foo',
    'lname': 'Bar',
}

for k in user:
    print("{} -> {}".format(k, user[k]))

# lname -> Bar
# fname -> Foo
```

Do not change dictionary in loop

```
user = {
    'fname': 'Foo',
    'lname': 'Bar',
}

for k in user.keys():
    user['email'] = 'foo@bar.com'
    print(k)

```

```
print('-----')

for k in user:
    user['birthdate'] = '1991'
    print(k)

# lname
# fname
# -----
# lname
# Traceback (most recent call last):
#   File "examples/dictionary/change_in_loop.py", line 13, in <module>
#     for k in user:
# RuntimeError: dictionary changed size during iteration
```

Default Dict

```
counter = {}

word = 'eggplant'

counter[word] += 1
# counter[word] = counter[word] + 1
```

```
Traceback (most recent call last):
  File "counter.py", line 5, in <module>
    counter[word] += 1
KeyError: 'eggplant'
```

```
counter = {}

word = 'eggplant'

if word not in counter:
    counter[word] = 0
counter[word] += 1

print(counter)
```

```
{'eggplant': 1}
```

```python
from collections import defaultdict

counter = defaultdict(int)

word = 'eggplant'

counter[word] += 1

print(counter)
```

```
defaultdict(<class 'int'>, {'eggplant': 1})
```

Sets

sets

- Sets in Python are used when we are primarily interested in operations that we know from the sets theory.
- See also the Venn diagrams.
- In day to day speach we often use the word “group” instead of “set” even though they are not the same.
- What are the common elements of two set (two groups).
- Is one group (set) the subset of the other?
- What are all the elements that exist in both groups (sets)?
- What are the elements that exist in exactly one of the groups (sets)?

set operations

- set
- issubset
- intersection
- symmetric difference
- union
- relative complement

- stdtypes: set

set intersection

```
english = set(['door', 'car', 'lunar', 'era'])
spanish = set(['era', 'lunar', 'hola'])

print('english: ', english)
print('spanish: ', spanish)

both = english.intersection(spanish)
print(both)
```

- `intersection` returns the elements that are in both sets.

```
english:  {'car', 'lunar', 'era', 'door'}
spanish:  {'lunar', 'era', 'hola'}
{'lunar', 'era'}
```

set subset

```
english = set(['door', 'car', 'lunar', 'era'])
spanish = set(['era', 'lunar', 'hola'])

words = set(['door', 'lunar'])


print('issubset: ', words.issubset( english ))
print('issubset: ', words.issubset( spanish ))
```

- `intersection` returns the elements that are in both sets.

```
issubset:  True
issubset:  False
```

set symmetric difference

```
english = set(['door', 'car', 'lunar', 'era'])
spanish = set(['era', 'lunar', 'hola'])

diff = english.symmetric_difference(spanish)
print('symmetric_difference: ', diff)
```

- Symmetric diffeerence is all the elements in either one of the sets, but not in both. “the ears of the elephant”.

```
symmetric_difference:  {'door', 'hola', 'car'}
```

set union

```
english = set(['door', 'car', 'lunar', 'era'])
spanish = set(['era', 'lunar', 'hola'])

all_the_words = english.union(spanish)

print(english)
print(spanish)
print(all_the_words)
```

```
{'era', 'door', 'lunar', 'car'}
{'era', 'hola', 'lunar'}
{'era', 'door', 'car', 'hola', 'lunar'}
```

set relative complement

```
english = set(['door', 'car', 'lunar', 'era'])
spanish = set(['era', 'lunar', 'hola'])


eng = english - spanish
spa = spanish - english

print(spa)
print(eng)

print(english)
print(spanish)
```

```
{'hola'}
{'door', 'car'}
```

```
{'door', 'era', 'car', 'lunar'}
{'hola', 'era', 'lunar'}
```

set examples

```
things = set(['table', 'chair', 'door', 'chair',
'chair'])
print(things)
print(things.__class__)
print(things.__class__.__name__)

if 'table' in things:
   print("has table")
```

```
{'door', 'chair', 'table'}
<class 'set'>
set
has table
```

defining an empty set

```
objects = set()
print(objects)
```

```
set()
```

```
set([])
```

Adding an element to a set (add)

```
objects = set()
print(objects)

objects.add('Mars')
print(objects)

objects.add('Mars')
```

```
print(objects)

objects.add('Neptun')
print(objects)
```

```
set()
{'Mars'}
{'Mars'}
{'Neptun', 'Mars'}
```

In Python 2:

```
set([])
set(['Mars'])
set(['Mars'])
set(['Neptun', 'Mars'])
```

Merging one set into another set (update)

```
set(['Neptun', 'Mars'])


objects  = set(['Mars', 'Jupiter', 'Saturn'])
internal = set(['Mercury', 'Venus', 'Earth', 'Mars'])

objects.update(internal)
print(objects)
print(internal)
```

```
{'Mars', 'Jupiter', 'Earth', 'Mercury', 'Saturn',
'Venus'}
{'Mars', 'Earth', 'Mercury', 'Venus'}
```

Functions (subroutines)

Defining simple function

```
def add(x, y):
    z = x + y
    return z

a = add(2, 3)
print(a)    # 5

q = add(23, 19)
print(q)   # 42
```

The function definition starts with the word “dev” followed by the name of the function (“add” in our example), followed by the list of parameters
in a pair of parentheses, followed by a colon “:”. Then the body of the function is indented to the right. The depth of indentation does not matter
but it must be the same for all the lines of the function. When we stop the indentation and start a new expression on the first column, that’s what tells
Python that the function defintion has ended.

Defining a function

```
def sendmail(From, To, Subject, Content):
    print('From:', From)
    print('To:', To)
    print('Subject:', Subject)
```

```
    print('')
    print(Content)

sendmail('gabor@szabgab.com',
    'szabgab@gmail.com',
    'self message',
    'Has some content too')
```

Positional parameters.

Parameters can be named

```
def sendmail(From, To, Subject, Content):
    print('From:', From)
    print('To:', To)
    print('Subject:', Subject)
    print('')
    print(Content)

sendmail(
    Subject = 'self message',
    Content = 'Has some content too',
    From = 'gabor@szabgab.com',
    To = 'szabgab@gmail.com',
)
```

The parameters of every function can be passed either as positional parameters or as named parameters.

Mixing positional and named parameters

```
def sendmail(From, To, Subject, Content):
    print('From:', From)
    print('To:', To)
    print('Subject:', Subject)
    print('')
    print(Content)

sendmail(
    Subject = 'self message',
    Content = 'Has some content too',
    To = 'szabgab@gmail.com',
    'gabor@szabgab.com',
)
```

```
def sendmail(From, To, Subject, Content):
    print('From:', From)
    print('To:', To)
    print('Subject:', Subject)
    print('')
    print(Content)

sendmail(
    'gabor@szabgab.com',
    Subject = 'self message',
    Content = 'Has some content too',
    To = 'szabgab@gmail.com',
)
```

```
  File "examples/functions/named_and_positional_params.py", line 14
    'gabor@szabgab.com',
    ^
SyntaxError: positional argument follows keyword argument
```

Default values

```
def prompt(question, retry=3):
    while retry > 0:
        inp = input('{} ({}): '.format(question, retry))
        if inp == 'my secret':
```

```
            return True
        retry -= 1
    return False

print(prompt("Type in your password"))

print(prompt("Type in your secret", 1))
```

Function parameters can have default values. In such case the parameters are optional.
In the function declaration, the parameters with the default values must come last.
In the call, the order among these arguments does not matter, and they are optional anyway.

Several defaults, using names

Parameters with defaults must come at the end of the parameter declaration.

```
def f(a, b=2, c=3):
    print(a, b , c)

f(1)                 # 1 2 3
f(1, b=0)            # 1 0 3
f(1, c=0)            # 1 2 0
f(1, c=0, b=5)       # 1 5 0

# f(b=0, 1)
# would generate:
# SyntaxError: non-keyword arg after keyword arg

f(b=0, a=1)          # 1 0 3
```

There can be several parameters with default values.
They are all optional and can be given in any order after the positional arguments.

Arbitrary number of arguments *

The values arrive as `tuple`.

```
def mysum(*numbers):
    print(numbers)
    total = 0
    for s in numbers:
        total += s
    return total

print(mysum(1))
print(mysum(1, 2))
print(mysum(1, 1, 1))

x = [2, 3, 5, 6]
print(mysum(*x))
```

```
(1,)
1
(1, 2)
3
(1, 1, 1)
3
```

Fixed parmeters before the others

The `*numbers` argument can be preceded by any number of regular arguments

```
def mysum(op, *numbers):
    print(numbers)
    if op == '+':
        total = 0
    elif op == '*':
        total = 1
    else:
        raise Exception('invalid operator {}'.format(op))

    for s in numbers:
        if op == '+':
            total += s
        elif op == '*':
            total *= s

    return total

print(mysum('+', 1))
print(mysum('+', 1, 2))
print(mysum('+', 1, 1, 1))
print(mysum('*', 1, 1, 1))
```

```
(1,)
1
(1, 2)
3
(1, 1, 1)
3
(1, 1, 1)
1
```

Arbitrary key-value pairs in parameters `**`

```
def f(**kw):
    print(kw)
```

```

f(a = 23, b = 12)
```

```
{'a': 23, 'b': 12}
```

Extra key-value pairs in parameters

```
def f(name, **kw):
    print(name)
    print(kw)

f(name="Foo", a = 23, b = 12)

# Foo
# {'a': 23, 'b': 12}
```

```
Foo
{'a': 23, 'b': 12}
```

Every parameter option

```
def f(op, count = 0, *things, **kw):
    print(op)
    print(count)
    print(things)
    print(kw)

f(2, 3, 4, 5, a = 23, b = 12)

# 2
# 3
# (4, 5)
# {'a': 23, 'b': 12}
```

Duplicate declaration of functions (multiple signatures)

```
def add(x, y):
    return x*y

print(add(2, 3))  # 6

def add(x):
    return x+x

# add(2, 3)
# TypeError: add() takes exactly 1 argument (2 given)

print(add(2))  # 4
```

The second declaration silently overrides the first declaration.

- pylint can find such problems, along with a bunch of others.

Recursive factorial

```
n! = n * (n-1) ... * 1

0! = 1
n! = n * (n-1)!

f(0) = 1
f(n) = n * f(n-1)
```

```
def f(n):
    if n == 0:
       return 1
    return n * f(n-1)

print(f(1))   # 1
print(f(2))   # 2
print(f(3))   # 6
print(f(4))   # 24
```

Recursive Fibonacci

```
fib(1) = 1
fib(2) = 1
fib(n) = fib(n-1) + fib(n-2)
```

```
def fib(n):
    if n == 1:
        return 1
    if n == 2:
        return 1
    return fib(n-1) + fib(n-2)

print(3, fib(3))     # 2
print(30, fib(30))   # 832040
```

Python also supports recursive functions.

Non-recursive Fibonacci

```
def fib(n):
    if n == 1:
        return [1]
    if n == 2:
        return [1, 1]
    fibs = [1, 1]
    for i in range(2, n):
        fibs.append(fibs[-1] + fibs[-2])
    return fibs

print(fib(1))  # [1]
print(fib(2))  # [1, 1]
print(fib(3))  # [1, 1, 2]
print(fib(10)) # [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Unbound recursion

- In order to protect us from unlimited recursion, Python limits the depth of recursion:

```
def recursion(n):
    print(f"In recursion {n}")
    recursion(n+1)

recursion(1)
```

```
...
In recursion 995
In recursion 996
Traceback (most recent call last):
  File "recursion.py", line 7, in <module>
    recursion(1)
  File "recursion.py", line 5, in recursion
    recursion(n+1)
  File "recursion.py", line 5, in recursion
    recursion(n+1)
  File "recursion.py", line 5, in recursion
    recursion(n+1)
  [Previous line repeated 992 more times]
  File "recursion.py", line 4, in recursion
    print(f"In recursion {n}")
RecursionError: maximum recursion depth exceeded while calling a Python object
```

Variable assignment and change - Immutable

Details showed on the next slide

```
a = 42      # number or string
b = a       # This is a copy
print(a)    # 42
print(b)    # 42
a = 1
print(a)    # 1
print(b)    # 42

a = (1, 2)    # tuple
b = a         # this is a copy
```

```
print(a)     # (1, 2)
print(b)     # (1, 2)
# a[0] = 42  TypeError: 'tuple' object does not support item assignment
a = (3, 4, 5)
print(a)     # (3, 4, 5)
print(b)     # (1, 2)
```

Variable assignment and change - Mutable

```
a = [5, 6]
b = a        # this is a copy of the *reference* only
             # if we change the list in a, it will
             # change the list connected to b as well
print(a)     # [5, 6]
print(b)     # [5, 6]
a[0] = 1
print(a)     # [1, 6]
print(b)     # [1, 6]


a = {'name' : 'Foo'}
b = a        # this is a copy of the *reference* only
             # if we change the dictionary in a, it will
             # change the dictionary connected to b as well
print(a)     # {'name' : 'Foo'}
print(b)     # {'name' : 'Foo'}
a['name'] = 'Jar Jar'
print(a)     # {'name' : 'Jar Jar'}
print(b)     # {'name' : 'Jar Jar'}
```

Parameter passing of functions

```
x = 3

def inc(n):
    n += 1
    return n

print(x)        # 3
```

```
print(inc(x))    # 4
print(x)         # 3
```

Passing references

```
numbers = [1, 2, 3]

def update(x):
    x[0] = 23

def change(y):
    y = [5, 6]
    return y

print(numbers)          # [1, 2, 3]

update(numbers)
print(numbers)          # [23, 2, 3]

print(change(numbers)) # [5, 6]
print(numbers)          # [23, 2, 3]
```

Function documentation

```
def f(name):
    """
    The documentation
    should have more than one lines.
    """
    print(name)


f("hello")
print(f.__doc__)
```

Immediately after the definition of the function, you can add a string - it can be a “”” string to spread multiple lines - that will include the documentation of the function. This string

can be accessed via the **doc** (2+2 underscores) attribute
of the function. Also, if you ‘import’ the file - as a module - in the interactive prompt of Python, you will be
able to read this documentation via the **help()** function.
help(mydocs) or **help(mydocs.f)**
in the above case.

Sum ARGV

```python
import sys

def mysum(*numbers):
    print(numbers)
    total = 0
    for s in numbers:
        total += s
    return total

v = [int(x) for x in sys.argv[1:] ]
r = mysum( *v )
print(r)
```

Copy-paste code

```python
a = [2, 3, 93, 18]
b = [27, 81, 11, 35]
c = [32, 105, 1]

total_a  = 0
for v in a:
    total_a += v
print("sum of a: {} average of a: {}".format(total_a, total_a / len(a)))

total_b  = 0
for v in b:
    total_b += v
print("sum of b: {} average of b: {}".format(total_b,
```

```
total_b / len(b)))

total_c  = 0
for v in c:
    total_c += v
print("sum of c: {} average of c: {}".format(total_c,
total_c / len(a)))
```

```
sum of a: 116 average of a: 29.0
sum of b: 154 average of b: 38.5
sum of c: 138 average of c: 34.5
```

Did you notice the bug?

Copy-paste code fixed

```
a = [2, 3, 93, 18]
b = [27, 81, 11, 35]
c = [32, 105, 1]

def calc(numbers):
    total  = 0
    for v in numbers:
        total += v
    return total, total / len(numbers)

total_a, avg_a = calc(a)
print("sum of a: {} average of a: {}".format(total_a,
avg_a))

total_b, avg_b = calc(b)
print("sum of b: {} average of b: {}".format(total_b,
avg_b))


total_c, avg_c = calc(c)
print("sum of c: {} average of c: {}".format(total_c,
avg_c))
```

```
sum of a: 116 average of a: 29.0
sum of b: 154 average of b: 38.5
sum of c: 138 average of c: 46.0
```

Copy-paste code further improvement

```python
data = {
    'a': [2, 3, 93, 18],
    'b': [27, 81, 11, 35],
    'c': [32, 105, 1],
}

def calc(numbers):
    total  = 0
    for v in numbers:
        total += v
    return total, total / len(numbers)

total = {}
avg   = {}
for name, numbers in data.items():
   total[name], avg[name] = calc(numbers)
   print("sum of {}: {} average of {}: {}".format(name, total[name], name, avg[name]\
))
```

Palindrome

An iterative and a recursive solution

```python
def is_palindrome(s):
    if s == '':
        return True
    if s[0] == s[-1]:
        return is_palindrome(s[1:-1])
    return False

def iter_palindrome(s):
    for i in range(0, int(len(s) / 2)):
        if s[i] != s[-(i+1)]:
            return False
```

```
    return True

print(is_palindrome(''))       # True
print(is_palindrome('a'))      # True
print(is_palindrome('ab'))     # False
print(is_palindrome('aa'))     # True
print(is_palindrome('aba'))    # True
print(is_palindrome('abc'))    # False

print()
print(iter_palindrome(''))       # True
print(iter_palindrome('a'))      # True
print(iter_palindrome('ab'))     # False
print(iter_palindrome('aa'))     # True
print(iter_palindrome('aba'))    # True
print(iter_palindrome('abc'))    # False
```

Exercise: statistics

Write a function that will accept any number of numbers and return a list of values:

- The sum
- Average
- Minimum
- Maximum

Exercise: recursive

Give a bunch of files that has list of requirement in them. Process them recursively and print the resulting full list of requirements

```
b
c
d
```

```
e
d
```

```
f
g
```

```
$ python traversing_dependency_tree.py a

Processing a
Processing b
Processing e
Processing d
Processing c
Processing f
Processing g
Processing d
```

Exercise: Tower of Hanoi

Tower of Hanoi

Exercise: Merge and Bubble sort

- Implement bubble sort
- Implement merge sort

Solution: statistics

```
def stats(*numbers):
    total = 0

    average = None  # there might be better solutions
here!
    minx = None
    maxx = None

    for val in numbers:
```

```
        total += val
        if minx == None:
            minx = maxx = val
        if minx > val:
            minx = val
        if maxx < val:
            maxx = val

    if len(numbers):
        average = total / len(numbers)


    return total, average, minx, maxx


ttl, avr, smallest, largest = stats(3, 5, 4)

print(ttl)
print(avr)
print(smallest)
print(largest)
```

Solution: recursive

```python
import sys
import os

if len(sys.argv) < 2:
    exit("Usage: {} NAME".format(sys.argv[0]))

start = sys.argv[1]

def get_dependencies(name):
    print("Processing {}".format(name))

    deps = set(name)
    filename = name + ".txt"
    if not os.path.exists(filename):
        return deps

    with open(filename) as fh:
        for line in fh:
            row = line.rstrip("\n")
```

```
            deps.add(row)
            deps.update( get_dependencies(row) )

    return deps

dependencies = get_dependencies(start)
print(dependencies)
```

Solution: Tower of Hanoi

```
def check():
    for loc in hanoi.keys():
        if hanoi[loc] != sorted(hanoi[loc], reverse=True):
            raise Exception(f"Incorrect order in {loc}: {hanoi[loc]}")

def move(depth, source, target, helper):
    if depth > 0:
        move(depth-1, source, helper, target)

        val = hanoi[source].pop()
        hanoi[target].append(val)
        print(f"Move {val} from {source} to {target} Status A:{str(hanoi['A']):10}\
  B:{str(hanoi['B']):10}  C:{str(hanoi['C']):10}")
        check()

        move(depth-1, helper, target, source)
    check()

hanoi = {
    'A': [4, 3, 2, 1],
    'B': [],
    'C': [],
}

check()
move(len(hanoi['A']), 'A', 'C', 'B')
check()
```

Solution: Merge and Bubble sort

```
def recursive_bubble_sort(data):
    data = data[:]
    if len(data) == 1:
        return data

    last = data.pop()
    sorted_data = recursive_bubble_sort(data)
    for i in range(len(sorted_data)):
        if last > sorted_data[i]:
            sorted_data.insert(i, last)
            break
    else:
        sorted_data.append(last)
    return sorted_data

def iterative_bubble_sort(data):
    data = data[:]
    for end in (range(len(data)-1, 0, -1)):
        for i in range(end):
            if data[i] < data[i+1]:
                data[i], data[i+1] = data[i+1], data[i]
    return data


old = [1, 5, 2, 4, 8]
new1 = recursive_bubble_sort(old)
new2 = iterative_bubble_sort(old)
print(old)
print(new1)
print(new2)
```

Modules

Before modules

```
def add(a, b):
    return a + b


z = add(2, 3)
print(z)       # 5
```

Create modules

A module is just a Python file with a set of functions that us usually not used by itself. For example the “my_calculator.py”.

```
def add(a, b):
    return a + b
```

A user made module is loaded exactly the same way as the built-in module.
The functions defined in the module are used as if they were methods.

```
import my_calculator

z = my_calculator.add(2, 3)
```

```

print(z)   # 5
```

We can import specific functions to the current name space (symbol table) and then we don't need to prefix it with the name of
the file every time we use it. This might be shorter writing, but if we import the same function name from two different modules then they will overwrite each other. So I usually prefer loading the module as in the previous example.

```
from my_calculator import add

print(add(2, 3))   # 5
```

path to load modules from - The module search path

1. The directory where the main script is located.
2. The directories listed in PYTHONPATH environment variable.
3. Directories of standard libraries.
4. Directories listed in .pth files.
5. The site-packages home of third-party extensions.

sys.path - the module search path

```
import sys

print(sys.path)
```

```
['/Users/gabor/work/training/python/examples/package',
 '/Users/gabor/python/lib/python2.7/site-packages/crypto-
1.1.0-py2.7.egg',
 ...
 '/Library/Python/2.7/site-packages',
'/usr/local/lib/python2.7/site-packages']
[Finished in 0.112s]
```

Flat project directory structure

> If our executable scripts and our modules are all in the same directory then we don't have to worry ad the directory of the script is included in the list of places
> where "import" is looking for the files to be imported.

```
project/
    script_a.py
    script_b.py
    my_module.py
```

Absolute path

> If we would like to load a module that is not installed in one of the standard locations, but we know where it is located on our disk,
> we can set the "sys.path" to the absolute path to this directory.
> This works on the specific computer, but if you'd like to distribute
> the script to other computers you'll have to make sure the module to be loaded is installed in the same location or you'll have to update the script to point to the location of the module in each computer. Not an ideal solution.

```python
import sys
sys.path.insert(0, "/home/foobar/python/libs")

# import module_name
```

Relative path

```
../project_root/
    bin/relative_path.py
    lib/my_module.py
```

> We can use a directory structure that is more complex than the flat structure we had earlier. In this case the location of the modules relatively to the scripts
>
> is fixed. In this case it is "../lib". We can compute the relative path in each of our scripts. That will ensure we pick up the right module every time we run the script.
> Regardless of the location of the whole project tree.

```python
print("Importing my_module")
```

```python
import os, sys

# import my_module   # ImportError: No module named my_module

print(__file__)   # examples/sys/bin/relative_path.py
project_root = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

mypath = os.path.join(project_root, 'lib')
print(mypath) # /Users/gabor/work/training/python/examples/sys/../lib
sys.path.insert(0, mypath)
```

```python

import my_module    # Importing my_module
```

Python modules are compiled

When libraries are loaded they are automatically compiled to `.pyc` files.
This provides moderate code-hiding and load-time speed-up. Not run-time speed-up.
Starting from Python 3.2 the pyc files are saved in the `__pycache__` directory.

How "import" and "from" work?

1. Find the file to load.
2. Compile to bytecode if necessary and save the bytecode if possible.
3. Run the code of the file loaded.
4. Copy names from the imported module to the importing namespace.

Runtime loading of modules

```python
def hello():
    print("Hello World")

print("Loading mygreet")
```

```python
print("Start running")   # Start running

import mygreet           # Loading mygreet

mygreet.hello()          # Hello World

print("DONE")            # DONE
```

Conditional loading of modules

```python
import random

print("Start running")
name = input("Your name:")

if name == "Foo":
    import mygreet
    mygreet.hello()
else:
    print('No loading')


print("DONE")
```

Duplicate importing of functions

```python
from mycalc import add
print(add(2, 3))   # 5

from mymath import add
print(add(2, 3))   # 6


from mycalc import add
print(add(2, 3))   # 5
```

The second declaration silently overrides the first declaration.

pylint can find such problems, along with a bunch of others.

Script or library

We can have a file with all the functions implemented and then launch the run() function only if the file was executed as a stand-alone script.

```
def run():
    print("run in ", __name__)

print("Name space in mymodule.py ", __name__)

if __name__ == '__main__':
    run()
```

```
$ python mymodule.py
Name space in mymodule.py  __main__
run in  __main__
```

Script or library - import

If it is imported by another module then it won't run automatically. We have to call it manually.

```
import mymodule

print("Name space in import_mymodule.py ", __name__)
mymodule.run()
```

```
$ python import_mymodule.py
Name space in mymodule.py  mymodule
Name space in import_mymodule.py  __main__
run in  mymodule
```

Script or library - from import

```
from mymodule import run

print("Name space in import_mymodule.py ", __name__)
run()
```

```
$ python import_from_mymodule.py
Name space in mymodule.py  mymodule
Name space in import_mymodule.py  __main__
run in  mymodule
```

assert to verify values

```
def add(x, y):
    return x * y

for x, y, z in [(2, 2, 4), (9, 2, 11), (2, 3, 5)]:
    print(f"add({x}, {y}) == {z}")
    if add(x, y) != z:
        raise Exception(f"add({x}, {y}) != {z}")
        #raise AssertionError
```

```
add(2, 2) == 4
add(9, 2) == 11
Traceback (most recent call last):
  File "examples/functions/raise_exception.py", line 7, in <module>
    raise Exception(f"add({x}, {y}) != {z}")
Exception: add(9, 2) != 11
```

```
def add(x, y):
    return x * y

for x, y, z in [(2, 2, 4), (9, 2, 11), (2, 3, 5)]:
    print(f"add({x}, {y}) == {z}")
    assert add(x, y) == z
```

```
add(2, 2) == 4
add(9, 2) == 11
Traceback (most recent call last):
  File "examples/functions/assert.py", line 6, in <module>
    assert add(x, y) == z
AssertionError
```

mycalc as a self testing module

```python
import mycalc
print(mycalc.add(19, 23))
```

```
$ python use_mycalc.py
42
```

```python
def test_add():
    print('Testing  {}'.format(__file__))
    assert add(1, 1) == 2
    assert add(-1, 1) == 0
    # assert add(-99, 1) == 0 # AssertionError

def add(a, b):
    return a + b

if __name__ == '__main__':
    test_add()
```

```
$ python mycalc.py
Self testing  mycalc.py
```

doctest

```python
def fib(n):
    '''
    Before the tests
    >>> fib(3)
    2
```

```
    >>> fib(10)
    55
    >>> [fib(n) for n in range(11)]
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

    >>> fib(11)
    89
    After the tests
    '''
    values = [0, 1]

    if n == 11:
        return 'bug'

    while( n > len(values) -1 ):
        values.append(values[-1] + values[-2])
    return values[n]

#if __name__ == "__main__":
#     import doctest
#     doctest.testmod()
```

```
python -m doctest fibonacci_doctest.py
```

```
python examples/functions/fibonacci_doctest.py


**********************************************************************
File ".../examples/functions/fibonacci_doctest.py", line 12, in __main__.fib
Failed example:
    fib(11)
Expected:
    89
Got:
    'bug'
**********************************************************************
1 items had failures:
   1 of   4 in __main__.fib
***Test Failed*** 1 failures.
```

doctest

Scope of import

```
def div(a, b):
    return a/b
```

```
from __future__ import print_function
from __future__ import division

import mydiv

print(mydiv.div(3, 2))    # 1

print(3/2)                # 1.5
```

The importing of functions, and the changes in the behavior of the compiler are file specific.
In this case the change in the behavior of division is only visible in the division.py script, but not in the mydiv.py module.

Export import

- from mod import a,b,_c - import ‘a’, ‘b’, and ‘_c’ from ‘mod’
- from mod import * - import every name listed in **all** of ‘mod’ if **all** is available.
- from mod import * - import every name that does NOT start with _ (if **all** is not available)
- import mod - import ‘mod’ and make every name in ‘mod’ accessible as ‘mod.a’, and ‘mod._c’

```python
def a():
    return "in a"

b = "value of b"

def _c():
    return "in _c"

def d():
    return "in d"
```

```python
from my_module import a,b,_c

print(a())      # in a
print(b)        # value of b
print(_c())     # in _c

print(d())
# Traceback (most recent call last):
#   File ".../examples/modules/x.py", line 7, in <module>
#     print(d())
# NameError: name 'd' is not defined
```

```python
from my_module import *

print(a())      # in a
print(b)        # value of b

print(d())      # in d


print(_c())

# Traceback (most recent call last):
#   File ".../examples/modules/y.py", line 9, in <module>
#     print(_c())    # in _c
# NameError: name '_c' is not defined
```

Export import with all

```
__all__ = ['a', '_c']

def a():
    return "in a"

b = "value of b"

def _c():
    return "in _c"

def d():
    return "in d"
```

```
from my_module2 import *

print(a())      # in a
print(_c())     # in _c

print(b)

# Traceback (most recent call last):
#   File ".../examples/modules/z.py", line 7, in <module>
#     print(b)        # value of b
# NameError: name 'b' is not defined
```

import module

```
import my_module

print(my_module.a())     # in a
print(my_module.b)       # value of b
print(my_module._c())    # in _c
print(my_module.d())     # in d
```

Execute at import time

```
import lib

print("Hello")
```

```python
print("import lib")

def do_something():
    print("do something")
```

```
import lib
Hello
```

Import multiple times

```python
import one
import two

print("Hello")
```

```python
import common
print("loading one")
```

```python
import common
print("loading two")
```

```python
print("import common")
```

```
import common
loading one
loading two
Hello
```

Exercise: Number guessing

Take the number guessing game from the earlier chapter and move the internal while() loop
to a function.

Once that's done, move the function out to a separate file and use it as a module.

Exercies: Scripts and modules

Take the number guessing game: if I run it as a script execute the whole game with repeated hidden numbers.
If I load it as a module, then let me call the function that runs a single game with one hidden number.
We should be able to even pass the hidden number as a parameter.

Exercise: Module my_sum

- Create a file called `my_simple_math.py` with two functions: `div(a, b)`, `add(a, b)`, that will divide and add the two numbers respectively.
- Add another two functions called `test_div` and `test_add` that will test the above two functions using assert.
- Add code that will run the tests if someone execute `python my_simple_math.py` running the file as if it was a script.
- Create another file called `use_my_simple_math.py` that will use the functions from `my_math` module to calculate 2 + 5 * 7
- Make sure when you run `python use_my_simple_math.py` the tests won't run.
- Add documentation to the "add" and "div" functions to examples that can be used with doctest.

- Can you run the tests when the file is loaded as a module?

Exercise: Convert your script to module

- Take one of your real script (from work). Create a backup copy.
- Change the script so it can be import-ed as a module and then it won't automatically execute anything, but that it still works when executed as a script.
- Add a new function to it called `self_test` and in that function add a few test-cases to your code using 'assert'.
- Write another script that will load your real file as a module and will run the `self_test`.
- Let me know what are the dificulties!

Exercise: Add doctests to your own code

- Pick a module from your own code and create a backup copy. (from work)
- Add a function called 'self_test' that uses 'assert' to test some of the real functions of the module.
- Add code that will run the 'self_test' when the file is executed as a script.
- Add documentation to one of the functions and convert the 'assert'-based tests to doctests.
- Convert the mechanism that executed the 'self_test' to run the doctests as well.
- Let me know what are the dificulties!

Solution: Module my_sum

```
def div(a, b):
    '''
    >>> div(8, 2)
    4
    '''
    return a/b
```

```python

def add(a, b):
    '''
    >>> add(2, 2)
    4
    '''
    return a * b    # bug added on purpose!

def test_div():
    assert div(6, 3) == 2
    assert div(0, 10) == 0
    assert div(-2, 2) == -1
    #assert div(10, 0) == ??

def test_add():
    assert add(2, 2) == 4
    #assert add(1, 1) == 2


if __name__ == "__main__":
    test_div()
    test_add()
```

```python
import my_simple_math
print(my_simple_math.my_sum(2, 3, 5))

print(dir(my_simple_math))
#my_sum_as_function.test_my_sum()
```

Regular Expressions

What are Regular Expressions (aka. Regexes)?

- An idea on how to match some pattern in some text.
- A tool/language that is available in many places.
- Has many different “dialects”
- Has many different modes of processing.
- The grand concept is the same.
- Uses the following symbols:

```
() [] {} . * + ? ^ $ | - \ \d \s \w \A \Z \1 \2 \3
```

What are Regular Expressions good for?

- Decide if a string is part of a larger string.
- Validate the format of some value (string) (e.g. is it a decimal number?, is it a hex?)
- Find if there are repetitions in a string.
- Analyze a string and fetch parts of if given some loose description.
- Cut up a string into parts.
- Change parts of a string.

Examples

```
Is the input given by the user a number?

```

```
(BTW which one is a number:  23, 2.3,  2.3.4, 2.4e3, abc ?)

Is there a word in the file that is repeated 3 or more times?

Replaces all occurrences of Python or python by Java ...
... but avoid replacing Monty Python.


Given a text message fetch all the phone numbers:
Fetch numbers that look like 09-1234567
then also fetch +972-2-1234567
and maybe also 09-123-4567


Check if in a given text passing your network there are credit card numbers....


Given a text find if the word "password" is in it and fetch the surrounding text.


Given a log file like this:

[Tue Jun 12 00:01:00 2019] - (3423) - INFO - ERROR log restarted
[Tue Jun 12 09:08:17 2019] - (3423) - INFO - System starts to work
[Tue Jun 13 08:07:16 2019] - (3423) - ERROR - Something is wrong

provide statistics on how many of the different levels of log messages
were seen. Separate the log messages into files.
```

Where can I use it ?

- grep, egrep
- Unix tools such as sed, awk, procmail
- vi, emacs, other editors

- text editors such as Multi-Edit
- .NET languages: C#, C++, VB.NET
- Java
- Perl
- **Python**
- PHP
- Ruby
- …
- Word, Open Office …
- PCRE

grep

grep gets a regex and one or more files. It goes over line-by-line all the files and displays the lines where the regex matched. A few examples:

```
grep python file.xml     # lines that have the string
python in them in file.xml.
grep [34] file.xml       # lines that have either 3 or 4
(or both) in file.xml.
grep [34] *.xml          # lines that have either 3 or 4
(or both) in every xml file.
grep [0-9] *.xml         # lines with a digit in them.
egrep '\b[0-9]' *.xml    # only highlight digits that are
at the beginning of a numbe\
r.
```

Regexes first match

```python
import re

text = 'The black cat climed'
```

```
match = re.search(r'lac', text)
if match:
    print("Matching")        # Matching
    print(match.group(0))    # lac

match = re.search(r'dog', text)
if match:
    print("Matching")
else:
    print("Did NOT match")
    print(match)      # None
```

The search method returns an object or **None**, if it could not find any match.
If there is a match you can call the **group()** method. Passing 0 to it will
return the actual substring that was matched.

Match numbers

```
import re

line = 'There is a phone number 12345 in this row and an age: 23'

match = re.search(r'\d+', line)
if match:
    print(match.group(0))   # 12345
```

Use raw strings for regular expression: r'a\d'. Especially because \ needs it.

- \d matches a digit.
- + is a quantifier and it tells \d to match one or more digits.

It matches the first occurrence.
Here we can see that the `group(0)` call is much more interesting than earlier.

Capture

```python
import re

line = 'There is a phone number 12345 in this row and an age: 23'

match = re.search(r'age: \d+', line)
if match:
  print(match.group(0))   # age: 23


match = re.search(r'age: (\d+)', line)
if match:
    print(match.group(0))   # age: 23
    print(match.group(1))   # 23      the first group of parentheses

    print(match.groups())   # ('23',)
    print(len(match.groups()))   # 1
```

Parentheses in the regular expression can enclose any sub-expression.
Whatever this sub-expression matches will be saved and can be accessed using the group() method.

Capture more

```python
import re

line = 'There is a phone number 12345 in this row and an
```

```
age: 23'

match = re.search(r'(\w+): (\d+)', line)
if match:
    print(match.group(0))   # age: 23
    print(match.group(1))   # age      the first group of parentheses
    print(match.group(2))   # 23       the second group of parentheses

    # print(match.group(3))  # IndexError: no such group
    print(match.groups())   # ('age', '23')
    print(len(match.groups()))   # 2
```

Some groups might match ‘’ or even not match at all, in which case we get None
in the appropriate match.group() call and in the match.groups() call

Capture even more

```
import re

line = 'There is a phone number 12345 in this row and an age: 23'

match = re.search(r'((\w+): (\d+))', line)
if match:
    print(match.group(0))   # age: 23
    print(match.group(1))   # age: 23
    print(match.group(2))   # age
    print(match.group(3))   # 23

    print(match.groups())   # ('age: 23', 'age', '23')
    print(len(match.groups()))   # 3
```

findall

```python
import re

line1 = 'There is a phone number 12345 in this row and another 42 number'
numbers1 = re.findall(r'\d+', line1)
print(numbers1) # ['12345', '42']

line2 = 'There are no numbers in this row. Not even one.'
numbers2 = re.findall(r'\d+', line2)
print(numbers2) # []
```

re.findall returns the matched substrings.

findall with capture

```python
import re

line = 'There is a phone number 12345 in this row and another 42 number'
match = re.search(r'\w+ \d+', line)
if match:
    print(match.group(0))    # number 12345

match = re.search(r'\w+ (\d+)', line)
if match:
    print(match.group(0))    # number 12345
    print(match.group(1))    # 12345

matches = re.findall(r'\w+ \d+', line)
print(matches)   # ['number 12345', 'another 42']

matches = re.findall(r'\w+ (\d+)', line)
print(matches)   # ['12345', '42']
```

findall with capture more than one

```python
import re

```

```python
line = 'There is a phone number 12345 in this row and another 42 number'
match = re.search(r'(\w+) (\d+)', line)
if match:
    print(match.group(1))    # number
    print(match.group(2))    # 12345

matches = re.findall(r'(\w+) (\d+)', line)
print(matches)   # [('number', '12345'), ('another', '42')]
```

If there are multiple capture groups then The returned list will consist of tuples.

Any Character

. matches any one character except newline.

For example: **#.#**

```python
import re

strings = [
    'abc',
    'text: #q#',
    'str: #a#',
    'text #b# more text',
    '#a  and this? #c#',
    '#a  and this? # c#',
    '#@#',
    '#.#',
    '# #',
    '##'
    '###'
]

for s in strings:
    print('str:  ', s)
```

```
    match = re.search(r'#.#', s)
    if match:
        print('match:', match.group(0))
```

If **re.DOTALL** is given newline will be also matched.

Match dot

```
import re

cases = [
    "hello!",
    "hello world.",
    "hello. world",
    ".",
]

for case in cases:
    print(case)
    match = re.search(r'.', case)    # Match any character
    if match:
        print(match.group(0))

print("----")

for case in cases:
    print(case)
    match = re.search(r'\.', case)   # Match a dot
    if match:
        print(match.group(0))

print("----")

for case in cases:
    print(case)
    match = re.search(r'[.]', case) # Match a dot
    if match:
        print(match.group(0))
```

Character classes

We would like to match any string that has any of the #a#, #b#, #c#, #d#, #e#, #f#, #@# or #.#

```
import re

strings = [
    'abc',
    'text: #q#',
    'str: #a#',
    'text #b# more text',
    '#ab#',
    '#@#',
    '#.#',
    '# #',
    '##'
    '###'
]


for s in strings:
    print('str:  ', s)
    match = re.search(r'#[abcdef@.]#', s)
    if match:
        print('match:', match.group(0))
```

```
r'#[abcdef@.]#'
r'#[a-f@.]#'
```

Common characer classes

- **\d** digit: [0-9] Use stand alone: \d or as part of a bigger character class: [abc\d]
- **\w** word character: [0-9a-zA-Z_]
- **\s** white space: [\f\t\n\r] form-feed, tab, newline, carriage return and SPACE

Negated character class

- **[^abc]** matches any one character that is not ‘a’, not ‘b’ and not ‘c’.
- D not digit [^\d]
- W not word character [^\w]
- S not white space [^\s]

Optional character

Match the word **color** or the word **colour**

```
Regex: r'colou?r'
```

```
Input: color
Input: colour
Input: colouur
```

Regex 0 or more quantifier

Any line with two - -es with anything in between.

```
Regex: r'-.*-'
Input: "ab"
Input: "ab - cde"
Input: "ab - qqqrq -"
Input: "ab -- cde"
Input: "--"
```

Quantifiers

Quantifiers apply to the thing in front of them

```
r'ax*a'      # aa, axa, axxa, axxxa, ...
r'ax+a'      #     axa, axxa, axxxa, ...
r'ax?a'      # aa, axa
r'ax{2,4}a'  #          axxa, axxxa, axxxxa
r'ax{3,}a'   #                axxxa, axxxxa, ...
```

```
r'ax{17}a'    # axxxxxxxxxxxxxxxxxa
```

*	0-
+	1-
?	0-1
	n-m
	n-
	n

Quantifiers limit

```python
import re

strings = (
    "axxxa",
    "axxxxa",
    "axxxxxa",
)

for text in strings:
    match = re.search(r'ax{4}', text)
    if match:
        print("Match")
        print(match.group(0))
    else:
        print("NOT Match")
```

Quantifiers on character classes

```python
import re

strings = (
    "-a-",
    "-b-",
    "-x-",
    "-aa-",
    "-ab-",
```

```python
    "--",
)

for line in strings:
    match = re.search(r'-[abc]-', line)
    if match:
        print(line)
print('==========================')

for line in strings:
    match = re.search(r'-[abc]+-', line)
    if match:
        print(line)
print('==========================')

for line in strings:
    match = re.search(r'-[abc]*-', line)
    if match:
        print(line)
```

Greedy quantifiers

```python
import re

match = re.search(r'xa*', 'xaaab')
print(match.group(0))

match = re.search(r'xa*', 'xabxaab')
print(match.group(0))

match = re.search(r'a*',  'xabxaab')
print(match.group(0))

match = re.search(r'a*',  'aaaxabxaab')
print(match.group(0))
```

They match ‘xaaa’, ‘xa’ and ‘’ respectively.

Minimal quantifiers

```
import re

match = re.search(r'a.*b', 'axbzb')
print(match.group(0))

match = re.search(r'a.*?b', 'axbzb')
print(match.group(0))


match = re.search(r'a.*b', 'axy121413413bq')
print(match.group(0))

match = re.search(r'a.*?b', 'axyb121413413q')
print(match.group(0))
```

Anchors

- A matches the beginning of the string
- Z matches the end of the string
- ^ matches the beginning of the row (see also re.MULTILINE)
- $ matches the end of the row but will accept a trailing newline (see also re.MULTILINE)

```
import re

lines = [
    "text with cat in the middle",
    "cat with dog",
    "dog with cat",
]

for line in lines:
    if re.search(r'cat', line):
        print(line)


print("---")
for line in lines:
```

```python
    if re.search(r'^cat', line):
        print(line)

print("---")
for line in lines:
    if re.search(r'\Acat', line):
        print(line)

print("---")
for line in lines:
    if re.search(r'cat$', line):
        print(line)

print("---")
for line in lines:
    if re.search(r'cat\Z', line):
        print(line)
```

```
text with cat in the middle
cat with dog
dog with cat
---
cat with dog
---
cat with dog
---
dog with cat
---
dog with cat
```

Anchors on both end

```python
import re

strings = [
    "123",
    "hello 456 world",
    "hello world",
]

for line in strings:
    if re.search(r'\d+', line):
```

```python
        print(line)

print('---')

for line in strings:
    if re.search(r'^\d+$', line):
        print(line)


print('---')

for line in strings:
    if re.search(r'\A\d+\Z', line):
        print(line)
```

```
123
hello 456 world
---
123
---
123
```

Match ISBN numbers

```

```

```python
import re

strings = [
    '99921-58-10-7',
    '9971-5-0210-0',
    '960-425-059-0',
    '80-902734-1-6',
    '85-359-0277-5',
    '1-84356-028-3',
    '0-684-84328-5',
    '0-8044-2957-X',
    '0-85131-041-9',
    '0-943396-04-2',
    '0-9752298-0-X',

```

```
    '0-975229-1-X',
    '0-9752298-10-X',
    '0-9752298-0-Y',
    '910975229-0-X',
    '-------------',
    '0000000000000',
]
for isbn in strings:
    print(isbn)

    if (re.search(r'^[\dX-]{13}$', isbn)):
        print("match 1")

    if (re.search(r'^\d{1,5}-\d{1,7}-\d{1,5}-[\dX]$', isbn) and len(isbn) == 13):
        print("match 2")
```

Matching a section

```
import re

text = "This is <a string> with some sections <marked> with special characters"

m = re.search(r'<.*>', text)
if m:
    print(m.group(0))
```

Matching a section - minimal

```
import re

text = "This is <a string> with some sections <marked> with special characters"

m = re.search(r'<.*?>', text)
if m:
    print(m.group(0))
```

Matching a section negated character class

```
import re

text = "This is <a string> with some sections <marked> with special characters"

m = re.search(r'<[^>]*>', text)
if m:
    print(m.group(0))
```

DOTALL S (single line)

if re.DOTALL is given, . will match any character. Including newlines.

```
import re

line = 'Before <div>content</div> After'

text = '''
Before
<div>
content
</div>
After
'''

if (re.search(r'<div>.*</div>', line)):
    print('line');
if (re.search(r'<div>.*</div>', text)):
    print('text');

print('-' * 10)

if (re.search(r'<div>.*</div>', line, re.DOTALL)):
    print('line');
if (re.search(r'<div>.*</div>', text, re.DOTALL)):
    print('text');
```

MULTILINE M

if re.MULTILNE is given, ^ will match beginning of line and $ will match end of line

```
import re

line = 'Start   blabla End'

text = '''
prefix
Start
blabla
End
postfix
'''

regex = r'^Start[\d\D]*End$'
m = re.search(regex, line)
if (m):
    print('line')

m = re.search(regex, text)
if (m):
    print('text')

print('-' * 10)

m = re.search(regex, line, re.MULTILINE)
if (m):
    print('line')

m = re.search(regex, text, re.MULTILINE)
if (m):
    print('text')
```

```
line
----------
line
text
```

```
re.MULTILINE | re.DOTALL
```

Two regex with logical or

All the rows with either ‘apple pie’ or ‘banana pie’ in them.

```
import re

strings = [
    'apple pie',
    'banana pie',
    'apple'
]

for s in strings:
    #print(s)
    match1 = re.search(r'apple pie', s)
    match2 = re.search(r'banana pie', s)
    if match1 or match2:
        print('Matched in', s)
```

Alternatives

Alternatives

```
import re

strings = [
    'apple pie',
    'banana pie',
    'apple'
]

for s in strings:
    match = re.search(r'apple pie|banana pie', s)
    if match:
        print('Matched in', s)
```

Grouping and Alternatives

Move the common part in one place and limit the alternation to the part within the parentheses.

```python
import re

strings = [
    'apple pie',
    'banana pie',
    'apple'
]

for s in strings:
    match = re.search(r'(apple|banana) pie', s)
    if match:
        print('Matched in', s)
```

Internal variables

```python
import re

strings = [
    'banana',
    'apple',
    'infinite loop',
]

for s in strings:
    match = re.search(r'(.)\1', s)
    if match:
        print(match.group(0), 'matched in', s)
        print(match.group(1))
```

More internal variables

```
(.)(.)\2\1

(\d\d).*\1

(\d\d).*\1.*\1
```

```

(.{5}).*\1
```

Regex DNA

- DNA is built from G, A, T, C
- Let’s create a random DNA sequence
- Then find the longest repeated sequence in it

```python
import re
import random

chars = ['G', 'A', 'T', 'C']
dna = ''
for i in range(100):
    dna += random.choice(chars)

print(dna)

'''
Generating regexes:

    ([GATC]{1}).*\1
    ([GATC]{2}).*\1
    ([GATC]{3}).*\1
    ([GATC]{4}).*\1
'''
length = 1
result = ''
while True:
    regex = r'([GATC]{' + str(length) + r'}).*\1'
    #print(regex)
    m = re.search(regex, dna)
    if m:
        result = m.group(1)
        length += 1
    else:
        break

print(result)
print(len(result))
```

Regex IGNORECASE

```
import re

s = 'Python'

if (re.search('python', s)):
    print('python matched')

if (re.search('python', s, re.IGNORECASE)):
    print('python matched with IGNORECASE')
```

Regex VERBOSE X

```
import re

email = "foo@bar.com"

m = re.search(r'\w[\w.-]*\@([\w-]+\.)+(com|net|org|uk|hu|il)', email)
if (m):
    print('match')


m = re.search(r'''
                  \w[\w.-]*              # username
                  \@
                  ([\w-]+\.)+            # domain
                  (com|net|org|uk|hu|il)   # gTLD
                  ''', email, re.VERBOSE)
if (m):
    print('match')
```

Substitution

```
import re

line = "abc123def"

print(re.sub(r'\d+', ' ', line)) # "abc def"
print(line)                      # "abc123def"
```

```python
print(re.sub(r'x', ' y', line))   # "abc123def"
print(line)                       # "abc123def"

print(re.sub(r'([a-z]+)(\d+)([a-z]+)', r'\3\2\1', line))  #  "def123abc"
print(re.sub(r'''
([a-z]+)      # letters
(\d+)         # digits
([a-z]+)      # more letters
''', r'\3\2\1', line, flags=re.VERBOSE))   #  "def123abc"

print(re.sub(r'...', 'x', line))              # "xxx"
print(re.sub(r'...', 'x', line, count=1))     # "x123def"

print(re.sub(r'(.)(.)', r'\2\1', line))             # "ba1c32edf"
print(re.sub(r'(.)(.)', r'\2\1', line, count=2))    # "ba1c23def"
```

---

findall capture

If there are parentheses in the regex, it will return tuples of the matches

---

```python
import re

line = 'There is a phone number 83795 in this row and another 42 number'
print(line)

search = re.search(r'(\d)(\d)', line)
if search:
  print(search.group(1))    # 8
  print(search.group(2))    # 3

matches = re.findall(r'(\d)(\d)', line)
if matches:
  print(matches)  # [('8', '3'), ('7', '9'), ('4', '2')]

matches = re.findall(r'(\d)\D*', line)
if matches:
  print(matches)  # [('8', '3', '7', '9', '5', '4', '2')]
```

```

matches = re.findall(r'(\d)\D*(\d?)', line)
print(matches)  # [('8', '3'), ('7', '9'), ('5', '4'), ('2', '')]

matches = re.findall(r'(\d).*?(\d)', line)
print(matches) # [('8', '3'), ('7', '9'), ('5', '4')]

matches = re.findall(r'(\d+)\D+(\d+)', line)
print(matches) # [('83795', '42')]

matches = re.findall(r'(\d+).*?(\d+)', line)
print(matches) # [('83795', '42')]

matches = re.findall(r'\d', line)
print(matches) # ['8', '3', '7', '9', '5', '4', '2']
```

Fixing dates

In the input we get dates like this 2010-7-5 but we would like to make sure we have two digits for both days and months: 2010-07-05

```
import re

def fix_date(date):
    return re.sub(r'-(\d)\b', r'-0\1', date)


dates = {
    '2010-7-5'   : '2010-07-05',
    '2010-07-5'  : '2010-07-05',
    '2010-07-05' : '2010-07-05',
    '2010-7-15'  : '2010-07-15',
}

for original in sorted(dates.keys()):
    result = fix_date(original)

    assert result == dates[original]

    print(f"      old: {original}")
```

```python
    print(f"      new: {result}")
    print(f" expected: {dates[original]}")
    print("")
```

```
      old: 2010-07-05
      new: 2010-07-05
 expected: 2010-07-05

      old: 2010-07-5
      new: 2010-07-05
 expected: 2010-07-05

      old: 2010-7-15
      new: 2010-07-15
 expected: 2010-07-15

      old: 2010-7-5
      new: 2010-07-05
 expected: 2010-07-05
```

Duplicate numbers

```python
import re

text = "This is 1 string with 3 numbers: 34"
new_text = re.sub(r'(\d+)', r'\1\1', text)
print(new_text)    # This is 11 string with 33 numbers: 3434

double_numbers = re.sub(r'(\d+)', lambda match: str(2 * int(match.group(0))), text)
print(double_numbers)   # This is 2 string with 6 numbers: 68
```

Remove spaces

```python
line = "  ab cd  "

res = line.lstrip(" ")
print(f"'{res}'")        # 'ab cd  '
```

```

res = line.rstrip(" ")
print(f"'{res}'")        # '  ab cd'

res = line.strip(" ")
print(f"'{res}'")        # 'ab cd'

res = line.replace(" ", "")
print(f"'{res}'")        # 'abcd'
```

Replace string in assembly code

```
mv A, R3
mv R2, B
mv R1, R3
mv B1, R4
add A, R1
add B, R1
add R1, R2
add R3, R3
add R21, X
add R12, Y
mv X, R2
```

```python
import sys
import re

if len(sys.argv) != 2:
    exit(f"Usage: {sys.argv[0]} FILENAME")

filename = sys.argv[1]

with open(filename) as fh:
    code = fh.read()

# assuming there are no R4 values then 4 substitutions will do
code = re.sub(r'R1', 'R4', code)
code = re.sub(r'R3', 'R1', code)
code = re.sub(r'R2', 'R3', code)
code = re.sub(r'R4', 'R2', code)
```

```

print(code)
```

```
import sys
import re

if len(sys.argv) != 2:
    exit(f"Usage: {sys.argv[0]} FILENAME")

filename = sys.argv[1]

with open(filename) as fh:
    code = fh.read()


# or without any assumption and in one substitution:
mapping = {
    'R1' : 'R2',
    'R2' : 'R3',
    'R3' : 'R1',
}

code = re.sub(r'\b(R[123])\b', lambda match: mapping[match.group(1)], code)

print(code)
```

```
import sys
import re

if len(sys.argv) != 2:
    exit(f"Usage: {sys.argv[0]} FILENAME")

filename = sys.argv[1]

with open(filename) as fh:
    code = fh.read()


# or without any assumption and in one substitution:
mapping = {
    'R1'  : 'R2',
    'R2'  : 'R3',
```

```python
    'R3'  : 'R1',
    'R12' : 'R21',
    'R21' : 'R12',
}

code = re.sub(r'\b(R1|R2|R3|R12)\b', lambda match: mapping[match.group(1)], code)

print(code)
```

```python
import sys
import re

if len(sys.argv) != 2:
    exit(f"Usage: {sys.argv[0]} FILENAME")

filename = sys.argv[1]

with open(filename) as fh:
    code = fh.read()


# or without any assumption and in one substitution:
mapping = {
    'R1'  : 'R2',
    'R2'  : 'R3',
    'R3'  : 'R1',
    'R12' : 'R21',
    'R21' : 'R12',
}

regex = r'\b(' + '|'.join(mapping.keys()) + r')\b'

code = re.sub(regex, lambda match: mapping[match.group(1)], code)

print(code)
```

Full example of previous

```python
import sys
import os
```

```python
import time
import re

if len(sys.argv) <= 1:
    exit(f"Usage: {sys.argv[0]} INFILEs")

conversion = {
    'R1'  : 'R2',
    'R2'  : 'R3',
    'R3'  : 'R1',
    'R12' : 'R21',
    'R21' : 'R12',
}
#print(conversion)

def replace(mapping, files):
    regex = r'\b(' + '|'.join(mapping.keys()) + r')\b'
    #print(regex)
    ts = time.time()

    for filename in files:
        with open(filename) as fh:
            data = fh.read()
        data = re.sub(regex, lambda match: mapping[match.group(1)], data)
        os.rename(filename, f"{filename}.{ts}")       # backup with current timestamp
        with open(filename, 'w') as fh:
            fh.write(data)

replace(conversion, sys.argv[1:]);
```

Split with regex

```
fname    =    Foo
lname    = Bar
email=foo@bar.com
```

```python
import sys
import re

# data: field_value_pairs.txt
```

```python
if len(sys.argv) != 2:
    exit(f"Usage: {sys.argv[0]} filename")

filename = sys.argv[1]

with open(filename) as fh:
    for line in fh:
        line = line.rstrip("\n")
        field, value = re.split(r'\s*=\s*', line)
        print(f"{value}={field}")
```

```
Foo=fname
Bar=lname
foo@bar.com=email
```

Exercises: Regexes part 1

Pick up a file with some text in it. Write a script (one for each item) that prints out every line from the file
that matches the requirement. You can use the script at the end of the page as a starting point but you will
have to change it!

- has a ‘q’
- starts with a ‘q’
- has ‘th’
- has an ‘q’ or a ‘Q’
- has a ‘*’ in it
- starts with an ‘q’ or an ‘Q’
- has both ‘a’ and ‘e’ in it
- has an ‘a’ and somewhere later an ‘e’
- does not have an ‘a’
- does not have an ‘a’ nor ‘e’
- has an ‘a’ but not ‘e’
- has at least 2 consecutive vowels (a,e,i,o,u) like in the word “bear”
- has at least 3 vowels
- has at least 6 characters
- has at exactly 6 characters
- all the words with either ‘Bar’ or ‘Baz’ in them
- all the rows with either ‘apple pie’ or ‘banana pie’ in them
- for each row print if it was apple or banana pie?
- Bonus: Print if the same word appears twice in the same line
- Bonus: has a double character (e.g. ‘oo’)

```
import sys
import re
```

```python

if len(sys.argv) != 2:
    print("Usage:", sys.argv[0], "FILE")
    exit()

filename = sys.argv[1]
with open(filename, 'r') as fh:
    for line in fh:
        print(line, end=" ")

        match = re.search(r'REGEX1', line)
        if match:
            print("   Matching 1", match.group(0))

        match = re.search(r'REGEX2', line)
        if match:
            print("   Matching 2", match.group(0))
```

Exercise: Regexes part 2

Write functions that returns true if the given value is a

- Hexadecimal number
- Octal number
- Binary number

Write a function that given a string it return true if the string is a number.
As there might be several definitions of what is the number create several solutions
one for each definition:

- Non negative integer.
- Integer. (Will you also allow + in front of the number or only - ?
- Real number. (Do you allow .3 ? What about 2. ?
- In scientific notation. (something like this: 2.123e4)

```
23
2.3
2.3.4
2.4e3
abc
```

Exercise: Sort SNMP numbers

Given a file with SNMP numbers (one number on every line)
print them in sorted order comparing the first number of
each SNMP number first.
If they are equal then comparing the second number, etc…

input:

```
1.2.7.6
4.5.7.23
1.2.7
1.12.23
2.3.5.7.10.8.9
1.2.7.5
```

output:

```
1.2.7
1.2.7.5
1.2.7.6
1.12.23
2.3.5.7.10.8.9
4.5.7.23
```

Exercise: parse hours log file and give report

The log file looks like this

```
09:20 Introduction
11:00 Exercises
11:15 Break
11:35 Numbers and strings
12:30 Lunch Break
13:30 Exercises
14:10 Solutions
14:30 Break
14:40 Lists
15:40 Exercises
17:00 Solutions
17:30 End

09:30 Lists and Tuples
10:30 Break
10:50 Exercises
12:00 Solutions
12:30 Dictionaries
12:45 Lunch Break
```

```
14:15 Exercises
16:00 Solutions
16:15 Break
16:30 Functions
17:00 Exercises
17:30 End
```

the report should look something like this:

```
09:20-11:00 Introduction
11:00-11:15 Exercises
11:15-11:35 Break
11:35-12:30 Numbers and strings
12:30-13:30 Lunch Break
13:30-14:10 Exercises
14:10-14:30 Solutions
14:30-14:40 Break
14:40-15:40 Lists
15:40-17:00 Exercises
17:00-17:30 Solutions

09:30-10:30 Lists and Tuples
10:30-10:50 Break
10:50-12:00 Exercises
12:00-12:30 Solutions
12:30-12:45 Dictionaries
12:45-14:15 Lunch Break
14:15-16:00 Exercises
16:00-16:15 Solutions
16:15-16:30 Break
16:30-17:00 Functions
17:00-17:30 Exercises

Break                      65 minutes    6%
Dictionaries               15 minutes    1%
Exercises                 340 minutes   35%
Functions                  30 minutes    3%
Introduction              100 minutes   10%
Lists                      60 minutes    6%
Lists and Tuples           60 minutes    6%
Lunch Break               150 minutes   15%
Numbers and strings        55 minutes    5%
Solutions                  95 minutes    9%
```

Exercise: Parse ini file

An ini file has sections starting by the name of the section in square brackets and within
each section there are key = value pairs with optional spaces around the "=" sign.
The keys can only contain letters, numbers, underscore or dash.
In addition there can be empty lines and lines starting with # which are comments.

Given a filename, generate a 2 dimensional hash and then print it out.
Example ini file:

```
# comment
[alpha]

base   = moon
ship= alpha 3

[earth]
   # ?
base=London
ship=  x-wing
```

If you print it, it should look like this (except of the nice formatting).

```
{
    'alpha': {
        'base': 'moon',
        'ship': 'alpha 3'
     },
     'earth': {
         'base': 'London',
         'ship': 'x-wing'
     }
}
```

Exercise: Replace Python

```
Replace all occurrences of Python or python by Java ...
... but avoid replacing Monty Python.
```

Exercise: Extract phone numbers

```
Given a text message fetch all the phone numbers:
Fetch numbers that look like 09-1234567
then also fetch +972-2-1234567
and maybe also 09-123-4567
This 123 is not a phone number.
```

Solution: Sort SNMP numbers

```python
import sys

def process(filename):
    snmps = []
    with open(filename) as fh:
        for row in fh:
            snmps.append({
                'orig': row.rstrip(),
            })
    #print(snmps)

    max_number_of_parts = 0
    max_number_of_digits = 0
    for snmp in snmps:
        snmp['split'] = snmp['orig'].split('.')
        max_number_of_parts  = max(max_number_of_parts, len(snmp['split']))
        for part in snmp['split']:
            max_number_of_digits = max(max_number_of_digits, len(part))

    padding = "{:0" + str(max_number_of_digits)  +  "}"
    #print(padding)
    for snmp in snmps:
        padded = []
        padded_split = snmp['split'] + ['0'] *
```

```
(max_number_of_parts - len(snmp['split\
']))

        for part in padded_split:
            padded.append(padding.format( int(part)))
        snmp['padded'] = padded
        snmp['joined'] = '.'.join(padded)


    #print(snmps)
    #print(max_number_of_parts)
    #print(max_number_of_digits)

    snmps.sort(key = lambda e: e['joined'])
    sorted_snmps = []
    for snmp in snmps:
        sorted_snmps.append( snmp['orig'] )
    for snmp in sorted_snmps:
       print(snmp)

# get the max number of all the snmp parts
# make each snmp the same length
# pad each part to that length with leading 0s

if len(sys.argv) < 2:
    exit("Usage: {} FILENAME".format(sys.argv[0]))
process(sys.argv[1])
```

Solution: parse hours log file and give report

```
import sys


if len(sys.argv) < 2:
    exit("Usage: {} FILENAME".format(sys.argv[0]))



data = {}

def read_file(filename):
    entries = []
    with open(filename) as fh:
```

```python
        for row in fh:
            row = row.rstrip("\n")
            if row == '':
                process_day(entries)
                entries = []
                continue
            #print(row)
            time, title = row.split(" ", 1)
            #print(time)
            #print(title)
            #print('')

            entries.append({
                'start': time,
                'title': title,
            })
        process_day(entries)

def process_day(entries):
    for i in range(len(entries)-1):
        start = entries[i]['start']
        title = entries[i]['title']
        end   = entries[i+1]['start']
        print("{}-{} {}".format(start, end, title))

        # manual way to parse timestamp and calculate elapsed time
        # as we have not learned to use the datetim module yet
        start_hour, start_min = start.split(':')
        end_hour, end_min = end.split(':')
        start_in_min = 60*int(start_hour) + int(start_min)
        end_in_min = 60*int(end_hour) + int(end_min)
        elapsed_time = end_in_min - start_in_min
        #print(elapsed_time)

        if title not in data:
            data[title] = 0
        data[title] += elapsed_time


    print('')

def print_summary():
    total = 0
```

```
    for val in data.values():
        total += val

    for key in sorted( data.keys() ):
        print("{:20}     {:4} minutes  {:3}%".format(key,
data[key], int(100 * data[k\
ey]/total)))


read_file( sys.argv[1] )
print_summary()
```

Solution: Processing INI file manually

```
# comment

      # deep comment

outer = 42

[person]
fname = Foo
lname=Bar
phone =    123

[company]
name = Acme Corp.
phone = 456
```

```
import sys
import re

# Sample input data.ini

def parse():
    if len(sys.argv) != 2:
        exit("Usage: {} FILEAME".format(sys.argv[0]))
    filename = sys.argv[1]
    data = {}
    # print("Dealing with " + filename)
    with open(filename) as fh:
        section = '__DEFAULT__'
```

```python
        for line in fh:
            if re.match(r'^\s*(#.*)?$', line):
                continue
            match = re.match(r'^\[([^\]]+)\]\s*$', line)
            if (match):
                # print('Section "{}"'.format(m.group(1)))
                section = match.group(1)
                continue
            match = re.match(r'^\s*(.+?)\s*=\s*(.*?)\s*$', line)
            if match:
                # print 'field :"{}"  value: "{}"'.format(m.group(1), m.group(2))
                if not data.get(section):
                    data[section] = {}
                data[section][ match.group(1) ] = match.group(2)

    return data

if __name__ == '__main__':
    ini = parse()
    print(ini)
```

Solution: Processing config file

```
[person]
fname = Foo
lname=Bar
phone =    123

# comment

      # deep comment


[company]
name = Acme Corp.
phone = 456
```

```python
import configparser
import sys

def parse():
  if len(sys.argv) != 2:
    print("Usage: " + sys.argv[0] + "  FILEAME")
    exit()
  filename = sys.argv[1]

  cp = configparser.RawConfigParser()
  cp.read(filename)
  return cp

ini = parse()

for section in ini.sections():
  print(section)
  for v in ini.items(section):
    print("  {}  = {}".format(v[0], v[1]))
```

Solution: Extract phone numbers

```python
import re

filename = "phone.txt"
with open(filename) as fh:
    for line in fh:
        match = re.search(r'''\b
            (
                \d\d-\d{7}
                |
                \d\d\d-\d-\d{7}
                |
                \d\d-\d\d\d-\d\d\d\d
            )\b''', line, re.VERBOSE)
        if match:
            print(match.group(1))
```

Regular Expressions Cheat sheet

Expression	Meaning
a	Just an ‘a’ character
.	any character except new-line
[bgh.]	one of the chars listed in the character class b,g,h or .
[b-h]	The same as [bcdefgh]
[a-z]	Lower case letters
[b-]	The letter b or -
[^bx]	Anything except b or x
\w	Word characters: [a-zA-Z0-9_]
\d	Digits: [0-9]
\s	[\f\t\n\r] form-feed, tab, newline, carriage return and SPACE
W	[^\w]
D	[^\d]
S	[^\s]
a*	0-infinite ‘a’ characters
a+	1-infinite ‘a’ characters
a?	0-1 ‘a’ characters
a	n-m ‘a’ characters
( )	Grouping and capturing
	Alternation
\1, \2	Capture buffers
^ $	Beginning and end of string anchors

re

Fix bad JSON

```
{
    subscriptions : [
        {
            name : "Foo Bar",
            source_name : "pypi",
            space names : [
                "Foo", "Bar"
            ]
        }
    ]
}
```

```python
import re, json, os

json_file = os.path.join(
    os.path.dirname(__file__),
    'bad.json'
)
with open(json_file) as fh:
    data = json.load(fh)
    # ValueError: Expecting property name: line 2 column 4 (char 5)
```

```python
import re, json, os

def fix(s):
    return re.sub(r'(\s)([^:\s][^:]+[^:\s])(\s+:)', r'\1"\2"\3', s)

json_file = os.path.join(
    os.path.dirname(__file__),
    'bad.json'
)
with open(json_file) as fh:
    bad_json_rows = fh.readlines()
    json_str = ''.join(map(fix, bad_json_rows))
    print(json_str)
    data = json.loads(json_str)
    print(data)
```

Fix very bad JSON

```
[
{
    TID : "t-0_login_sucess"
    Test :
    [
        {SetValue : { uname : "Zorg", pass : "Rules"} },
        {DoAction : "login"},
        {CheckResult: [0, LOGGED_IN]}
    ]
},
{ TID : "t-1_login_failure", Test : [ {SetValue :
{ uname : "11", pass : "im2happy78"} },
{DoAction : "login"}, {CheckResult: [-1000, LOGGED_OUT]} ] }
]
```

```python
import re, json, os

json_file = os.path.join(
    os.path.dirname(__file__),
    'very_bad.json'
)
with open(json_file, 'r') as fh:
    bad_json = fh.read()
    #print(bad_json)
    improved_json  = re.sub(r'"\s*$', '",', bad_json, flags=re.MULTILINE)
    #print(improved_json)

    # good_json = re.sub(r'(?<!")(?P<word>[\w-]+)\b(?!")', '"\g<word>"',
    #    improved_json)
    # good_json = re.sub(r'(?<[\{\s])(?P<word>[\w-]+)(?=[:\s])', '"\g<word>"',
    #    improved_json)
    # good_json = re.sub(r'([\{\[\s])(?P<word>[\w-]+)([:,\]\s])', '\1"\g<word>"\3',
    #    improved_json)
    good_json = re.sub(r'(?<=[\{\[\s])(?P<word>[\w-]+)(?=[:,\]\s])', '"\g<word>"',
      improved_json)
    #print(good_json)

# with open('out.js', 'w') as fh:
```

```
#      fh.write(good_json)

data = json.loads(good_json)
print(data)
```

Raw string or escape

Let's try to check if a string contains a back-slash?

```
import re

txt = 'text with slash \ and more text'
print(txt)          # text with slash \ and more text

# m0 = re.search('\', txt)
    # SyntaxError: EOL while scanning string literal

# m0 = re.search('\\', txt)
    # Exception:  sre_constants.error: bogus escape (end of line)
    # because the regex engine does not know what to do with a single \

m1 = re.search('\\\\', txt)
if m1:
    print('m1')     # m1

m2 = re.search(r'\\', txt)
if m2:
    print('m2')     # m2
```

Remove spaces regex

This is not necessary as we can use rstrip, lstrip, and replace.

```
import re

line = "  ab cd  "

res = re.sub(r'^\s+', '', line)  # leading
print(f"'{res}'")
```

```python

res = re.sub(r'\s+$', '', line)   # trailing
print(f"'{res}'")
```

both ends:

```
re.sub(r'\s*(.*)\s*$', r'\1', line)   #  " abc " =>  "abc "  because of the greediness
```

```
re.sub('^\s*(.*?)\s*$', '\1', line)   #   " abc " => "abc"  minimal match
```

Regex Unicode

Python 3.8 required

```python
print("\N{GREEK CAPITAL LETTER DELTA}")

print("\u05E9")
print("\u05DC")
print("\u05D5")
print("\u05DD")
print("\u262E")
print("\u1F426")    # "bird"

print("\u05E9\u05DC\u05D5\u05DD \u262E")
```

```
Hello World!
Szia Világ!
שלום עולם!
```

```python
import re

filename = "mixed.txt"

with open(filename) as fh:
    lines = fh.readlines()
for line in lines:
```

```
    if re.search('\N{IN HEBREW}', line):
        print(line)
```

Anchors Other example

```
import re

strings = [
    "123-XYZ-456",
    "a 123-XYZ-456 b",
    "a 123-XYZ-456",
    "123-XYZ-456 b",
    "123-XYZ-456\n",
]

regexes = [
    r'\d{3}-\w+-\d{3}',
    r'^\d{3}-\w+-\d{3}',
    r'\d{3}-\w+-\d{3}$',
    r'^\d{3}-\w+-\d{3}$',
    r'^\d{3}-\w+-\d{3}\Z',
    r'\A\d{3}-\w+-\d{3}\Z',
]

for r in regexes:
    print(r)
    for s in strings:
        #print(r, s)
        if (re.search(r, s)):
            print('    ', s)
    print('-' * 10)
```

Python standard modules

Some Standard modules

- sys - System specific
- os - Operating System
- stat - inode table
- shutil - File Operations
- glob - Unix style pathname expansion
- subprocess - Processes

- argparse - Command Line Arguments
- re - Regexes
- math - Mathematics
- time - timestamp and friends
- datetime - time management
- random - Random numbers

sys

```
import sys,os

print(sys.argv) # the list of the values
    # on the command line sys.argv[0] is the name of the Python script

print(sys.executable)  # path to the python interpreter

# print(sys.path)
    # list of file-system path strings for searching for modules
    # hard-coded at compile time but can be changed via the PYTHONPATH
```

```
    # environment variable or during execution by modifying sys.path

print(sys.version_info)
# sys.version_info(major=2, minor=7, micro=12, releaselevel='final', serial=0)

print(sys.version_info.major)  # 2 or 3

print(sys.platform)    # darwin   or  linux2   or  win32

print(os.uname())
# On Mac:
# ('Darwin', 'air.local', '16.3.0', 'Darwin Kernel Version 16.3.0: Thu Nov 17 20:23:\
58 PST 2016; root:xnu-3789.31.2~1/RELEASE_X86_64', 'x86_64')

# On Linux:
# posix.uname_result(sysname='Linux', nodename='thinkpad', release='5.0.0-32-generic\
', version='#34-Ubuntu SMP Wed Oct 2 02:06:48 UTC 2019', machine='x86_64')
```

```
['examples/sys/mysys.py']

/usr/bin/python

['/Users/gabor/work/training/python/examples/sys',
 '/Users/gabor/python/lib/python2.7/site-packages/crypto-1.1.0-py2.7.egg',
 ...,
 '/Users/gabor/python',
 '/Users/gabor/python/lib/python2.7/site-packages',
 ...]
```

Writing to standard error (stderr)

```
import sys

print("on stdout (Standard Output)")
```

```
print("on stderr (Standard Error)", file=sys.stderr)
sys.stderr.write("in stderr again\n")
```

Redirection:

```
python stderr.py > out.txt  2> err.txt
python stderr.py 2> /dev/null
python stderr.py > out.txt 2>&1
```

Current directory (getcwd, pwd, chdir)

```
import os

this_dir = os.getcwd()
print(this_dir)

# os.chdir('/path/to/some/dir')
os.chdir('..')
```

OS dir (mkdir, makedirs, remove, rmdir)

```
os.mkdir(path_to_new_dir)
os.makedirs(path_to_new_dir)

os.remove()       remove a file
os.unlink()       (the same)

os.rmdir()         single empty directory
os.removedirs()    empty subdirectories as well
shutil.rmtree()    rm -rf
```

python which OS are we running on (os, platform)

```
import os
import platform

```

```python
print(os.name)
print(platform.system())
print(platform.release())

# posix
# Linux
# 5.3.0-24-generic
```

Get process ID

```python
import os

print(os.getpid())
print(os.getppid())
```

```
93518
92859
```

```
echo $$
```

OS path

```python
import os

os.path.basename(path_to_thing)
os.path.dirname(path_to_thing)
os.path.abspath(path_to_file)

os.path.exists(path_to_file)
os.path.isdir(path_to_thing)

os.path.expanduser('~')
```

Traverse directory tree - list directories recursively

```python
import os
import sys

if len(sys.argv) != 2:
    exit("Usage: {} PATH_TO_DIRECTORY".format(sys.argv[0]))

root = sys.argv[1]

for dirname, dirs, files in os.walk(root):
    #print(dirname)      # relative path (from cwd) to the directory being processed
    #print(dirs)         # list of subdirectories in the currently processed directory
    #print(files)        # list of files in the currently processed directory

    for filename in files:
        print(os.path.join(dirname, filename))    # relative path to the "current" fi\
le
```

os.path.join

```python
import os

path = os.path.join('home', 'foo', 'work')
print(path)   # home/foo/work
```

Directory listing

```python
import os
import sys

if len(sys.argv) != 2:
    exit("Usage: {} directory".format(sys.argv[0]))

path = sys.argv[1]
files = os.listdir(path)
for name in files:
```

```
    print(name)
    print(os.path.join(path, name))
```

expanduser - handle tilde ~

```
import os

print( os.path.expanduser("~") )
print( os.path.expanduser("~/work") )
print( os.path.expanduser("~/other") )
print( os.path.expanduser("some/other/dir/no/expansion") )
```

Listing specific files using glob

```
import glob

files = glob.glob("*.py")
print(files)

files = glob.glob("/usr/bin/*.sh")
print(files)
```

External command with system

```
import os

command = 'ls -l'

os.system(command)
```

If you wanted to list the content of a directory in an os independent way you'd use `os.listdir('.')`
or you could use the `glob.glob("*.py")` function to have a subset of files.

subprocess

Run external command and capture the output

```python
import time
import sys

for i in range(3):
    print("OUT {}".format(i))
    print("ERR {}".format(i), file=sys.stderr)
    time.sleep(1)
```

```python
import subprocess
import sys

command = [sys.executable, 'slow.py']

proc = subprocess.Popen(command,
    stdout = subprocess.PIPE,
    stderr = subprocess.PIPE,
)

out,err = proc.communicate()   # runs the code

# out and err are two strings

print('exit code:', proc.returncode)

print('out:')
for line in out.decode('utf8').split('\n'):
    print(line)

print('err:')
for line in err.decode('utf8').split('\n'):
    print(line)
```

In this example p is an instance of the subprocess.PIPE class. The command is executed when the object is created.

subprocess in the background

```python
import subprocess
import sys
import time


proc = subprocess.Popen([sys.executable, 'slow.py'],
    stdout = subprocess.PIPE,
    stderr = subprocess.PIPE,
)

#out, err = proc.communicate()  # this is when the code starts executing
#print(out)
#print(err)

timeout = 6
while True:
    poll = proc.poll()
    print(poll)
    time.sleep(0.5)
    timeout -= 0.5
    if timeout <= 0:
        break
    if poll is not None:
        break

print("Final: {}".format(poll))
if poll is None:
    pass
else:
    out, err = proc.communicate()
    print(out)
    print(err)
```

Accessing the system environment variables from Python

```python
import os

print(os.environ['HOME']) # /Users/gabor
print(os.environ.get('HOME')) # /Users/gabor

```

```python
for k in os.environ.keys():
    print("{:30} {}".format(k , os.environ[k]))
```

os.environ is a dictionary where the keys are the environment variables and the values are, well, the values.

Set env and run command

```python
import os

os.system("echo hello")
os.system("echo $HOME")

os.system("echo Before $MY_TEST")
os.environ['MY_TEST'] = 'qqrq'
os.system("echo After $MY_TEST")
```

We can change the environment variables and that change will be visible in subprocesses,
but once we exit from ou Python program, the change will not persist.

shutil

```python
import shutil

shutil.copy(source, dest)
shutil.copytree(source, dest)
shutil.move(source, dest)
shutil.rmtree(path)
```

time

```
import time

print(time.time())      # 1351178170.85

print(time.timezone)    # 7200 = 2*60*60  (GMT + 2)
print(time.daylight)    # 1 (DST or Daylight Saving Time)

print(time.gmtime())    # time.struct_time
    # time.struct_time(tm_year=2012, tm_mon=10, tm_mday=25,
    # tm_hour=17, tm_min=25, tm_sec=34, tm_wday=3, tm_yday=299, tm_isdst=0)

t = time.gmtime()
print(t.tm_year) # 2012

print(time.strftime('%Y-%m-%d %H:%M:%S')) # with optional timestamp
```

sleep in Python

```
import time

start = time.time()
print("hello " + str(start))

time.sleep(3.5)

end = time.time()
print("world " + str(end))
print("Elapsed time:" + str(end-start))
```

```
hello 1475217162.472256
world 1475217165.973437
Elapsed time:3.501181125640869
```

timer

More time-related examples.

```
import random
import time

# https://docs.python.org/3/library/time.html#time.struct_time

print(time.time())      # time since the epoch in seconds
print(time.asctime())   # current local time in human-readable format
print(time.strftime("%Y-%m-%d %H:%M:%S"))  # create your own human-readable format

print(time.gmtime(0))  # epoch
print(time.asctime(time.gmtime(0)))  # epoch in human-readable format

print(time.localtime()) # local time now
print(time.gmtime()) # time in London



print(time.process_time())
print(time.process_time_ns())

s = time.perf_counter()
ps = time.process_time()
print(time.monotonic())
time.sleep(0.1)
print(time.monotonic())
e = time.perf_counter()
for _ in range(100000):
    random.random()
pe = time.process_time()
print(s)
print(e)
print(e-s)
print(pe-ps)

# print(time.get_clock_info('monotonic'))
```

Current date and time datetime now

```
import datetime

now   = datetime.datetime.now()
print(now)                 # 2015-07-02 16:28:01.762244
print(type(now))           # <type 'datetime.datetime'>

print(now.year)            # 2015
print(now.month)           # 7
print(now.day)             # 2
print(now.hour)            # 16
print(now.minute)          # 28
print(now.second)          # 1
print(now.microsecond)     # 762244

print(now.strftime("%Y%m%d-%H%M%S-%f"))   # 20150702-162801-762244
print(now.strftime("%B %b %a %A"))        # July Jul Thu Thursday
print(now.strftime("%c"))                 # Thu Jul  2 16:28:01 2015
```

Converting string to datetime

```
import datetime

usa_date_format   = "%m/%d/%Y" # MM/DD/YYYY
world_date_format = "%d/%m/%Y" # DD/MM/YYYY
other_date_format = "%Y/%m/%d" # YYYY/MM/DD


d = "2012-12-19"
some_day = datetime.datetime.strptime(d, '%Y-%m-%d') # YYYY-MM-DD
print(some_day)             # 2012-12-19
print(type(some_day))       # <type 'datetime.datetime'>

t = "2013-11-04 11:23:45"   # YYYY-MM-DD HH:MM:SS
some_time = datetime.datetime.strptime(t, '%Y-%m-%d %H:%M:%S')
print(type(some_time))      # <type 'datetime.date'>
```

```python
print(some_time)         # 2013-11-04
print(some_time.minute)  # 23
```

datetime arithmeticis

```python
import datetime

t1 = "2013-12-29T11:23:45"
t2 = "2014-01-02T10:19:49"
dt1 = datetime.datetime.strptime(t1, '%Y-%m-%dT%H:%M:%S')
dt2 = datetime.datetime.strptime(t2, '%Y-%m-%dT%H:%M:%S')
print(dt1)       # 2013-12-29 11:23:45
print(dt2)       # 2014-01-02 10:19:49

d = dt2-dt1
print(d)         # 3 days, 22:56:04
print(type(d))   # <type 'datetime.timedelta'>
print(d.total_seconds())   # 341764.0

nd = dt1 + datetime.timedelta(days = 3)
print(nd)        # 2014-01-01 11:23:45
```

Rounding datetime object to nearest second

```python
import datetime

d = datetime.datetime.now()
x = d - datetime.timedelta(microseconds=d.microsecond)
print(d) # 2019-11-01 07:11:19.930974
print(x) # 2019-11-01 07:11:19
```

Signals and Python

- man 7 signal (on Linux)
- Unix: kill PID, kill -9 PID, Ctrl-C, Ctrl-Z
- os.kill
- signal

Sending Signal

```python
import signal
import os

print("before")
os.kill(os.getpid(), signal.SIGUSR1)
print("after")
```

```
before
User defined signal 1: 30
```

Catching Signal

```python
import signal
import os

def handler(signum, frame):
    print('Signal handler called with signal', signum)

signal.signal(signal.SIGUSR1, handler)

print("before")
os.kill(os.getpid(), signal.SIGUSR1)
print("after")
```

```
before
('Signal handler called with signal', 30)
after
```

Catching Ctrl-C on Unix

```python
username = input('Username:')
print(username)
```

```
$  python ctrl_c.py
```

```
Username:^CTraceback (most recent call last):
  File "ctrl_c.py", line 3, in <module>
    username = input('Username:')
KeyboardInterrupt
```

```python
import signal

def handler(signum, frame):
    print('Signal handler called with signal', signum)

signal.signal(signal.SIGINT, handler)

username = input('Username:')
print(username)
```

- Cannot stop using Ctrl-C !
- Ctrl-Z and then kill %1
- kill PID

Catching Ctrl-C on Unix confirm

```python
import signal
import time

def handler(signum, frame):
    answer = input('We are almost done. Do you really want to exit? [yes]:')
    if answer == 'yes':
        print('bye')
        exit()
    print("Then let's keep running")

signal.signal(signal.SIGINT, handler)

for _ in range(10):
    time.sleep(5)
```

Alarm signal and timeouts

```python
import signal

class MyTimeout(Exception):
    pass

def handler(signum, frame):
    print('Signal handler called with signal', signum)
    raise MyTimeout

try:
    signal.signal(signal.SIGALRM, handler)
    signal.alarm(5)
    number = input("Divide by (5 sec):")
    signal.alarm(0)
    print(42/int(number))
except MyTimeout:
    print('timeout')
except Exception as e:
    print(e)
    #raise

print("Still working")
```

deep copy list

```python
a = [
    {
      'name': 'Joe',
      'email': 'joe@examples.com',
    },
    {
      'name': 'Mary',
      'email': 'mary@examples.com',
    },
]


b = a
a[0]['phone'] = '1234'
a[0]['name'] = 'Jane'
a.append({
    'name': 'George'
})
```

```

print(a)
print(b)
```

```
[{'name': 'Jane', 'email': 'joe@examples.com', 'phone': '1234'}, {'name': 'Mary', 'e\
mail': 'mary@examples.com'}, {'name': 'George'}]
[{'name': 'Jane', 'email': 'joe@examples.com', 'phone': '1234'}, {'name': 'Mary', 'e\
mail': 'mary@examples.com'}, {'name': 'George'}]
```

```
a = [
    {
      'name': 'Joe',
      'email': 'joe@examples.com',
    },
    {
      'name': 'Mary',
      'email': 'mary@examples.com',
    },
]


b = a[:]
a[0]['phone'] = '1234'
a[0]['name'] = 'Jane'
a.append({
    'name': 'George'
})

print(a)
print(b)
```

```
[{'name': 'Jane', 'email': 'joe@examples.com', 'phone': '1234'}, {'name': 'Mary', 'e\
mail': 'mary@examples.com'}, {'name': 'George'}]
[{'name': 'Jane', 'email': 'joe@examples.com', 'phone': '1234'}, {'name': 'Mary', 'e\
mail': 'mary@examples.com'}]
```

```python
from copy import deepcopy

a = [
    {
      'name': 'Joe',
      'email': 'joe@examples.com',
    },
    {
      'name': 'Mary',
      'email': 'mary@examples.com',
    },
]


b = deepcopy(a)
a[0]['phone'] = '1234'
a[0]['name'] = 'Jane'
a.append({
    'name': 'George'
})

print(a)
print(b)
```

```
[{'name': 'Jane', 'email': 'joe@examples.com', 'phone': '1234'}, {'name': 'Mary', 'e\
mail': 'mary@examples.com'}, {'name': 'George'}]
[{'name': 'Joe', 'email': 'joe@examples.com'}, {'name': 'Mary', 'email': 'mary@examp\
les.com'}]
```

deep copy dictionary

```python
a = {
    'name': 'Foo Bar',
    'grades': {
       'math': 70,
       'art' : 100,
    },
    'friends': ['Mary', 'John', 'Jane', 'George'],
}

```

```
b = a
a['grades']['math'] = 90
a['email'] = 'foo@bar.com'
print(a)
print(b)
```

```
{'name': 'Foo Bar', 'grades': {'math': 90, 'art': 100}, 'friends': ['Mary', 'John', \
'Jane', 'George'], 'email': 'foo@bar.com'}
{'name': 'Foo Bar', 'grades': {'math': 90, 'art': 100}, 'friends': ['Mary', 'John', \
'Jane', 'George'], 'email': 'foo@bar.com'}
```

- [deepcopy]()

```python
from copy import deepcopy

a = {
    'name': 'Foo Bar',
    'grades': {
        'math': 70,
        'art' : 100,
    },
    'friends': ['Mary', 'John', 'Jane', 'George'],
}

b = deepcopy(a)
a['grades']['math'] = 90
a['email'] = 'foo@bar.com'
print(a)
print(b)
```

```
{'name': 'Foo Bar', 'grades': {'math': 90, 'art': 100}, 'friends': ['Mary', 'John', \
'Jane', 'George'], 'email': 'foo@bar.com'}
{'name': 'Foo Bar', 'grades': {'math': 70, 'art': 100}, 'friends': ['Mary', 'John', \
'Jane', 'George']}
```

Exercise: Catching Ctrl-C on Unix 2nd time

- When Ctrl-C is pressed display: “In order to really kill the application press Ctrl-C again” and keep running. If the user presses Ctrl-C again, then let id die.
- Improve the previous that if 5 sec within the first Ctrl-C there is no 2nd Ctrl-C then any further Ctrl-C will trigger the above message again.

Exercise: Signals

- What signal is sent when you run **kill PID**?
- Write a script that will disable the **kill PID** for your process. How can you kill it then?
- What signal is sent when we press Ctrl-Z ?

Ctrl-z

```python
import signal
import os

print(os.getpid())

username = input('Username:')
print(username)
```

```
kill PID
```

```python
import signal
import os

print(os.getpid())

def handler(signum, frame):
    print('Signal handler called with signal', signum)

signal.signal(signal.SIGTERM, handler)

```

```python
username = input('Username:')
print(username)
```

JSON

JSON - JavaScript Object Notation

JSON is basically the data format used by JavaScript. Because its universal availability it became the de-facto standard for data communication between many different languages. Most dynamic languages have an fairly good mapping between JSON and their own data structures.
Lists and dictionaries in the case of Python.

Documentation of the
Python json library.

```
{"lname": "Bar", "email": null, "fname": "Foo", "children": ["Moo", "Koo", "Roo"]}
```

dumps

```
import json

a = {
  "fname" : 'Foo',
  "lname" : 'Bar',
  "email" : None,
  "children" : [
     "Moo",
     "Koo",
     "Roo"
  ]
}
print(a)

json_str = json.dumps(a)
```

```python
print(json_str)

with open('data.json', 'w') as fh:
    fh.write(json_str)
```

```
{'lname': 'Bar', 'email': None, 'fname': 'Foo',
  'children': ['Moo', 'Koo', 'Roo']}

{"lname": "Bar", "email": null, "fname": "Foo",
  "children": ["Moo", "Koo", "Roo"]}
```

(lines were broken for readability on the slides)

`dumps` can be used to take a Python data structure and generate a string in JSON format. That string can then be saved in a file, inserted in a database, or sent over the wire.

loads

```python
import json

with open('examples/json/data.json') as fh:
    json_str = fh.read()

print(json_str)
b = json.loads(json_str)
print(b)
```

```
{"lname": "Bar", "email": null, "fname": "Foo",
    "children": ["Moo", "Koo", "Roo"]}

{u'lname': u'Bar', u'email': None, u'fname': u'Foo',
    u'children': [u'Moo', u'Koo', u'Roo']}
```

u is the Unicode prefix used in Python 2. In Python 3 it won’t appear as Unicode is the default there.

dump

```
import json

a = {
  "fname" : 'Foo',
  "lname" : 'Bar',
  "email" : None,
  "children" : [
     "Moo",
     "Koo",
     "Roo"
  ]
}

print(a)

with open('data.json', 'w') as fh:
    json.dump(a, fh)
```

```
{'lname': 'Bar', 'email': None, 'fname': 'Foo',
  'children': ['Moo', 'Koo', 'Roo']}

{"lname": "Bar", "email": null, "fname": "Foo",
  "children": ["Moo", "Koo", "Roo"]}
```

(lines were broken for readability on the slides)

As a special case **dump** will save the string in a file or in other stream.

load

```
import json

with open('examples/json/data.json', 'r') as fh:
    a = json.load(fh)
print(a)
```

```
{u'lname': u'Bar', u'email': None, u'fname': u'Foo',
    u'children': [u'Moo', u'Koo', u'Roo']}
```

Round trip

```python
import json
import os
import time

data = {}
filename = 'mydata.json'

if os.path.exists(filename):
    with open(filename) as fh:
        json_str = fh.read()
        print(json_str)
        data = json.loads(json_str)

data['name'] = 'Foo Bar'
data['time'] = time.time()


with open(filename, 'w') as fh:
   json_str = json.dumps(data)
   fh.write(json_str)
```

Pretty print JSON

```python
import json

data = {
    "name" : "Foo Bar",
    "grades" : [23, 47, 99, 11],
    "children" : {
        "Peti Bar" : {
            "email": "peti@bar.com",
        },
        "Jenny Bar" : {
            "phone": "12345",
        },
    }
```

```
}

print(data)
print(json.dumps(data))
print(json.dumps(data, indent=4, separators=(',', ': ')))
```

```
{'name': 'Foo Bar', 'grades': [23, 47, 99, 11],
'children': {'Peti Bar': {'email': '\
peti@bar.com'}, 'Jenny Bar': {'phone': '12345'}}}
{"name": "Foo Bar", "grades": [23, 47, 99, 11],
"children": {"Peti Bar": {"email": "\
peti@bar.com"}, "Jenny Bar": {"phone": "12345"}}}
{
    "name": "Foo Bar",
    "grades": [
        23,
        47,
        99,
        11
    ],
    "children": {
        "Peti Bar": {
            "email": "peti@bar.com"
        },
        "Jenny Bar": {
            "phone": "12345"
        }
    }
}
```

Sort keys in JSON

```
import json

data = {
    "name" : "Foo Bar",
    "grades" : [23, 47, 99, 11],
    "children" : {
        "Peti Bar" : {
            "email": "peti@bar.com",
        },
        "Jenny Bar" : {
```

```
                "phone": "12345",
            },
        }
}

print(json.dumps(data, sort_keys=True, indent=4, separators=(',', ': ')))
```

```
{
    "children": {
        "Jenny Bar": {
            "phone": "12345"
        },
        "Peti Bar": {
            "email": "peti@bar.com"
        }
    },
    "grades": [
        23,
        47,
        99,
        11
    ],
    "name": "Foo Bar"
}
```

Set order of keys in JSON - OrderedDict

```
from collections import OrderedDict

d = {}
d['a'] = 1
d['b'] = 2
d['c'] = 3
d['d'] = 4
print(d)

planned_order = ('b', 'c', 'd', 'a')
e = OrderedDict(sorted(d.items(), key=lambda x: planned_order.index(x[0])))
print(e)

```

```
print('-----')
# Create index to value mapping dictionary from a list of values
planned_order = ('b', 'c', 'd', 'a')
plan = dict(zip(planned_order, range(len(planned_order))))
print(plan)

f = OrderedDict(sorted(d.items(), key=lambda x: plan[x[0]]))
print(f)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
OrderedDict([('b', 2), ('c', 3), ('d', 4), ('a', 1)])
-----
{'b': 0, 'c': 1, 'd': 2, 'a': 3}
OrderedDict([('b', 2), ('c', 3), ('d', 4), ('a', 1)])
```

Exercise: Counter in JSON

Write a script that will provide several counters. The user can provide an argument on the command
line and the script will increment and display that counter.
Keep the current values of the counters in a single JSON file.
The script should behave like this:

```
$ python counter.py foo
1

$ python counter.py foo
2

$ python counter.py bar
1

$ python counter.py foo
3
```

Exercise: Phone book

Write a script that acts as a phonebook. As “database” use a file in JSON format.

```
$ python phone.py Foo 123
Foo added

$ python phone.py Bar
Bar is not in the phnebook

$ python phone.py Bar 456
Bar added

$ python phone.py Bar
456

$ python phone.py Foo
123
```

Can it handle changes in phone numbers?
Can it remove a name from the “database”?

Exercise: Processes

Write a program that will do “some work” that can be run in parallel
and collect the data. Make the code work in a single process by default
and allow the user to pass a number that will be the number of child processes
to be used. When the child process exits it should save the results in
a file and the parent process should read them in.

The “some work” can be accessing 10-20 machines using “ssh machine uptime”
and creating a report from the results.

It can be fetching 10-20 URLs and reporting the size of each page.

It can be any other network intensive task.

Measure the time in both cases

Solution: Counter in JSON

```
import json
import sys
import os

filename = 'counter.json'

if len(sys.argv) != 2:
    print("Usage: " + sys.argv[0] + " COUNTER")
    exit()

counter = {}

if os.path.exists(filename):
    with open(filename) as fh:
        json_str = fh.read()
        counter = json.loads(json_str)

name = sys.argv[1]
if name in counter:
    counter[name] += 1
else:
    counter[name] = 1

print(counter[name])


with open(filename, 'w') as fh:
    json_str = json.dumps(counter)
    fh.write(json_str)
```

Solution: Phone book

```python
import sys
import json
import os

def main():
    filename = 'phonebook.json'
    phonebook = {}
    if os.path.exists(filename):
        with open(filename) as fh:
            json_str = fh.read()
            phonebook = json.loads(json_str)

    if len(sys.argv) == 2:
        name = sys.argv[1]
        if name in phonebook:
            print(phonebook[name])
        else:
            print("{} is not in the phonebook".format(name))
        return

    if len(sys.argv) == 3:
        name = sys.argv[1]
        phone = sys.argv[2]
        phonebook[name] = phone
        with open(filename, 'w') as fh:
            json_str = json.dumps(phonebook)
            fh.write(json_str)
        return

    print("Invalid number of parameters")
    print("Usage: {} username [phone]".format(sys.argv[0]))

if __name__ == '__main__':
    main()
```

Command line arguments with argparse

Modules to handle the command line

You would like to allow the user to pass arguments on the command line. For example:

```
myprog.py --machine server_name --test name --verbose --debug
myprog.py -v -d
myprog.py -vd
myprog.py file1 file2 file3
```

- sys.argv manual parsing?
- optparse (deprecated)
- argparse

argparse

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--name')      # optional named parameter that requires a value
parser.add_argument('--name', help="Some description")

parser.add_argument('--max', help='max number of somthing', type=int) # check and co\
nvert to integer
parser.add_argument('--verbose', action='store_true') # "flag" no value is expected

parser.add_argument('--color', '-c') # short name also
```

```
accepted


parser.add_argument('files', help="filenames(s)")    # a required positional argument
parser.add_argument('files', nargs="*")    # 0 or more positional
parser.add_argument('files', nargs="+")    # 1 or more positional

parser.add_argument('--files', nargs="+")  # --files a.txt b.txt c.txt


args = parser.parse_args()

print(args.name)
print(args.files)
```

Basic usage of argparse

Setting up the argparse already has some (little) added value.

```
import argparse

parser = argparse.ArgumentParser()
parser.parse_args()

print('the code...')
```

Running the script without any parameter will not interfere…

```
$ python argparse_basic.py
the code...
```

If the user tries to pass some parameters on the command line, the argparse will
print an error message and stop the execution.

```
$ python argparse_basic.py foo
usage: argparse_basic.py [-h]
argparse_basic.py: error: unrecognized arguments: foo
```

```
$ python argparse_basic.py -h
usage: argparse_basic.py [-h]

optional arguments:
  -h, --help  show this help message and exit
```

The minimal set up of the argparse class already provides a (minimally) useful help message.

Positional argument

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('name', help='your full name')
args = parser.parse_args()

print(args.name)
```

```
$ python argparse_positional.py
usage: argparse_positional.py [-h] name
argparse_positional.py: error: too few arguments
```

```
$ python argparse_positional.py -h
usage: argparse_positional.py [-h] name

```

```
positional arguments:
  name        your full name

optional arguments:
  -h, --help  show this help message and exit
```

```
$ python argparse_positional.py Foo
Foo
```

```
$ python argparse_positional.py Foo Bar
usage: argparse_positional.py [-h] name
argparse_positional.py: error: unrecognized arguments:
Bar
```

```
$ python argparse_positional.py "Foo Bar"
Foo Bar
```

Many positional argument

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('files', help='filename(s)',
nargs='+')
args = parser.parse_args()

print(args.files)
```

```
$ python argparse_positional_many.py
usage: argparse_positional_many.py [-h] files [files ...]
argparse_positional_many.py: error: too few arguments
```

```
air:python gabor$ python argparse_positional_many.py
a.txt b.txt
['a.txt', 'b.txt']
```

Convert to integers

```python
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('number', help='the number to take to the square')
args = parser.parse_args()

print(args.number * args.number)
```

```
$ python argparse_number.py abc
```

```
Traceback (most recent call last):
  File "examples/argparse/argparse_number.py", line 10, in <module>
    print(args.number * args.number)
TypeError: can't multiply sequence by non-int of type 'str'
```

Trying to the argument received from the command line as an integer, we get a TypeError. The same would happen even if a number was passed, but you could call `int()` on the parameter to convert to an integer. However there is a better solution.

The same with the following

```
$ python argparse_number.py 23
```

```
Traceback (most recent call last):
  File "examples/argparse/argparse_number.py", line 10, in <module>
    print(args.number * args.number)
TypeError: can't multiply sequence by non-int of type 'str'
```

Convert to integer

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('number', help='the number to take to the square', type=int)
args = parser.parse_args()

print(args.number * args.number)
```

```
$ argparse_type.py abc
usage: argparse_type.py [-h] number
argparse_type.py: error: argument number: invalid int value: 'abc'
```

We got a much better error message as argparse already found out the
argument was a string and not a number as expected.

```
$ argparse_type.py 23
529
```

The `type` parameter can be used to define the type restriction and type conversion of the attributes.

Named arguments

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--color', help='The name of the color')
args = parser.parse_args()

print(args.color)
```

python argparse_named.py –color Blue

```
Blue
```

python argparse_named.py

```
None
```

Named parameters are optional by default. You can pass the `required=True` parameter to make them required.

Boolean Flags

```python
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--color',   help='The name of the color')
parser.add_argument('--verbose', help='Print more data',
    action='store_true')
args = parser.parse_args()

print(args.color)
print(args.verbose)
```

python argparse_boolean.py –color Blue –verbose

```
Blue
True
```

python argparse_boolean.py

```
None
False
```

Short names

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--color', '-c', help='The name of the color')
parser.add_argument('--verbose', '-v', help='Print more data',
    action='store_true')
args = parser.parse_args()

print(args.color)
print(args.verbose)
```

python argparse_shortname.py -c Blue -v
python argparse_shortname.py -vc Blue

Exercise: Command line parameters

Take the code from the color selector exercise in the files section and change it so
the user can supply the name of the file where the colors are listed using the
`--file filename` option.

If the user supplies an incorrect color name (which is not listed among the accepted colors)
give an error message and stop execution.

Allow the user to supply a flag called `--force` that will
override the color-name-validity checking and will allow any color name.

Exercise: argparse positional and named

Create a script that can accept any number of filenames, the named parameter `--machine` and the flag `--verbose`.
Like this:

```
python ex.py file1 file2 file3 --machine MACHINE --
verbose
```

Exception handling

Hierarchy of calls

```
main()
    some_process()
        for filename in some_list:
            handle_file(filename)
                private_module.deal_with_file(filename)
private_module._helper_function(filename)
public_module.process_file(filename)
                            with open(filename) as fh:
                                pass
```

Handling errors as return values

- Each function that fails returns some error indicator. **None** ? An object that has and attribute “error”?
- None would be bad as that cannot indicate different errors.
- Every called needs to check if the function returned error. If at any point we forget our system might run with hidden failures.

```
main()
   ......
             result = do_something(filename)
             if result:
                 do_something_else(result)
```

```
main()
   ......
             result = do_something(filename)
             do_something_else(result)
```

Handling errors as exceptions

- Only need to explicitely check for it at the level where we know what to do with the problem.
- But: Do we want our pacemaker to stop totally after missing one beat? Probably not. Or better yet: not when it is in production.

```
main()
   try:
   ......
            result = do_something(filename)
            do_something_else(result)
    except Exception:
        # decide what to do
```

A simple exception

When something goes wrong, Python throws (raises) an exception. For example,
trying to divide a number by 0 won't work. If the exception is not
handled, it will end the execution.

In some programming languags we use the expression "throwing an exception" in other languages the expression is "raising an exception".
I use the two expressions interchangeably.

In the next simple example, Python will print the string before the division,
then it will throw an exception, printing it to the standard error that is
the screen by default. Then the script stops working and the string “after” is not printed.

```
def div(a, b):
    print("before")
    print(a/b)
    print("after")

div(1, 0)

# before
# Traceback (most recent call last):
#   File "examples/exceptions/divide_by_zero.py", line 8, in <module>
#     div(1, 0)
#   File "examples/exceptions/divide_by_zero.py", line 5, in div
#     print(a/b)
# ZeroDivisionError: integer division or modulo by zero
```

Working on a list

In a slightly more interesting example we have a list of values. We would like to divide a number by each one of the values.

As you can see one of the values is 0 which will generate and exception.

The loop will finish early.

```
def div(a, b):
    print("dividing {} by {} is {}".format(a, b, a/b))

a = 100
values = [2, 5, 0, 4]

for v in values:
    div(a, v)

# dividing 100 by 2 is 50.0
# dividing 100 by 5 is 20.0
# Traceback (most recent call last):
# ...
# ZeroDivisionError: division by zero
```

We can’t repair the case where the code tries to divide by 0, but it would be nice
if we could get the rest of the results as well.

Catch ZeroDivisionError exception

For that, we’ll wrap the critical part of the code in a “try” block.
After the “try” block we need to provide a list of exception that are
caught by this try-block.

You could say something like “Try this code and let all the exceptions propagate, except of the ones I listed”.

As we saw in the previous example, the specific error is called ZeroDivisionError.

If the specified exception occurs within the try: block, instead of the script ending,
only the try block end and the except: block is executed.

```
def div(a, b):
    print("dividing {} by {} is {}".format(a, b, a/b))

a = 100
values = [2, 5, 0, 4]

for v in values:
    try:
        div(a, v)
    except ZeroDivisionError:
        print("Cannot divide by 0")

# dividing 100 by 2 is 50.0
# dividing 100 by 5 is 20.0
# Cannot divide by 0
# dividing 100 by 4 is 25.0
```

Module to open files and calculate something

Of course in the previous example, it would be probably much easier if we just checked if the number was 0, before trying to divide with it. There are many other cases when this is not possible. For example it is impossible to check if open a file will succeed, without actually trying to open the file.

In this example we open the file, read the first line which is a number and use that for division.

When the open() fails, Python throws an IOError exception.

```
def read_and_divide(filename):
    print("before " + filename)
    with open(filename, 'r') as fh:
        number = int(fh.readline())
        print(100 / number)
    print("after  " + filename)
```

File for exception handling example

If we have a list of files and we would like to make sure we process as many as possible without any problem caused in the middle, we can catch the exception.

We have the following list of files.

Notice that “two.txt” is missing and “zero.txt” has a 0 in it.

```
0
```

```
1
```

File two.txt is missing on purpose.

```
3
```

Open files - exception

```
import sys
import module

# python open_list_of_files.py one.txt zero.txt two.txt three.txt
files = sys.argv[1:]

for filename in files:
    module.read_and_divide(filename)

# before one.txt
# 100.0
# after  one.txt
# before zero.txt
# Traceback (most recent call last):
# ...
# ZeroDivisionError: division by zero
```

Handle divide by zero exception

Running this code will the ZeroDivisionError exception, but it will die with a IOError exception.

```
import sys
import module

# python handle_divide_by_zero.py one.txt zero.txt two.txt three.txt
files = sys.argv[1:]

for filename in files:
    try:
        module.read_and_divide(filename)
    except ZeroDivisionError:
        print("Cannot divide by 0 in file {}".format(filename))


# before one.txt
# 100.0
# after  one.txt
# before zero.txt
# Cannot divide by 0 in file zero.txt
# before two.txt
# IOError: [Errno 2] No such file or directory: 'two.txt'
```

Handle files - exception

We can add multiple “except” statement at the end of the “try” block and handle several exceptions. Each one in a different way.

```
import sys
import module

# python handle_both_exceptions.py one.txt zero.txt two.txt three.txt
files = sys.argv[1:]

for filename in files:
    try:
```

```
        module.read_and_divide(filename)
    except ZeroDivisionError:
        print("Cannot divide by 0 in file {}".format(filename))
    except IOError:
        print("Cannot open file {}".format(filename))


# before one.txt
# 100.0
# after  one.txt
# before zero.txt
# Cannot divide by 0 in file zero.txt
# before two.txt
# Cannot open file two.txt
# before three.txt
# 33.333333333333336
# after  three.txt
```

Catch all the exceptions and show their type

We can also use the “except Exception” to catch all exceptions. In this case we might want to also print out the text and the type of the exception by ourselves.

```
import sys
import module

# python show_exceptions_type.py one.txt zero.txt two.txt three.txt
files = sys.argv[1:]

for filename in files:
    try:
        module.read_and_divide(filename)
    except Exception as err:
        print("  There was a problem in " + filename)
        print("  Text: {}".format(err))
        print("  Name: {}".format(type(err).__name__))
```

```
# before one.txt
# 100.0
# after  one.txt
# before zero.txt
#    There was a problem in zero.txt
#    Text: division by zero
#    Name: ZeroDivisionError
# before two.txt
#    There was a problem in two.txt
#    Text: [Errno 2] No such file or directory: 'two.txt'
#    Name: FileNotFoundError
# before three.txt
# 33.333333333333336
# after  three.txt
```

List exception types

> We can list more than one exceptions to be caught one after the other in a single “except” statement.

```
except (IOError, ZeroDivisionError):
```

```
import sys
import module

# python handle_both_exceptions.py one.txt zero.txt two.txt three.txt
files = sys.argv[1:]

for filename in files:
    try:
        module.read_and_divide(filename)
    except (ZeroDivisionError, IOError):
        print("We have a problem with file {}".format(filename))


```

```
# before one.txt
# 100.0
# after  one.txt
# before zero.txt
# We have a problem with file zero.txt
# before two.txt
# We have a problem with file two.txt
# before three.txt
# 33.333333333333336
# after  three.txt
```

Exceptions

There are many kinds of exceptions in Python and each module can define its own exception types as well.
On this page you'll find the list and hierarchy of exceptions in Python.

- [exceptions]

How to raise an exception

As you create more and more complex applications you'll reach a point where you write a function, probably in a module that needs to report some error condition.
You can raise an exception in a simple way.

```
def some():
    raise Exception("Some Error")

def main():
    try:
```

```
        some()
    except Exception as err:
        print(err)
        print("Type: " + type(err).__name__)

main()

# Some Error
# Type: Exception
```

Stack trace

```python
import traceback

def bar():
    foo()

def foo():
    raise Exception("hi")

def main():
    try:
        bar()
    except Exception as err:
        track = traceback.format_exc()
        print(track)

    print("---------------------")
    bar()


main()
```

```
Traceback (most recent call last):
  File "stack_trace.py", line 11, in main
    bar()
  File "stack_trace.py", line 4, in bar
    foo()
  File "stack_trace.py", line 7, in foo
    raise Exception("hi")
Exception: hi

```

```
---------------------
Traceback (most recent call last):
  File "stack_trace.py", line 20, in <module>
    main()
  File "stack_trace.py", line 17, in main
    bar()
  File "stack_trace.py", line 4, in bar
    foo()
  File "stack_trace.py", line 7, in foo
    raise Exception("hi")
Exception: hi
```

Exercies: Exception int conversion

- In the earlier example we learned how to handle both ZeroDivisionError and IOError exceptions. Now try this

```
cd examples/exceptions
python handle_both_exceptions.py one.txt zero.txt two.txt text.txt three.txt
```

```
before one.txt
100.0
after  one.txt
before zero.txt
Cannot divide by 0 in file zero.txt
before two.txt
Cannot open file two.txt
before text.txt
Traceback (most recent call last):
  File "handle_both_exceptions.py", line 9, in <module>
    module.read_and_divide(filename)
  File "/home/gabor/work/slides/python/examples/exceptions/module.py", line 4, in re\
ad_and_divide
    number = int(fh.readline())
ValueError: invalid literal for int() with base 10: '3.14\n'
```

- This will raise a ValueError exception before handling file three.txt
- Fix it by capturing the spcific exception.
- Fix by capturing “all other exceptions”.

```
3.14
```

Exercies: Raise Exception

- Write a function that expects a positive integer as its single parameter.
- Raise exception if the parameter is not a number.
- Raise a different exception if the parameter is not positive.
- Raise a different exception if the parameter is not whole number.

Solution: Exception int conversion (specific)

```
import sys
import module

# python handle_both_exceptions.py one.txt zero.txt two.txt three.txt
files = sys.argv[1:]

for filename in files:
    try:
        module.read_and_divide(filename)
    except ZeroDivisionError:
        print("Cannot divide by 0 in file {}".format(filename))
    except IOError:
        print("Cannot open file {}".format(filename))
    except ValueError as ex:
        print("ValueError {} in file {}".format(ex, filename))
```

```
before one.txt
100.0
after  one.txt
before zero.txt
Cannot divide by 0 in file zero.txt
before two.txt
Cannot open file two.txt
before text.txt
ValueError invalid literal for int() with base 10: '3.14\n' in file text.txt
before three.txt
33.333333333333336
after  three.txt
```

Solution: Exception int conversion (all other)

```python
import sys
import module

# python handle_both_exceptions.py one.txt zero.txt two.txt three.txt
files = sys.argv[1:]

for filename in files:
    try:
        module.read_and_divide(filename)
    except ZeroDivisionError:
        print("Cannot divide by 0 in file {}".format(filename))
    except IOError:
        print("Cannot open file {}".format(filename))
    except Exception as ex:
        print("Exception type {} {} in file {}".format(type(ex).__name__, ex, filena\
me))
```

```
before one.txt
100.0
after  one.txt
before zero.txt
Cannot divide by 0 in file zero.txt
before two.txt
```

```
Cannot open file two.txt
before text.txt
Exception type ValueError invalid literal for int() with
base 10: '3.14\n' in file t\
ext.txt
before three.txt
33.333333333333336
after  three.txt
```

Solution: Raise Exception

```
def positive(num):
    if type(num).__name__ == 'float':
        raise Exception("The given parameter {} is a float
and not an int.".format(nu\
m))

    if type(num).__name__ != 'int':
        raise Exception("The given parameter {} is of type
{} and not int.".format(nu\
m, type(num).__name__))

    if num < 0:
        raise Exception("The given number {} is not
positive.".format(num))

for val in [14, 24.3, "hi", -10]:
    print(val)
    print(type(val).__name__)
    try:
        positive(val)
    except Exception as ex:
        print("Exception: {}".format(ex))
```

Classes - OOP - Object Oriented Programming

Why Object Oriented Programming?

- Better encapsulation of intent.
- Integration between data and functionality (attributes and methods)
- Better modelling for some part of the world.
- Another level of code-reuse.
- Clearer separation between “usage” and “implementation”. (Private data in some cases)
- Clearer connection between “classes” of things.
- In reality: avoid using “global”.

Generic Object Oriented Programming terms

- OOP differs a lot among programming languages!
- Classes (blueprints)
- Objectes / instances (actual)
- Members: Attributes and Methods
- Attributes / Properties (variables - data)
- Methods (functions) (private, public, virtual)
- Inheritance (is a)
- Composition (has a)
- Constructor
- Destructor

OOP in Python

- Everything is an object
- Numbers, strings, list, … even classes are objects.
- Class objects
- Instance objects
- Nothing is private.

OOP in Python (numbers, strings, lists)

```
# numbers
print((255).bit_length())     # 8
print((256).bit_length())     # 9

# strings
print( "hello WOrld".capitalize() )   # Hello world
print( ":".join(["a", "b", "c"]) )    # a:b:c


# lists
numbers = [2, 17, 4]
print(numbers)           # [2, 17, 4]

numbers.append(7)
print(numbers)           # [2, 17, 4, 7]

numbers.sort()
print(numbers)           # [2, 4, 7, 17]
```

OOP in Python (argparse)

```
import argparse
def get_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--name')
    parser.add_argument('--email')

    print(type(parser).__name__)
    print(parser.__class__)
```

```python

    # print(dir(parser))
    print( parser.format_help() )
    parser.print_help()

    return parser.parse_args()

args = get_args()
print(args.__class__)
print(args.name)
```

Create a class

```python
# class Person(object):
#    pass

class Person:
    pass

if __name__ == '__main__':
    p = Person()
    print(p)
    print(type(p))
    print(p.__class__.__name__)

    members = dir(p)
    print(members)
```

```
<__main__.Person object at 0x7fc4e3ec1da0>
<class '__main__.Person'>
Person
['__class__', '__delattr__', '__dict__', '__dir__',
'__doc__', '__eq__', '__format__\
', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__init_subclass_\
_', '__le__', '__lt__', '__module__', '__ne__',
'__new__', '__reduce__', '__reduce_e\
x__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__wea\
kref__']
```

In Python 2.x classes needed to inherit from ‘object’ in order to become ‘new style’ classes.

Import module containing class

```python
import ppl

p = ppl.Person()
print(p)                      # <person.Person object at 0x101a8a190>
print(type(p))                # <class 'person.Person'>
print(p.__class__.__name__)   # Person
```

```
<ppl.Person object at 0x7f973024a780>
<class 'ppl.Person'>
Person
```

Import class from module

```python
from ppl import Person

p = Person()
print(p)                      # <person.Person object at 0x101a8a190>
print(type(p))                # <class 'person.Person'>
print(p.__class__.__name__)   # Person
```

Initialize a class - constructor, attributes

```python
class Person():
    def __init__(self, given_name):
        self.name = given_name

if __name__ == '__main__':
    p1 = Person("Joe")
    print(p1)                      # <__main__.Person object at 0x0000021EC664B358>
    print(p1.__class__.__name__)   # Person
```

```
    print(p1.name)                # Joe

    p2 = Person("Jane")
    print(p2)                     # <__main__.Person object at 0x0000021EC664B470>
    print(p2.name)                # Jane

    p1.name = "Joseph"
    print(p1)                     # <__main__.Person object at 0x0000021EC664B358>
    print(p1.name)                # Josheph
```

Attributes are not special

```
class Person():
    def __init__(self, given_name):
        self.name = given_name

if __name__ == '__main__':
    p1 = Person("Joe")
    print(p1.__class__.__name__)  # Person
    print(p1.name)                # Joe

    p2 = Person("Jane")
    print(p2.name)                # Jane

    p1.address = "Main street 12"
    print(p1.address)             # Main street 12


    print(p2.address)             # AttributeError: 'Person' object has no attribute\
 'address'
```

Create Point class

```
import shapes

p = shapes.Point()
print(p)            # <shapes.Point instance at 0x7fb58c31ccb0>
```

```python
class Point():
    pass
```

Initialize a class - constructor, attributes

```python
import shapes

p1 = shapes.Point(2, 3)
print(p1)               # <shapes.Point instance at 0x7fb58c31ccb0>
print(p1.x)             # 2
print(p1.y)             # 3

p1.x = 7
print(p1.x)             # 7
```

```python
class Point():
    def __init__(self, a, b):
        self.x = a
        self.y = b
```

Methods

```python
import shapes

p1 = shapes.Point(2, 3)

print(p1.x)      # 2
print(p1.y)      # 3

p1.move(4, 5)
print(p1.x)      # 6
print(p1.y)      # 8


print(p1)        # <shapes.Point object at 0x7fb0691c3e48>
```

```python
class Point():
    def __init__(self, a, b):
```

```
        self.x = a
        self.y = b

    def move(self, dx, dy):
        self.x += dx
        self.y += dy
```

Stringify class

- **repr** “should” return Python-like code
- **str** should return readable representation
- If **str** does not exist, **repr** is called instead.

```
import shapes

p1 = shapes.Point(2, 3)
print(p1)     # Point(2, 3)
```

```
class Point():
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return 'Point({}, {})'.format(self.x, self.y)

    def move(self, dx, dy):
        self.x += dx
        self.y += dy
```

Inheritance

```
class Point():
    def __init__(self, x, y):
        print('__init__ of Point')
        self.x = x
        self.y = y

    def move(self, dx, dy):
```

```python
        self.x += dx
        self.y += dy

class Circle(Point):
    def __init__(self, x, y, r):
        print('__init__ of Circle')
        super().__init__(x, y)
        self.r = r

    def area(self):
        return self.r * self.r * 3.14
```

```python
import shapes

c = shapes.Circle(2, 3, 10)    # __init__ of Circle
                               # __init__ of Point
print(c)            # <shapes.Circle instance at 0x7fb58c31ccb0>
print(c.x)          # 2
print(c.y)          # 3
print(c.r)          # 10

c.move(4, 5)
print(c.x)          # 6
print(c.y)          # 8
print(c.area())     # 314.0
```

Inheritance - another level

```python
class Point():
    def __init__(self, x, y):
        print('__init__ of Point')
        self.x = x
        self.y = y

class Circle(Point):
    def __init__(self, x, y, r):
        print('__init__ of Circle')
        super().__init__(x, y)
        self.r = r

    def area(self):
```

```python
        return self.r * self.r * 3.14

class Ball(Circle):
    def __init__(self, x, y, r, z):
        print('__init__ of Ball')
        super().__init__(x, y, r)
        self.z = z


b = Ball(2, 3, 9, 7)
print(b)
print(b.area())

# __init__ of Ball
# __init__ of Circle
# __init__ of Point
# <__main__.Ball object at 0x103dea190>
# 254.34
```

Modes of method inheritance

- Implicit
- Override
- Extend
- Delegate - Provide

Modes of method inheritance - implicit

Inherit method

```python
class Parent():
    def greet(self):
        print("Hello World")

class Child(Parent):
    pass

p = Parent()
p.greet()     # Hello World

```

```
c   = Child()
c.greet()     # Hello World
```

Modes of method inheritance - override

Replace method

```
class Parent():
    def greet(self):
        print("Hello World")

class Child(Parent):
    def greet(self):
        print("Hi five!")

p = Parent()
p.greet()

c   = Child()
c.greet()

super(Child, c).greet()
```

```
Hello World
Hi five!
Hello World
```

Modes of method inheritance - extend

Extend method before or after calling original.

```
class Parent():
    def greet(self):
        print("Hello World")

class Child(Parent):
    def greet(self):
        print("Hi five!")
        super().greet()
        print("This is my world!")
```

```

p = Parent()
p.greet()     # Hello World

c  = Child()
c.greet()

# Hi five!
# Hello World
# This is my world!
```

Modes of method inheritance - delegate - provide

Let the child implement the functionality.

```
class Parent():
    def greet(self):
        print("Hello", self.get_name())

class Child(Parent):
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.name

# Should not create instance from Parent
# p = Parent()
# p.greet()     # AttributeError: 'Parent' object has no attribute 'get_name'

c  = Child('Foo')
c.greet()     # Hello Foo
```

- Should we have a version of greet() in the Parent that throws an exception?
- Do we want to allow the creation of instance of the Parent class?

- Abstract Base Class (abc)

Composition - Line

When an object holds references to one or more other objects.

- Pythagorean theorem

```
class Point():
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Line():
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def length(self):
        return ((self.a.x - self.b.x) ** 2 + (self.a.y - self.b.y) ** 2) ** 0.5

p1 = Point(2, 3)
p2 = Point(5, 7)
blue_line = Line(p1, p2)

print(blue_line.a) # <__main__.Point object at 0x0000022174B637B8>
print(blue_line.b) # <__main__.Point object at 0x0000022174B3C7B8>
print(blue_line.length())   # 5.0
```

Some comments

- There are no private attributes. The convention is to use leading underscore to communicate to other developers what is private.
- Using the name **self** for the current object is just a consensus.

Class in function

```
def creator():
    class MyClass():
        pass
    o = MyClass()
    print(o.__class__.__name__) # MyClass

creator()
# MyClass()   # NameError: name 'MyClass' is not defined
```

Serialization of instances with pickle

```
import pickle

class aClass(object):
    def __init__(self, amount, name):
        self.amount = amount
        self.name = name


the_instance = aClass(42, "FooBar")

a = {
    "name": "Some Name",
    "address" : ['country', 'city', 'street'],
    'repr' : the_instance,
}

print(a)

pickle_string = pickle.dumps(a)

b = pickle.loads(pickle_string)

print(b)

print(b['repr'].amount)
print(b['repr'].name)
```

Quick Class definition and usage

```python
class Quick(object):
    def __init__(self, name, email):
        self.name = name
        self.email = email

q = Quick(name = "Foo", email = "foo@bar.com")
print(q.name)
print(q.email)
```

Exercise: Add move_rad to based on radians

- From the **Python: Methods** take the examples/classes/methods/shapes.py and add a method called **move_rad(dist, angle)** that accpets a distance and an angle and moved the point accordingly.

```python
delta_x = dist * cos(angle)
delta_y = dist * sin(angle)
```

Exercise: Improve previous examples

- Take the previous example **Python: Inheritance - another level** and the example file called examples/classes/inheritance/ball_shape.py and change it so the **Ball** class will accept **x, y, z, r**.
- Add a method called move to the new Ball class that will accept dx, dy, dz.
- Implement a method that will return the volume of the ball.

Exercise: Polygon

- Implement a class representing a Point.
- Make the printing of a point instance nice.

- Implement a class representing a Polygon. (A list of points)
- Allow the user to "move a polygon" calling poly.move(dx, dy) that will change the coordinates of every point by (dx, dy)

```
class Point():
    pass

class Polygon():
    pass

p1 = Point(0, 0)   # Point(0, 0)
p2 = Point(5, 7)   # Point(5, 7)
p3 = Point(4, 9)   # Point(4, 9)
print(p1)
print(p2)
print(p3)
p1.move(2, 3)
print(p1)          # Point(2, 3)

poly = Polygon(p1, p2, p3)
print(poly)        # Polygon(Point(2, 3), Point(5, 7), Point(4, 9))
poly.move(-1, 1)
print(poly)        # Polygon(Point(1, 4), Point(4, 8), Point(3, 10))
```

Exercise: Number

Turn the Number guessing game into a class. Replace every print statement with a call to an output method.
Do the same with the way you get the input.
Then create a subclass where you override these methods.
You will be able to launch the game with a hidden value you decide upon.
The input will feed a pre-defined list of values as guesses to the game
and the output methods will collect the values that the game prints in a list.

Exercise: Library

Create a class hierarchy to represent a library that will be able to represent the following entities.

- Author (name, birthdate, books)
- Book (title, author, language, who_has_it_now?, is_on_waiting_list_for_whom?)
- Reader (name, birthdate, books_currently_lending)

Methods:

- write_book(title, language,)

Exercise: Bookexchange

It is like the library example, but instead of having a central library with books,
each person owns and lends out books to other people.

Exercise: Represent turtle graphics

There is a cursor (or turtle) in the x-y two-dimensional sphere. It has some (x,y) coordinates.
It can go forward n pixels. It can turn left n degrees. It can lift up the pencil or put it down.

Solution - Polygon

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
```

```
        return "Point({}, {})".format(self.x, self.y)

    def move(self, dx, dy):
        self.x += dx
        self.y += dy

class Polygon:
    def __init__(self, *args):
        self.points = args

    def __repr__(self):
        return 'Polygon(' + ', '.join(map(lambda p: str(p), self.points)) + ')'

    def move(self, dx, dy):
        for p in self.points:
            p.move(dx, dy)

p1 = Point(0, 0)   # Point(0, 0)
p2 = Point(5, 7)   # Point(5, 7)
p3 = Point(4, 9)   # Point(4, 9)
print(p1)
print(p2)
print(p3)
p1.move(2, 3)
print(p1)          # Point(2, 3)

poly = Polygon(p1, p2, p3)
print(poly)        # Polygon(Point(2, 3), Point(5, 7), Point(4, 9))
poly.move(-1, 1)
print(poly)        # Polygon(Point(1, 4), Point(4, 8), Point(3, 10))
```

PyPi - Python Package Index

What is PyPi?

- pypi

Easy Install

- setuptools

```
$ easy_install module_name
```

pip

```
$ pip install package_name
```

Upgrade pip

- **pip install –upgrade pip** Will probably not work on Windows because file is in use…
- **easy_install pip** Will work on Windows as well.

PYTHONPATH

```
export PYTHONPATH=~/python
easy_install -d ~/python Genshi
```

Virtualenv

```
$ pip install virtualenv

$ cd project_dir
$ virtualenv venv
$ source venv/bin/activate
$ ...
$ deactivate
```

On Windows:

```
venv\Source\activate.bat
```

The **virtualenv** command will create a copy of python in the given directory inside the current directory.
In the above example it will create the copy in the 'venv' directory inside the 'project_dir'.
After source-ing the 'activate' file the PATH will include the local python with a local version of **pip**
and **easy_install**. This requires bash or zsh.

See also the Python guide.

Virtualenv for Python 3

```
virtualenv -p python3 venv3
source venv3/bin/activate
...
deactivate
```

SQLite Database Access

SQLite

- sqlite3

Connecting to SQLite database

```python
import sqlite3

conn = sqlite3.connect("sample.db")
c = conn.cursor()

# use the database here

conn.close()
```

Create TABLE in SQLite

execute and commit

```python
import sqlite3

conn = sqlite3.connect("sample.db")
c = conn.cursor()

try:
    c.execute('''CREATE TABLE companies (
        id PRIMARY KEY,
        name VARCRCHAR(100) UNIQUE NOT NULL,
        employees INTEGER DEFAULT 0)''')
except sqlite3.OperationalError as e:
    print('sqlite error:', e.args[0])  # table companies already exists

```

```
conn.commit()

conn.close()

print('done')
```

INSERT data into SQLite database

Use placeholders (?) supply the data in tuples.

```
import sqlite3

conn = sqlite3.connect("sample.db")
c = conn.cursor()

my_company = 'Acme'

try:
  c.execute('''INSERT INTO companies (name) VALUES (?)''', (my_company,))
except sqlite3.IntegrityError as e:
  print('sqlite error: ', e.args[0]) # column name is not unique
conn.commit()

companies = [
  ('Foo', 12),
  ('Bar', 7),
  ('Moo', 99),
]

try:
  sql = '''INSERT INTO companies (name, employees) VALUES (?, ?)'''
  c.executemany(sql, companies)
except sqlite3.IntegrityError as e:
  print('sqlite error: ', e.args[0]) # column name is not unique
conn.commit()


conn.close()
```

```
print('done')
```

UPDATE works quite similar, but it might have a WHERE clause.

SELECT data from SQLite database

```
import sqlite3

conn = sqlite3.connect("sample.db")
c = conn.cursor()

minimum = 0

sql = '''SELECT * FROM companies WHERE employees >= ?'''
for company in c.execute(sql, (minimum,)):
  print(company)

sql = '''SELECT COUNT(*) FROM companies WHERE employees >= ?'''
c.execute(sql, (minimum,))
print(c.fetchone()[0])

conn.close()
```

Use the result as an iterator, or call the fetchone method. If the result set might be empty,
then the fetchone might return None. Check for it!

A counter

```
"""
    Counter using an SQLite backend
    --list              list all the counters
    --start name       creates the counter for 'name'
    name               counts for 'name'
"""

import sys
```

```python
import os
import sqlite3

database_file = "counter.db"

def usage():
    print('TODO print doc')
    conn.close()
    exit()

def main():
    global conn
    conn = sqlite3.connect(database_file)
    c = conn.cursor()
    try:
        c.execute('''CREATE TABLE counters (
          id PRIMARY KEY,
          name VARCRCHAR(100) UNIQUE NOT NULL,
          count INTEGER NOT NULL
          )''')
    except sqlite3.OperationalError as e:
        pass
        # print('sqlite error:', e.args[0])  # table counters already exists

    # print(len(sys.argv))
    # print(sys.argv)

    if len(sys.argv) == 1:
        usage()

    if len(sys.argv) == 2:
        if sys.argv[1] == '--list':
            print('List counters:')
            for r in c.execute("SELECT name FROM counters"):
                print(r[0])
            exit()
        name = sys.argv[1]
        c.execute("SELECT count FROM counters WHERE name = ?", (name,))
        line = c.fetchone()
        if line == None:
            print("Invalid counter name '{}'".format(name))
```

```
            exit()
        value = line[0]
        value = value +1
        c.execute("UPDATE counters SET count=? WHERE name = ?", (value, name))
        conn.commit()
        print("{} {}".format(name, value))
        #print("increment counter {} was: {}".format(name, value))
        exit()

    if len(sys.argv) == 3 and sys.argv[1] == '--start':
        name = sys.argv[2]
        print("Start counter", name)
        try:
            c.execute("INSERT INTO counters (name, count) VALUES(?,?)", (name, 0))
            conn.commit()
        except sqlite3.IntegrityError:
            print("Name '{}' already exists".format(name))
            exit()

        exit()

    print('none')
    usage()

main()

#print "TODO get the value of 'name' from the database"
# if it was not there then add


#try:
#   c.execute('''INSERT INTO companies (name) VALUES ('Stonehenge')''')
#except sqlite3.IntegrityError as e:
#   print 'sqlite error: ', e.args[0] # column name is not unique

#conn.commit()

#conn.close()
```

```
#print "done"
```

MySQL

Install MySQL support

- Anaconda on MS Windows: **conda install mysql-connector-python**
- Otherwise: **pip install mysql-connector**

Create database user (manually)

```
$ mysql -u root -p

  SHOW DATABASES;

  CREATE USER 'foobar'@'localhost' IDENTIFIED BY 'no secret';
  GRANT ALL PRIVILEGES ON fb_db . * TO 'foobar'@'localhost';
  GRANT ALL PRIVILEGES ON * . * TO 'foobar'@'%' IDENTIFIED BY 'no secret';
  FLUSH PRIVILEGES;

  exit
```

```
vim /etc/mysql/mysql.conf.d/mysqld.cnf
comment out
# bind-address          = 127.0.0.1

service mysql restart
```

Create database (manually)

```
$ mysql -u foobar -p

```

```
 CREATE DATABASE fb_db;

 DROP DATABASE fb_db;
 exit
```

Create table (manually)

```
$ mysql -u foobar -p

 USE fb_db;
 CREATE TABLE person (
   id INTEGER PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(255),
   birthdate DATE,
   score REAL
 );

 INSERT INTO person (name, birthdate, score)
     VALUES ("Foo Bar", "1998-05-23", 42.1)
```

Connect to MySQL

```python
import mysql.connector

def main():
    conn = mysql.connector.connect(
        host = 'localhost',
        database = 'fb_db',
        user = 'foobar',
        password='no secret')

    print("Connected:", conn)

    conn.close()

if __name__ == "__main__":
    main()
```

```
$ python3 examples/mysql/connect.py
```

- Change some of the parameters and try again

Connect to MySQL and Handle exception

```
import mysql.connector

def main():
    try:
        conn = mysql.connector.connect(
            host = 'localhost',
            database = 'fb_db',
            user = 'foobar',
            password='no secret')
    except mysql.connector.Error as e:
        print("MySQL exception: ", e)
        return
    #except Exception as e:
    #    print("Other exception", e);
    #    return

    print("Connected:", conn)

    conn.close()

if __name__ == "__main__":
    main()
```

Select data

```
import mysql.connector


def main():
    conn = mysql.connector.connect(
        host = 'localhost',
        database = 'fb_db',
        user = 'foobar',
        password='no secret')

    cursor = conn.cursor()
    cursor.execute("SELECT * FROM person")

```

```python
    row = cursor.fetchone()
    print(row)

    # cursor.close()   # mysql.connector.errors.InternalError: Unread result found.
    conn.close()

if __name__ == "__main__":
    main()
```

Select more data

```python
import mysql.connector


def main():
    conn = mysql.connector.connect(
        host = 'localhost',
        database = 'fb_db',
        user = 'foobar',
        password='no secret')

    cursor = conn.cursor()
    cursor.execute("SELECT * FROM person")

    while True:
        row = cursor.fetchone()
        if not row:
            break
        print(row)

    cursor.close()
    conn.close()

if __name__ == "__main__":
    main()
```

Select all data fetchall

```python
import mysql.connector

```

```python

def main():
    conn = mysql.connector.connect(
        host = 'localhost',
        database = 'fb_db',
        user = 'foobar',
        password='no secret')

    cursor = conn.cursor()
    cursor.execute("SELECT * FROM person")

    rows = cursor.fetchall()

    print(len(rows))
    for row in rows:
        print(row)

    cursor.close()
    conn.close()

if __name__ == "__main__":
    main()
```

Select some data fetchmany

```python
import mysql.connector


def main():
    conn = mysql.connector.connect(
        host = 'localhost',
        database = 'fb_db',
        user = 'foobar',
        password='no secret')

    cursor = conn.cursor()
    cursor.execute("SELECT * FROM person")

    size = 2

    while True:
        rows = cursor.fetchmany(size)
        if not rows:
```

```
            break
        print(len(rows))
        for row in rows:
            print(row)

    cursor.close()
    conn.close()

if __name__ == "__main__":
    main()
```

Select some data WHERE clause

[Bobby Tables]

```
import mysql.connector


def main(min_score):
    conn = mysql.connector.connect(
        host = 'localhost',
        database = 'fb_db',
        user = 'foobar',
        password='no secret')

    cursor = conn.cursor()
    cursor.execute("SELECT * FROM person WHERE score > %s", (min_score,))

    size = 2

    while True:
        rows = cursor.fetchmany(size)
        if not rows:
            break
        print(len(rows))
        for row in rows:
            print(row)

    cursor.close()
    conn.close()

```

```python
if __name__ == "__main__":
    main(40)
```

Select into dictionaries

```python
import mysql.connector


def main():
    conn = mysql.connector.connect(
        host = 'localhost',
        database = 'fb_db',
        user = 'foobar',
        password='no secret')

    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM person")

    for row in cursor:
        print(row)

    cursor.close()
    conn.close()

if __name__ == "__main__":
    main()
```

Insert data

```python
import mysql.connector


def main(name, birthdate, score):
    conn = mysql.connector.connect(
        host = 'localhost',
        database = 'fb_db',
        user = 'foobar',
        password='no secret')

    cursor = conn.cursor()
    cursor.execute(
```

```
        "INSERT INTO person (name, birthdate, score) VALUES (%s, %s, %s)",
        (name, birthdate, score))

    if cursor.lastrowid:
        print('last insert id', cursor.lastrowid)
    else:
        print('last insert id not found')
    conn.commit()

    conn.close()

if __name__ == "__main__":
    main('Monty Python', '1969-10-05', 100)
```

Update data

```
import mysql.connector


def main(uid, score):
    conn = mysql.connector.connect(
        host = 'localhost',
        database = 'fb_db',
        user = 'foobar',
        password='no secret')

    cursor = conn.cursor()
    cursor.execute("UPDATE person SET score=%s WHERE id=%s",
        (score, uid))
    conn.commit()

    conn.close()

if __name__ == "__main__":
    main(12, 32)
```

Delete data

```
import mysql.connector


def main(uid):
    conn = mysql.connector.connect(
        host = 'localhost',
        database = 'fb_db',
        user = 'foobar',
        password='no secret')

    cursor = conn.cursor()
    cursor.execute("DELETE FROM person WHERE id=%s", (uid,))
    conn.commit()

    conn.close()

if __name__ == "__main__":
    main(11)
```

Exercise MySQL

1. Create a user with a password manually.
2. Create a database manually.
3. Create a table manually for describing fleet of cars: id, license-plate, year-built, brand, owner. (Owner is the name of the owner)
4. Create a program that accepts values on the command line and insterts the data in the database
5. Create another program that lists all the cars.
6. Improve the selector program to accept command line paramter –minage N and –maxage N and show the cars within those age limits (N is a number of years e.g. 3)
7. Create program to delete a car.
8. Create program to change the owner of a car.

Exercise: MySQL Connection

Instead of hard-coding the connection details in the script, let’s create an INI file that contains the connection information and use that.

```
[development]
host     = localhost
database = fb_db
user     = foobar
password = no secret
```

Solution: MySQL Connection

```python
import configparser
import mysql.connector

config_file = 'examples/mysql/connect.ini'

def read_config(section = 'development'):
    print(section)
    cp = configparser.ConfigParser()
    cp.read(config_file)
    if not cp.has_section(section):
        raise Exception("No configuration found for '{}'".format(section))

    return cp[section]

def main():
    try:
        db = read_config()
        print(db['password'])
        print(db)
        conn = mysql.connector.connect(**db)
    except mysql.connector.Error as e:
        print("MySQL exception: ", e)
        return
    except Exception as e:
        print("Other exception", e);
        return
```

```python

    if conn.is_connected():
        print("is connected")
    print("Connected:", conn)

    conn.close()

if __name__ == "__main__":
    main()
```

PostgreSQL

PostgreSQL install

```
$ sudo aptitude install postgresql

$ sudo -i -u postgres
$ createuser --interactive
  Add "ubuntu" as superuser   (we need a username that matches our Linux username)
$ createdb testdb

$ psql
$ sudo -u postgres psql

$ psql testdb
testdb=# CREATE TABLE people (id INTEGER PRIMARY KEY, name VARCHAR(100));
```

Python and Postgresql

```
$ sudo aptitude install python3-postgresql
$ sudo aptitude install python3-psycopg2
```

PostgreSQL connect

```
import psycopg2

try:
    conn = psycopg2.connect("postgresql:///testdb")
    #conn = psycopg2.connect("dbname='testdb' user='ubuntu' host='localhost' passwor\
d='secret'")
except Exception as e:
    print("I am unable to connect to the database: ", e)
```

INSERT

```python
import psycopg2

try:
    conn = psycopg2.connect("postgresql:///testdb")
except Exception as e:
    print("I am unable to connect to the database: ", e)

cur = conn.cursor()

uid = 1
name = 'Foo'

try:
    cur.execute("INSERT INTO people (id, name) VALUES (%s, %s)", (uid, name))
    conn.commit()
except Exception as e:
    print(e)
```

```
duplicate key value violates unique constraint "people_pkey"
DETAIL:  Key (id)=(1) already exists.
```

INSERT (from command line)

```python
import psycopg2
import sys

if len(sys.argv) != 3:
    exit("Usage: {} ID NAME".format(sys.argv[0]))

uid, name = sys.argv[1:]


try:
    conn = psycopg2.connect("postgresql:///testdb")
except Exception as e:
    print("I am unable to connect to the database: ", e)

```

```python
cur = conn.cursor()

try:
    cur.execute("INSERT INTO people (id, name) VALUES (%s, %s)", (uid, name))
    conn.commit()
except Exception as e:
    print(e)
```

SELECT

```python
import psycopg2

try:
    conn = psycopg2.connect("postgresql:///testdb")
except Exception as e:
    print("I am unable to connect to the database: ", e)

cur = conn.cursor()

try:
    cur.execute("SELECT * from people")
    for r in cur.fetchall():
        print(r)
except Exception as e:
    print(e)
```

DELETE

```python
import psycopg2

try:
    conn = psycopg2.connect("postgresql:///testdb")
except Exception as e:
    print("I am unable to connect to the database: ", e)

cur = conn.cursor()

try:
    cur.execute("DELETE FROM people")
    conn.commit()
```

```python
except Exception as e:
    print(e)

try:
    cur.execute("SELECT * from people")
    for r in cur.fetchall():
        print(r)
except Exception as e:
    print(e)
```

SQLAlchemy

SQLAlchemy hierarchy

- ORM
- Table, Metadata, Reflection, DDL - standardized language
- Engine - standardize low-level access (placeholders)

SQLAlchemy engine

```
engine = create_engine('sqlite:///test.db') # relative path
engine = create_engine('sqlite:////full/path/to/test.db') # full path
engine = create_engine('sqlite://') # in memory database
```

PostgreSQL

```
engine = create_engine('postgresql://user:password@hostname/dbname')
engine = create_engine('postgresql+psycopg2://user:password@hostname/dbname')
```

MySQL

```
engine = create_engine("mysql://user:password@hostname/dbname", encoding='latin1') #\
 defaults to utf-8
```

SQLAlchemy autocommit

Unlike the underlying database engines, SQLAlchemy uses autocommit.
That is, usually we don’t need to call `commit()`, but if we would like to have a transaction we need to
start it using `begin()` and end it either with `commit()` or with `rollback()`.

SQLAlchemy engine CREATE TABLE

```python
import os
from sqlalchemy import create_engine

dbname = 'test.db'
if os.path.exists(dbname):
    os.unlink(dbname)

engine = create_engine('sqlite:///' + dbname)   # Engine

engine.execute('''
    CREATE TABLE person (
        id INTEGER PRIMARY KEY,
        name VARCHAR(100) UNIQUE,
        balance INTEGER NOT NULL
    );
''')
```

SQLAlchemy engine INSERT

```python
import os
from sqlalchemy import create_engine

dbname = 'test.db'

engine = create_engine('sqlite:///' + dbname)

engine.execute('INSERT INTO person (name, balance) VALUES (:name, :balance)', name =\
```

```
 'Joe', balance = 100)
engine.execute('INSERT INTO person (name, balance) VALUES (:name, :balance)', name =\
 'Jane', balance = 100)
engine.execute('INSERT INTO person (name, balance) VALUES (:name, :balance)', name =\
 'Melinda', balance = 100)
engine.execute('INSERT INTO person (name, balance) VALUES (:name, :balance)', name =\
 'George', balance = 100)
```

SQLAlchemy engine SELECT

```python
from sqlalchemy import create_engine

dbname = 'test.db'

engine = create_engine('sqlite:///' + dbname)
result = engine.execute('SELECT * FROM person WHERE id=:id', id=3)

print(result)             # <sqlalchemy.engine.result.ResultProxy object at 0x1013c9d\
a0>

row = result.fetchone()
print(row)                # (3, 'Melinda', 100)  - Its a tuple
print(row['name'])        # Melinda              - And a dictionary
print(row.name)           # Melinda   - and object with methods for the columns

for k in row.keys():      # keys also works on it
    print(k)              # id, name, balance

result.close()
```

SQLAlchemy engine SELECT all

```
import os
from sqlalchemy import create_engine

dbname = 'test.db'

engine = create_engine('sqlite:///' + dbname)
result = engine.execute('SELECT * FROM person')

for row in result:
    print(row)

result.close()

# (1, 'Joe', 100)
# (2, 'Jane', 100)
# (3, 'Melinda', 100)
# (4, 'George', 100)
```

SQLAlchemy engine SELECT fetchall

```
from sqlalchemy import create_engine

dbname = 'test.db'

engine = create_engine('sqlite:///' + dbname)
result = engine.execute('SELECT * FROM person WHERE id >= :id', id=3)

rows = result.fetchall()
print(rows)         # [(3, 'Melinda', 100), (4, 'George', 100)]

result.close()
```

SQLAlchemy engine SELECT aggregate

```
from sqlalchemy import create_engine

dbname = 'test.db'

engine = create_engine('sqlite:///' + dbname)
```

```
result = engine.execute('SELECT COUNT(*) FROM person')

r = result.fetchone()[0]
print(r)

result.close()
```

SQLAlchemy engine SELECT IN

```
from sqlalchemy import create_engine

dbname = 'test.db'

engine = create_engine('sqlite:///' + dbname)

results = engine.execute("SELECT * FROM person WHERE name IN ('Joe', 'Jane')")
print(results.fetchall()) # [(2, 'Jane', 100), (1, 'Joe', 100)]

# engine.execute("SELECT * FROM person WHERE name IN (:a0, :a1)", a0 = 'Joe', a1 = '\
Jane')
```

SQLAlchemy engine SELECT IN with placeholders

```
from sqlalchemy import create_engine

dbname = 'test.db'

engine = create_engine('sqlite:///' + dbname)


names = ['Joe', 'Jane']
placeholders = []
data = {}
for i in range(len(names)):
    placeholders.append(':a' + str(i))
    data['a' + str(i)] = names[i]

```

```
# print(placeholders)   # [':a0', ':a1']
# print(data)           # {'a0': 'Joe', 'a1': 'Jane'}

sql = "SELECT * FROM person WHERE name IN ({})".format(', '.join(placeholders))
# print(sql)   #   SELECT * FROM person WHERE name IN (:a0, :a1)

#results = engine.execute(sql, a0 = 'Jane', a1 = 'Joe')
results = engine.execute(sql, **data)
print(results.fetchall()) # [(2, 'Jane', 100), (1, 'Joe', 100)]
```

SQLAlchemy engine connection

```
from sqlalchemy import create_engine

dbname = 'test.db'

engine = create_engine('sqlite:///' + dbname)

conn = engine.connect()
results = conn.execute('SELECT balance, name FROM person WHERE id < :id', id = 3)
print(results.fetchall())    # [(100, 'Joe'), (100, 'Jane')]
conn.close()
```

SQLAlchemy engine transaction

```
from sqlalchemy import create_engine

dbname = 'test.db'

engine = create_engine('sqlite:///' + dbname)

conn = engine.connect()

trans = conn.begin()

src = 'Joe'
```

```
dst = 'Jane'
payment = 3

results = conn.execute("SELECT balance, name FROM person WHERE name = :name", name =\
 src)
src_balance = results.fetchone()[0]
results.fetchall()
print(src_balance)


results = conn.execute("SELECT balance, name FROM person WHERE name = :name", name =\
 dst)
dst_balance = results.fetchone()[0]
results.fetchall()
print(dst_balance)

conn.execute('UPDATE person SET balance = :balance WHERE name=:name', balance = src_\
balance - payment, name = src)
conn.execute('UPDATE person SET balance = :balance WHERE name=:name', balance = dst_\
balance + payment, name = dst)

trans.commit()

# trans.rollback()

conn.close()

results = engine.execute("SELECT * FROM person")
print(results.fetchall())
```

SQLAlchemy engine using context managers

```
with engine.begin() as trans:
    conn.execute(...)
    conn.execute(...)
    raise Exception()  # for rollback
```

Exercise: Create table

Create the following schema

```
CREATE TABLE node (
    id      INTEGER PRIMARY KEY,
    name    VARCHAR(100)
);

CREATE TABLE interface (
    id        INTEGER PRIMARY KEY,
    node_id   INTEGER NOT NULL,
    ipv4      VARCHAR(15) UNIQUE,
    ipv6      VARCHAR(80) UNIQUE,
    FOREIGN KEY (node_id) REFERENCES node(id)
);

CREATE TABLE connection (
    a         INTEGER NOT NULL,
    b         INTEGER NOT NULL,
    FOREIGN KEY (a) REFERENCES interface(id),
    FOREIGN KEY (b) REFERENCES interface(id)
);
```

Insert a few data items. Write a few select statements.

SQLAlchemy Metada

Describe the Schema, the structure of the database (tables, columns, constraints, etc.) in Python.

- SQL generation from the metadata, generate to a schema.
- Reflection (Introspection) - Create the metadata from an existing database, from an existing schema.

```python
from sqlalchemy import MetaData
from sqlalchemy import Table, Column
from sqlalchemy import Integer, String

metadata = MetaData()
```

```
user_table = Table('user', metadata,
                    Column('id', Integer,
primary_key=True),
                    Column('name', String(100),
unique=True),
                    Column('balance', Integer,
nullable=False)
                    )
print(user_table.name)
print(user_table.c.name)
print(user_table.c.id)

print(user_table.c)
print(user_table.columns)   # A bit like a Python
dictionary, but it is an associativ\
e array



from sqlalchemy import create_engine
engine = create_engine('sqlite://')
metadata.create_all(engine)

from sqlalchemy import ForeignKey

address_table = Table('address', metadata,
                Column('id', Integer, primary_key=True),
                Column('stree', String(100)),
                Column('user_id', Integer,
ForeignKey('user.id'))
                )
address_table.create(engine)

from sqlalchemy import Unicode, UnicodeText,
ForeignKeyConstraint, DateTime

story_table = Table('story', metadata,
                    Column('id', Integer,
primary_key=True),
                    Column('version', Integer,
primary_key=True),
                    Column('headline', Unicode(100),
nullable=False),
                    Column('body', UnicodeText)
                    )
```

```
published_table = Table('published', metadata,
                    Column('id', Integer, primary_key=True),
                    Column('timestamp', DateTime, nullable=False),
                    Column('story_id', Integer, nullable=False),
                    Column('version', Integer, nullable=False),
                    ForeignKeyConstraint(
                        ['story_id', 'version_id'],
                        ['story.story_id', 'story.version_id']
                    )
                )


conn.execute(user_table.insert(), [
    {'username': 'Jack', 'fullname': 'Jack Burger'},
    {'username': 'Jane', 'fullname': 'Jane Doe'}
])

from sqlalchemy import select
select_stmt = select([user_table.c.username, user_table.c.fullname]).where(user_tabl\
e.c.username == 'ed')
result = conn.execute(select_stmt)
for row in result:
    print(row)

select_stmt = select([user_table])
conn.execute(select_stmt).fetchall()

select_stmt = select([user_table]).where(
    or_(
        user_table.c.username == 'ed',
        user_table.c.usernane == 'wendy'
    )
)

joined_obj = user_table.join(address_table, user_table.c.id = address_table.c.user_i\
d)
```

SQLAlchemy types

- Integer() - INT
- String() - ASCII strings - VARCHAR
- Unicode() - Unicode string - VARCHAR or NVARCHAR depending on database
- Boolean() - BOOLEAN, INT, TINYINT depending on db support for boolean type
- DateTime() - DATETIME or TIMESTAMP returns Python datetime() objects.
- Float() - floating point values
- Numeric() - precision numbers using Python Decimal()

SQLAlchemy ORM - Object Relational Mapping

- Domain model
- Mapping between Domain Object - Table Row

SQLAlchemy ORM create

```python
import os
from sqlalchemy import Column, ForeignKey, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy import create_engine

Base = declarative_base()


class Person(Base):
    __tablename__ = 'person'
    id   = Column(Integer, primary_key=True)
    name = Column(String(250), nullable=False, unique=True)
```

```python

class Genre(Base):
    __tablename__ = 'genre'
    id = Column(Integer, primary_key=True)
    name = Column(String(250), nullable=False, unique=True)

class Movie(Base):
    __tablename__ = 'movie'
    id = Column(Integer, primary_key=True)
    title = Column(String(250), nullable=False, unique=True)
    genre_id = Column(Integer, ForeignKey('genre.id'))
    genre = relationship(Genre)

class Cast(Base):
    __tablename__ = 'cast'
    id = Column(Integer, primary_key=True)
    character = Column(String(250))
    person_id = Column(Integer, ForeignKey('person.id'))
    movie_id = Column(Integer, ForeignKey('movie.id'))



if __name__ == '__main__':
    dbname = 'imdb.db'
    if os.path.exists(dbname):
        os.unlink(dbname)
    engine = create_engine('sqlite:///' + dbname)
    Base.metadata.create_all(engine)
```

SQLAlchemy ORM schema

```
    echo .schema | sqlite3 imdb.db
```

```sql
CREATE TABLE person (
      id INTEGER NOT NULL,
      name VARCHAR(250) NOT NULL,
      PRIMARY KEY (id)
);
CREATE TABLE genre (
      id INTEGER NOT NULL,
```

```
        title VARCHAR(250),
        PRIMARY KEY (id)
);
CREATE TABLE movie (
        id INTEGER NOT NULL,
        title VARCHAR(250),
        genre_id INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY(genre_id) REFERENCES genre (id)
);
CREATE TABLE "cast" (
        id INTEGER NOT NULL,
        character VARCHAR(250),
        person_id INTEGER,
        movie_id INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY(person_id) REFERENCES person (id),
        FOREIGN KEY(movie_id) REFERENCES movie (id)
);
```

SQLAlchemy ORM reflection

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session
from sqlalchemy.ext.automap import automap_base

Base = automap_base()

dbname = 'imdb.db'
engine = create_engine('sqlite:///' + dbname)

Base.prepare(engine, reflect=True)
Genre = Base.classes.genre

print(Genre.metadata.sorted_tables)

for c in Base.classes:
    print(c)

#session = Session(engine)
#session.add(Address(email_address="foo@bar.com",
user=User(name="foo")))
#session.commit()
```

SQLAlchemy ORM INSERT after automap

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import Session
from sqlalchemy.ext.automap import automap_base

Base = automap_base()

dbname = 'imdb.db'
engine = create_engine('sqlite:///' + dbname)

Base.prepare(engine, reflect=True)
Genre  = Base.classes.genre
Movie  = Base.classes.movie
Person = Base.classes.person
Cast   = Base.classes.cast



session = Session(engine)
for name in ('Action', 'Animation', 'Comedy', 'Documentary', 'Family', 'Horror'):
    session.add(Genre(name = name))

session.add(Movie(title = "Sing", genre_id=2))
session.add(Movie(title = "Moana", genre_id=2))
session.add(Movie(title = "Trolls", genre_id=2))
session.add(Movie(title = "Power Rangers", genre_id=1))

session.commit()
```

SQLAlchemy ORM INSERT

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import Session
from orm_create_db import Base, Genre, Movie, Person, Cast

dbname = 'imdb.db'
engine = create_engine('sqlite:///' + dbname)

Base.metadata.bind = engine
```

```

session = Session(engine)
genre = {}
for name in ('Action', 'Animation', 'Comedy', 'Documentary', 'Family', 'Horror'):
    genre[name] = Genre(name = name)
    session.add(genre[name])

print(genre['Animation'].name) # Animation
print(genre['Animation'].id)   # None
session.commit()

print(genre['Animation'].name) # Animation
print(genre['Animation'].id)   # 2
session.add(Movie(title = "Sing", genre = genre['Animation']))
session.commit()
```

SQLAlchemy ORM SELECT

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session
from orm_create_db import Base, Genre, Movie, Person, Cast

dbname = 'imdb.db'
engine = create_engine('sqlite:///' + dbname)

Base.metadata.bind = engine

session = Session(engine)

for g in session.query(Genre).all():
    print(g.name, g.id)

print("---")
animation = session.query(Genre).filter(Genre.name == 'Animation').one()
print(animation.name, animation.id)
```

SQLAlchemy ORM SELECT cross tables

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import Session
from orm_create_db import Base, Genre, Movie, Person, Cast

dbname = 'imdb.db'
engine = create_engine('sqlite:///' + dbname)

Base.metadata.bind = engine

session = Session(engine)

movies = session.query(Movie).all()
for m in movies:
    print(m.title, "-", m.genre.name)
```

SQLAlchemy ORM SELECT and INSERT

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import Session
from orm_create_db import Base, Genre, Movie, Person, Cast

dbname = 'imdb.db'
engine = create_engine('sqlite:///' + dbname)

Base.metadata.bind = engine

session = Session(engine)

animation = session.query(Genre).filter(Genre.name == 'Animation').one()
session.add(Movie(title = "Moana", genre = animation))
session.add(Movie(title = "Trolls", genre = animation))

action = session.query(Genre).filter(Genre.name == 'Action').one()
session.add(Movie(title = "Power Rangers", genre = action))

comedy = session.query(Genre).filter(Genre.name == 'Comedy').one()
session.add(Movie(title = "Gostbuster", genre = comedy))
```

```
session.commit()
```

SQLAlchemy ORM UPDATE

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import Session
from orm_create_db import Base, Genre, Movie, Person, Cast

dbname = 'imdb.db'
engine = create_engine('sqlite:///' + dbname)

Base.metadata.bind = engine

session = Session(engine)

movie = session.query(Movie).filter(Movie.title == 'Gostbuster').one()
print(movie.title)
movie.title = 'Ghostbusters'
session.commit()

print(movie.title)
```

SQLAlchemy ORM logging

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import Session
from orm_create_db import Base, Genre, Movie, Person, Cast

import logging

logging.basicConfig()
logging.getLogger('sqlalchemy.engine').setLevel(logging.INFO)

logger = logging.getLogger('demo')
```

```
logger.setLevel(logging.INFO)

dbname = 'imdb.db'
engine = create_engine('sqlite:///' + dbname)

Base.metadata.bind = engine

session = Session(engine)


logger.info("Selecting all")
movies = session.query(Movie).all()
for m in movies:
    logger.info("------------")
    #print(m.title, "-", m.genre_id)
    print(m.title, "-", m.genre.name)
```

Solution: Create table

Create the followig schema

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table, Column
from sqlalchemy import Integer, String
from sqlalchemy import ForeignKey

metadata = MetaData()

node_table = Table('node', metadata,
                     Column('id', Integer, primary_key=True),
                     Column('name', String(100), unique=True)
                     )

interface_table = Table('interface', metadata,
                     Column('id', Integer, primary_key=True),
                     Column('node_id', Integer, ForeignKey('node.id'), nullable=False),
                     Column('ipv4', String(14), unique=True),
```

```
                        Column('ipv6', String(80), unique=True),
                        )

connection_table = Table('connection', metadata,
                         Column('a', Integer, ForeignKey('interface.id'), nullable=False),
                         Column('b', Integer, ForeignKey('interface.id'), nullable=False)
                               )

engine = create_engine('sqlite://', echo=True)
metadata.create_all(engine)
```

Exercise: Inspector

Use the inspector to list all the tables and all the columns in every table.

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table, Column
from sqlalchemy import Integer, String
from sqlalchemy import ForeignKey

metadata = MetaData()

node_table = Table('node', metadata,
                        Column('id', Integer, primary_key=True),
                        Column('name', String(100), unique=True)
                        )

interface_table = Table('interface', metadata,
                        Column('id', Integer, primary_key=True),
                        Column('node_id', Integer, ForeignKey('node.id'), nullable=False),
                        Column('ipv4', String(14), unique=True),
                        Column('ipv6', String(80), unique=True),
```

```
                     )

connection_table = Table('connection', metadata,
                      Column('a', Integer, ForeignKey('interface.id'), nullable=False),
                      Column('b', Integer, ForeignKey('interface.id'), nullable=False)
                           )

engine = create_engine('sqlite://', echo=True)
metadata.create_all(engine)


m2 = MetaData()
m2_node_table = Table('node', m2, autoload=True, autoload_with=engine)
m2_interface_table = Table('interface', m2, autoload=True, autoload_with=engine)
print(m2_node_table.columns)
print(m2_interface_table.columns)
print(m2_node_table.__repr__())

from sqlalchemy import inspect

inspector = inspect(engine)
inspector.get_columns('address')
inspector.get_foreign_keys('address')
```

SQLAlchemy CREATE and DROP

- metadata.create_all(engine, checkfirst=True|False) emits CREATE statement for all tables.
- table.create(engine, checkfirst=False|True) emits CREATE statement for a single table.
- metadata.drop_all(engine, checkfirst=True|False) emits DROPT statement for all the tables.
- table.drop(engine, checkfirst=False|True) emits DROPT statement for a single table.

metada can create (or drop) the tables in the correct order to maintain the dependencies.

SQLAlchemy Notes

- Multi-column primary key (composite primary key).
- Composite foreign key.

SQLAlchemy Meta SQLite CREATE

```
from sqlalchemy import create_engine
import os
from sqlite_meta_schema import get_meta

dbname = 'test.db'
if os.path.exists(dbname):
    os.unlink(dbname)
engine = create_engine('sqlite:///test.db')

metadata = get_meta()
metadata.create_all(engine)
```

```
from sqlalchemy import MetaData
from sqlalchemy import Table, Column
from sqlalchemy import Integer, String
from sqlalchemy import ForeignKey


def get_meta():
    metadata = MetaData()

    node_table = Table('node', metadata,
                       Column('id', Integer, primary_key=True),
                       Column('name', String(100), unique=True)
                       )

    interface_table = Table('interface', metadata,
                            Column('id', Integer,
```

```
primary_key=True),
                              Column('node_id', Integer,
ForeignKey('node.id'), nullab\
le=False),
                              Column('ipv4', String(14),
unique=True),
                              Column('ipv6', String(80),
unique=True),
                              )

    connection_table = Table('connection', metadata,
                               Column('a', Integer,
ForeignKey('interface.id'), nullab\
le=False),
                               Column('b', Integer,
ForeignKey('interface.id'), nullab\
le=False)
                               )
    return metadata
```

SQLAlchemy Meta Reflection

```
from sqlalchemy import create_engine
import os
#from sqlalchemy import inspect
from sqlalchemy.engine import reflection

dbname = 'test.db'
if not os.path.exists(dbname):
    exit("Database file '{}' could not be
found".format(dbname))

engine = create_engine('sqlite:///test.db')
# inspector = inspect(engine)
# print(inspector)
# print(inspector.get_columns('address'))
# print(inspector.get_foreign_keys('address'))

insp = reflection.Inspector.from_engine(engine)
print(insp.get_table_names())
```

SQLAlchemy Meta INSERT

```

```

SQLAlchemy Meta SELECT

```

```

NoSQL

Types of NoSQL databases

- Document oriented - MongoDB
- Key-Value store - Redis
- Graph - Neo4j
- Tuple store - Apache River, TIBCO

MongoDB

MongoDB CRUD

- Create, Read, Update, Delete

Install MongoDB support

- Otherwise: **pip install pymongo**

Python MongoDB insert

```
from pymongo import MongoClient
import datetime

client = MongoClient()
db = client.demo

foo = {
    'name'      : 'Foo',
    'email'     : 'foo@example.com',
    'birthdate' : datetime.datetime.strptime('2002-01-02', '%Y-%m-%d'),
    'student'   : True,
}

bar = {
    'name'      : 'Bar',
    'email'     : 'bar@example.com',
    'birthdate' : datetime.datetime.strptime('1998-08-03', '%Y-%m-%d'),
    'student'   : True,
    'teacher'   : False,
}

zorg = {
```

```
    'name'      : 'Zorg',
    'email'     : 'zorg@corp.com',
    'birthdate' : datetime.datetime.strptime('1995-12-12', '%Y-%m-%d'),
    'teacher'   : True,
}


db.people.insert(foo)
db.people.insert(bar)
db.people.insert(zorg)
```

MongoDB CLI

```
$ mongo
> help
...
> show dbs
admin  (empty)
demo   0.078GB
local  0.078GB

> use demo      (name of db)
switched to db demo

> show collections
people
system.indexes

> db.people.find()
{ "_id" : ObjectId("58a3e9b2962d747a9c6e676c"), "email" : "foo@example.com", "studen\
t" : true,
    "birthdate" : ISODate("2002-01-02T00:00:00Z"), "name" : "Foo" }
{ "_id" : ObjectId("58a3e9b2962d747a9c6e676d"), "email" : "bar@example.com", "name" \
: "Bar", "student" : true,
    "birthdate" : ISODate("1998-08-03T00:00:00Z"), "teacher" : false }
{ "_id" : ObjectId("58a3e9b2962d747a9c6e676e"), "email" : "zorg@corp.com",
    "birthdate" : ISODate("1995-12-12T00:00:00Z"),
```

```
"teacher" : true, "name" : "Zorg"\
 }

> db.people.drop()       (drop a collection)
> db.dropDatabase()      (drop a whole database)
```

Python MongoDB find

```python
from pymongo import MongoClient
import datetime

client = MongoClient()
db = client.demo

for p in db.people.find():
    print(p)
```

Python MongoDB find refine

```python
from pymongo import MongoClient
import datetime

client = MongoClient()
db = client.demo

for p in db.people.find({ 'name' : 'Foo'}):
    print(p)
```

Python MongoDB update

```python
from pymongo import MongoClient
import datetime

client = MongoClient()
db = client.demo

db.people.update({ 'name' : 'Zorg'}, { '$set' : {
'salary' : 1000 } })
for p in db.people.find({ 'name' : 'Zorg'}):
    print(p)
```

Python MongoDB remove (delete)

```python
from pymongo import MongoClient
import datetime

client = MongoClient()
db = client.demo

db.people.remove({ 'name' : 'Zorg'})
for p in db.people.find():
    print(p)
```

Redis

Redis CLI

[redis-cli]

```
$ redis-cli
> set name foo
> get name
> set name "foo bar"
> get name

> set a 1
> get a
> incr a
> get a

> set b 1
> keys *
> del b
```

Redis list keys

```python
import redis
r = redis.StrictRedis()

for k in r.keys('*'):
    print(k)
```

Redis set get

```python
import redis
r = redis.StrictRedis()

```

```
r.set("name", "some value")
print(r.get("name"))
```

Redis incr

```
import redis
r = redis.StrictRedis()

r.set("counter", 40)
print(r.get("counter"))
print(r.incr("counter"))
print(r.incr("counter"))
print(r.get("counter"))
```

Redis incrby

```
import redis
r = redis.StrictRedis()

r.set("counter", 19)
print(r.get("counter"))
print(r.incrby("counter", 23))
print(r.get("counter"))
```

Redis setex

Set with expiration time in seconds.

```
import redis
import time
r = redis.StrictRedis()


r.setex("login", 2, 'foobar')
print(r.get("login"))    # 'foobar'
time.sleep(1)
print(r.get("login"))    # 'foobar'
time.sleep(1)
print(r.get("login"))    # None
```

Web client

urllib the web client

```
import urllib

# f is like a filehand for http requests, but it cannot be user "with"
# Only works in Python 2
f = urllib.urlopen('http://python.org/')
html = f.read()   # is like a get() request
f.close()

print(html)


# retrieve a file and save it locally:
urllib.urlretrieve('http://www.python.org/images/python-logo.gif', 'logo.gif')
```

urllib2 the web client

urllib2 is better than urllib as it will indicate if there was an error retreiving

```
import urllib2

# f is like a filehand for http requests
f = urllib2.urlopen('http://python.org/')
html = f.read()   # is like a get() request
f.close()
print(html)

```

```python

try:
    f = urllib2.urlopen('http://python.org/some_missing_page')
    html = f.read()
    f.close()
    print(html)
except urllib2.HTTPError as e:
    print(e)    # HTTP Error 404: OK
```

httpbin.org

- httpbin.org
- source

requests get

```python
import requests

r = requests.get('http://httpbin.org/')
print(type(r))
print(r.status_code)
print(r.headers)
print(r.headers['content-type'])
```

- HTTP status codes
- Python requests

Download image using requests

```python
import requests

url = 'https://bloximages.newyork1.vip.townnews.com/wpsdlocal6.com/content/tncms/ass\
ets/v3/editorial/7/22/722f8401-e134-5758-9f4b-a542ed88a101/5d41b45d92106.image.jpg'
filename = "source.jpg"
```

```
res = requests.get(url)
print(res.status_code)
with open(filename, 'wb') as fh:
    fh.write(res.content)
```

Download image as a stream using requests

```
import requests
import shutil

url = 'https://bloximages.newyork1.vip.townnews.com/wpsdlocal6.com/content/tncms/ass\
ets/v3/editorial/7/22/722f8401-e134-5758-9f4b-a542ed88a101/5d41b45d92106.image.jpg'
filename = "source.jpg"
res = requests.get(url, stream=True)
print(res.status_code)
with open(filename, 'wb') as fh:
    res.raw.decode_content
    shutil.copyfileobj(res.raw, fh)
```

Download zip file

```
import requests
import shutil

url = "https://code-maven.com/public/developer_survey_2019.zip"
filename = "developer_survey_2019.zip"

res = requests.get(url, stream=True)
print(res.status_code)
if res.status_code == 200:
    with open(filename, 'wb') as fh:
        res.raw.decode_content
        shutil.copyfileobj(res.raw, fh)
```

Extract zip file

```python
import zipfile

path = "developer_survey_2019.zip"
z = zipfile.ZipFile(path)
z.extractall()
```

Interactive Requests

```python
import requests

r = requests.get('http://httpbin.org/')

import code
code.interact(local=locals())
```

requests get JSON

```python
import requests

r = requests.get('http://httpbin.org/ip')
print(r.headers['content-type'])
print(r.text)
data = r.json()
print(data)
print(data['origin'])
```

requests get JSON UserAgent

```python
import requests

r = requests.get('http://httpbin.org/user-agent')
print(r.headers['content-type'])
print(r.text)
data = r.json()
print(data)
print(data['user-agent'])
```

requests get JSON UserAgent

```
import requests

r = requests.get('http://httpbin.org/user-agent',
  headers = {'User-agent': 'Internet Explorer/2.0'})
print(r.headers['content-type'])
print(r.text)
data = r.json()
print(data)
print(data['user-agent'])
```

requests get header

```
import requests

r = requests.get('http://httpbin.org/headers')
print(r.text)

# {
#   "headers": {
#     "Accept": "*/*",
#     "Accept-Encoding": "gzip, deflate",
#     "Host": "httpbin.org",
#     "User-Agent": "python-requests/2.3.0 CPython/2.7.12 Darwin/16.3.0"
#   }
# }
```

requests change header

```
import requests

r = requests.get('http://httpbin.org/headers',
    headers = {
      'User-agent'  : 'Internet Explorer/2.0',
      'SOAPAction'  : 'http://www.corp.net/some/path/CustMsagDown.Check',
      'Content-type': 'text/xml'
    }
  )
```

```
print(r.text)

# {
#    "headers": {
#      "Accept": "*/*",
#      "Accept-Encoding": "gzip, deflate",
#      "Content-Type": "text/xml",
#      "Host": "httpbin.org",
#      "Soapaction": "http://www.corp.net/some/path/CustMsagDown.Check",
#      "User-Agent": "Internet Explorer/2.0"
#    }
# }
```

---

requests post

```
import requests

payload = '''
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:cu\
s="http://www.corp.net/Request.XSD">
    <soapenv:Header/>
    <soapenv:Body>
       <cus:CustMsagDown.Check>
           <cus:MainCustNum>327</cus:MainCustNum>
           <cus:SourceSystem></cus:SourceSystem>
       </cus:CustMsagDown.Check>
    </soapenv:Body>
</soapenv:Envelope>
'''

r = requests.post('http://httpbin.org/post',
    headers = {
        'User-agent'   : 'Internet Explorer/2.0',
        'SOAPAction'   : 'http://www.corp.net/some/path/CustMsagDown.Check',
        'Content-type': 'text/xml'
    },
    data = payload,
)
```

```
print(r.headers['content-type'])
print(r.text)
```

Tweet

```
import configparser
import twitter
import os

config = configparser.ConfigParser()
config.read(os.path.join(os.path.dirname(os.path.abspath(__file__)), 'api.cfg'));
api = twitter.Api( **config['twitter'] )

status = api.PostUpdate('My first Tweet using Python')
print(status.text)
```

API config file

```
[twitter]
consumer_key=
consumer_secret=
access_token_key=
access_token_secret=

[bitly]
access_token=
```

bit.ly

```
import configparser
import os
import requests

def shorten(uri):
    config = configparser.ConfigParser()
    #config.read(os.path.join(os.path.expanduser('~'), 'api.cfg'))

```

```
config.read(os.path.join(os.path.dirname(os.path.abspath(__file__)), 'api.cfg'))

    query_params = {
        'access_token': bitly_config['bitly']['access_token'],
        'longUrl': uri
    }

    endpoint = 'https://api-ssl.bitly.com/v3/shorten'
    response = requests.get(endpoint, params=query_params, verify=False)

    data = response.json()

    if not data['status_code'] == 200:
        exit("Unexpected status_code: {} in bitly response. {}".format(data['status_\
code'], response.text))
    return data['data']['url']

print(shorten("http://code-maven.com/"))
```

Exercise: Combine web server and client

Write a web application that can get a site and a text as input (e.g. http://cnn.com and ‘Korea’)
check if on the given site the word appears or not?

Extended version: Only get the URL as the input and create statistics, which are the most
frequent words on the given page.

Python Web server

Hello world web

```
from wsgiref.util import setup_testing_defaults
from wsgiref.simple_server import make_server

import time

def hello_world(environ, start_response):
    setup_testing_defaults(environ)

    status = '200 OK'
    headers = [('Content-type', 'text/plain')]

    start_response(status, headers)

    return "Hello World " + str(time.time())

port = 8080
httpd = make_server('0.0.0.0', port, hello_world)
print("Serving on port {}...".format(port))
httpd.serve_forever()
```

Dump web environment info

```
from wsgiref.util import setup_testing_defaults
from wsgiref.simple_server import make_server

# A relatively simple WSGI application. It's going to print out the
# environment dictionary after being updated by setup_testing_defaults
def simple_app(environ, start_response):
    setup_testing_defaults(environ)

    status = '200 OK'
```

```python
    headers = [('Content-type', 'text/plain')]

    start_response(status, headers)

    ret = ["{}: {}\n".format(key, value)
           for key, value in environ.iteritems()]
    return ret

httpd = make_server('', 8000, simple_app)
print("Serving on port 8000...")
httpd.serve_forever()

# taken from the standard documentation of Python
```

Web echo

```python
from wsgiref.util import setup_testing_defaults
from wsgiref.simple_server import make_server

import time
import cgi

def hello_world(environ, start_response):
    setup_testing_defaults(environ)

    status = '200 OK'
    headers = [('Content-type', 'text/html')]

    start_response(status, headers)

    form = cgi.FieldStorage(fp=environ['wsgi.input'], environ=environ)
    if 'txt' in form:
        return 'Echo: ' + form['txt'].value

    return """
<form>
<input name="txt" />
<input type="submit" value="Echo" />
</form>
"""
httpd = make_server('', 8000, hello_world)
```

```
print("Serving on port 8000...")
httpd.serve_forever()
```

Web form

```
from wsgiref.util import setup_testing_defaults
from wsgiref.simple_server import make_server

import time
import cgi

def hello_world(environ, start_response):
    setup_testing_defaults(environ)

    status = '200 OK'
    headers = [('Content-type', 'text/html')]

    start_response(status, headers)

    form = cgi.FieldStorage(fp=environ['wsgi.input'], environ=environ)
    html = ''
    for f in form:
       html += f + '==' + form[f].value + '<br>'

    if not html:
        html = """
<a href="?fname=Foo&lname=Bar">click</a>
<form>
Username: <input name="username" /><br>
Password: <input type="password" name="pw" /><br>
Age group: Under 18 <input type="radio" name="age" value="kid" >
18-30 <input type="radio" name="age" value="young" >
30- <input type="radio" name="age" value="old" >
<input type="submit" value="Send" />
</form>
"""
    return html

httpd = make_server('', 8000, hello_world)
print("Serving on port 8000...")
httpd.serve_forever()
```

Resources

- wsgi tutorial

Python Flask

Python Flask intro

- Flask
- Jinja
- Werkzeug

Python Flask installation

```
virtualenv venv -p python3
source venv/bin/activate

pip install flask
```

Flask: Hello World

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

```
$ python hello_world.py
Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Flask hello world + test

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

```
FLASK_APP=app FLASK_DEBUG=1 flask run

Visit: http://127.0.0.1:5000/
curl http://localhost:5000/
```

Windows on the command line or in the terminal of Pycharm.

```
set FLASK_APP=app
set FLASK_DEBUG=1
flask run
```

```python
import app

def test_app():
    web = app.app.test_client()

    rv = web.get('/')
    assert rv.status == '200 OK'
    assert rv.data == b'Hello World!'
```

```
pytest
```

Flask generated page - time

```python
from flask import Flask
import time

app = Flask(__name__)

@app.route("/")
```

```python
def main():
    return '<a href="/time">time</a>'

@app.route("/time")
def echo():
    return str(time.time())
```

```python
import app
import re

def test_home():
    web = app.app.test_client()

    rv = web.get('/')
    assert rv.status == '200 OK'
    assert rv.data == b'<a href="/time">time</a>'

def test_time():
    web = app.app.test_client()

    rv = web.get('/time')
    assert rv.status == '200 OK'
    assert re.search(r'\d+\.\d+$', rv.data.decode('utf-
8'))
```

Flask: Echo GET

```python
from flask import Flask, request
app = Flask(__name__)

@app.route("/")
def main():
    return '''
     <form action="/echo" method="GET">
         <input name="text">
         <input type="submit" value="Echo">
     </form>
     '''

@app.route("/echo")
def echo():
    return "You said: " + request.args.get('text', '')
```

```python
import app

def test_app():
    web = app.app.test_client()

    rv = web.get('/')
    assert rv.status == '200 OK'
    assert '<form action="/echo" method="GET">' in rv.data.decode('utf-8')

    rv = web.get('/echo')
    assert rv.status == '200 OK'
    assert b"You said: " == rv.data

    rv = web.get('/echo?text=foo+bar')
    assert rv.status == '200 OK'
    assert b"You said: foo bar" == rv.data
```

```
curl http://localhost:5000/
curl http://localhost:5000/echo?text=Sanch+Panza
```

```python
import requests

res = requests.get('http://localhost:5000/')
print(res.status_code)
print(res.text)

res = requests.get('http://localhost:5000/echo?text=Hello World!')
print(res.status_code)
print(res.text)
```

Flask: Echo POST

```python
from flask import Flask, request

app = Flask(__name__)

@app.route("/")
def main():
    return '''
```

```
    <form action="/echo" method="POST">
        <input name="text">
        <input type="submit" value="Echo">
    </form>
    '''

@app.route("/echo", methods=['POST'])
def echo():
    if 'text' in request.form:
        return "You said: " + request.form['text']
    else:
        return "Nothing to say?"
```

```
import app

def test_app():
    web = app.app.test_client()

    rv = web.get('/')
    assert rv.status == '200 OK'
    assert '<form action="/echo" method="POST">' in rv.data.decode('utf-8')


    rv = web.get('/echo')
    assert rv.status == '405 METHOD NOT ALLOWED'
    assert '<title>405 Method Not Allowed</title>' in rv.data.decode('utf-8')


    rv = web.post('/echo')
    assert rv.status == '200 OK'
    assert b"Nothing to say?" == rv.data


    rv = web.post('/echo', data={ "text": "foo bar" })
    assert rv.status == '200 OK'
    assert b"You said: foo bar" == rv.data
```

```
curl --data "text=Sancho Panza" http://localhost:5000/echo
```

```python
import requests

res = requests.get('http://localhost:5000/')
print(res.status_code)
print(res.text)


res = requests.post('http://localhost:5000/echo', data=
{"text": "Hello World!"})
print(res.status_code)
print(res.text)
```

Flask: templates

```python
from flask import Flask, request
app = Flask(__name__)

@app.route("/")
def main():
    return render_template('index.html')

@app.route("/echo", methods=['POST'])
def echo():
    return "You said: " + request.form['text']
```

```html
<form action="/echo" method="POST">
<input name="text">
<input type="submit" value="Echo">
</form>
```

```
FLASK_APP=echo_post FLASK_DEBUG=0 flask run
```

Internal Server Error

```
FLASK_APP=echo_post FLASK_DEBUG=1 flask run
```

Flask: templates

```python
from flask import Flask, request, render_template
app = Flask(__name__)

@app.route("/")
def main():
    return render_template('index.html')

@app.route("/echo", methods=['POST'])
def echo():
    return "You said: " + request.form['text']
```

Flask: templates with parameters

```python
from flask import Flask, request, render_template
app = Flask(__name__)

@app.route("/")
def main():
    return render_template('echo.html')

@app.route("/echo", methods=['POST'])
def echo():
    return render_template('echo.html', text=request.form['text'])
```

```html
<form action="/echo" method="POST">
<input name="text">
<input type="submit" value="Echo">
</form>

{% if text %}
  You said: {{ text }}
{% endif %}
```

```python
import echo

def test_app():
    web = echo.app.test_client()

    rv = web.get('/')
    assert rv.status == '200 OK'
```

```python
    assert '<form action="/echo" method="POST">' in rv.data.decode('utf-8')

    rv = web.post('/echo', data={ "text": "foo bar" })
    assert rv.status == '200 OK'
    assert "You said: foo bar" in rv.data.decode('utf-8')
```

Flask: runner

```
$ cd examples/flask/params
```

```
$ export FLASK_APP=echo
$ export FLASK_DEBUG=1
$ flask run
```

or

```
$ FLASK_APP=echo.py FLASK_DEBUG=1  flask run
```

on windows

```
> set FLASK_APP=echo
> set FLASK_DEBUG=1
> flask run
```

Other parameters

```
$ FLASK_APP=echo.py FLASK_DEBUG=1  flask run --port 8080 --host 0.0.0.0
```

Exercise: Flask calculator

Write a web application that has two entry boxes and a button and that will add the two numbers inserted into the entry boxes.

Static files

```python
from flask import Flask, request, render_template, url_for
app = Flask(__name__)

@app.route("/")
def main():
    return render_template('main.html')

@app.route("/other")
def other():
    return render_template('other.html',
        img_path = url_for('static', filename='img/python.png'))
```

```html
<h1>Main page</h1>
<img src="/static/img/python.png">
<p>
<a href="/other">other</a>
```

```html
<h2>Other page</h2>
img_path: {{ img_path }}
<p>
<img src="{{ img_path }}">
<p>
<a href="/">main</a>
```

```
.
├── app.py
├── static
│   └── img
│       └── python.png
└── templates
    ├── main.html
    └── other.html
```

Flask Logging

```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def main():
    app.logger.debug("Some debug message")
    app.logger.warning("Some warning message")
    app.logger.error("Some error message")
    return "Hello World"
```

Flask: Counter

```python
from flask import Flask
app = Flask(__name__)

counter = 1

@app.route("/")
def main():
    global counter
    counter += 1
    return str(counter)
```

Access the page from several browsers. There is one single counter that lives as long as the process lives.

Color selector without session

```python
from flask import Flask, request, render_template
import re
app = Flask(__name__)

@app.route("/",methods=['GET', 'POST'] )
def main():
    color = "FFFFFF"
    new_color = request.form.get('color', '')
    if re.search(r'^[0-9A-F]{6}$', new_color):
        color = new_color

    return render_template('main.html', color = color)
```

```
<style>
* {
  background-color: #{{ color }};
}
</style>

<form method="POST">
<input name="color" value="{{ color }}">
<input type="submit" value="Set">
</form>
<p>
<a href="/">home</a>
```

Session management

```
from flask import Flask, request, render_template, session
import re
app = Flask(__name__)
app.secret_key = 'blabla'

@app.route("/",methods=['GET', 'POST'] )
def main():
    color = session.get('color', 'FFFFFF')
    app.logger.debug("Color: " + color)

    new_color = request.form.get('color', '')
    if re.search(r'^[0-9A-F]{6}$', new_color):
        app.logger.debug('New color: ' + new_color)
        session['color'] = new_color
        color = new_color

    return render_template('main.html', color = color)
```

Flask custom 404 page

```
from flask import Flask
app = Flask(__name__)

```

```
@app.route("/")
def main():
    return '''
Main
<a href="/not">404 page</a>
'''
```

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def main():
    return '''
Main
<a href="/not">404 page</a>
'''

@app.errorhandler(404)
def not_found(e):
    return "Our Page not found", 404
```

```
curl -I http://localhost:5000/not

HTTP/1.0 404 NOT FOUND
```

Flask Error page

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def main():
    return '''
Main
<a href="/bad">bad</a>
'''

@app.route("/bad")
def bad():
    raise Exception("This is a bad page")
    return 'Bad page'
```

Will not trigger in debug mode!

```
$ FLASK_APP=echo.py FLASK_DEBUG=0  flask run
```

```
curl -I http://localhost:5000/not

HTTP/1.0 500 INTERNAL SERVER ERROR
```

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def main():
    return '''
Main
<a href="/bad">bad</a>
'''

@app.route("/bad")
def bad():
    raise Exception("This is a bad page")
    return 'Bad page'

@app.errorhandler(500)
def not_found(err):
    #raise Exception("Oups")
    return "Our Page crashed", 500
```

Flask URL routing

The mapping of the path part of a URL, so the one that comes after the domain name and
after the port number (if it is included) is the path. Mapping that to a function call
is called routing.

In the following pages we are going to see several examples on how to map routes to functions.

It is also called “url route registration”.

Flask Path params

```
from flask import Flask, jsonify
app = Flask(__name__)

@app.route("/")
def main():
    return '''
Main<br>
<a href="/user/23">23</a><br>
<a href="/user/42">42</a><br>
<a href="/user/Joe">Joe</a><br>
'''

@app.route("/user/<uid>")
def api_info(uid):
    return uid
```

```
FLASK_APP=app.py FLASK_DEBUG=0  flask run
```

Flask Path params (int)

```
from flask import Flask, jsonify
app = Flask(__name__)

@app.route("/")
def main():
    return '''
Main<br>
<a href="/user/23">23</a><br>
<a href="/user/42">42</a><br>
<a href="/user/Joe">Joe</a><br>
'''
```

```python

@app.route("/user/<int:uid>")
def api_info(uid):
    return str(uid)
```

```
FLASK_APP=app.py FLASK_DEBUG=0  flask run
```

Flask Path params add (int)

```python
from flask import Flask, jsonify
app = Flask(__name__)

@app.route("/")
def main():
    return '''
Main <a href="/add/23/19">add</a>
'''

@app.route("/add/<int:a>/<int:b>")
def api_info(a, b):
    return str(a+b)
```

```
FLASK_APP=app.py FLASK_DEBUG=0  flask run
```

Flask Path params add (path)

- Accept any path, including slashes:

```python
from flask import Flask, jsonify
app = Flask(__name__)

@app.route("/")
def main():
    return '''
Main<br>
<a href="/user/name">/user/name</a><br>
<a href="/user/other/dir">/user/other/dir</a><br>
<a href="/user/hi.html">/usre/hi.html</a><br>
'''
```

```python

@app.route("/user/<path:fullpath>")
def api_info(fullpath):
    return fullpath
```

```
FLASK_APP=app.py FLASK_DEBUG=0  flask run
```

Jinja loop, conditional, include

```
.
├── app.py
└── templates
    ├── incl
    │   ├── footer.html
    │   └── header.html
    └── main.html
```

```python
from flask import Flask, render_template
app = Flask(__name__)

@app.route("/")
def main():
    languages = [
        {
            "name": "Python",
            "year": 1991,
        },
        {
            "name": "JavaScript",
            "year": 1995,
        },
        {
            "name": "C",
        }
    ]
    return render_template('main.html',
        title      = "Code Maven Jinja example",
        languages = languages,
    )
```

```
{% include 'incl/header.html' %}


<h2>Languages</h2>
<ul>
  {% for lang in languages %}
     <li>{{ lang.name }}
        {% if lang.year %}
             {{ lang.year }}
        {% else %}
             Timeless
        {% endif %}
     </li>
  {% endfor %}
</ul>

{% include 'incl/footer.html' %}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1, user-scalabl\
e=yes">
    <title>{{ title }}</title>
</head>
<body>
<h1>{{ title }}</h1>
```

```
</body>
</html>
```

Exercise: Flask persistent

Create a Flask-based application with a persistent counter that even after restarting the application
the counter will keep increasing.

Exercise: Flask persistent

Create a Flask-based application with a persistent counter that even after restarting the application
the counter will keep increasing. For each user have its own counter as identified by the username they type in.

Flask Exercises

- [Shopping list]
- [TODO]

Flask login

```python
from flask import Flask, render_template, url_for, redirect, request, session
app = Flask(__name__)
app.secret_key = 'loginner'

users = {
    'admin' : 'secret',
    'foo'   : 'myfoo',
}

@app.route("/")
def main():
    return render_template('main.html')

@app.route('/login')
def login_form():
    return render_template('login.html')

@app.route('/login', methods=['POST'])
def login():
    username = request.form.get('username')
    password = request.form.get('password')
    if username and password and username in users and users[username] == password:
        session['logged_in'] = True
```

```
        return redirect(url_for('account'))

    return render_template('login.html')

@app.route("/account")
def account():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    return render_template('account.html')

@app.route('/logout')
def logout():
    if not session.get('logged_in'):
        return "Not logged in"
    else:
        session['logged_in'] = False
    return render_template('logout.html')
```

```
{% include 'header.html' %}
Account information.
```

```
<div>
<a href="/">home</a> | <a href="/login">login</a> | <a href="/logout">logout</a> | <\
a href="/account">account</a>
</div>
```

```
{% include 'header.html' %}
Home page
```

```
{% include 'header.html' %}
<form method="POST">
<input name="username" placeholder="username">
<input name="password" placeholder="password" type="password">
<input type="submit" value="Login">
</form>
```

```
{% include 'header.html' %}
Bye bye
```

```
{% include 'header.html' %}
Home
```

Flask JSON API

```python
from flask import Flask, jsonify
app = Flask(__name__)

@app.route("/")
def main():
    return '''
Main
<a href="/api/info">info</a>
'''

@app.route("/api/info")
def api_info():
    info = {
       "ip" : "127.0.0.1",
       "hostname" : "everest",
       "description" : "Main server",
       "load" : [ 3.21, 7, 14 ]
    }
    return jsonify(info)
```

```
$ curl -I http://localhost:5000/api/info
HTTP/1.0 200 OK
Content-Type: application/json
```

Flask and AJAX

```python
from flask import Flask, jsonify, render_template, request
import time
app = Flask(__name__)

```

```python
@app.route("/")
def main():
    return render_template('main.html', reload = time.time())

@app.route("/api/info")
def api_info():
    info = {
       "ip" : "127.0.0.1",
       "hostname" : "everest",
       "description" : "Main server",
       "load" : [ 3.21, 7, 14 ]
    }
    return jsonify(info)

@app.route("/api/calc")
def add():
    a = int(request.args.get('a', 0))
    b = int(request.args.get('b', 0))
    div = 'na'
    if b != 0:
        div = a/b
    return jsonify({
        "a"        :  a,
        "b"        :  b,
        "add"      :  a+b,
        "multiply" :  a*b,
        "subtract" :  a-b,
        "divide"   :  div,
    })
```

```javascript
(function() {
    var ajax_get = function(url, callback) {
        xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
                console.log('responseText:' + xmlhttp.responseText);
                try {
                    var data = JSON.parse(xmlhttp.responseText);
                } catch(err) {
                    console.log(err.message + " in " +
```

```
xmlhttp.responseText);
                        return;
                    }
                    callback(data);
                }
            };

            xmlhttp.open("GET", url, true);
            xmlhttp.send();
        };

        ajax_get('/api/info', function(data) {
            console.log('get info');
            document.getElementById('info').innerHTML =
JSON.stringify(data, null, '    '\
);
            document.getElementById('description').innerHTML
= data['description'];
        });

        var calc = document.getElementById('calc');
        calc.addEventListener('click', function() {
            document.getElementById('info').style.display =
"none";

document.getElementById('description').style.display =
"none";
            var url = '/api/calc?a=' +
document.getElementById('a').value + '&b=' + docu\
ment.getElementById('b').value;
            //console.log(url);
            ajax_get(url, function(data) {
                document.getElementById('add').innerHTML =
data['a'] + ' + ' + data['b']\
 + ' = ' + data['add'];
                document.getElementById('subtract').innerHTML
= data['a'] + ' - ' + data\
['b'] + ' = ' + data['subtract'];
                document.getElementById('multiply').innerHTML
= data['a'] + ' * ' + data\
['b'] + ' = ' + data['multiply'];
                document.getElementById('divide').innerHTML =
data['a'] + ' / ' + data['\
b'] + ' = ' + data['divide'];
            });
```

```
    });
})()
```

```
<html>
<head>
</head>
<body>
<input type="number" id="a">
<input type="number" id="b">
<button id="calc">Calc</button>
<div id="results">
   <div id="add"></div>
   <div id="subtract"></div>
   <div id="multiply"></div>
   <div id="divide"></div>
</div>

<pre id="info"></pre>
<div id="description"></div>

<script src="/static/math.js?r={{reload}}"></script>
</body>
</html>
```

Flask and AJAX

```
from flask import Flask, jsonify, render_template, request
import time
app = Flask(__name__)

@app.route("/")
def main():
    return render_template('main.html', reload = time.time())

@app.route("/api/info")
def api_info():
    info = {
       "ip" : "127.0.0.1",
       "hostname" : "everest",
       "description" : "Main server",
```

```
        "load" : [ 3.21, 7, 14 ]
    }
    return jsonify(info)

@app.route("/api/calc")
def add():
    a = int(request.args.get('a', 0))
    b = int(request.args.get('b', 0))
    div = 'na'
    if b != 0:
        div = a/b
    return jsonify({
        "a"        :  a,
        "b"        :  b,
        "add"      :  a+b,
        "multiply" :  a*b,
        "subtract" :  a-b,
        "divide"   :  div,
    })
```

```
$(function() {
    $.ajax({
        url: '/api/info',
        success: function(data) {
            console.log('get info');
            $('#info').html(JSON.stringify(data, null, '
'));
            $('#description').html(data['description']);
        }
    });

    $('#calc').click(function() {
        $('#info').css('display', "none");
        $('#description').css('display', "none");
        //console.log(url);
        $.ajax({
            url : '/api/calc?a=' +
document.getElementById('a').value + '&b=' + docu\
ment.getElementById('b').value,
            success: function(data) {
                $('#add').html(data['a'] + ' + ' +
data['b'] + ' = ' + data['add']);
                $('#subtract').html(data['a'] + ' - ' +
data['b'] + ' = ' + data['su\
```

```
btract']);
                    $('#multiply').html(data['a'] + ' * ' + data['b'] + ' = ' + data['mu\
ltiply']);
                    $('#divide').html(data['a'] + ' / ' + data['b'] + ' = ' + data['divi\
de']);
                }
            });
        });
})
```

```html
<html>
<head>
</head>
<body>
<input type="number" id="a">
<input type="number" id="b">
<button id="calc">Calc</button>
<div id="results">
   <div id="add"></div>
   <div id="subtract"></div>
   <div id="multiply"></div>
   <div id="divide"></div>
</div>

<pre id="info"></pre>
<div id="description"></div>

<script src="/static/jquery-3.1.1.min.js"></script>
<script src="/static/math.js?r={{reload}}"></script>
</body>
</html>
```

passlib

```python
from passlib.hash import pbkdf2_sha256
import sys

if len(sys.argv) != 2:
   exit("Usage: {} PASSWORD".format(sys.argv[0]))

```

```python
pw = sys.argv[1]

hash1 = pbkdf2_sha256.hash(pw)
print(hash1)

hash2 = pbkdf2_sha256.hash(pw)
print(hash2)

print(pbkdf2_sha256.verify(pw, hash1))
print(pbkdf2_sha256.verify(pw, hash2))
```

```
$ python use_passlib.py "my secret"
$pbkdf2-sha256$29000$svZ.7z2HEEJIiVHqPeecMw$QAWd8P7MaPDXlEwtsv9AqhFEP2hp8MvZ9QxasIw4\
Pgw
$pbkdf2-sha256$29000$XQuh9N57r9W69x6jtDaG0A$VtD35zfeZhXsE/jxGl6wB7Mjwj/5iDGZv6QC7XBJ\
jrI
True
True
```

Flask Testing

```python
from flask import Flask, jsonify
myapp = Flask(__name__)

@myapp.route("/")
def main():
    return '''
Main <a href="/add/23/19">add</a>
'''

@myapp.route("/add/<int:a>/<int:b>")
def api_info(a, b):
    return str(a+b)
```

```python
from app import myapp
import unittest

```

```python
# python -m unittest test_app


class TestMyApp(unittest.TestCase):
    def setUp(self):
        self.app = myapp.test_client()

    def test_main(self):
        rv = self.app.get('/')
        assert rv.status == '200 OK'
        assert b'Main' in rv.data
        #assert False

    def test_add(self):
        rv = self.app.get('/add/2/3')
        self.assertEqual(rv.status, '200 OK')
        self.assertEqual(rv.data, '5')

        rv = self.app.get('/add/0/10')
        self.assertEqual(rv.status, '200 OK')
        self.assertEqual(rv.data, '10')

    def test_404(self):
        rv = self.app.get('/other')
        self.assertEqual(rv.status, '404 NOT FOUND')
```

Flask Deploy app

```python
from flask import Flask
myapp = Flask(__name__)

@myapp.route("/")
def main():
    return 'Main'
```

uwsgi

```
[uwsgi]
socket    = :9091
plugin    = python
wsgi-file = /home/gabor/work/my/app.py
```

```
process    = 3
callable = myapp
```

nginx

```
server {
      server_name example.com;


      access_log  /var/log/nginx/example.log  main;
      error_log /var/log/nginx/example.error.log;

      location ~ /.git/ {
        deny all;
      }

      #error_page 404 /404.html;

      location '/' {
              include uwsgi_params;
              uwsgi_pass 127.0.0.1:9091;
      }

      root /home/gabor/work/example.com/html/;
 }
```

Flask Simple Authentication + test

```
from flask import Flask
from flask_httpauth import HTTPBasicAuth
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)
auth = HTTPBasicAuth()

users = {
    "john": generate_password_hash("nhoj"),
    "jane": generate_password_hash("enaj")
}

@app.route("/")
```

```python
def hello():
    return "Hello World!"

@auth.verify_password
def verify_password(username, password):
    if username in users:
        return check_password_hash(users.get(username), password)
    return False


@app.route("/admin")
@auth.login_required
def admin():
    return "Hello Admin"
```

```python
import app
import base64

def test_app():
    web = app.app.test_client()

    rv = web.get('/')
    assert rv.status == '200 OK'
    assert rv.data == b'Hello World!'

def test_admin_unauth():
    web = app.app.test_client()

    rv = web.get('/admin')
    assert rv.status == '401 UNAUTHORIZED'
    assert rv.data == b'Unauthorized Access'
    assert 'WWW-Authenticate' in rv.headers
    assert rv.headers['WWW-Authenticate'] == 'Basic realm="Authentication Required"'

def test_admin_auth():
    web = app.app.test_client()

    credentials = base64.b64encode(b'john:nhoj').decode('utf-8')
    rv = web.get('/admin', headers={
            'Authorization': 'Basic ' + credentials
    })
```

```python

    assert rv.status == '200 OK'
    assert rv.data == b'Hello Admin'
```

Flask REST API

- [flask-restful]()

Flask REST API - Echo

```python
from flask import Flask, request
from flask_restful import Api, Resource

app = Flask(__name__)

api = Api(app)

class Echo(Resource):
    def get(self):
        return { "prompt": "Type in something" }
    def post(self):
        return { "echo": "This" }

api.add_resource(Echo, '/echo')
```

```python
import api

def test_echo():
    web = api.app.test_client()

    rv = web.get('/echo')
    assert rv.status == '200 OK'
    assert rv.headers['Content-Type'] == 'application/json'
    assert rv.json == {"prompt": "Type in something"}


    rv = web.post('/echo')
    assert rv.status == '200 OK'
    assert rv.headers['Content-Type'] ==
```

```
'application/json'
    assert rv.json == {"echo": "This"}
```

Flask REST API - parameters in path

```python
from flask import Flask, request
from flask_restful import Api, Resource

app = Flask(__name__)

api = Api(app)

class Echo(Resource):
    def get(self, me):
        return { "res": f"Text: {me}" }

    def post(self, me):
        return { "Answer": f"You said: {me}" }


api.add_resource(Echo, '/echo/<me>')
```

```python
import api

def test_echo():
    web = api.app.test_client()

    rv = web.get('/echo/hello')
    assert rv.status == '200 OK'
    assert rv.headers['Content-Type'] == 'application/json'
    assert rv.json ==  {'res': 'Text: hello'}


    rv = web.post('/echo/ciao')
    assert rv.status == '200 OK'
    assert rv.headers['Content-Type'] == 'application/json'
    assert rv.json == {'Answer': 'You said: ciao'}
```

Flask REST API - parameter parsing

```python
from flask import Flask, request
from flask_restful import Api, Resource, reqparse

app = Flask(__name__)

api = Api(app)


class Echo(Resource):
    def get(self):
        parser = reqparse.RequestParser()
        parser.add_argument('text', help='Type in some text')
        args = parser.parse_args()
        return { "res": f"Text: {args['text']}" }

    def post(self):
        parser = reqparse.RequestParser()
        parser.add_argument('text', help='Type in some text')
        args = parser.parse_args()
        return { "Answer": f"You said: {args['text']}" }


api.add_resource(Echo, '/echo')
```

```python
import api

def test_echo():
    web = api.app.test_client()

    rv = web.get('/echo?text=hello')
    assert rv.status == '200 OK'
    assert rv.headers['Content-Type'] == 'application/json'
    assert rv.json ==  {'res': 'Text: hello'}

    rv = web.post('/echo', data={'text': 'ciao'})
    assert rv.status == '200 OK'
    assert rv.headers['Content-Type'] == 'application/json'
```

```python
    assert rv.json == {'Answer': 'You said: ciao'}


    # If the parameter is missing the parser just returns None
    rv = web.get('/echo')
    assert rv.status == '200 OK'
    assert rv.headers['Content-Type'] == 'application/json'
    assert rv.json ==  {'res': 'Text: None'}
```

Flask REST API - parameter parsing - required

```python
from flask import Flask, request
from flask_restful import Api, Resource, reqparse

app = Flask(__name__)

api = Api(app)


class Echo(Resource):
    def get(self):
        parser = reqparse.RequestParser()
        parser.add_argument('text', help='Type in some text', required=True)
        args = parser.parse_args()
        return { "res": f"Text: {args['text']}" }

    def post(self):
        parser = reqparse.RequestParser()
        parser.add_argument('text', help='Type in some text')
        args = parser.parse_args()
        return { "Answer": f"You said: {args['text']}" }


api.add_resource(Echo, '/echo')
```

```python
import api

def test_echo():
    web = api.app.test_client()

    rv = web.get('/echo?text=hello')
    assert rv.status == '200 OK'
    assert rv.headers['Content-Type'] == 'application/json'
    assert rv.json ==  {'res': 'Text: hello'}

    rv = web.post('/echo', data={'text': 'ciao'})
    assert rv.status == '200 OK'
    assert rv.headers['Content-Type'] == 'application/json'
    assert rv.json == {'Answer': 'You said: ciao'}


    # If the parameter is missing the parser just returns None
    rv = web.get('/echo')
    assert rv.status == '400 BAD REQUEST'
    assert rv.headers['Content-Type'] == 'application/json'
    assert rv.json == {'message': {'text': 'Type in some text'}}
```

Networking

Secure shell

- subprocess + external ssh client
- Paramiko
- Fabric

ssh

- On Windows install putty

```python
import subprocess
import sys

if len(sys.argv) !=2:
    exit("Usage: " + sys.argv[0] + " hostname")

host = sys.argv[1]
command = "uname -a"

ssh = subprocess.Popen(["ssh", host, command],
                       shell=False,
                       stdout=subprocess.PIPE,
                       stderr=subprocess.PIPE)
result = ssh.stdout.readlines()
error = ssh.stderr.readlines()
if error:
    for err in error:
        sys.stderr.write("ERROR: {}\n".format(err))
if result:
    print(result)
```

ssh from Windows

```
$ ssh foobar@hostname-or-ip
  -o "StrictHostKeyChecking no"

$ plink.exe -ssh foobar@hostname-or-ip -pw "password" -C "uname -a"
$ plink.exe", "-ssh", "foobar@username-or-ip", "-pw", "no secret", "-C", "uname -a"
```

```python
import subprocess
import sys

ssh = subprocess.Popen([r"c:\Users\foobar\download\plink.exe", "-ssh",
                        "foobar@username-or-ip",
                        "-pw", "password",
                        "-C", "uname -a"],
                          shell=False,
                          stdout=subprocess.PIPE,
                          stderr=subprocess.PIPE)
result = ssh.stdout.readlines()
error = ssh.stderr.readlines()
if error:
    for err in error:
        sys.stderr.write("ERROR: {}\n".format(err))
if result:
    print(result)
```

Parallel ssh

- parallel-ssh
- **pip install parallel-ssh**

```python
from pssh import ParallelSSHClient
hosts = ['myhost1', 'myhost2']
client = ParallelSSHClient(hosts)
output = client.run_command('ls -ltrh /tmp/', sudo=True)
```

telnet

```
import telnetlib

hostname  = '104.131.87.33'
user = 'gabor'
password = 'robag'

tn = telnetlib.Telnet(hostname)
tn.read_until("login: ")
tn.write(user + "\n")

tn.read_until("Password: ")
tn.write(password + "\n")
tn.read_until("~$")

tn.write("hostname\n")
print(tn.read_until("~$"))
print("-------");


tn.write("uptime\n")
print(tn.read_until("~$"))
print("-------");


print("going to exit")
tn.write("exit\n")

print("--------")
print(tn.read_all())
```

prompt for password

```
import getpass

password = getpass.getpass("Password:")

print(password)
```

Python nmap

```
import nmap
nm = nmap.PortScanner()
nm.scan('127.0.0.1', '20-1024')
print(nm.command_line())

for host in nm.all_hosts():
    print('--------------------------------------------------')
    print('Host : {} ({})'.format(host, nm[host].hostname()))
    print('State : {}'.format(nm[host].state()))
    for proto in nm[host].all_protocols():
        print('----------')
        print('Protocol : {}'.format(proto))

        lport = nm[host][proto].keys()
        for port in lport:
            print ('port : {}\tstate : {}'.format(port, nm[host][proto][port]['state\
']))
```

```
nmap -oX - -p 10-1024 -sV 127.0.0.1
--------------------------------------------------
Host : 127.0.0.1 ()
State : up
----------
Protocol : tcp
port : 21   state : open
port : 22   state : open
port : 23   state : open
```

ftp

```
$ sudo aptitude install proftpd
$ sudo /etc/init.d/proftpd start
$ sudo adduser   (user: foo pw: bar)
```

```
from ftplib import FTP
ftp = FTP('localhost')
ftp.login("foo", "bar")

```

```python
print(ftp.retrlines('LIST'))

print('-------')
for f in ftp.nlst():
    print("file: " + f)

filename = 'ssh.py'

ftp.storlines("STOR " + filename, open(filename))

print('-------')
for f in ftp.nlst():
    print("file: " + f)

ftp.delete(filename)

print('-------')
for f in ftp.nlst():
    print("file: " + f)
```

```
-rw-rw-r--   1 foo      foo             6 Feb 18 19:18 a.txt
-rw-rw-r--   1 foo      foo             6 Feb 18 19:18 b.txt
226 Transfer complete
-------
file: b.txt
file: a.txt
-------
file: b.txt
file: a.txt
file: ssh.py
-------
file: b.txt
file: a.txt
```

Interactive shell

The Python interactive shell

Type `python` without any arguments on the command line and you'll get into the Interactive shell of Python.
In the interactive shell you can type:

```
>>> print "hello"
hello

>>> "hello"
'hello'

>>> 6
6

>>> len("abc")
3

>>> "abc" + 6
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects

>>> "abc" + str(6)
'abc6'
```

REPL - Read Evaluate Print Loop

A variable comes to existence the first time we assign a value to it. It points to an object and that object knows about its type.

```
>>> a = "abc"
>>> len(a)
3
```

```
>>> a = '3'
>>> a + 3
Traceback (most recent call last):
  File "<stdin>", line 1, in &lt;module>
TypeError: cannot concatenate 'str' and 'int' objects

>>> int(a) + 3
6

>>> a = '2.3'
>>> float(a) + 1
3.3
```

Using Modules

Python has lots of standard (and not standard) modules. You can load one of them using the `import` keyword. Once loaded, you can use functions from the module or access its objects. For example the `sys` module has a `sys.version` and a `sys.executable` variable.

```
>>> import sys
>>> sys.version
'2.7.3 (default, Apr 10 2012, 23:24:47) [MSC v.1500 64 bit (AMD64)]'
```

```
>>> sys.executable
'c:\\Python27\\python.exe'
```

You can also load specific object directly into your code.

```
>>> from sys import executable
>>> executable
'c:\\Python27\\python.exe'
```

To quit the interpreter call the `exit()` function.

```
>>> exit
Use exit() or Ctrl-Z plus Return to exit
```

The `import` binds the word sys to whatever it loaded from the file.

Getting help

```
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> help()    - entering an internal shell:
...
help> dir     - explains about the dir command.  Navigate using SPACE/ENTER/q
help> Ctrl-D  - to quite, (Ctrl-Z ENTER on Windows)
>>> help(dir) - the same explanation as before

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>> dir("")    - list of string related methods
['__add__', '__class__', ... 'upper', 'zfill']

>>> dir(1)     - list of integer related methods
['__abs__', '__add__', ... 'numerator', 'real']

>>> dir(__builtins__)
...                       - functions available in python

>>> help(abs)             - exlain how abs() works
>>> help(sum)
>>> help(zip)
>>> help(int)
>>> help(str)

>>> help("".upper)    - explain how the upper method of strings work

>>> import sys
>>> dir(sys)
>>> help(sys)
```

```
>>> help(sys)
>>> help(sys.path)
>>> help(sys.path.pop)
```

Exercise: Interactive shell

- Start the REPL and check the examples.
- Check the documentation in the REPL.

Testing Demo

How do you test your code?

* What kind of things do you test?

- Web application?
- Command line application?
- Databases?
- …

What is testing?

- Fixture + Input = Expected Output

What is testing really?

- Fixture + Input = Expected Output + Bugs

Testing demo - AUT - Application Under Test

Given the following module with a single function, how can we use this function and how can
we test it?

```
def add(x, y):
    return x * y

# Yes, I know there is a bug in this code!
```

Testing demo - use the module

```python
import mymath
import sys

if len(sys.argv) != 3:
    exit(f"Usage {sys.argv[0]} NUMBER NUMBER")

a = int(sys.argv[1])
b = int(sys.argv[2])

print(mymath.add(a, b) )
```

```
python use_mymath.py 2 2
4
```

Testing demo: doctets

```python
def add(x, y):
    """
    >>> add(2, 2)
    4
    """
    return x * y
```

```
python -m doctest mymath_doctest_first.py
echo $?
0

echo %ERRORLEVEL%
0
```

```python
def add(x, y):
    """
    >>> add(2, 2)
    4
    >>> add(3, 3)
    6
```

```
    """
    return x * y
```

```
python -m doctest mymath_doctest.py
echo $?
1
```

```
**********************************************************************
File "/home/gabor/work/slides/python/examples/testing-demo/mymath_doctest.py", line \
5, in mymath_doctest.add
Failed example:
    add(3, 3)
Expected:
    6
Got:
    9
**********************************************************************
1 items had failures:
   1 of   2 in mymath_doctest.add
***Test Failed*** 1 failures.
```

Testing demo: Unittest success

```
import unittest
import mymath

class TestMath(unittest.TestCase):
    def test_math(self):
        self.assertEqual(mymath.add(2, 2), 4)
```

```
python -m unittest test_one_with_unittest.py
echo $?
0
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Testing demo: Unittest failure

```
import unittest
import mymath

class TestMath(unittest.TestCase):
    def test_math(self):
        self.assertEqual(mymath.add(2, 2), 4)

    def test_more_math(self):
        self.assertEqual(mymath.add(3, 3), 6)
```

```
python -m unittest test_with_unittest.py
echo $?
1
```

```
.F
======================================================================
FAIL: test_more_math (test_with_unittest.TestMath)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/gabor/work/slides/python/examples/testing-demo/test_with_unittest.py",\
 line 9, in test_more_math
    self.assertEqual(mymath.add(3, 3), 6)
AssertionError: 9 != 6

----------------------------------------------------------------------
Ran 2 tests in 0.000s
```

```

FAILED (failures=1)
```

Testing demo: pytest using classes

```python
import mymath

class TestMath():
    def test_math(self):
        assert mymath.add(2, 2) == 4

    def test_more_math(self):
        assert mymath.add(3, 3) == 6
```

```
pytest test_with_pytest_class.py
```

```
============================= test session starts ==============================
platform linux -- Python 3.7.3, pytest-5.1.1, py-1.8.0, pluggy-0.13.0
rootdir: /home/gabor/work/slides/python/examples/testing-demo
plugins: flake8-1.0.4
collected 2 items

test_with_pytest_class.py .F [100%]

=================================== FAILURES ===================================
____________________________ TestMath.test_more_math ____________________________

self = <test_with_pytest_class.TestMath object at 0x7fc1ea617828>

    def test_more_math(self):
>       assert mymath.add(3, 3) == 6
E       assert 9 == 6
E        +  where 9 = <function add at 0x7fc1ea6caf28>(3, 3)
```

```
E         +    where <function add at 0x7fc1ea6caf28> = mymath.add

test_with_pytest_class.py:8: AssertionError
========================= 1 failed, 1 passed in 0.03s ==========================
```

Testing demo: pytest without classes

```python
import mymath

def test_math():
    assert mymath.add(2, 2) == 4

def test_more_math():
    assert mymath.add(3, 3) == 6
```

```
pytest test_with_pytest.py
```

```
============================= test session starts ==============================
platform linux -- Python 3.7.3, pytest-5.1.1, py-1.8.0, pluggy-0.13.0
rootdir: /home/gabor/work/slides/python/examples/testing-demo
plugins: flake8-1.0.4
collected 2 items

test_with_pytest.py .F [100%]

=================================== FAILURES ===================================
________________________________ test_more_math ________________________________

    def test_more_math():
>       assert mymath.add(3, 3) == 6
E       assert 9 == 6
E        +  where 9 = <function add at 0x7f36e78db0d0>(3, 3)
```

```
E         +    where <function add at 0x7f36e78db0d0> = mymath.add

test_with_pytest.py:7: AssertionError
========================= 1 failed, 1 passed in 0.02s ==========================
```

Testing demo: pytest run doctests

```
pytest --doctest-modules mymath_doctest_first.py
pytest --doctest-modules mymath_doctest.py
```

Testing demo: pytest run unittest

```
pytest -v test_with_unittest.py
```

Exercise: Testing demo

- An anagram is a pair of words that are created from exactly the same set of characters, but of different order.
- For example **listen** and **silent**
- Or **bad credit** and **debit card**
- Given the following module with the **is_anagram** function write tests for it. (in a file called test_anagram.py)
- Write a failing test as well.
- Try **doctest**, **unittest**, and **pytest** as well.

```
def is_anagram(a_word, b_word):
    return sorted(a_word) == sorted(b_word)
```

Sample code to use the Anagram module.

```
from anagram import is_anagram
import sys
```

```python

if len(sys.argv) != 3:
    exit(f"Usage {sys.argv[0]} WORD WORD")

if is_anagram(sys.argv[1], sys.argv[2]):
    print("Anagram")
else:
    print("NOT")
```

Solution: Testing demo

```python
from anagram import is_anagram

def test_anagram():
    assert is_anagram("silent", "listen")
    assert is_anagram("bad credit", "debit card")

def test_not_anagram():
    assert not is_anagram("abc", "def")

def test_should_be_anagram_spaces():
    assert is_anagram("anagram", "nag a ram")


def test_should_be_anagram_case():
    assert is_anagram("Silent", "Listen")
```

Types in Python

mypy

- mypy
- Type Checking
- type hints

```
pip install mypy
```

Types of variables

```
x :int = 0

x = 2
print(x)

x = "hello"
print(x)
```

```
python variables.py
```

```
2
hello
```

```
mypy variables.py
```

```
variables.py:7: error: Incompatible types in assignment (expression has type "str", \
variable has type "int")
Found 1 error in 1 file (checked 1 source file)
```

Types of function parameters

```
def add(a :int, b :int) -> int:
    return a+b

print(add(2, 3))
print(add("Foo", "Bar"))
```

```
5
FooBar
```

```
function.py:6: error: Argument 1 to "add" has incompatible type "str"; expected "int"
function.py:6: error: Argument 2 to "add" has incompatible type "str"; expected "int"
Found 2 errors in 1 file (checked 1 source file)
```

Types used properly

```
def add(a :int, b :int) -> int:
    return a+b

print(add(2, 3))

x :int = 0

x = 2
print(x)
```

```
5
2
```

```
Success: no issues found in 1 source file
```

TODO: mypy

- Complex data structures?
- My types/classes?
- Allow None (or not) for a variable.

Testing Intro

The software testing equasion

```
INPUT + PROCESS = EXPECTED_OUTPUT
```

The software testing equasion (fixed)

```
INPUT + PROCESS = EXPECTED_OUTPUT + BUGS
```

The pieces of your software?

- Web application with HTML interface?
- Web application with HTML + JavaScript? Which frameworks?
- Web application with JSON interface? (API)
- What kind of databases do you have in the system? SQL? NoSQL? What size is the database?
- Source and the format of your input? Zip? CSV? XML? SQL Dump? JSON?
- The format of your output? HTML/PDF/CSV/JSON/XML/Excel/Email/..?
- Are you pushing out your results or are your cliens pulling them? e.g. via an API?
- What external dependencies do you have? Slack, Twilio, What kind of APIs?

Manual testing

- How do you check your application now?

What to tests?

- Happy path
- Sad path

- Valid input
- Valid edge cases (0, -1, empty string, etc.)
- Broken input (string instead of number, invalid values, too long input, etc.)
- Extreme load
- System failure (power failure, network outage, lack of memory, lack of disk, …)
- Third-party error or failure - How does your system work if the 3rd party API returns rubbish?

Continuous Integration

- Reduce feedback cycle
- Avoid regression
- On every push
- Every few hours full coverage

Functional programming

Functional programming

- Immutability (variables don't change)
- Separation of data and functions.
- First-class functions (you can assign function to another name and you can pass function to other functions and return them as well. We can also manipulate functions)

- Higher order functions: a functions that either takes a function as a parameter or returns a function as a parameter.

Iterators (Iterables)

You already know that there are all kinds of objects in Python that you can iterate over using the **for in** construct.
For example you can iterate over the characters of a string, or the elements of a list, or whatever **range()** returns.
You can also iterate over the lines of a file
and you have probably seen the **for in** construct in other cases as well. The objects that can be iterated over are collectively called
iterables.
You can do all kind of interesting things on such **iterables**.
We'll see a few now.

- A few data type we can iterate over using the **for … in …** construct. (strings, files, tuples, lists, list comprehension)

```
numbers = [101, 2, 3, 42]
for num in numbers:
    print(num)
print(numbers)

print()

name = "FooBar"
for cr in name:
    print(cr)
print(name)

print()

rng = range(3, 9, 2)
for num in rng:
    print(num)
print(rng)
```

```
101
2
3
42
[101, 2, 3, 42]

F
o
o
B
a
r
FooBar

3
5
7
range(3, 9, 2)
```

range

So what does **range** really return?

Instead of returning the list of numbers (as it used to do in Python 2), now it returns a **range object** that provides “the opportunity to go over
the specific series of numbers” without actually creating the **list** of numbers. Getting an object instead of the whole list has a number of advantages.
One is space. In the next example we’ll see how much memory is needed for the object returned by the **range** function and how much would it take to have the corresponding list of numbers in memory. For now let’s see how can we use it:

- `range(start, end, step)`
- `range(start, end)` - step defaults to 1
- `range(end)` - start defaults to 0, step defaults to 1

```
rng = range(3, 9, 2)
print(rng)
print(type(rng).__name__)

print()

for num in rng:
    print(num)

print()

for num in rng:
    print(num)
```

```

print()

print(rng[2])
```

```
range(3, 9, 2)
range

3
5
7

3
5
7

7
```

range with list

Using the **list** function we can tell the **range** object to generate the whole list immediately. Either using
the variable that holds the **range** object, or wrapping the **range()** call in a **list()** call.

You might recall at the beginning of the course we saw the **sys.getsizeof()** function that returns the size of a Python object in the memory. If you don't recall, no problem, we'll see it used now. As you can see the size of the range object is only 48 bytes
while the size of the 3-element list is already 112 bytes. It seems the range object is better than even such a short lists. On the next page we'll see a more detailed analyzis.

```
import sys

rng = range(3, 9, 2)
numbers = list(rng)
print(rng)          # range(3, 9, 2)
print(numbers)      # [3, 5, 7]

others = list(range(2, 11, 3))
print(others)    # [2, 5, 8]

print(sys.getsizeof(rng))      # 48
print(sys.getsizeof(numbers)) # 112
```

range vs. list size

In this example we have a loop iterating over **range(21)**, but that's only for the convenience, the interesting part is inside the loop.
On every iteration call **range()** with the current number, then we convert the resulting object into a list of numbert. Finally we print out
the current number and the size of both the object returned by **range()** and the list generated from the object. As you can see the memory usage
of the **range** object remains the same 48 byttes, while the memory usage of the list growth as the list gets longer.

```
import sys

for ix in range(21):
    rng = range(ix)
    numbers = list(rng)
    print("{:>3} {:>3} {:>4}".format(ix,
sys.getsizeof(rng), sys.getsizeof(numbers)))
```

```
 0  48   64
 1  48   96
 2  48  104
 3  48  112
 4  48  120
 5  48  128
 6  48  136
 7  48  144
 8  48  160
 9  48  192
10  48  200
11  48  208
12  48  216
13  48  224
14  48  232
15  48  240
16  48  256
17  48  264
18  48  272
19  48  280
20  48  288
```

for loop with transformation

There are many cases when we have a list of some values and we need to apply some transformation to each value. At the end we would like to have the list of the resulting values.

A very simple such transformation would be to double each value. Other, more interesting examples might be reversing each string, computing some more complex function on each number, etc.)

In this example we just double the values and use **append** to add each value to the list containing the results.

```
def double(n):
    return 2 * n

numbers = [1, 2, 3, 4]
name = "FooBar"

double_numbers = []

for num in numbers:
    double_numbers.append( double(num) )
print(double_numbers)

double_letters = []
for cr in name:
    double_letters.append( double(cr) )
print(double_letters)
```

```
[2, 4, 6, 8]
['FF', 'oo', 'oo', 'BB', 'aa', 'rr']
```

There are better ways to do this.

map

- `map(function, iterable, ...)`

The map function of Python applies a function to every item in an iterable and returns an iterator
that can be used to iterate over the results. Wow, how many times I repeated the word iter…something. Instead of trying to untangle that sentence,
let’s look at the following exampe:

We have a list of numbers in the brilliantly named variable `numbers` with 1, 2, 3, 4 as the content. We could like to ceate a list of all the doubles (so that would be 2, 4, 6, 8 in this casse) and then iterate over them printing them on the screen. Sure, you probably have some more complex operation to do on the numbers than simple double them, but in this example I did not want to complicate
that part. Suffice to say that you have some computation to do in every element.

So you encapsulate your computation in a regular Python function (in our case the function is called **double**). Then you call **map** and pass to it two parameters. The first parameter is the **double** function itself, the second parameter is the list of the values you would like to work on. `map` will no go over all the values in the **numbers** list, call the **double** function with each number and provide allow you to iterate over the results. Something like this:

double_numbers = [double(1), double(2), double(3), double(4)]

Except, that the above is not true.

When Python executes the `double_numbers = map(double, numbers)` line, no computation happens and no resulting list is created. Python only prepars “the possibility to do the computations”. In the upcoming examples we’ll see what does this sentence really mean, for now let’s see what do we have in this example: `double_numbers` contains a **map object*, but when you iterate
over it using the **for num in double_numbers** construct you get the expected values.

In the second half of the example you can see the same works on strings as well.

```
def double(n):
    return 2 * n

numbers = [1, 2, 3, 4]
name = "FooBar"

double_numbers = map(double, numbers)
print(double_numbers)   # <map object at 0x7f8eb2d849e8>
```

```
for num in double_numbers:
    print(num)

double_letters = map(double, name)
print(double_letters)   # <map object at 0x7f8eb2d84cc0>
for cr in double_letters:
    print(cr)
```

```
<map object at 0x7ff0c0d89da0>
2
4
6
8
<map object at 0x7ff0c0d89a20>
FF
oo
oo
BB
aa
rr
```

map delaying function call

In this example we have added a call to `print` in the double function in order to see when is it really executed. You can see that the first output
comes from the `print` statement that was **after** the `map` call. Then on each iteration we see the output from inside the "double" function and then the
result from the loop. In a nutshell Python does not execute the "double" function at the point where we called `map`. It only executes it when we iterate over
the resulting object.

```
def double(n):
    print(f"double {n}")
```

```
    return 2 * n

numbers = [1, 4, 2, -1]

double_numbers = map(double, numbers)
print(double_numbers)

for num in double_numbers:
    print(num)
```

```
<map object at 0x7f90df760f98>
double 1
2
double 4
8
double 2
4
double -1
-2
```

map on many values

Now imagine you have a very long list. I know this is not such a long list, but I trust you can imagin a long list of numbers. We would like to run some function on each element and then iterate over the results, but what if at one point in the iteration we decide to `break` out of the loop?

```
import sys

def double(n):
    print(f"double {n}")
    return 2 * n

numbers = [1, 4, 2, -1, 23, 12, 5, 6, 34, 143123, 98,
213]
```

```

double_numbers = map(double, numbers)
print(double_numbers)
for num in double_numbers:
    print(num)
    if num > 42:
        break

print()
print(sys.getsizeof(numbers))
print(sys.getsizeof(double_numbers))
```

```
<map object at 0x7fe5c5270d68>
double 1
2
double 4
8
double 2
4
double -1
-2
double 23
46

160
56
```

You can see that it did not need to waste time calculating the doubles of all the values, as it was calculating on-demand. You can also see that the object returned from `map` takes up only 56 bytes. Regardless of the size of the original array.

map with list

Here too you can use the **list** function to convert all the values at once, but there is an advantage of keeping it as a **map object**. Not only the size that we already saw with the **range** case, but also the processing time saved by not calculating the results till you actually need it.

Imagine a case where you apply several expensive (time consuming) transformations to some original list and then you iterate over the end-results looking for the first value that matches some condition. What if you find the value you were looking for after only a few iteration. Then making all that expensive calculations to the whole list was a waste of time.

This lazy evaluation can help you save both memory and time and you always have the option to force the immediate calculation by calling the **list** function.

```
def double(num):
    return 2 * num

numbers = [1, 2, 3, 4]
name = "FooBar"

double_numbers = list(map(double, numbers))
print(double_numbers)

double_letters = list(map(double, name))
print(double_letters)
```

```
[2, 4, 6, 8]
['FF', 'oo', 'oo', 'BB', 'aa', 'rr']
```

double with lambda

There are many other cases besides **map** where we need to pass a function as a parameter to some other function. Many cases the function we pass is some almost trivial function with a single operation in it. In those cases creating a named function like the “double” function in the previous examples is an overkill.

In this example we also used the **list** function to force the full evaluation of the map object to make it easier to show the results. Normally you probably would not use the **list** function here.

```
numbers = [1, 2, 3, 4]
name = "FooBar"


double_numbers = list( map( lambda n: n * 2, numbers) )
print(double_numbers)


double_letters = map( lambda n: n * 2, name)
for cr in double_letters:
    print(cr)
```

```
[2, 4, 6, 8]
FF
```

```
oo
oo
BB
aa
rr
```

What is lambda in Python?

Lambda creates simple anonymous function. It is simple because it can only have one statement in its body. It is anonymous because usually it does not have a name.

The usual use is as we saw earlier when we passed it as a parameter to the `map` function. However, in the next example we show that you can assign the lambda-function to a name and then you could used that name just as any other function you would define using **def**.

```
def dbl(n):
    return 2*n
print(dbl(3))

double = lambda n: 2*n
print(double(3))
```

```
6
6
```

lambda returning tuple

A lambda function can return complex data structures as well. e.g. a tuple.

```
dbl = lambda n: (n, 2*n)

ret = dbl(12)

print(ret)
```

```
(12, 24)
```

map returning tuples

```
numbers = [1, 2, 3, 4]

pairs = map(lambda n: (n, 2*n), numbers)
print(pairs)

for pair in pairs:
    print(pair)
```

```
<map object at 0x7fcd264a15f8>
(1, 2)
(2, 4)
(3, 6)
(4, 8)
```

lambda with two parameters

A **lambda**-function can have more than one parameters:

```
add = lambda x,y: x+y
print(add(2, 3))
```

```
5
```

map for more than one iterable

Lets “add” together two lists of numbers. Using + will just join the two lists together, but we can use the “map” function to add the values pair-wise.

```
v1 = [1, 3, 5, 9]
v2 = [2, 6, 4, 8]

v3 = v1 + v2
print(v3)

sums = map(lambda x,y: x+y, v1, v2)
print(sums)
print(list(sums))
```

```
[1, 3, 5, 9, 2, 6, 4, 8]
<map object at 0x7fcbecc8c668>
[3, 9, 9, 17]
```

map on uneven lists

In **Python 3** the iterator stops when the shortest iterable is exhausted.

In **Python 2** it used to extend the shorter lists by **None** values.

```
v1 = [1, 3, 5, 9]
v2 = [2, 6, 4, 8, 10]

sums = map(lambda x,y: x+y, v1, v2)
print(sums)

print(list(sums))
```

```
<map object at 0x7ff9469a8da0>
[3, 9, 9, 17]
```

replace None (for Python 2)

In Python 2 map used to extend the shorter lists by **None** values.
So to avoid exceptions, we had some exra code replacing the None values by 0, using the ternary operator.

```
v1 = [1, 3, 5, 9]
v2 = [2, 6, 4, 8, 10]

print(map(lambda x,y: (0 if x is None else x) + (0 if y
is None else y), v1, v2))
# [3, 9, 9, 17, 10]
```

map on uneven lists - fixed (for Python 2)

A nicer fix was this:

```
v1 = [1, 3, 5, 9]
v2 = [2, 6, 4, 8, 10]
```

```

print(map(lambda x,y: (x or 0) + (y or 0), v1, v2))
# [3, 9, 9, 17, 10]
```

map mixed iterators

map works on any iterable, so we might end up passing one list and one string to it.

```
v1 = ['foo', 'bar', 'baz']
v2 = 'abc'

result = map(lambda x,y: x+y, v1, v2)
print(result)
print( list(result) )
```

```
<map object at 0x7fc5e9ff4e80>
['fooa', 'barb', 'bazc']
```

map fetch value from dict

```
people = [
    {
        'name': 'Foo',
        'phone': '123',
    },
    {
        'name': 'Bar',
        'phone': '456',
    },
    {
        'name': 'SnowWhite',
        'phone': '7-dwarfs',
    }
]

```

```
names = map(lambda d: d['name'], people)

print(names)
print(list(names))
```

```
<map object at 0x7f5afffaeb00>
['Foo', 'Bar', 'SnowWhite']
```

Exercise: string to length

Given a list of strings, create an iterator that will provide the length of each string.

Exercise: row to length

Given a file, create an iterator that will provide the length of each row. Can you do it without actually reading the file?

Exercise: compare rows

Create an iterator that given two files will return true for each line where the first space in the first file is earlier than the first space in the second file. So

- given: “ab cd” vs “abc d” the value is true
- given: “ab cd” vs “ab cd” the value is false
- given: “ab cd” vs “a bcd” the value is false

Solution: string to length

```
animals = ['chicken', 'cow', 'snail', 'elephant', 'pig',
'zebra', 'gnu', 'praying ma\
ntiss', 'snake']

length = map(len, animals)
```

```
print(length)
print(list(length))
```

Solution: row to length

```
filename = __file__ # just being lazy and using ourselves as the input file

with open(filename) as fh:
    length = map(len, fh)
    print(length)
    for ln in length:
        print(ln)
        # if ln > 10:
        #     break
```

Solution: compare rows

```python
import sys

file_a = 'map_string_to_len.py'
file_b = 'map_row_to_length.py'

def compare(row_a, row_b):
    a = row_a.find(' ')
    b = row_b.find(' ')
    return a < b

with open(file_a) as fh_a, open(file_b) as fh_b:
    results = map(compare, fh_a, fh_b)
    print(results)
    print(sys.getsizeof(results))

    truth = list(results)
    print(truth)
    print(sys.getsizeof(truth))
```

```
<map object at 0x7f0858d3f8d0>
56
```

```
[False, True, False, True, True]
128
```

filter

- `filter(function, iterable)`

> `filter` will return an iterable object that will return all the items of the original iterable that evaluate the function to **True**. This can have only one iterable!

```
numbers = [1, 3, 27, 10, 38]
def big(n):
    return n > 10

reduced = filter(big, numbers)
print(reduced)
print(list(reduced))
```

```
<filter object at 0x7f4bc37355c0>
[27, 38]
```

filter with lambda

```
numbers = [1, 3, 27, 10, 38]

reduced = filter(lambda n: n > 10, numbers)
print(reduced)
print(list(reduced))
```

```
<filter object at 0x7faed0fe57b8>
[27, 38]
```

filter - map example

```
numbers = [1, 7, 19, 5, 57,  23, 8]

def big(x):
    print(f"filtering {x}")
    return x > 10

def double(y):
    print(f"double {y}")
    return 2*y

big_numbers = filter(big, numbers)
print(big_numbers)

doubles = map(double,  big_numbers)
print(doubles)

for num in doubles:
    print(num)
```

```
<filter object at 0x7ffad9f82f28>
<map object at 0x7ffad9f829e8>
filtering 1
filtering 7
filtering 19
double 19
38
filtering 5
filtering 57
double 57
114
filtering 23
double 23
46
filtering 8
```

filter - map in one expression

```
numbers = [1, 7, 19, 5, 57,  23, 8]

```

```
def big(x):
    print(f"filtering {x}")
    return x > 10

def double(y):
    print(f"double {y}")
    return 2*y


for num in map(double,  filter(big, numbers)):
    print(num)
```

```
filtering 1
filtering 7
filtering 19
double 19
38
filtering 5
filtering 57
double 57
114
filtering 23
double 23
46
filtering 8
```

Get indexes of values

`filter` can help us get a sublist of values from an iterable, eg. from a list that match some condition.
In this example we see how to get all the names that are exactly 3 characters long.

What if, however if instead of the values themselves, you would like to know their location? The indexes of the places where these value can be found. In that case, you would run the `filter` on the indexes from 0 till the last valid index of the list. You can do that using the `range` function.

Finally there is another example that shows how to get the indexes of all the names that have an “e” in them.
Just to show you that we can use any arbitray condition there.

```
names = ["Helen", "Ann", "Mary", "Harry", "Joe", "Peter"]
names3 = filter(lambda w: len(w) == 3, names)
print( list(names3) )

loc3 = filter(lambda i: len(names[i]) == 3, range(len(names)))
print( list(loc3) )


has_e = filter(lambda i: "e" in names[i], range(len(names)))

print( list(has_e) )
```

```
['Ann', 'Joe']
[1, 4]
[0, 4, 5]
```

reduce

In Python 2 it was still part of the language.

```
reduce(function, iterable[, initializer])
```

```python
from functools import reduce

numbers = [1, 2, 3, 4]

print(reduce(lambda x,y: x+y, numbers))     # 10 = ((1+2)+3)+4
print(reduce(lambda x,y: x*y, numbers))     # 24 = ((1*2)*3)*4
print(reduce(lambda x,y: x/y, [8, 4, 2]))   # 1.0

print(reduce(lambda x,y: x+y, [2]))         # 2
print()

# print(reduce(lambda x,y: x+y, []))
    # TypeError: reduce() of empty sequence with no initial value
print(reduce(lambda x,y: x+y, [], 0))       # 0
print(reduce(lambda x,y: x+y, [2,4], 1))    # 7
print()

mysum = 0
for num in numbers:
    mysum += num
print(mysum)          # 10

mymultiple = 1
for num in numbers:
    mymultiple *= num
print(mymultiple)     #24
```

```
10
24
1.0
2

0
7

10
24
```

The initializer is used as the 0th element returned by the iterable. It is mostly interesting in case the iterable is empty.

reduce with default

```
from functools import reduce

print( reduce(lambda x,y: x+y, [], 0) )      # 0
print( reduce(lambda x,y: x+y, [1, 2], 0) )  # 3

print( reduce(lambda x,y: x*y, [1, 2], 0) )  # 0
print( reduce(lambda x,y: x*y, [2, 3], 1) )  # 6
print( reduce(lambda x,y: x*y, [], 0) )      # 0
```

```
0
3
0
6
0
```

zip

```
fname = ['Graham',          'Eric',            'Terry',
          'Terry',          'John',            'Michael']
lname = ['Chapman',         'Idle',            'Gilliam',
          'Jones',          'Cleese',          'Palin']
born  = ['8 January 1941',  '29 March 1943',   '22 November 1940',
          '1 February 1942', '27 October 1939', '5 May 1943']

for f_name, l_name, b_date in zip(fname, lname, born):
    print("{:10} {:10} was born {}".format(f_name, l_name, b_date))
```

```
Graham     Chapman    was born 8 January 1941
Eric       Idle       was born 29 March 1943
Terry      Gilliam    was born 22 November 1940
Terry      Jones      was born 1 February 1942
```

```
John        Cleese     was born 27 October 1939
Michael     Palin      was born 5 May 1943
```

[Monty Python](#)

Creating dictionary from two lists using zip

```
names = ['Jan', 'Feb', 'Mar', 'Apr']
days =  [31, 28, 31, 30]

zipped = zip(names, days)
print(zipped)

pairs = list(zipped)
print(pairs)

month = dict(zipped)
print(month)    # this is empty because zipped was already
exhausted by the "list" ca\
ll

zipped = zip(names, days)
month = dict(zipped)
print(month)
```

```
<zip object at 0x7ff021949788>
[('Jan', 31), ('Feb', 28), ('Mar', 31), ('Apr', 30)]
{}
{'Jan': 31, 'Feb': 28, 'Mar': 31, 'Apr': 30}
```

all, any

- `all(iterable)` - returns True if all the elements of iterable return True
- `any(iterable)` - returns True if any of the elements in iterable return True

```
a = [True, True]
b = [True, False]
c = [False, False]

print(all(a))   # True
print(all(b))   # False
print(all(c))   # False
print()
print(any(a))   # True
print(any(b))   # True
print(any(c))   # False
```

Compare elements of list with scalar

```
print(2 > 1) # True
print(0 > 1) # False
print()

numbers = [2, 4]
# Comparing different types does not make sense, but nevertheless Python 2 would sti\
ll do it.
# Python 3 raises exception:
# TypeError: '>' not supported between instances of 'list' and 'int'
# print(numbers > 1) # True
# print(numbers > 7) # True
# print()

# compare each element with the scalar and then check if 'all' were True
print(all(map(lambda x: x > 1, numbers)))  # True
print(all(map(lambda x: x > 2, numbers)))  # False
```

List comprehension - double

We take the original example where we had a function called double, and this time we write a different expression to run the function on every element of an iterable.

```
def double(n):
    return 2*n

numbers = [1, 2, 3, 4]
name = "FooBar"

double_numbers = [double(n) for n in numbers]
print(double_numbers) # [2, 4, 6, 8]


double_chars = [double(n) for n in name]
print(double_chars)   # ['FF', 'oo', 'oo', 'BB', 'aa',
'rr']
```

List comprehension - simple expression

```
import sys

numbers  = [0, 1, 2, 3]

sqrs = map(lambda n: n*n, numbers)
print(sqrs)          # <map object at 0x7fdcab2f5940>
print(list(sqrs))    # [0, 1, 4, 9]
print(sys.getsizeof(sqrs))
print()

squares = [n*n for n in numbers]
print(squares)   # [0, 1, 4, 9]
print(sys.getsizeof(squares))
```

```
<map object at 0x7fa9cf2eb9e8>
[0, 1, 4, 9]
56

[0, 1, 4, 9]
96
```

List generator

Going over the values of the generator will empty the generator.

```
import sys

numbers  = [0, 1, 2, 3, 4, 5, 6]

gn = (n*n for n in numbers)
print(gn)
print(sys.getsizeof(gn))
print()

for num in gn:
    print(num)
print()

gn = (n*n for n in numbers)
squares = list(gn)
print(sys.getsizeof(squares))
print(squares)

print(list(gn)) # the generator was already exhausted
```

```
<generator object <genexpr> at 0x7f8c0bda2930>
120

0
1
4
9
16
25
36

160
[0, 1, 4, 9, 16, 25, 36]
[]
```

List comprehension

```
text = ['aaaa', 'bb', 'ccc ccc']

length_1 = map(lambda x: len(x), text)
print(length_1)         # <map object at 0x7f60ceb90f98>
print(list(length_1)) # [4, 2, 7]


length_2 = map(len, text)
print(length_2)         # <map object at 0x7f60ceb90fd0>
print(list(length_2)) # [4, 2, 7]


length_3 = [ len(s)  for s in text ]
print(length_3) # [4, 2, 7]
```

In LISP this would be a mapcar.

Dict comprehension

```
people = {
    'Foo':       123,
    'Bar':       456,
    'SnowWhite': 7,
}

doubles = { k:v*2 for (k, v) in people.items() }
print(doubles)  # {'Foo': 246, 'Bar': 912, 'SnowWhite':
14}
```

Lookup table with lambda

```
import sys

table = {
    "cat"  : lambda : print("miau"),
    "dog"  : lambda : print("hauhau"),
```

```
    "duck" : lambda : print("hap hap"),
}


def main():
    if len(sys.argv) != 2:
        exit(f"Usage: {sys.argv[0]} NAME")

    animal = sys.argv[1]
    if animal in table:
        table[animal]()

main()
```

Read lines without newlines

```
import sys

if len(sys.argv) != 2:
    exit(f"Usage: {sys.argv[0]}")

filename = sys.argv[1]

with open(filename) as fh:
    rows = map(lambda s: s.rstrip("\n"), fh.readlines())

for row in rows:
    print(row)
```

Read key-value pairs

```
name=Foo Bar
email=foo@bar.com
address=Foo street 42
```

```
import sys

if len(sys.argv) != 2:
    exit(f"Usage: {sys.argv[0]}")

```

```
filename = sys.argv[1]

with open(filename) as fh:
    pairs = dict(map(lambda x: x.split('='), map(lambda s: s.rstrip("\n"), fh.readlines())))

print(pairs)
```

```
{'name': 'Foo Bar', 'email': 'foo@bar.com', 'address': 'Foo street 42'}
```

Create index-to-value mapping in a dictionary based on a list of values

```
planned_order = ('b', 'c', 'd', 'a')
plan = dict(zip(range(len(planned_order)), planned_order))
print(plan)
```

```
{0: 'b', 1: 'c', 2: 'd', 3: 'a'}
```

Exercise: min, max, factorial

- Implement an expression to calculate “min”, and another expression to calculate “max” of lists.
- Implement an expression that will calculate factorial. f(n) should return the value of n! (n! = n * (n-1) * (n-2) * … * 1)
- Implement an expression that given 2 lists will return a new list in which each element is the max() for each pair from the input lists. E.g. given [1, 3, 6] and [2, 4, 5] the result is [2, 4, 6]
- Use reduce, map, lambda

Exercise: Prime numbers

Calculate and print the prime numbers between 2 and N. Use filter.

Exercise: Many validator functions

Given several validator functions (that get a parameter and return True or False),
and given a list of values, return a sublist of values that pass all the validation
checks. See the sekeleton:

```
def is_big(x):
    return x > 100

def is_even(x):
    return not x % 2

numbers = [90, 102, 101, 104]

cond = [is_big, is_even]

# z = ...
print(z) # [102, 104]
```

Exercise: Calculator using lookup table

Write a script that will accept a math expression such as `python calc.py 2 + 3` and will print the result.
Use lookup tables select the implementation of the actual computation. (supporting +, - , *, /) is enought

Exercise: parse file

In the following file we have lines:

```
SOURCE/FILENAME.json,TARGET
```

read in the file and create

- a single dictionary where the `SOURCE/FILENAME.json` is the key and the TARGET is the value.
- list of dictionaries in which the keys are ‘source’, ‘filename’, and ‘target’ and the values are from the respective columns (SOURCE, FILENAME.json, and TARGET)

You can solve this `for`-loop or with `map` and list-comprehensions. Do it in both ways.

```
agile/agile.json,agile
ansible/ansible.json,ansible
ansible-intro/ansible.json,ansible-intro
aws-lambda/aws.json,aws-lambda
bash/bash.json,bash
css/css.json,css
collab-dev/collab.json,collab-dev
data-science/data.json,data-science
dart-programming/dart.json,dart-programming
docker/docker.json,docker
google-gcp/gcp.json,google-gcp
git/git.json,git
git-intro/git.json,git-intro
github-ci/github-ci.json,github-ci
golang/go.json,golang
groovy/groovy.json,groovy
java-programming/java.json,java-programming
javascript-programming/javascript.json,javascript-programming
jenkins/jenkins.json,jenkins
jenkins-intro/jenkins.json,jenkins-intro
linux/linux.json,linux
linux-intro/linux.json,linux-intro
mobile/mobile.json,mobile
mojolicious/mojolicious.json,mojolicious
mongodb/mongodb.json,mongodb
nodejs/nodejs.json,nodejs
nosql/nosql.json,nosql
pair-programming/pair.json,pair-programming
perl-intro/perl.json,perl-intro
```

```
perl-programming/perl.json,perl-programming
perl-programming/testing.json,test-automation-using-perl
php-programming/php.json,php-programming
programming/programming.json,programming
python-mocking/python.json,python-mocking
python-programming/python.json,python-programming
ruby-programming/ruby.json,ruby=programming
sql/sql.json,sql
value/value.json,value
vim/vim.json,vim
web/web.json,web
windows-cmd/windows.json,windows-cmd
talks/real_world.json,real-world
talks/github-pages.json,github-pages
talks/python-pair-programming-and-tdd-workshop.json,python-pair-programming-and-tdd-\
workshop
```

Solution: min, max, factorial

```python
from functools import reduce

numbers = [2, 1, 4, 3]

# min
print(reduce(lambda x,y: x if x < y else y, numbers))   # 1
# max
print(reduce(lambda x,y: x if x > y else y, numbers))   # 4

# factorial
n = 4
print(reduce(lambda x,y: x*y, range(1, n+1), 1))    # 24
# The 1 at the end is the initializor of reduce to provide
# correct results for n = 0.

a = [1, 3, 6]
b = [2, 4, 5]
c = map(lambda x,y: x if x > y else y, a, b)
print(list(c))  # [2, 4, 6]
```

Solution: Prime numbers

Calculating the prime numbers

```
n = 50

nums = range(2, n)
for i in range(2, 1+int(n ** 0.5)):
    nums = filter(lambda x: x == i or x % i, nums)

print(nums)
```

Solution: Many validator functions

```
def is_big(x):
    return x > 100

def is_even(x):
    return not x % 2

numbers = [90, 102, 101, 104]

cond = [is_big, is_even]

z = filter( lambda n: all([f(n) for f in cond]), numbers)
print(z) # [102, 104]
```

Solution: Calculator using lookup table

```
import sys

table = {
    "+" : lambda x, y: x+y,
    "-" : lambda x, y: x-y,
    "*" : lambda x, y: x*y,
    "/" : lambda x, y: x/y,
}


def main():
```

```python
    if len(sys.argv) != 4:
        exit(f"Usage: {sys.argv[0]} NUMBER OP NUMBER")
    action = table[sys.argv[2]]
    print( action(int(sys.argv[1]), int(sys.argv[3])) )

main()
```

map with condtion

The conversion function can do anything. It can have a condition inside.

```python
numbers = [1, 2, 3, 4]

def cond_double(n):
   if n % 2:
      return 2*n
   else:
      return n

cd = map(cond_double, numbers)
print(cd)                          # [2, 2, 6, 4]
```

map with lambda

```python
numbers = [1, 2, 3, 4]

def dbl(x):
   return 2*x
d1 = map(dbl, numbers)
print(d1)   # [2, 4, 6, 8]

double = lambda x: 2*x
d2 = map(double, numbers)
print(d2)   # [2, 4, 6, 8]

```

```
d3 = map(lambda n: 2*n, numbers)
print(d3)    # [2, 4, 6, 8]
```

map with lambda with condition

```
numbers = [1, 2, 3, 4]

a = map(lambda n: 2*n if n % 2 else n, numbers)
print(a)    # [2, 2, 6, 4]
```

List comprehension - complex

```
numbers = [1, 3, 2, 4]

t = filter(lambda n: n > 2, numbers)
print(t)  # [3, 4]

n1 = map(lambda n: n*n, t)
print(n1) # [9, 16]


n2 = map(lambda n: n*n, filter(lambda n: n > 2, numbers))
print(n2)  # [9, 16]



n3 = [ n*n for n in numbers if n > 2 ]
print(n3) # [9, 16]
```

Iterators - with and without Itertools

Advantages of iterators and generators

- Lazy evaluation
- Save processing (or at least delay the use)
- Save memory
- Handle an infinite series of information
- Turn complex operations into a simple matter of `for` loop.

The Fibonacci research institute

- We have a bunch of mathematicians who research the Fibonacci series.
- We have a bunch of people who research a series of DNA sequences.
- ???

Fibonacci plain

- We don't call this as this has an infinite loop

```
def fibonacci():
    a, b = 0, 1
    while True:
        a, b = b, a+b

# fibonacci()
```

Fibonacci copy-paste

```
def fibonacci():
    a, b = 0, 1
    while True:
        a, b = b, a+b

        print(a)
        if a % 17 == 0:
            print('found')
            break

        if a > 200:
            print('not found')
            break

fibonacci()
```

Iterators Glossary

- iterable (Can be iterated over using a `for` loop.)
- iterator
- Every iterator is also iterable
- Iterators (and iterables) are not necessarily addressable like lists with the `thing[index]` construct.
- Iterator Types
- The standard type hierarchy

What are iterators and iterables?

- All of them are iterables
- A filehandle and the map object are also iterators. (Side note: You should always `open` files using the `with` statement and not like this.)
- `iter()` would return the iterator from an iterable. We don’t need this.

```
from collections.abc import Iterator, Iterable

a_string = "Hello World"
a_list   = ["Tiger", "Mouse"]
a_tuple  = ("Blue", "Red")
a_range  = range(10)
a_fh     = open(__file__)
a_map    = map(lambda x: x*2, a_list)

for thing in [a_string, a_list, a_tuple, a_range, a_map, a_fh]:
    print(thing.__class__.__name__)
    print(issubclass(thing.__class__, Iterator))
    print(issubclass(thing.__class__, Iterable))
    zorg = iter(thing)
    print(zorg.__class__.__name__)
    print(issubclass(zorg.__class__, Iterator))

    print()

a_fh.close()
```

```
str
False
True
str_iterator
True

list
False
True
list_iterator
True

tuple
False
True
tuple_iterator
True

range
False
True
range_iterator
```

```
True

TextIOWrapper
True
True
TextIOWrapper
True
```

A file-handle is an iterator

> This slightly a repetition of the previous statement, that filehandles are iterators.

```python
from collections.abc import Iterator, Iterable
from io import TextIOWrapper

with open(__file__) as fh:
    print(fh.__class__.__name__)
    print(issubclass(fh.__class__, TextIOWrapper))
    print(issubclass(fh.__class__, Iterator))
    print(issubclass(fh.__class__, Iterable))

    for line in fh:
        pass
        #print(line, end="")
```

```
TextIOWrapper
True
True
True
```

range is iterable but it is not an iterator

Just as a string or a list, the `range` function in Python is also an "iterable" but it is not an "iterator".
In many aspects it behaves as an iterator. Specifically it allows us to iterate over numbers.
Range Is Not An Iterator

- range

```
for n in range(2, 12, 3):
    print(n)
print()

for n in range(3):
    print(n)
print()

for n in range(2, 5):
    print(n)
print()

from collections.abc import Iterator, Iterable
rng = range(2, 5)
print(issubclass(rng.__class__, Iterator))
print(issubclass(rng.__class__, Iterable))
```

```
2
5
8
11

0
1
2

2
3
4

```

```
False
True
```

Iterator: a counter

We can create a iterator using a class. We are required to implement the `__iter__` method that returns the iterator object and the `__next__` method that returns the next element in our iteration. We can indicated that the iteration was exhaused by raising a `StopIteration` exception.

The instance-object that is created from this class-object is the iterator, not the class-object itself!

- `__iter__`
- `__next__` (in Python 2 this used to called `next`)
- `raise StopIteration`

```python
class Counter():
    def __init__(self):
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        self.count += 1
        if self.count > 3:
            raise StopIteration
        return self.count
```

Using iterator

The class returned an iterator, we could use a `for` loop to iterate over the element.
We tried to run through the iterator again, but it did not print anything. It was exhausted.

```
from counter import Counter

cnt = Counter()
for c in cnt:
    print(c)

for c in cnt:
    print(c)
```

```
1
2
3
```

Iterator without temporary variable

```
from counter import Counter

for c in Counter():
    print(c)
```

```
1
2
3
```

The type of the iterator

How can we know it is an iterator? We check it.

```python
from collections.abc import Iterator, Iterable
from counter import Counter

cnt = Counter()
print(cnt.__class__.__name__)
print(issubclass(cnt.__class__, Iterator))
print(issubclass(cnt.__class__, Iterable))
```

```
Counter
True
True
```

Using iterator with next

> A feature of any iterator is that we could iterate over it using the `next` call.

```python
from counter import Counter

cnt = Counter()

while True:
    try:
        a = next(cnt)
        print(a)
    except Exception as ex:
        print(ex.__class__.__name__)
        break
```

```
1
2
3
StopIteration
```

Mixing for and next

You can even use `next` inside a `for` loop, but then you will have to handle the `StopIteration` exception that migh happen during your call of `next`.

I am not really sure when would we want to use this.

```
from counter import Counter

cnt = Counter()

for i in cnt:
    print(f"i: {i}")
    try:
        n = next(cnt)
        print(f"n: {n}")
    except Exception as ex:
        print(ex.__class__.__name__)
        break
```

```
i: 1
n: 2
i: 3
StopIteration
```

Iterable which is not an iterator

```
from counter import Counter

class GetMyIterable():
    def __init__(self):
        pass
    def __iter__(self):
        return Counter()


thing = GetMyIterable()

```

```python
from collections.abc import Iterator, Iterable
print(issubclass(thing.__class__, Iterator))
print(issubclass(thing.__class__, Iterable))

for i in thing:
    print(i)
```

```
False
True
1
2
3
```

Iterator returning multiple values

```python
class SquareCounter():
    def __init__(self):
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        self.count += 1
        if self.count > 5:
            raise StopIteration
        return self.count, self.count ** 2

for cnt, sqr in SquareCounter():
    print(f"{cnt}  {sqr}")
```

```
1  1
2  4
3  9
4  16
5  25
```

Range-like iterator

```python
class Range():
    def __init__(self, start, end):
        self.current = start
        self.end = end

    def __iter__(self):
        return self

    def __next__(self):
        if self.current >= self.end:
            raise StopIteration
        v = self.current
        self.current += 1
        return v
```

```python
import it

r = it.Range(1, 4)
for n in r:
    print(n)

print('---')

for n in it.Range(2, 5):
    print(n)
```

```
1
2
3
---
2
3
4
```

Unbound or infinite iterator

So far each iterator had a beginning and an end. However we can also create infinte or unbounded iterators.

The nice thing about them is that we can pass them around as we do with any other object and we can execute operations on them without burning our CPU.

Of course the user will have to be carefull not to try to flatten the iterator, not to try to get all the values from it, as that will only create an infinite loop or a never ending operation.

In this very simple example we count from 0 and we never stop.

When we use the `Counter` in the `for` loop we need to include a stop-condition, otherwise our loop will never end.

```
class Counter():
    def __init__(self):
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        self.count += 1
        return self.count

for c in Counter():
    print(c)
    if c > 10:
        break
```

```
1
2
3
4
5
6
7
8
```

```
9
10
11
```

Unbound iterator Fibonacci

Now we can get back to our original problem, the slightly more complex Fibonacci series. In this example we created an unbounded iterator that on every iteration will return the next element of the Fibonacci series.

```python
class Fibonacci():
    def __init__(self):
        self.values = []

    def __iter__(self):
        return self

    def __next__(self):
        if len(self.values) == 0:
            self.values.append(1)
            return 1

        if len(self.values) == 1:
            self.values.append(1)
            return 1

        self.values.append(self.values[-1] + self.values[-2])
        self.values.pop(0)

        return self.values[-1]
```

```python
from fibonacci import Fibonacci
for v in Fibonacci():
    print(v)
```

```python
    if v > 10:
        break
```

```
1
1
2
3
5
8
13
```

Operations on Unbound iterator

```python
from fibonacci import Fibonacci

fib = Fibonacci()

#odd = [x for x in fib if x % 2 == 1]
odd = filter(lambda x: x % 2 == 1, fib)

print("Let's see")

for v in odd:
    print(v)
    if v > 10:
        break
```

```
Let's see
1
1
3
5
13
```

itertools

- [itertools]()

itertools is a standard Python library that provides a number of interesting iterators.
We are going to see a few examples here:

itertools - count

- Unbound counter: Count from N to infinity.

```python
import itertools

for c in itertools.count(start=19, step=1):
    print(c)
    if c > 23:
        break

# 19
# 20
# 21
# 22
# 23
# 24
```

itertools - cycle

```python
import itertools

ix = 0
for c in itertools.cycle(['A', 'B', 'C']):
    print(c)
    ix += 1
    if ix >= 5:
        break

print('')

ix = 0
for c in itertools.cycle('DEF'):
    print(c)
```

```python
    ix += 1
    if ix >= 5:
        break
```

```
A
B
C
A
B

D
E
F
D
E
```

Exercise: iterators - reimplement the range function

In one of the first slides of this chapter we saw a partial implementation of the `range` function.
Change that code to have a full implementation, that can accept 1, 2, or 3 parameters.

Exercise: iterators - cycle

- Reimplement the cycle functions of itertools using iterator class.

Exercise: iterators - alter

- Implement the alter functions as an iterator that will return

```
1
-2
3
```

```
-4
5
-6
...
```

- Optionally provide a start and end parameters
- start defaults to 1
- end defaults to unlimited

Exercise: iterators - limit Fibonacci

Change the Iterator version of the Fibonacci series so optionally you will be able to provide a parameter called “limit” to the constructor. If the limit is provided, the iterator should stop when the value passes the limit.

Exercise: iterators - Fibonacci less memory

Change the Iterator version of the Fibonacci series so it will NOT hold the previous values in memory.

Exercise: read char

Create an iterator that given a filename will return an object that on every iteration will return a single character. As an option let the user skip newlines, or maybe any pre-defined character.

Exercise: read section

- Create an iterator that given the name of a file like the following, will return once section at a time.

- It will return a list one each iteration and each element of the list will be a line from the current section.
- Other ideas what should be returned on each iteration?

```
name     = Mercury
distance = 0.4
mass     = 0.055


name     = Venus
distance = 0.7
mass     = 0.815


name     = Earth
distance = 1
mass     = 1

name     = Mars
distance = 1.5
mass     = 0.107
```

Exercise: collect packets

- You get a series of packets (e.g. lines in a file)
- In each line you have several fields: id, seqid, maxseq, content
- id is a unique identifier of a series of packets (lines)
- seqid is the seuence id of a packet in a series. (an integer)
- maxseq is the length of the sequence.
- content is the actual content.

In each iteration return a message that is built up from all the packages in the given sequence.

```
12,1,5,First of Twelve
12,2,5,Second of Twelve
12,3,5,Third of Twelve
12,4,5,Fourth of Twelve
```

```
12,5,5,Fifth of Twelve

9,1,4,First of Nine
9,2,4,Second of Nine
9,3,4,Third of Nine
9,4,4,Fourth of Nine

11,1,3,First of Eleven
11,2,3,Second of Eleven
11,3,3,Third of Eleven
```

```
['First of Twelve', 'Second of Twelve', 'Third of Twelve', 'Fourth of Twelve', 'Fift\
h of Twelve']
['First of Nine', 'Second of Nine', 'Third of Nine', 'Fourth of Nine']
['First of Eleven', 'Second of Eleven', 'Third of Eleven']
```

```
12,1,5,First of Twelve
11,1,3,First of Eleven
9,1,4,First of Nine
12,2,5,Second of Twelve
9,2,4,Second of Nine
11,2,3,Second of Eleven
12,3,5,Third of Twelve
9,3,4,Third of Nine
12,4,5,Fourth of Twelve
12,5,5,Fifth of Twelve
9,4,4,Fourth of Nine
11,3,3,Third of Eleven
```

```
11,2,3,Second of Eleven
11,1,3,First of Eleven
9,1,4,First of Nine
12,1,5,First of Twelve
9,3,4,Third of Nine
9,2,4,Second of Nine
12,3,5,Third of Twelve
12,4,5,Fourth of Twelve
12,2,5,Second of Twelve

```

```
12,5,5,Fifth of Twelve
9,4,4,Fourth of Nine
11,3,3,Third of Eleven
```

Exercise: compare files

Compare two files line-by-line, and create a 3rd file listing the lines that are different.

```
One
Two
Three
Four
Five
```

```
One
Two
Tree
Four
Five
```

Expected output:

```
2,Three,Tree
```

Solution: iterators - limit Fibonacci

```
class Fibonacci:
    def __init__(self, limit=0):
        self.values = []
        self.limit = limit
    def __iter__(self):
        return self
    def next(self):
        if self.limit and len(self.values) >= self.limit:
            raise StopIteration
        if len(self.values) == 0:
            self.values.append(1)
```

```
            return 1
        if len(self.values) == 1:
            self.values.append(1)
            return 1
        self.values.append(self.values[-1] + self.values[-2])
        return self.values[-1]
```

```
import fibonacci
f = fibonacci.Fibonacci(limit = 10)
print(f)
for v in f:
    print(v)

print('-----')
f = fibonacci.Fibonacci()
for v in f:
    print(v)
    if v > 30:
        break
```

Solution: iterators - Fibonacci less memory

```
class Fibonacci:
    def __init__(self, limit=0):
        self.values = ()
        self.limit = limit
    def __iter__(self):
        return self
    def next(self):
        if self.limit and len(self.values) and self.values[-1] >= self.limit:
            raise StopIteration
        if len(self.values) == 0:
            self.values = (1,)
            return 1
        if len(self.values) == 1:
            self.values = (1, 1)
            return 1
        self.values = (self.values[-1], self.values[-1] + self.values[-2])
        return self.values[-1]
```

```python
import fibonacci
f = fibonacci.Fibonacci(limit = 10)
print(f)
for v in f:
    print(v)

print('-----')
f = fibonacci.Fibonacci()
for v in f:
    print(v)
    if v > 30:
        break
```

Solution: read section

```python
import re

class SectionReader():
    def __init__(self, filename):
        self.filename = filename
        self.fh       = open(filename)

    def __iter__(self):
        return self

    def __next__(self):
        self.section = []
        while True:
            line = self.fh.readline()
            if not line:
                if self.section:
                    return self.section
                else:
                    self.fh.close()
                    raise StopIteration
            if re.search(r'\A\s*\Z', line):
                if self.section:
                    return self.section
                else:
                    continue
            self.section.append(line)


```

```python
filename = 'planets.txt'
for sec in SectionReader(filename):
    print(sec)
```

Solution: compare files

```python
import sys

def main():
    if len(sys.argv) != 4:
        exit(f"Usage: {sys.argv[0]} IN_FILE IN_FILE OUT_FILE")
    infile_a, infile_b = sys.argv[1:3]
    outfile = sys.argv[3]

    with open(outfile, 'w') as out_fh, open(infile_a) as in_a, open(infile_b) as in_\
b:
        cnt = 0
        for lines in zip(in_a, in_b):
            #print(lines)
            lines = list(map(lambda s: s.rstrip('\n'), lines))
            #print(lines)
            if lines[0] != lines[1]:
                out_fh.write(f"{cnt},{lines[0]},{lines[1]}\n")
            cnt += 1

main()
```

```
python diff.py first.txt second.txt diff.txt
```

Solution: collect packets

The implementation

```python
class Packets():
    def __init__(self, filename):
        self.filename = filename
```

```
        self.fh = open(filename)
        self.packets = {}
        self.max = {}

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            line = self.fh.readline()
            #print(f"line: {line}")
            if line == '':
                raise StopIteration

            line = line.rstrip("\n")
            if line == '':
                continue

            pid, seqid, maxseq, content = line.split(",")
            pid = int(pid)
            seqid = int(seqid)
            maxseq = int(maxseq)
            if pid not in self.packets:
                self.packets[pid] = {}
                self.max[pid] = maxseq
            if seqid in self.packets[pid]:
                raise Exception("pid arrived twice")
            if maxseq != self.max[pid]:
                raise Exception("maxseq changed")
            self.packets[pid][seqid] = content
            if len(self.packets[pid].keys()) == self.max[pid]:
                content = list(map(lambda i: self.packets[pid][i+1], range(self.max[\
pid])))
                del(self.max[pid])
                del(self.packets[pid])
                return content
```

The use:

```
import sys
from packets import Packets
```

```python

if len(sys.argv) < 2:
    exit(f"Usage: {sys.argv[0]} FILENAME")

for packet in Packets(sys.argv[1]):
    print(packet)
```

The test to verify it

```python
import os
import json
import pytest

from packets import Packets

root = os.path.dirname(os.path.abspath(__file__))

with open(os.path.join(root, 'packets.json')) as fh:
    expected_results = json.load(fh)

@pytest.mark.parametrize('filename', ['packets.txt', 'packets1.txt', 'packets2.txt'])
def test_packetes(filename):
    filepath = os.path.join(root, filename)

    results = []
    for packet in Packets(filepath):
        results.append(packet)
    assert results == expected_results
```

Expected result:

```
[["First of Twelve", "Second of Twelve", "Third of Twelve", "Fourth of Twelve", "Fif\
th of Twelve"], ["First of Nine", "Second of Nine", "Third of Nine", "Fourth of Nine\
"], ["First of Eleven", "Second of Eleven", "Third of Eleven"]]
```

Generators and Generator Expressions

Generators Glossary

- generator (a function that returns a “generator iterator”)
- generator-iterator (an object created by a generator)
- Generator types
- generator-expression

- Generators are basically a way to create iterators without a class.

Iterators vs Generators

- a generator is an iterator
- an iterator is an iterable

```
from collections.abc import Iterator, Iterable
from types import GeneratorType

print( issubclass(GeneratorType, Iterator) )   # True
print( issubclass(Iterator, Iterable) )        # True
```

- Genarators are a simpler way to create an iterable object than iterators, but iterators allow for more complex iterables.
- To create an iterator we need a class with two methods: `__iter__` and `__next__`, and a `raise StopIteration`.

- To create a generator we only need a single function with `yield .

List comprehension and Generator Expression

> However, before learning about `yield` let's see an even simpler way to create a generator. What we call a generator expression.
>
> You are probably already familiar with list comprehensions where you have a an `for` expression inside square brackets. That returns a `list` of values.
>
> If you replace the square brackets with parentheses then you get a **generator expression**.
>
> You can iterate over either of those. So what's the difference?

```
a_list = [i*2 for i in range(3)]
print(a_list)
for x in a_list:
    print(x)
print()

a_generator = (i*2 for i in range(3))
print(a_generator)
for x in a_generator:
    print(x)
```

```
[0, 2, 4]
0
2
4

<generator object <genexpr> at 0x7f0af6f97a50>
```

```
0
2
4
```

List comprehension vs Generator Expression - less memory

Let’s use a bigger range of numbers and create the corresponding list and generator. Then check the size of both of them.
You can see the list is much bigger. That’s becuse the list already contains all the elements, while the generator contains only the promise to give you all the elements.

As we could see in the previous example, this is not an empty promise, you can indeed iterate over the elements of a generator
just as you can iterate over the elements of a list.

However, you cannot access an arbitrary element of a generator because the generator is not **subscriptable**.

```python
import sys

lst = [n*2 for n in range(1000)] # List comprehension
gen = (n*2 for n in range(1000)) # Generator expression

print(sys.getsizeof(lst))
print(sys.getsizeof(gen))
print()

print(type(lst))
print(type(gen))
print()

```

```
print(lst[4])
print()

print(gen[4])
```

```
9016
112

<class 'list'>
<class 'generator'>

8

Traceback (most recent call last):
  File "generator_expression.py", line 17, in <module>
    print(gen[4])
TypeError: 'generator' object is not subscriptable
```

List Comprehension vs Generator Expressions

List comprehension vs Generator Expression - lazy evaluation

The second big difference between list comprehension and generator expressions is that the latter has lazy evaluation. In this example you can see that once we assign to list comprehension to a variable the `sqr` function is called on each element.

In the case of the generator expression, only when we iterate over the elements will Python call the `sqr` function. If we exit from the loop before we go over all the values than we saved time by not executing the expression on every element up-front. If the computation is complex and if our list is long, this can have a substantial impact.

```python
def sqr(n):
    print(f"sqr {n}")
    return n ** 2

numbers = [1, 3, 7]

# list comprehension
n1 = [ sqr(n) for n in numbers ]
print("we have the list")
for i in n1:
    print(i)
print("-------")

# generator expression
n2 = ( sqr(n) for n in numbers )
print("we have the generator")
for i in n2:
    print(i)
```

```
sqr 1
sqr 3
sqr 7
we have the list
1
9
49
-------
we have the generator
sqr 1
1
sqr 3
9
sqr 7
49
```

Generator: function with yield - call next

We can create a function that has multiple `yield` expressions inside.
We call the function and what we get back is a `generator`.
A `generator` is also an `iterator` so we can call the `next` function on it and it will give us the next `yield` value.

If we call it one too many times we get a `StopIteration` exception.

```
def number():
    yield 42
    yield 19
    yield 23

num = number()
print(type(num))
print(next(num))
print(next(num))
print(next(num))
print(next(num))
```

```
<class 'generator'>
42
19
23
Traceback (most recent call last):
  File "simple_generator_next.py", line 11, in <module>
    print(next(num))
StopIteration
```

Generators - call next

We can also use a `for` loop on the `generator` and then we don’t need to worry about the exception.

```
def number():
    yield 42
    yield 19
    yield 23

num = number()
print(type(num))
for n in num:
    print(n)
```

```
<class 'generator'>
42
19
23
```

Generator with yield

We don’t even need to use a temporary variable for it.

```
def number():
    yield 42
    yield 19
    yield 23

for n in number():
  print(n)
```

```
42
19
23
```

Generators - fixed counter

```python
def counter():
    n = 1
    yield n

    n += 1
    yield n

    n += 1
    yield n

for c in counter():
    print(c)
```

```
1
2
3
```

Generators - counter

```python
def counter():
    n = 1
    while True:
        yield n
        n += 1

for c in counter():
    print(c)
    if c >= 10:
        break
```

```
1
2
3
4
5
6
7
8
9
10
```

Generators - counter with parameter

```
def counter(n = 1):
    while True:
        yield n
        n += 1

for c in counter():
    print(c)
    if c >= 4:
        break
print()

for c in counter(8):
    print(c)
    if c >= 12:
        break
```

```
1
2
3
4

8
9
10
11
12
```

Generators - my_range

```
import sys

def my_range(limit = 1):
    n = 0
    while n < limit:
        yield n
        n += 1

for i in my_range(5):
    print(i)
```

```
print()

print(sum(my_range(10)))
print()

x = my_range(10000)
print(x)
print(sys.getsizeof(x))
```

```
0
1
2
3
4

45

<generator object my_range at 0x7f36f6089930>
120
```

Fibonacci - generator

```
def fibonacci():
    a, b = 0, 1
    while True:
        a, b = b, a+b
        yield a

for a in fibonacci():
    print(a)
    if a % 17 == 0:
        print('found')
        break

    if a > 200:
        print('not found')
        break
```

The fibonacci() function is called 5 times. When it reached the 'yield' command it returns the value
as if it was a normal return call, but when the function is called again, it will be executed starting
from the next statement. Hence the word 'after' will be printed after each call.

Infinite series

- The Fibonacci was already infinite, let's see a few more.

Integers

```
from series import integers

for i in integers():
    print(i)
    if i >= 10:
        break
```

```
1
2
3
4
5
6
7
8
9
10
```

Integers + 3

```python
from series import integers

n3   = (n+3 for n in integers())
# n3 = integers(3)
for i in n3:
    print(i)
    if i >= 10:
        break
```

```
4
5
6
7
8
9
10
```

Integers + Integers

```python
from series import integers

def mysum(nums):
    print(nums)
    total = 0
    for n in nums:
        total += n
    return total

n3 = integers(3)
n7 = integers(7)
d = ( mysum(p) for p in zip(n3, n7) )

print("start")
for i in d:
    print(i)
    if i >= 20:
        break
```

```
start
(3, 7)
10
```

```
(4, 8)
12
(5, 9)
14
(6, 10)
16
(7, 11)
18
(8, 12)
20
```

Filtered Fibonacci

```
from series import fibonacci

even = ( fib for fib in fibonacci() if fib % 2 == 0 )
for e in even:
    print(e)
    if e > 40:
        break
```

```
0
2
8
34
144
```

The series.py

This is the module behind the previous examples.

```
def integers(n = 1):
   while True:
       yield n
       n += 1

def fibonacci():
    a, b = 0, 1
    while True:
        yield a
```

```python
        a, b = b, a+b


def gfibonacci(size = 2):
    """Generalized Fibonacci. """
    values = [0]
    while True:
        yield values[-1]
        if len(values) < size:
            values.append(1)
        else:
            values.append(sum(values))
            values = values[1:]

def pascal():
    values = [1]
    while True:
        yield values
        new = [1]
        for i in range(0, len(values)-1):
            new.append(values[i] + values[i+1])
        new.append(1)
        values = new
```

generator - unbound count (with yield)

```python
def count(start=0, step=1):
    n = start
    while True:
        yield n
        n += step


for c in count(start=19, step=1):
    print(c)
    if c > 23:
        break
```

```
19
20
21
22
```

```
23
24
```

iterator - cycle

```python
def cycle(values=[]):
    my_values = []
    for v in values:
        my_values.append(v)
        yield v
    while True:
        for v in my_values:
            yield v

i = 0
for c in cycle(['A', 'B', 'C']):
    print(c)
    i += 1
    if i >= 4:
        break
```

```
A
B
C
A
```

Exercise: Alternator

Create a generator for the following number series: 1, -2, 3, -4, 5, -6, …

Exercise: Prime number generator

Create a generator that will return the prime numbers:
2, 3, 5, 7, 11, 13, 17, …

Exercise: generator

Take the two generator examples (increment number and Fibonacci) and change them to provide infinite iterations.
Then try to run them in a for loop. Just make sure you have some other condition to leave the for-loop.

Exercise: Tower of Hanoi

There are 3 sticks. On the first stick there are n rings of different sizes. The smaller the ring the higher it is on the stick.
Move over all the rings to the 3rd stick by always moving only one ring and making sure that never will there be a large ring on top of a smaller ring.

- [Tower of Hanoi]

Exercise: Binary file reader

Create a generator that given a filename and a number n will return the content of the file in chunks of n characters.

Exercise: File reader with records

In a file we have "records" of data. Each record starts with three bytes in which we have the length of the record.
Then the content.

```
8 ABCDEFGH 5 XYZQR
```

Given this source file

```
First line
Second record
Third row of the records
Fourth
```

```
5
END
```

using this code

```
filename = "rows.txt"
records  = "records.txt"

with open(filename) as in_fh:
    with open(records, 'w') as out_fh:
        for line in in_fh:
            line = line.rstrip("\n")
            out_fh.write("{:>3}{}".format(len(line), line))
```

we can create this file:

```
 10First line 13Second record 24Third row of the records  6Fourth  15  3END
```

The exercise is to create an iterator/generator that can read such a file record-by-record.

Logging

Simple logging

```python
import logging

logging.debug("debug")
logging.info("info")
logging.warning("warning")
logging.error("error")
logging.critical("critical")

logging.log(logging.WARNING, "another warning")
logging.log(40, "another error")
```

```
WARNING:root:warning
ERROR:root:error
CRITICAL:root:critical
WARNING:root:another warning
ERROR:root:another error
```

- Written on STDERR

Simple logging - set level

```python
import logging

logging.basicConfig(level = logging.INFO)

logging.debug("debug")
logging.info("info")
logging.warning("warning")
logging.error("error")
logging.critical("critical")
```

```
INFO:root:info
WARNING:root:warning
ERROR:root:error
CRITICAL:root:critical
```

Simple logging to a file

```
import logging
import time

logging.basicConfig(level = logging.INFO, filename =
time.strftime("my-%Y-%m-%d.log"\
))

logging.debug("debug")
logging.info("info")
logging.warning("warning")
logging.error("error")
logging.critical("critical")
```

Simple logging format

```
import logging

logging.basicConfig( format = '%(asctime)s  %
(levelname)-10s %(processName)s  %(name\
)s %(message)s')

logging.debug("debug")
logging.info("info")
logging.warning("warning")
logging.error("error")
logging.critical("critical")
```

Simple logging change date format

```
import logging

logging.basicConfig( format = '%(asctime)s  %
```

```
(levelname)-10s %(processName)s  %(name\
)s %(message)s', datefmt =  "%Y-%m-%d-%H-%M-%S")

logging.debug("debug")
logging.info("info")
logging.warning("warning")
logging.error("error")
logging.critical("critical")
```

```
2020-04-22-18-59-16  WARNING    MainProcess  root warning
2020-04-22-18-59-16  ERROR      MainProcess  root error
2020-04-22-18-59-16  CRITICAL   MainProcess  root
critical
```

getLogger

```
import logging

logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

fh = logging.FileHandler('my.log')
fh.setLevel(logging.INFO)
fh.setFormatter( logging.Formatter('%(asctime)s - %
(name)s - %(levelname)-10s - %(me\
ssage)s') )
logger.addHandler(fh)


sh = logging.StreamHandler()
sh.setLevel(logging.DEBUG)
sh.setFormatter(logging.Formatter('%(asctime)s - %
(levelname)-10s - %(message)s'))
logger.addHandler(sh)



log = logging.getLogger(__name__)
log.debug("debug")
log.info("info")
log.warning("warning")
```

```
log.error("error")
log.critical("critical")
```

Time-based logrotation

```
import logging

log_file = "my.log"

logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

ch = logging.handlers.TimedRotatingFileHandler(log_file, when='M', backupCount=2)
ch.setLevel(logging.INFO)
ch.setFormatter( logging.Formatter('%(asctime)s - %(name)s - %(levelname)-10s - %(me\
ssage)s') )
logger.addHandler(ch)


log = logging.getLogger(__name__)
log.debug("debug")
log.info("info")
log.warning("warning")
log.error("error")
log.critical("critical")
```

Size-based logrotation

```
import logging

log_file = "my.log"

logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

ch = logging.handlers.RotatingFileHandler(log_file, maxBytes=100, backupCount=2)
ch.setLevel(logging.INFO)
ch.setFormatter( logging.Formatter('%(asctime)s - %
```

```
(name)s - %(levelname)-10s - %(me\
ssage)s') )
logger.addHandler(ch)


log = logging.getLogger(__name__)
log.debug("debug")
log.info("info")
log.warning("warning")
log.error("error")
log.critical("critical")
```

Closures

Counter local - not working

```
def counter():
    count = 0
    count += 1
    return count

print(counter())
print(counter())
print(counter())
```

```
1
1
1
```

Counter with global

```
count = 0
def counter():
    global count
    count += 1
    return count

print(counter())
print(counter())
print(counter())

count = -42
print(counter())
```

```
1
2
```

```
3
-41
```

Create incrementors

In order to use in various map-expressions, we need a couple of functions that - for simplicity - need to increment a number:

```
def f3(x):
    return x + 3

def f7(x):
    return x + 7

def f23(x):
    return x + 23

print(f3(2))
print(f7(3))
print(f3(4))
print(f7(10))
print(f23(19))
```

```
5
10
7
17
42
```

Create internal function

```
def create_func():
    def internal():
        print("Hello world")
    internal()


func = create_func()
internal()
```

```
Hello world
Traceback (most recent call last):
  File "create_internal_func.py", line 8, in <module>
    internal()
NameError: name 'internal' is not defined
```

Create function by a function

```
def create_func():
    def internal():
        print("Hello world")
    #internal()

    return internal

func = create_func()
#internal()
func()
```

```
Hello world
```

Create function with parameters

```
def create_func(name):
    def internal():
        print(f"Hello {name}")

    return internal

foo = create_func("Foo")
foo()


bar = create_func("Bar")
bar()
```

```
Hello Foo
Hello Bar
```

Counter closure

```python
def create_counter():
    count = 0
    def internal():
        nonlocal count
        count += 1
        return count
    return internal

counter = create_counter()

print(counter())
print(counter())
print(counter())
print()

other = create_counter()
print(counter())
print(other())
print(counter())
print(other())

print()
print(count)
```

```
1
2
3

4
1
5
2

Traceback (most recent call last):
  File "counter.py", line 23, in <module>
    print(count)
NameError: name 'count' is not defined
```

Make incrementor with def (closure)

```
def make_incrementor(n):
    def inc(x):
        return x + n
    return inc

f3 = make_incrementor(3)
f7 = make_incrementor(7)

print(f3(2))
print(f7(3))
print(f3(4))
print(f7(10))
```

```
5
10
7
17
```

Make incrementor with lambda

```
def make_incrementor(n):
    return lambda x: x + n

f3 = make_incrementor(3)
f7 = make_incrementor(7)

print(f3(2))
print(f7(3))
print(f3(4))
print(f7(10))
```

```
5
10
7
17
```

Exercise: closure bank

- Create a closure that returns a function that holds a number (like a bank account) that can be incremented or decremented as follows:
- Allow for an extra paramter called `prev` that defaults to `False`. If `True` is passed then instead of returning the new balance, return the old balance.

```
bank = create_bank(20)

print(bank())    # 20
print(bank(7))   # 27
print(bank())    # 27
print(bank(-3))  # 24
print(bank())    # 24


print(bank(10, prev=True))   # 24
print(bank())    # 34
```

Exercise: counter with parameter

Change the counter example to accept a parameter and start counting from that number.

Solution: closure bank

```
def create_bank(n = 0):
    balance = n
    def bnk(change = 0, prev=False):
        nonlocal balance
        prev_balance = balance
        balance += change
        if prev:
            return prev_balance
        else:
            return balance
    return bnk


```

```
bank = create_bank(20)

print(bank())    # 20
print(bank(7))   # 27
print(bank())    # 27
print(bank(-3))  # 24
print(bank())    # 24


print(bank(10, prev=True))   # 24
print(bank())    # 34
```

```
20
27
27
24
24
24
34
```

Solution: counter with parameter

```
def create_counter(count=0):
    def internal():
        nonlocal count
        count += 1
        return count
    return internal

counter = create_counter()

print(counter())
print(counter())
print(counter())
print()

other = create_counter(42)
print(counter())
print(other())
print(counter())
print(other())
```

```
1
2
3

4
43
5
44
```

Decorators

Function assignment

Before we learn about decorators let's remember that we can assign function names
to other names and then use the new name:

```
say = print
say("Hello World")

print = lambda n: n**n
res = print(3)
say("Hi")
say(res)


def add(x, y):
    return x + y

combine = add

say( combine(2, 3) )
```

```
Hello World
Hi
27
5
```

Function inside other function

> Let’s also remember that we can defind a function inside another function and then the internally defined function only exists in the scope of the function where it was defined in. Not outside.

```
def f():
    def g():
        print("in g")
    print("start f")
    g()
    print("end f")

f()
g()
```

```
start f
in g
end f
Traceback (most recent call last):
  File "examples/decorators/function_in_function.py", line 9, in <module>
    g()
NameError: name 'g' is not defined
```

Decorator

- A function that changes the behaviour of other functions.
- The input of a decorator is a function.
- The returned value of a decorator is a modified version of the same function.

```
from some_module import some_decorator

@some_decorator
def f(...):
    ...
```

```
def f(...):
    ...
```

```
f = some_decorator(f)
```

Use cases for decorators in Python

- Common decorators are classmethod() and staticmethod().
- Flask uses them to mark and configure the routes.
- Pytest uses them to add marks to the tests.

- Logging calls with parameters.
- Logging elapsed time of calls.
- Access control in Django or other web frameworks. (e.g. login required)
- Memoization (caching)
- Retry
- Function timeout
- Locking for thread safety
- Decorator Library

A recursive Fibonacci

```
def fibo(n):
    if n in (1,2):
        return 1
    return fibo(n-1) + fibo(n-2)

print(fibo(5))  # 5
```

trace fibo

```python
import decor

@decor.tron
def fibo(n):
    if n in (1,2):
        return 1
    return fibo(n-1) + fibo(n-2)

print(fibo(5))
```

```
Calling fibo(5)
Calling fibo(4)
Calling fibo(3)
Calling fibo(2)
Calling fibo(1)
Calling fibo(2)
Calling fibo(3)
Calling fibo(2)
Calling fibo(1)
5
```

tron decorator

```python
def tron(func):
    def new_func(v):
        print("Calling {}({})".format(func.__name__, v))
        return func(v)
    return new_func
```

Decorate with direct call

```python
import decor

def fibo(n):
    if n in (1,2):
        return 1
    return fibo(n-1) + fibo(n-2)

fibo = decor.tron(fibo)
```

```
print(fibo(5))
```

Decorate with parameter

```
import decor_param

@decor_param.tron('foo')
def fibo(n):
    if n in (1,2):
        return 1
    return fibo(n-1) + fibo(n-2)

print(fibo(5))
```

```
foo Calling fibo(5)
foo Calling fibo(4)
foo Calling fibo(3)
foo Calling fibo(2)
foo Calling fibo(1)
foo Calling fibo(2)
foo Calling fibo(3)
foo Calling fibo(2)
foo Calling fibo(1)
5
```

Decorator accepting parameter

```
def tron(prefix):
    def real_tron(func):
        def new_func(v):
            print("{} Calling {}({})".format(prefix, func.__name__, v))
            return func(v)
        return new_func
    return real_tron
```

Decorate function with any signature

- How can we decorate a function that is flexible on the number of arguments?
- Accept `*args` and `**kwargs` and pass them on.

```python
from decor_any import tron


@tron
def one(param):
    print(f"one({param})")

@tron
def two(first, second = 42):
    print(f"two({first}, {second})")


one("hello")
one(param = "world")

two("hi")
two(first = "Foo", second = "Bar")
```

Decorate function with any signature - implementation

```python
def tron(func):
    def new_func(*args, **kw):
        params = list(map(lambda p: str(p), args))
        for (k, v) in kw.items():
            params.append(f"{k}={v}")
        print("Calling {}({})".format(func.__name__, ', '.join(params)))
        return func(*args, **kw)
    return new_func
```

```
Calling one(hello)
one(hello)
Calling one(param=world)
one(world)
Calling two(hi)
```

```
two(hi, 42)
Calling two(first=Foo, second=Bar)
two(Foo, Bar)
```

Exercise: Logger decorator

- In the previous pages we created a decorator that can decorate arbitrary function logging the call and its parameters.
- Add time measurement to each call to see how long each function took.

Exercise: memoize decorator

Write a function that gets a functions as attribute and returns a new functions while memoizing (caching) the input/output pairs.
Then write a unit test that checks it.
You probably will need to create a subroutine to be memoized.

- Write tests for the fibonacci functions.
- Implement the memoize decorator for a function with a single parameter.
- Apply the decorator.
- Run the tests again.
- Check the speed differences.
- or decorate with tron to see the calls…

Solution: Logger decorator

```
import time
def tron(func):
    def new_func(*args, **kwargs):
        start = time.time()
        print("Calling {}({}, {})".format(func.__name__,
args, kwargs))
```

```
        out = func(*args, **kwargs)
        end = time.time()
        print("Finished {}({})".format(func.__name__,
out))
        print("Elapsed time: {}".format(end - start))
        return out
    return new_func
```

Solution: Logger decorator (testing)

```
from logger_decor import tron

@tron
def f(a, b=1, *args, **kwargs):
    print('a:     ', a)
    print('b:     ', b)
    print('args:  ', args)
    print('kwargs:', kwargs)
    return a + b

f(2, 3, 4, 5, c=6, d=7)
print()
f(2, c=5, d=6)
print()
f(10)
```

```
Calling f((2, 3, 4, 5), {'c': 6, 'd': 7})
a:      2
b:      3
args:   (4, 5)
kwargs: {'c': 6, 'd': 7}
Finished f(5)
Elapsed time: 1.3589859008789062e-05

Calling f((2,), {'c': 5, 'd': 6})
a:      2
b:      1
args:   ()
kwargs: {'c': 5, 'd': 6}
Finished f(3)
Elapsed time: 5.245208740234375e-06

```

```
Calling f((10,), {})
a:      10
b:      1
args:   ()
kwargs: {}
Finished f(11)
Elapsed time: 4.291534423828125e-06
```

Solution memoize decorator

```python
import sys
import memoize_attribute
import memoize_nonlocal
import decor_any

#@memoize_attribute.memoize
#@memoize_nonlocal.memoize
#@decor_any.tron
def fibonacci(n):
    if n == 1:
        return 1
    if n == 2:
        return 1
    return fibonacci(n-1) + fibonacci(n-2)

if __name__ == '__main__':
    if len(sys.argv) != 2:
        sys.stderr.write("Usage: {} N\n".format(sys.argv[0]))
        exit(1)
    print(fibonacci(int(sys.argv[1])))
```

```python
def memoize(f):
    data = {}
    def caching(n):
        nonlocal data
        key = n
        if key not in data:
            data[key] = f(n)
        return data[key]

    return caching
```

```
def memoize(f):
    def caching(n):
        key = n
        #if 'data' not in caching.__dict__:
        #    caching.data = {}
        if key not in caching.data:
            caching.data[key] = f(n)
        return caching.data[key]
    caching.data = {}

    return caching
```

Before

```
$ time python fibonacci.py 35
9227465

real   0m3.850s
user   0m3.832s
sys    0m0.015s
```

After

```
$ time python fibonacci.py 35
9227465

real   0m0.034s
user   0m0.019s
sys    0m0.014s
```

Context managers (with statement)

Why use context managers?

In certain operations you might want to ensure that when the operation is done there will be an opportunity to clean up after it. Even if decided to end the operation early or if there is an exception in the middle of the operation.

In the following pseudo-code example you can see that `cleanup` must be called both at the end and before the `early-end`, but that still leaves the bad-code that raises exception avoiding the cleanup. That forces us to wrap the whole section in a try-block.

```
start
do
do
do
do
cleanup
```

What is we have some conditions for early termination?

```
start
do
do
if we are done early:
   cleanup
   early-end
do
do
cleanup
```

What if we might have an exception in the code?

```
start
try:
   do
   do
   if we are done early:
      cleanup
      early-end
   do
   bad-code    (raises exception)
   do
   cleanup
finally:
   cleanup
```

It is a lot of unnecessary code duplication and we can easily forget to add it in every location where we early-end our code.

Context Manager examples

A few examples where context managers can be useful:

- Opening a file - close it once we are done with it so we don't leak file descriptors.
- Changing directory - change back when we are done.
- Create temporary directory - remove when we are done.
- Open connection to database - close connection.
- Open SSH connection - close connection.

- More information about context managers

cd in a function

In this example we have a function in which we change to a directory and then when we are done we change back to the original directory.

For this to work first we save the current working directory using the `os.getcwd` call. Unfortunatelly in the middle of the code there is a conditional call to `return`. If that condition is `True` we won't change back to the original directory. We could fix this by calling `os.chdir(start_dir)` just before calling `return`. However this would still not solve the problem if there is an exception in the function.

```python
import sys
import os

def do_something(path):
    start_dir = os.getcwd()
    os.chdir(path)

    content = os.listdir()
    number = len(content)
    print(number)
    if number < 15:
        return

    os.chdir(start_dir)

def main():
    if len(sys.argv) != 2:
        exit(f"Usage: {sys.argv[0]} PATH")
    path = sys.argv[1]
    print(os.getcwd())
    do_something(path)
    print(os.getcwd())

main()
```

```
$ python no_context_cd.py /tmp/

/home/gabor/work/slides/python-programming/examples/advanced
19
/home/gabor/work/slides/python-programming/examples/advanced
```

```
$ python no_context_cd.py /opt/

/home/gabor/work/slides/python-programming/examples/advanced
9
/opt
```

- In the second example `return` was called and thus we stayed on the /opt directory.:w

open in function

This is not the recommended way to open a file, but this is how it was done before the introduction of the `with` context manager.
Here we have the same issue. We have a conditional call to `return` where we forgot to close the file.

```
import sys
import re

def do_something(filename):
    fh = open(filename)

    while True:
        line = fh.readline()
```

```
        if line is None:
            break
        line = line.rstrip("\n")

        if re.search(r'\A\s*\Z', line):
            return
        print(line)

    fh.close()

def main():
    if len(sys.argv) != 2:
        exit(f"Usage: {sys.argv[0]} FILENAME")
    filename = sys.argv[1]
    do_something(filename)

main()
```

open in for loop

Calling `write` does not immediately write to disk. The Operating System provides buffering as an optimization to avoid frequent access to the disk. In this case it means the file has not been saved before we already check its size.

```
import os

for ix in range(10):
    filename = f'data{ix}.txt'
    fh = open(filename, 'w')
    fh.write('hello')
    if ix == 0:
        break
    fh.close()
stat = os.stat(filename)
print(stat.st_size)    # 0,    the file has not been saved yet
```

open in function using with

If we open the file in the recommended way using the `with` statement then we can be sure that the `close` method of the `fh` object will be called when we leave the context of the `with` statement.

```python
import sys
import re

def do_something(filename):
    with open(filename) as fh:

        while True:
            line = fh.readline()
            if line is None:
                break
            line = line.rstrip("\n")

            if re.search(r'\A\s*\Z', line):
                return
            print(line)


def main():
    if len(sys.argv) != 2:
        exit(f"Usage: {sys.argv[0]} FILENAME")
    filename = sys.argv[1]
    do_something(filename)

main()
```

Plain context manager

```python
from contextlib import contextmanager

@contextmanager
```

```python
def my_plain_context():
    print("start context")
    yield
    print("end context")

print("START")
with my_plain_context():
    print("  In plain context")
    print("  More work")

print("END")
```

```
START
start context
  In plain context
  More work
end context
END
```

Param context manager

```python
from contextlib import contextmanager

@contextmanager
def my_param_context(name):
   print(f"start {name}")
   yield
   print(f"end {name}")

with my_param_context("foo"):
   print("In param context")
```

```
start foo
In param context
end foo
```

Context manager that returns a value

```python
from contextlib import contextmanager

import time
import random
import os
import shutil


@contextmanager
def my_tempdir():
    print("start return")
    tmpdir = '/tmp/' + str(time.time()) + str(random.random())
    os.mkdir(tmpdir)
    try:
        yield tmpdir
    finally:
        shutil.rmtree(tmpdir)
        print("end return")
```

```python
import os
from my_tempdir import my_tempdir

with my_tempdir() as tmp_dir:
    print(f"In return context with {tmp_dir}")
    with open(tmp_dir + '/data.txt', 'w') as fh:
        fh.write("hello")
    print(os.listdir(tmp_dir))

print('')
print(tmp_dir)
print(os.path.exists(tmp_dir))
```

```
start return
In return context with /tmp/1578211890.49409370.6063140788762365
['data.txt']
end return

/tmp/1578211890.49409370.6063140788762365
False
```

Use my tempdir - return

```python
import os
from my_tempdir import my_tempdir

def some_code():
    with my_tempdir() as tmp_dir:
        print(f"In return context with {tmp_dir}")
        with open(tmp_dir + '/data.txt', 'w') as fh:
            fh.write("hello")
        print(os.listdir(tmp_dir))
        return

    print('')
    print(tmp_dir)
    print(os.path.exists(tmp_dir))

some_code()
```

```
start return
In return context with /tmp/1578211902.3545020.7667694368935928
['data.txt']
end return
```

Use my tempdir - exception

```python
import os
from my_tempdir import my_tempdir

with my_tempdir() as tmp_dir:
    print(f"In return context with {tmp_dir}")
    with open(tmp_dir + '/data.txt', 'w') as fh:
        fh.write("hello")
    print(os.listdir(tmp_dir))
    raise Exception('trouble')

print('')
print(tmp_dir)
print(os.path.exists(tmp_dir))
```

```
start return
In return context with /tmp/1578211921.12552210.9000097350821897
['data.txt']
end return
Traceback (most recent call last):
  File "use_my_tempdir_exception.py", line 9, in <module>
    raise Exception('trouble')
Exception: trouble
```

cwd context manager

```python
import os
from contextlib import contextmanager

@contextmanager
def cwd(path):
    oldpwd = os.getcwd()
    os.chdir(path)
    try:
        yield
    finally:
        os.chdir(oldpwd)
```

```python
import sys
import os
from mycwd import cwd

def do_something(path):
    with cwd(path):
        content = os.listdir()
        if len(content) < 10:
            return

def main():
    if len(sys.argv) != 2:
        exit(f"Usage: {sys.argv[0]} PATH")
    path = sys.argv[1]
    print(os.getcwd())
    do_something(path)
    print(os.getcwd())
```

```

main()
```

```
$ python context_cd.py /tmp
/home/gabor/work/slides/python/examples/advanced
/home/gabor/work/slides/python/examples/advanced

$ python context_cd.py /opt
/home/gabor/work/slides/python/examples/advanced
/home/gabor/work/slides/python/examples/advanced
```

tempdir context manager

```python
import os
from contextlib import contextmanager
import tempfile
import shutil

@contextmanager
def tmpdir():
    dd = tempfile.mkdtemp()
    try:
        yield dd
    finally:
        shutil.rmtree(dd)
```

```python
from mytmpdir import tmpdir
import os

with tmpdir() as temp_dir:
    print(temp_dir)
    with open( os.path.join(temp_dir, 'some.txt'), 'w')
as fh:
        fh.write("hello")
    print(os.path.exists(temp_dir))
    print(os.listdir(temp_dir))

print(os.path.exists(temp_dir))
```

```
/tmp/tmprpuywa3_
True
['some.txt']
False
```

Context manager with class

```python
class MyCM:
    def __init__(self, name):
        self.name = name

    def __enter__(self):
        print(f'__enter__ {self.name}')
        return self

    def __exit__(self, exception_type, exception, traceback):
        print(f'__exit__  {self.name}')

    def something(self):
        print(f'something {self.name}')

def main():
    with MyCM('Foo') as cm:
        print(cm.name)
        cm.something()
        #raise Exception('nono')
    print('in main - after')

main()
print('after main')
```

Context managers with class

Even if there was en exception in the middle of the process, the **exit** methods of each object will be called.

```python
class MyCM:
    def __init__(self, n):
        self.name = n
```

```

    def __enter__(self):
        print('__enter__', self.name)

    def __exit__(self, exception_type, exception, traceback):
        print('__exit__ ', self.name)

    def something(self):
        print('something', self.name)

def main():
    a = MyCM('a')
    b = MyCM('b')
    with a, b:
        a.partner = b
        b.partner = a
        a.something()
        raise Exception('nono')
        b.something()
    print('in main - after')

main()
print('after main')
```

```
__enter__ a
__enter__ b
something a
__exit__  b
__exit__  a
Traceback (most recent call last):
  File "context-managers.py", line 27, in <module>
    main()
  File "context-managers.py", line 23, in main
    raise Exception('nono')
Exception: nono
```

Context manager: with for file

```
import sys

if len(sys.argv) != 2:
```

```
    sys.stderr.write('Usage: {} FILENAME\n'.format(sys.argv[0]))
    exit()

file = sys.argv[1]
print(file)
with open(file) as f:
    for line in f:
        val = 30/int(line)

print('done')
```

With - context managers

```
class WithClass:
    def __init__(self, name='default'):
        self.name = name

    def __enter__(self):
        print('entering the system')
        return self.name

    def __exit__(self, exc_type, exc_value, traceback):
        print('exiting the system')

    def __str__(self):
        return 'WithObject:'+self.name

x = WithClass()
with x as y:
    print(x,y)
```

Exercise: Context manager

Create a few CSV file likes these:

```
a11,a12
a21,a22
```

```
b13,b14
b23,b24
```

```
c15,c16
c25,c26
```

Merge them horizontally to get this:

```
a11,a12,b13,b14,c15,c16
a21,a22,b23,b24,c25,c26
```

- Do it without your own context manager
- Create a context manager called myopen that accepts N filenames. It opens the first one to write and the other N-1 to read

```
with myopen(outfile, infile1, infile2, infile3) as out, ins:
    ...
```

Exercise: Tempdir on Windows

Make the tempdir context manager example work on windows as well. Probably need to cd out of the directory.

Solution: Context manager

```python
import sys
from contextlib import contextmanager

if len(sys.argv) < 3:
    exit(f"Usage: {sys.argv[0]} OUTFILE INFILEs")

outfile = sys.argv[1]
infiles = sys.argv[2:]
#print(outfile)
```

```python
#print(infiles)

@contextmanager
def myopen(outfile, *infiles):
    #print(len(infiles))
    out = open(outfile, 'w')
    ins = []
    for filename in infiles:
        ins.append(open(filename, 'r'))
    try:
        yield out, ins
    except Exception as ex:
        print(ex)
        pass
    finally:
        out.close()
        for fh in ins:
            fh.close()


with myopen(outfile, *infiles) as (out_fh, input_fhs):
    #print(out_fh.__class__.__name__)
    #print(len(input_fhs))
    while True:
        row = ''
        done = False
        for infh in (input_fhs):
            line = infh.readline()
            #print(f"'{line}'")
            if not line:
                done = True
                break
            if row:
                row += ','
            row += line.rstrip("\n")
        if done:
            break
        out_fh.write(row)
        out_fh.write("\n")
```

Advanced lists

Change list while looping: endless list

```
numbers = [1, 1]
for n in numbers:
    print(n)
    numbers.append(numbers[-1] + numbers[-2])

    if n > 100:
        break

print(numbers)
```

Creating a Fibonacci series in a crazy way.

Change list while looping

Probably not a good idea…

```
numbers = [1, 2, 3, 4]
for n in numbers:
    print(n)
    if n == 2:
        numbers.remove(2)


print(numbers)
```

```
1
2
4
[1, 3, 4]
```

Note, the loop only iterated 3 times, and it skipped value 3

Copy list before iteration

It is better to copy the list using list slices before the iteration starts.

```
numbers = [1, 2, 3, 4]
for n in numbers[:]:
    print(n)
    if n == 2:
        numbers.remove(2)


print(numbers)
```

```
1
2
3
4
[1, 3, 4]
```

for with flag

```
names = ['Foo', 'Bar', 'Baz']

ok = False
for i in range(3):
    name = input('Your name please: ')
    if name in names:
        ok = True
        break

if not ok:
    print("Not OK")
    exit()

print("OK....")
```

for else

The else statement of the for loop is executed when the iteration ends normally. (without calling break)

```
names = ['Foo', 'Bar', 'Baz']


for i in range(3):
    name = input('Your name please: ')
    if name in names:
        break
else:
    print("Not OK")
    exit()

print("OK....")
```

enumerate

```
names = ['Foo', 'Bar', 'Baz']

for i in range(len(names)):
    print(i,  names[i])

print('')

for i, n  in enumerate(names):
    print(i, n)
```

```
0 Foo
1 Bar
2 Baz

0 Foo
1 Bar
2 Baz
```

do while

There is no do-while in Python, but you can emulate it:

```
while True:
    do_stuff()
    if not loop_condition():
        break
```

```
x = 0

while True:
    x += 1
    print(x)
    if x > 0:
        break
```

list slice is copy

```
x = [1, 1, 2, 3, 5, 8, 13, 21, 34]
y = x[2:5]
print(y)    # [2, 3, 5]

x[2] = 20
print(x)    # [1, 1, 20, 3, 5, 8, 13, 21, 34]
print(y)    #  [2, 3, 5]
```

Advanced Exception handling

Exceptions else

- The else part will be execute after each successful “try”. (So when there was no exception.)

```python
import sys
import module

# python else.py one.txt zero.txt two.txt three.txt
files = sys.argv[1:]

for filename in files:
    try:
        module.read_and_divide(filename)
    except ZeroDivisionError as err:
        print("Exception {} of type {} in file {}".format(err, type(err).__name__, f\
ilename))
    else:
        print("In else part after trying file {} and succeeding".format(filename))
        # Will run only if there was no exception.
    print()
```

```
before one.txt
100.0
after  one.txt
In else part after trying file one.txt and succeeding

before zero.txt
Exception division by zero of type ZeroDivisionError in file zero.txt

before two.txt
Traceback (most recent call last):
```

```
  File "else.py", line 9, in <module>
    module.read_and_divide(filename)
  File "/home/gabor/work/slides/python-programming/examples/exceptions/module.py", l\
ine 3, in read_and_divide
    with open(filename, 'r') as fh:
FileNotFoundError: [Errno 2] No such file or directory: 'two.txt'
```

Exceptions finally

- We can add a “finally” section to the end of the “try” - “except” construct.
- The code in this block will be executed after **every** time we enter the **try**.
- When we finish it successfully. When we catch an exception. (In this case a ZeroDivisionError exception in file zero.txt)
- Even when we don’t catch an exception. Before the exception propagates up in the call stack, we still see the “finally” section executed.

```python
import sys
import module

# python finally.py one.txt zero.txt two.txt three.txt
files = sys.argv[1:]

for filename in files:
    try:
        module.read_and_divide(filename)
    except ZeroDivisionError as err:
        print("Exception {} of type {} in file {}".format(err, type(err).__name__, f\
ilename))
    finally:
        print("In finally after trying file {}".format(filename))
    print('')
```

```
before one.txt
100.0
after  one.txt
In finally after trying file one.txt

before zero.txt
Exception division by zero of type ZeroDivisionError in file zero.txt
In finally after trying file zero.txt

before two.txt
In finally after trying file two.txt
Traceback (most recent call last):
  File "finally.py", line 9, in <module>
    module.read_and_divide(filename)
  File "/home/gabor/work/slides/python-programming/examples/exceptions/module.py", l\
ine 3, in read_and_divide
    with open(filename, 'r') as fh:
FileNotFoundError: [Errno 2] No such file or directory: 'two.txt'
```

Exit and finally

> The “finally” part will be called even if we call “return” or “exit” in the “try” block.

```
def f():
    try:
        return
    finally:
       print("finally in f")

def g():
    try:
        exit()
    finally:
       print("finally in g")

```

```
print("before")
f()
print("after f")
g()
print("after g")

# before
# finally in f
# after f
# finally in g
```

Catching exceptions

```
def divide(x, y):
    return x/y

def main():
    cnt = 6
    for num in [2, 0, 'a']:
        try:
            divide(cnt, num)
        except ZeroDivisionError:
            pass
        except (IOError, MemoryError) as err:
            print(err)
        else:
            print("This will run if there was no exception at all")
        finally:
            print("Always executes. {}/{} ended.".format(cnt, num))

    print("done")


main()
```

```
This will run if there was no exception at all
Always executes. 6/2 ended.
Always executes. 6/0 ended.
Always executes. 6/a ended.
Traceback (most recent call last):
```

```
  File "try.py", line 22, in <module>
    main()
  File "try.py", line 9, in main
    divide(cnt, num)
  File "try.py", line 3, in divide
    return x/y
TypeError: unsupported operand type(s) for /: 'int' and
'str'
```

Home made exception

> You can create your own exception classes that will allow the user to know what kind of an exception was caught or to capture only the exceptions of that type.

```python
class MyException(Exception):
    pass

def some():
    raise MyException("Some Error")

def main():
    try:
        some()
    except Exception as err:
        print(err)
        print("Type: " + type(err).__name__)

    try:
        some()
    except MyException as err:
        print(err)

main()
```

```
Some Error
Type: MyException
Some Error
```

Home made exception with attributes

```python
class MyException(Exception):
    def __init__(self, name, address):
        self.name    = name
        self.address = address
    def __str__(self):
        return 'Have you encountered problems? name:{} address:{}'.format(self.name\
, self.address)


def some():
    raise MyException(name = "Foo Bar", address = "Somewhere deep in the code")

def main():
    try:
        some()
    except Exception as err:
        print(err)
        print("Type: " + type(err).__name__)
        print(err.name)
        print(err.address)

main()

# Have you encountered problems? name:Foo Bar address:Somewhere deep in the code
# Type: MyException
# Foo Bar
# Somewhere deep in the code
```

Home made exception hierarcy

```python
class MyError(Exception):
    pass

class MyGreenError(MyError):
    pass
```

```python

class MyBlueError(MyError):
    pass


def green():
    raise MyGreenError('Hulk')

def blue():
    raise MyBlueError('Frozen')

def red():
     red_alert()
```

Home made exception hierarcy - 1

```python
import colors as cl

def main():
    print("start")
    try:
        cl.green()
    except Exception as err:
        print(err)
        print(type(err).__name__)
    print("done")


main()
```

```
start
Hulk
MyGreenError
done
```

Home made exception hierarcy - 2

```python
import colors as cl

def main():
```

```
    print("start")
    try:
        cl.green()
    except cl.MyGreenError as err:
        print(err)
        print(type(err).__name__)
    print("done")


main()
```

```
start
Hulk
MyGreenError
done
```

Home made exception hierarcy - 3

```
import colors as cl

def main():
    print("start")

    try:
        cl.green()
    except cl.MyError as err:
        print(err)
        print(type(err).__name__)

    try:
        cl.blue()
    except cl.MyError as err:
        print(err)
        print(type(err).__name__)

    try:
        cl.red()
    except cl.MyError as err:
        print(err)
        print(type(err).__name__)


```

```python


    print("done")


main()
```

```
start
Hulk
MyGreenError
Frozen
MyBlueError
Traceback (most recent call last):
  File "hierarchy3.py", line 30, in <module>
    main()
  File "hierarchy3.py", line 19, in main
    cl.red()
  File
"/home/gabor/work/slides/python/examples/exceptions/colors.
py", line 18, in r\
ed
    red_alert()
NameError: name 'red_alert' is not defined
```

Exercise: spacefight with exceptions

Take the number guessing game (or one-dimensional space-fight) and add exceptions
for cases when the guess is out of space (0-200 by default), or when the guess is not
a number.

```python
import random

class Game:
    def __init__(self):
        self.lower_limit = 0
        self.upper_limit = 200

        self.number = random.randrange(self.lower_limit,
```

```
self.upper_limit)
        self.is_debug = False
        self.running = True

    def debug(self):
        self.is_debug = not self.is_debug

    def guess(self, num):
        if num == 'd':
            self.debug()
            return

        if self.is_debug:
            print("Hidden number {}. Your guess is
{}".format(self.number, num))

        if num < self.number:
            print("Too small")
        elif num > self.number:
            print("Too big")
        else:
            print("Bingo")
            self.running = False


g = Game()
g.guess('d')

try:
    g.guess('z')
except Exception as e:
    print(e)

try:
    g.guess('201')
except Exception as e:
    print(e)

try:
    g.guess('-1')
except Exception as e:
    print(e)
```

Exercies: Raise My Exception

This is very similar to the exercise the first chapter about exceptions, but
in this case you need to create your own hierarchy of exception classes.

- Write a function that expects a positive integer as its single parameter.
- Raise exception if the parameter is not a number.
- Raise a different exception if the parameter is not positive.
- Raise a different exception if the parameter is not whole number.
- In each case make sure both the text and the type of the exceptions are different.
- Include the actual value received as an attribute in the exception object.

Solution: spacefight with exceptions

```
import random

class SpaceShipError(Exception):
    def __init__(self, inp):
        self.inp = inp

class NumberTooBigError(SpaceShipError):
    def __str__(self):
        return "Number {} is too big".format(self.inp)

class NumberTooSmallError(SpaceShipError):
    def __str__(self):
        return "Number {} is too small".format(self.inp)


class NotANumberError(SpaceShipError):
    def __str__(self):
```

```python
        return "Not a Number {}".format(self.inp)


class Game:
    def __init__(self):
        self.lower_limit = 0
        self.upper_limit = 200

        self.number = random.randrange(self.lower_limit, self.upper_limit)
        self.is_debug = False
        self.running = True

    def debug(self):
        self.is_debug = not self.is_debug

    def guess(self, num):
        if num == 'd':
            self.debug()
            return

        if self.is_debug:
            print("Hidden number {}. Your guess is {}".format(self.number, num))

        try:
            num =  int(num)
        except Exception:
            raise NotANumberError(num)

        if num > self.upper_limit:
            raise NumberTooBigError(num)

        if num < self.upper_limit:
            raise NumberTooSmallError(num)

        if num < self.number:
            print("Too small")
        elif num > self.number:
            print("Too big")
        else:
            print("Bingo")
            self.running = False


```

```
g = Game()
g.guess('d')

try:
    g.guess('z')
except Exception as e:
    print(e)

try:
    g.guess('201')
except Exception as e:
    print(e)

try:
    g.guess('-1')
except Exception as e:
    print(e)



#while g.running:
#    guess = input("Please type in your guess: ")
#    g.guess(int(guess))
```

```
Hidden number 137. Your guess is z
Not a Number z
Hidden number 137. Your guess is 201
Number 201 is too big
Hidden number 137. Your guess is -1
Number -1 is too small
```

Solution: Raise My Exception

```
class MyValueError(ValueError):
   def __init__(self, val):
       self.value = val

class MyFloatError(MyValueError):
   def __str__(self):
       return "The given parameter {} is a float and not an int.".format(self.value)

```

```
class MyTypeError(MyValueError):
    def __init__(self, val, val_type):
        self.value_type = val_type
        super(MyTypeError, self).__init__(val)

    def __str__(self):
        return "The given parameter {} is of type {} and not int.".format(self.value,\
 self.value_type)

class MyNegativeError(MyValueError):
    def __str__(self):
        return "The given number {} is not positive.".format(self.value)

def positive(num):
    if type(num).__name__ == 'float':
        raise MyFloatError(num)

    if type(num).__name__ != 'int':
        raise MyTypeError(num, type(num).__name__)

    if num < 0:
        raise MyNegativeError(num)

for val in [14, 24.3, "hi", -10]:
    print(val)
    print(type(val).__name__)
    try:
        positive(val)
    except MyValueError as ex:
        print("Exception: {}".format(ex))
        print("Exception type {}".format(type(ex).__name__))

    # Exception, ValueError
```

Exception finally return

```
def div(a, b):
    try:
        print("try")
        c = a / b
```

```
    except Exception:
        print("exception")
        return
    finally:
        print("finally")

div(2, 1)
print('---')
div(2, 0)
```

Warnings

Warnings

```python
from warnings import warn

def foo():
    warn("foo will be deprecated soon. Use bar() instead", DeprecationWarning)
    print("foo still works")


def main():
    foo()
    print("afterfoo")

main()
```

CSV

Reading CSV the naive way

```
Tudor;Vidor;10;Hapci
Szundi;Morgo;7;Szende
Kuka;Hofeherke;100;Kiralyno
Boszorkany;Herceg;9;Meselo
```

```python
import sys, csv

if len(sys.argv) != 2:
    sys.stderr.write("Usage: {} FILENAME\n".format(sys.argv[0]))
    exit()

file = sys.argv[1]
fh = open(file, 'rb')

count = 0
for line in fh:
    line = line.rstrip("\n")
    row = line.split(';')
    print(row)
    count += int(row[2])

print("Total: {}".format(count))
```

python examples/csv/read_csv_split.py examples/csv/process_csv_file.csv

CSV with quotes and newlines

```
Tudor;Vidor;10;Hapci
Szundi;Morgo;7;Szende
```

```
Kuka;"Hofeherke; alma";100;Kiralyno
Boszorkany;Herceg;9;Meselo
```

```
Tudor;Vidor;10;Hapci
Szundi;Morgo;7;Szende
Kuka;"Hofeherke;
alma";100;Kiralyno
Boszorkany;Herceg;9;Meselo
```

Reading a CSV file

```
import sys, csv

if len(sys.argv) != 2:
    sys.stderr.write("Usage: {} FILENAME\n".format(sys.argv[0]))
    exit()

file = sys.argv[1]
count = 0
with open(file) as fh:    # Python 2 might need 'rb'
    rd = csv.reader(fh, delimiter=';')

    for row in rd:
        print(row)
        count += int(row[2])

print("Total: {}".format(count))
```

python examples/csv/read_csv.py examples/csv/process_csv_file.csv

Dialects of CSV files. See also: [csv](csv)

CSV dialects

```python
import csv

for dname in csv.list_dialects():
    print(dname)
    d = csv.get_dialect(dname)
    for n in ['delimiter', 'doublequote', 'escapechar',
            'lineterminator', 'quotechar',
            'quoting', 'skipinitialspace', 'strict']:
        attr = getattr(d, n)
        if attr == '\t':
            attr = '\\t'
        if attr == '\r\n':
            attr = '\\r\\n'
        print("  {:16} '{}'".format(n, attr))
```

```
excel-tab
  delimiter        '\t'
  doublequote      '1'
  escapechar       'None'
  lineterminator   '\r\n'
  quotechar        '"'
  quoting          '0'
  skipinitialspace '0'
  strict           '0'
excel
  delimiter        ','
  doublequote      '1'
  escapechar       'None'
  lineterminator   '\r\n'
  quotechar        '"'
  quoting          '0'
  skipinitialspace '0'
  strict           '0'
```

CSV to dictionary

```
fname,lname,born
Graham,Chapman,8 January 1941
Eric,Idle,29 March 1943
Terry,Gilliam,22 November 1940
Terry,Jones,1 February 1942
```

```
John,Cleese,27 October 1939
Michael,Palin,5 May 1943
```

```python
import csv

file = 'examples/csv/monty_python.csv'
with open(file) as fh:
    rd = csv.DictReader(fh, delimiter=',')
    for row in rd:
        print(row)
```

```
{'lname': 'Chapman', 'born': '8 January 1941', 'fname': 'Graham'}
{'lname': 'Idle', 'born': '29 March 1943', 'fname': 'Eric'}
{'lname': 'Gilliam', 'born': '22 November 1940', 'fname': 'Terry'}
{'lname': 'Jones', 'born': '1 February 1942', 'fname': 'Terry'}
{'lname': 'Cleese', 'born': '27 October 1939', 'fname': 'John'}
{'lname': 'Palin', 'born': '5 May 1943', 'fname': 'Michael'}
```

Exercise: CSV

Given the CSV file of Monty Python troupe, create a dictionary where we can look up information
about them based on the first name. For example:

```python
people = read_csv_file()
print(people["Graham"]["lname"])   # Champman
print(people["John"]["born"])      # 27 October 1939
print(people["Michael"])
     # {'lname': 'Palin', 'born': '5 May 1943', 'fname': 'Michael'}
print(people["Terry"]["lname"])   # Gilliam
```

For extra bonus create another dictionary where we can look up the information based on their fname and lname.

Solution: CSV

```python
import csv

def read_csv_file():
    file = 'examples/csv/monty_python.csv'
    name_of = {}
    with open(file) as fh:
        rd = csv.DictReader(fh, delimiter=',')
        for row in rd:
            name_of[ row['fname'] ] = row
    print(name_of)
    return name_of

people = read_csv_file()
print(people["Graham"]["lname"])  # Champman
print(people["John"]["born"])     # 27 October 1939
print(people["Michael"])
     # {'lname': 'Palin', 'born': '5 May 1943', 'fname': 'Michael'}
print(people["Terry"]["lname"])  # Gilliam
```

Excel

Spreadsheets

- CSV files - use the standard csv library
- Microsoft Excel files (various versions and formats)
- Open Office / Libre Office Calc

Python Excel

- Python Excel
- openpyxl
- xlsxwriter
- xlrd
- xlwt
- xlutils using xlrd and xlwt. Mostly obsolete.

Create an Excel file from scratch

```
import openpyxl
import datetime

wb = openpyxl.Workbook()

ws = wb.active

ws['A1'] = 42

ws['A2'] = datetime.datetime.now()
#ws.column_dimensions['A'].width = 20.0

wb.save("first.xlsx")
```

Worksheets in Excel

```
import openpyxl
import datetime

wb = openpyxl.Workbook()
ws = wb.active
ws['A1'] = 42
ws.title = "First"

ws2 = wb.create_sheet()
ws2.title = "Second sheet"
ws2['A1'] = datetime.datetime.now()
ws2.sheet_properties.tabColor = "1072BA"

wb.save("two_worksheets.xlsx")
```

Add expressions to Excel

Nothing special needed.

```
import openpyxl
import datetime

wb = openpyxl.Workbook()

ws = wb.active

ws['A1'] = 19
ws['A2'] = 23

ws['A3'] = "=A1+A2"

wb.save("expression.xlsx")
```

Format field

```
import openpyxl
import datetime

wb = openpyxl.Workbook()
```

```
ws = wb.active

ws['A1'] = 123456.78
ws['A2'] = 123456.78
ws['A3'] = 123456.78
ws['A4'] = -123456.78
ws['A5'] = datetime.datetime.now()
ws.column_dimensions['A'].width = 20.0

ws['A2'].number_format = '0.00E+00'
ws['A3'].number_format = '#,##0_);[RED](#,##0)'
ws['A4'].number_format = '#,##0_);[RED](#,##0)'

wb.save("format.xlsx")
```

Number series and chart

```
import openpyxl

wb = openpyxl.Workbook()

ws = wb.active
ws.title = "Chart"

a = ["First", 20, 28, 30, 37, 18, 47]
b = ["Second", 35, 30, 40, 40, 38, 35]

# write them as columns
for i in range(len(a)):
    ws.cell(row=i+1, column=1).value = a[i]
    ws.cell(row=i+1, column=2).value = b[i]

lc = openpyxl.chart.LineChart()
lc.title = "Two Lines Chart"
#lc.style=13
data = openpyxl.chart.Reference(ws,
                                min_col=1,
                                min_row=1,
                                max_col=2,
                                max_row=len(a))
lc.add_data(data, titles_from_data=True)

```

```
ws.add_chart(lc, "D1")
wb.save("chart.xlsx")
```

Read Excel file

```
import openpyxl
wb = openpyxl.load_workbook(filename = 'chart.xlsx')
for ws in wb.worksheets:
    print(ws.title)

ws = wb.worksheets[0]
print(ws['A1'].value)
```

Update Excel file

```
import openpyxl

wb = openpyxl.load_workbook(filename = 'chart.xlsx')
for ws in wb.worksheets:
    print(ws.title)

ws = wb.worksheets[0]
c = ["Third", 40, 20, 35, 25, 20, 35]

for i in range(len(c)):
    ws.cell(row=i+1, column=3).value = c[i]

lc = openpyxl.chart.LineChart()
lc.title = "Three Lines Chart"
data = openpyxl.chart .Reference(ws,
                                 min_col=1,
                                 min_row=1,
                                 max_col=3,
                                 max_row=len(c))
lc.add_data(data, titles_from_data=True)

ws.add_chart(lc, "D1")

wb.save("chart.xlsx")
```

Exercise: Excel

- Create a series of 10 random numbers between 1 and 100 and save them in an Excel file in a column.
- Create a graph showing the values.
- Add a second series of 10 random numbers, add them to the Excel file as a second column next to the first one.
- Add a 3rd colum containing the average of the first two columns.
- Update the graph to include all 3 number serieses

XML

XML Data

```xml
<?xml version="1.0"?>
<main>
  <person id="1">
    <fname>Foo</fname>
    <lname>Bar</lname>
  </person>
  <person id="3">
    <fname>Moo</fname>
    <lname>Zorg</lname>
    <email id="home">moo@zorghome.com</email>
    <email id="work">moo@work.com</email>
  </person>
</main>
```

Expat - Callbacks

```python
import xml.parsers.expat

file = 'examples/xml/data.xml'


def start_element(name, attrs):
    print('Start element: {} {}'.format(name, attrs))


def end_element(name):
    print('End element: {}'.format(name))


def char_data(data):
    print('Character data: {}'.format(repr(data)))


```

```
p = xml.parsers.expat.ParserCreate()

p.StartElementHandler = start_element
p.EndElementHandler = end_element
p.CharacterDataHandler = char_data

p.ParseFile(open(file, 'rb'))

print('done')
```

XML DOM - Document Object Model

```
import xml.dom.minidom

file = 'examples/xml/data.xml'

dom = xml.dom.minidom.parse(file)

root = dom.firstChild
print(root.tagName)

print('')

for node in root.childNodes:
    if node.nodeType != node.TEXT_NODE:
        print('name: ', node.tagName)
        print('id: ', node.getAttribute('id'))

print('')

emails = dom.getElementsByTagName("email")
for e in emails:
    print('email', e.getAttribute('id'),
e.firstChild.data)
```

```
main

name:  person
id:  1
name:  person
id:  3

```

```
email home moo@zorghome.com
email work moo@work.com
```

- [xml.dom]()
- [xml.dom.minidom]()

XML SAX - Simple API for XML

```python
import xml.sax

file = 'examples/xml/data.xml'


class EventHandler(xml.sax.ContentHandler):
    def startElement(self, name, attrs):
        print('start', (name, attrs._attrs))

    def characters(self, text):
        if not text.isspace():
            print('text', text)

    def endElement(self, name):
        print('end', name)


xml.sax.parse(file, EventHandler())
```

```
start (u'main', {})
start (u'person', {u'id': u'1'})
start (u'fname', {})
text Foo
end fname
start (u'lname', {})
text Bar
end lname
end person
start (u'person', {u'id': u'3'})
start (u'fname', {})
text Moo
end fname
start (u'lname', {})
```

```
text Zorg
end lname
start (u'email', {u'id': u'home'})
text moo@zorghome.com
end email
start (u'email', {u'id': u'work'})
text moo@work.com
end email
end person
end main
```

* [xml.sax](#)
* [xml.sax.hanldler](#)
* [xml.sax.reader](#)

SAX collect

```python
import xml.sax

file = 'examples/xml/data.xml'

class EventHandler(xml.sax.ContentHandler):
    def __init__(self, c):
        self.path = []
        self.collector = c

    def startElement(self, name, attrs):
        self.path.append({ 'name' : name, 'attr' :
attrs._attrs })

    def characters(self, text):
        self.path[-1]['text'] = text

    def endElement(self, name):
        element = self.path.pop()
        print('End name: ', name)
        if element['name'] == 'email':
            collector.append(element)

collector = []
```

```
xml.sax.parse(file, EventHandler(collector))
print(collector)
```

```
End name:  fname
End name:  lname
End name:  person
End name:  fname
End name:  lname
End name:  email
End name:  email
End name:  person
End name:  main
[{'text': u'moo@zorghome.com', 'name': u'email', 'attr':
{u'id': u'home'}},
 {'text': u'moo@work.com', 'name': u'email', 'attr':
{u'id': u'work'}}]
```

XML elementtree

```
import xml.etree.ElementTree as ET

file = 'examples/xml/data.xml'

tree = ET.parse(file)
root = tree.getroot()
print(root.tag)

for p in root.iter('person'):
    print(p.attrib)

print('')

for p in root.iter('email'):
    print(p.attrib, p.text)

print('')

elements = tree.findall(".//*[@id='home']")
for e in elements:
    print(e.tag, e.attrib)
```

```
main
{'id': '1'}
{'id': '3'}

{'id': 'home'} moo@zorghome.com
{'id': 'work'} moo@work.com

email {'id': 'home'}
```

- xml.etree.elementtree

SciPy - for Scientific Computing in Python

Data Science tools in Python

- SciPy ecosystem of open-source software for mathematics, science, and engineering.
- Biopython tools for biological computation.
- NumPy to handle N-dimensional arrays.
- Pandas Python Data Analysis Library. (Data Frames)
- Matplotlib a 2D plotting library.
- Seaborn data visualization library based on matplotlib.
- Bokeh interactive visualization library.
- SciKit-Learn Machine Learning in Python.
- TensorFlow Machine learning framework. (developed by Google engineer)
- Keras Python Deep learning (neural-network) library. (On top of Tensorflow.)
- Orange machine learning and data visualization tool. Written partially in Python.

- Airflow Workflow management platform
- Luigi Data pipelines (from Spotify)

- Showing speed improvement using a GPU with CUDA and Python with numpy on Nvidia Quadro 2000D

- Octave (Open Source Matlab replacement - not related to Python)

Data Analysis resources

- Exploratory data analysis by John Tukey
- Think Bayes - Bayesian Statistics Made Simple
- Statistical Signal Extraction and Filtering: Structual Time Series Models
- Panel Data

For Econometrics

- Econometric Analysis
- Microeconometric Modeling and Discrete Choice Analysis with Cross Section and Panel Data

For Intro Stats,

- Applied Statistics with R
- Statistics: A Fresh Approach

Datasets

- Climate
- Open Weather map
- PRB

Python and Biology

Biopython

- [Biopython]
- [Biopython GitHub project]
- [Biopython Tutorial and Cookbook]

Biopython background

- Sequence formats (FASTA, FASTQ, EMBL, …)
- FASTA
- FASTQ
- EMBL European Molecular Biology Laboratory
- Gene names symbols

Bio python sequences

```python
from Bio.Seq import Seq

# Nucleotide Sequences
my_dna = Seq("AGTACACTGGTAGGCCTTACAG_T")
print(my_dna)                        # AGTACACTGGTAGGCCTTACAG_T
print(my_dna.complement())           # TCATGTGACCATCCGGAATGTC_A
print(my_dna.reverse_complement())   # A_CTGTAAGGCCTACCAGTGTACT
print(my_dna.transcribe())           # AGUACACUGGUAGGCCUUACAG_U

my_rna = Seq("GAC_U")
print(my_rna)                        # GAC_U
```

```python
print(my_rna.reverse_complement())   # A_GUC
print(my_rna.reverse_complement())   # A_GUC
print(my_rna.transcribe())           # GAC_U
```

```python
from Bio.Seq import Seq

what_is_this = Seq("AGTC_U")
what_is_this.complement()   # ValueError: Mixed RNA/DNA found
```

Download data

Use the NCBI (National Center for Biotechnology Information) database to search manually for nucleotide or tons of other types of data. Then one can download the files manually from the web site.

Read FASTA, GenBank files

For example the data about Orchids in two formats:

- ls_orchid.fasta in FASTA format
- ls_orchid.gbk in GenBank format

Download those files and use them:

```python
from Bio import SeqIO
import requests

def get_file(url, filename):
    res = requests.get(url)
    if res.status_code != 200:
        raise Exception("Could not get file")

    with open(filename, 'w') as fh:
        fh.write(res.text)


```

```python
def process_file(filename, file_type):
    for seq_record in SeqIO.parse(filename, file_type):
        print(seq_record.id)
        print(repr(seq_record.seq))
        print(len(seq_record))


fasta_url = 'https://raw.githubusercontent.com/biopython/biopython/master/Doc/exampl\
es/ls_orchid.fasta'
filename = "ls_orchid.fasta"
file_type = "fasta"
get_file(fasta_url, filename)
process_file(filename, file_type)


genbank_url = "https://raw.githubusercontent.com/biopython/biopython/master/Doc/exam\
ples/ls_orchid.gbk"
filename = "ls_orchid.gbk"
file_type = "genbank"
get_file(genbank_url, filename)
process_file(filename, file_type)
```

Search nucleotids

You can also search the same database programmatically.

```python
from Bio import Entrez
Entrez.email = "gabor@szabgab.com"

term = "Cypripedioideae[Orgn] AND matK[Gene]"

handle = Entrez.esearch(db="nucleotide", term=term, idtype="acc", retmax=30)
record = Entrez.read(handle)
print(record["Count"])        # 538
print(record["IdList"])       # ['MK792700.1', 'MK792699.1', 'MK792698.1', ..., 'MK79\
2681.1']
print(len(record["IdList"])) # 30
```

```python
handle.close()


# term = "Orchid"
# 530077
# ['NZ_SELD00000000.2', 'NZ_SELD02000072.1',
```

Download nucleotids

```python
from Bio import Entrez, SeqIO

Entrez.email = "gabor@szabgab.com"

#doc_id = 'MK792700.1'
doc_id = "EU490707"

# rettype="fasta"
handle = Entrez.efetch(db="nucleotide", id=doc_id, rettype="gb", retmode="text")
data = handle.read()
handle.close()
#print(data)

filename = "temp.data"
with open(filename, 'w') as fh:
    fh.write(data)

file_type = "genbank"
for seq_record in SeqIO.parse(filename, file_type):
    print(seq_record.id)
    print(repr(seq_record.seq))  # A short part of the sequence
    print()
    print(seq_record.seq)    # The full sequence
    print()
    print(len(seq_record.seq))
    print()
    print(seq_record.name)
    print()
    print(seq_record.annotations)
    #print()
    #print(dir(seq_record))
```

Exercise: Nucleotid

- Search for your favorite nucleotid
- Print out the number of results
- Download the 3 different sequences from the list (using the id) in GeneBank format and save them in files using the id as the name of the file and .gb as the extension
- Write a separate script that reads and displays the sequences.

Biology background

- Genetics - inheritance
- Genetic inheritance
- What’s a genome Chp2 1
- What’s a genome Chp4 1
- alleles, genotype, phenotype

Chemistry

Chemistry links

- Python for Chemistry students
- Open Babel The Open Source Chemistry Toolbox
- Chemical table file to describe molecules and chemical reactions.
- Pytim Interfacial Analysis of Molecular Simulations

- Awesome Python Chemistry (article)
- Awesome Python Chemistry (list on GitHub)

- downloads
- Open Babel module
- Pybel

```
import sdf
import pybel
```

Bond length

- Bond length
- Distance between two points Pythagorean theorem
- Video
- XYZ fileformat to specify the molecule geometry.

Covalent radius

- Covalent radius
- Video
- tmpchem/computational_chemistry

Python energy landscape explorer

- Python energy landscape explorer

Other chemistry links

- Periodic table
- Diatomic molecule
- VMD - Visual Molecular Dynamics and application to visualize molecules.

numpy

What is NumPy

- numpy
- High-level mathematical functions to operate on large, multi-dimensional arrays and matrices. **ndarray**

Numpy - vector

```
import numpy as np

a = np.array([3, 4, 7])
print(a)            # [3 4 7]
print(a * 3)        # [ 9 12 21]
print(a + 4)        # [ 7  8 11]
print(a.dtype)      # int64
print(a.ndim)       # 1
print(a.shape)      # (3,)

b = np.array([2, 3.14, -1])
print(b.dtype)      # float64
print(b.shape)      # (3,)

c = np.array(['one', 'two', 'three'])
print(c.dtype)      # <U5    (Unicode less than 5 characters)
```

- Basic types
- dtypes

NumPy 2D arrays

```python
import numpy as np

a = np.array([
    [ 1,  2,  3,  4,  5],
    [ 2,  3,  4,  5,  6]
])

print(a)
# [[1 2 3 4 5]
#  [2 3 4 5 6]]

print(a.shape) # (2, 5)
print(a.ndim)  # 2


print(a * 3)
# [[ 3  6  9 12 15]
#  [ 6  9 12 15 18]]

print(a + 7)
# [[ 8  9 10 11 12]
#  [ 9 10 11 12 13]]
```

Numpy - set type

```python
import numpy as np

a = np.array([3, 4, 7], dtype='int8')
print(a)          # [3 4 7]
print(a * 3)      # [ 9 12 21]
print(a + 4)      # [ 7  8 11]
print(a.dtype)    # int8
```

NumPy arrays: ones and zeros

```python
import numpy as np

c = np.ones(4, dtype='int32')
print(c)            # [1 1 1 1]
print(c.dtype)      # int32
print(c.shape)      # (4,)
```

```python
print()


d = np.zeros(3, dtype='float32')
print(d)            # [ 0.  0.  0.]
print(d.dtype)      # float32
print(d.shape)      # (3,)
print()


a = np.ones([2, 3])
print(a)
# [[1., 1., 1.],
# [1., 1., 1.]]
print(a.dtype)      # float64
print(a.shape)      # (2, 3)
```

Numpy: eye

```python
import numpy as np

a = np.eye(4)
print(a)
print()

b = np.eye(3, 5)
print(b)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]]
```

NumPy array random

```python
import numpy as np

a = np.random.random((2, 5))  # in the range [0.0, 1.0)
print(a)
print()

rng = np.random.default_rng()
b = rng.random(size=(3, 4))
print(b)
```

```
[[0.32151126 0.07688622 0.95666894 0.42396291 0.93592235]
 [0.71406863 0.95152079 0.20199695 0.72628099 0.33545885]]

[[0.46643834 0.71350899 0.40279583 0.85148985]
 [0.19367868 0.53288449 0.97181597 0.86311691]
 [0.70687485 0.78534671 0.16654183 0.9371896 ]]
```

- [random sampling]

NumPy Random integers

```python
import numpy as np

a = np.random.randint(10, size=(3, 4))
print(a)

rng = np.random.default_rng()
b = rng.integers(42, size=(3, 4))
print(b)
```

```
[[1 2 2 6]
 [2 2 9 8]
 [8 8 9 5]]
[[13 31  7 11]
 [22  2  6 18]
 [24 10 12  0]]
```

- integer generator

NumPy array type change by division (int to float)

```python
import numpy as np

a = np.array([3, 4, 7])
print(a.dtype) # int64
print(a.shape)  # (3,)

x = (a / 2)
print(x)        # [ 1.5  2.   3.5]
print(x.dtype)  # float64
print(x.shape)  # (3,)
```

Numpy: Array methods: transpose

```python
import numpy

a = numpy.array([
    [ 1,  2,  3,  4,  5],
    [ 2,  3,  4,  5,  6]
])

b = a.transpose()

print(b)
# [[1 2]
#  [2 3]
#  [3 4]
#  [4 5]
#  [5 6]]

print(a)
# [[1 2 3 4 5]
#  [2 3 4 5 6]]
```

Numpy: reference, not copy

```python
import numpy

a = numpy.array([
    [ 1,   2,   3,   4,   5],
    [ 2,   3,   4,   5,   6]
])

b = a.transpose()
a[0][0] = 42

print(b)
# [[42  2]
#  [2  3]
#  [3  4]
#  [4  5]
#  [5  6]]

print(a)
# [[42  2  3  4  5]
#  [2  3  4  5  6]]
```

Numpy: copy array

```python
import numpy

a = numpy.array([
    [ 1,   2,   3,   4,   5],
    [ 2,   3,   4,   5,   6]
])

b = a.copy().transpose()
a[0][0] = 42

print(b)
# [[1  2]
#  [2  3]
#  [3  4]
#  [4  5]
#  [5  6]]

print(a)
# [[42  2  3  4  5]
#  [2  3  4  5  6]]
```

Numpy: Elementwise Operations on Arrays

```
import numpy as np

a = np.array([
    [ 1,   2,   3,   4,   5],
    [ 2,   3,   4,   5,   6]
])
b = np.array([
    [ 7,   3,   8,   9,   4],
    [ 1,   3,   6,   1,   2]
])

print(a+b)
# [[ 8   5 11 13   9]
#   [ 3   6 10   6   8]]

print(a*b)
# [[ 7   6 24 36 20]
#   [ 2   9 24   5 12]]
```

Numpy: multiply, matmul, dot for vectors

- multiply
- matmul
- dot

```
import numpy as np

a = np.array([3, 4, 7])
b = np.array([6, 5, 2])
print(a)  # [3 4 7]
print(b)  # [6 5 2]

c = np.multiply(a, b)
print(c)  # [18 20 14]

d = np.dot(a, b)
print(d)    # 52
```

```
m = np.matmul(a, b)
print(m)    # 52
```

Numpy: multiply, matmul, dot for vector and matrix

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([1, 2, 4])
print(a)
print(b)
print()

print(a*b)
print(b*a)
print()

print(np.multiply(a, b))

print()
print( np.dot(a, b) )
print( np.matmul(a, b) )
```

```
[[1 2 3]
 [4 5 6]]
[1 2 4]

[[ 1  4 12]
 [ 4 10 24]]
[[ 1  4 12]
 [ 4 10 24]]

[[ 1  4 12]
 [ 4 10 24]]

[17 38]
[17 38]
```

Numpy: multiply, matmul, dot for matrices

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([[1, 3, 4], [7, 8, 0]])
print(a)
print(b)
print()

print(a*b)
print(b*a)
print()

print(np.multiply(a, b))

print()
print( np.dot(a, b.transpose()) )
print( np.matmul(a, b.transpose()) )

print()
print( np.dot(a.transpose(), b) )
print( np.matmul(a.transpose(), b) )
```

```
[[1 2 3]
 [4 5 6]]
[[1 3 4]
 [7 8 0]]

[[ 1  6 12]
 [28 40  0]]
[[ 1  6 12]
 [28 40  0]]

[[ 1  6 12]
 [28 40  0]]

[[19 23]
 [43 68]]
[[19 23]
 [43 68]]

[[29 35  4]
```

```
 [37 46  8]
 [45 57 12]]
[[29 35  4]
 [37 46  8]
 [45 57 12]]
```

Numpy: casting - converting from strings to integer.

```python
import numpy as np

a = np.array([
    [ "12",  "23",  "3",  "4"],
    [ "2",  "3",  "4",  "5"]
])

print(a)
#[['12' '23' '3' '4']
# ['2' '3' '4' '5']]

try:
    b = a + 1
except Exception as e:
    print(e)
# TypeError: ufunc 'add' did not contain a loop with
#    signature matching types dtype('<U3') dtype('<U3') dtype('<U3')


c = a.astype(np.int) + 1
print(c)
# [[13 24  4  5]
# [ 3  4  5  6]]
```

Numpy: indexing 1d array

```python
import numpy as np

a = np.array([1, 1, 2, 3, 5, 8, 13, 21, 34])
print(a)        # [ 1  1  2  3  5  8 13 21 34]

```

```
print(a[4])      # 5
print(a[2:5])    # [2 3 5]
```

Numpy: slice is a reference

The slice in numpy does not copy the data structure

```
import numpy as np

a = np.array([1, 1, 2, 3, 5, 8, 13, 21, 34])
print(a)          # [ 1  1  2  3  5  8 13 21 34]

b = a[2:5]
print(b)    # [2 3 5]

a[2] = 20
print(a)          # [ 1  1 20  3  5  8 13 21 34]
print(b)          # [20  3  5]
```

Numpy: slice - copy

```
import numpy as np

a = np.array([1, 1, 2, 3, 5, 8, 13, 21, 34])
print(a)          # [ 1  1  2  3  5  8 13 21 34]

b = a[2:5].copy()
print(b)    # [2 3 5]

a[2] = 20
print(a)          # [ 1  1 20  3  5  8 13 21 34]
print(b)          # [2  3  5]
```

Numpy: abs value on a Numpy array

```
import numpy as np

a = np.array([[-1, 2, -3], [-4, 5, -7]])
print(a)
```

```
print(a.dtype)
print()

abs_a = np.absolute(a)
print(abs_a)
print(abs_a.dtype)
```

```
[[-1  2 -3]
 [-4  5 -7]]
int64

[[1 2 3]
 [4 5 7]]
int64
```

- absolute

Numpy: Logical not on a Numpy array

```
import numpy as np

a = np.array([True, True, False])
print(a.dtype)
print(a)
print()

not_a = np.logical_not(a)
print(not_a.dtype)
print(not_a)
print()

b = np.array([True, True, False, 0, 42])
print(b.dtype)
print(b)
print()

not_b = np.logical_not(b)
print(not_b.dtype)
print(not_b)
print()
```

```
bool
[ True  True False]

bool
[False False  True]

int64
[ 1  1  0  0 42]

bool
[False False  True  True False]
```

- logical not

Numpy: Vectorize a function

```python
import numpy as np

def fibo(n):
    if n == 1 or n == 2:
        return 1
    a, b = 1, 1
    for _ in range(n-2):
        a, b = b, a + b
    return b

vfibo = np.vectorize(fibo)
a = np.array([
    [1, 2, 3, 4, 5, 6],
    [7, 8, 9, 10, 11, 12],
    ])
print(a)
print(a.dtype)
print()

b = vfibo(a)
print(b)
print(b.dtype)
```

```
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
```

```
int64

[[  1   1   2   3   5   8]
 [ 13  21  34  55  89 144]]
int64
```

- vectorize

Numpy: Vectorize len

```
import numpy as np

animals = np.array(['Cow', 'Elephant', 'Snake', 'Camel',
'Praying Mantis'])
print(animals)

vlen = np.vectorize(len)
print(vlen(animals))
```

```
['Cow' 'Elephant' 'Snake' 'Camel' 'Praying Mantis']
[ 3  8  5  5 14]
```

Numpy: Vectorize lambda

```
import numpy as np

animals = np.array(['Cow', 'Elephant', 'Snake', 'Camel',
'Praying Mantis'])
print(animals)

longer_than_5 = np.vectorize(lambda x: len(x) > 5)
long_animals_bool = longer_than_5(animals)
print(long_animals_bool)
```

```
['Cow' 'Elephant' 'Snake' 'Camel' 'Praying Mantis']
[False  True False False  True]
```

Numpy: Filtering array

```
import numpy as np

animals = np.array(['Cow', 'Elephant', 'Snake', 'Camel',
'Praying Mantis'])
print(animals)

longer_than_5 = np.vectorize(lambda x: len(x) > 5)
long_animals_bool = longer_than_5(animals)
print(long_animals_bool)

long_animals = animals[long_animals_bool]
print(long_animals)
```

```
['Cow' 'Elephant' 'Snake' 'Camel' 'Praying Mantis']
[False  True False False  True]
['Elephant' 'Praying Mantis']
```

Numpy: Filter matrix values

```
import numpy as np
import re

scores = np.array([
    [23, 37, 18, 97, 13, 40],
    [10, 15, 20, 30, 39, 50],
    [99, 20, 83, 42, 19, 31],
    [19, 11, 55, 78, 39, 27]
])
print(scores)
print()

high_scores_boolean = (scores > 20)
print(high_scores_boolean)
print()

high_scores = scores[high_scores_boolean]
print(high_scores)
```

```
[[23 37 18 97 13 40]
 [10 15 20 30 39 50]
 [99 20 83 42 19 31]
 [19 11 55 78 39 27]]

[[ True  True False  True False  True]
 [False False False  True  True  True]
 [ True False  True  True False  True]
 [False False  True  True  True  True]]

[23 37 97 40 30 39 50 99 83 42 31 55 78 39 27]
```

Numpy: Filter matrix rows

```python
import numpy as np

names = np.array(['Mary', 'Bar', 'Joe', 'Jane'])
print(names)
print()

def has_ar(text):
    return "ar" in text
    # if "ar" in text:
        # return True
    # else:
        # return False

names_with_ar_selector = np.vectorize(has_ar)
names_with_ar_bool = names_with_ar_selector(names)
print(names_with_ar_bool)
print()

scores = np.array([
    [23, 37, 18, 97, 13, 40],
    [10, 15, 20, 30, 39, 50],
    [99, 20, 83, 42, 19, 31],
    [19, 11, 55, 78, 39, 27]
])

print(scores[names_with_ar_bool])
```

```
['Mary' 'Bar' 'Joe' 'Jane']

[ True  True False False]

[[23 37 18 97 13 40]
 [10 15 20 30 39 50]]

[[23 37 18 97 13 40]
 [10 15 20 30 39 50]]
```

Numpy: Stat

```python
import numpy as np

scores = np.array([23, 37, 18, 97, 13, 40])
print(scores.sum())            # 228
print(len(scores))             # 6
print(scores.mean())           # 38.0

print(scores.std())            # 28.0950766743 standard deviation
print(scores.var())            # 789.333333333 variance
print(np.median(scores))       # 30.0
print(scores.max())            # 97
print(scores.min())            # 13

print(scores.cumsum())         # [ 23  60  78 175 188 228]
```

Numpy: Serialization

```python
import numpy as np

scores = np.array([
    [23, 37, 18, 97, 13, 40],
    [10, 15, 20, 30, 39, 50],
    [99, 20, 83, 42, 19, 31],
    [19, 11, 55, 78, 39, 27]
])
filename = 'scores.npy'
np.save(filename, scores)

```

```python
s = np.load(filename)
print(s)
```

Numpy: Load from Matlab file

```python
import scipy.io

file_path = 'data.mat'
mat = scipy.io.loadmat(file_path)
data = mat['data']
print(type(data))
print(data)
```

- numpy.ndarray

Numpy: Save as Matlab file

```python
import scipy.io
import numpy as np

data = np.random.random((2, 5))
print(data)

file_path = 'data.mat'
scipy.io.savemat(file_path, {'data': data})
```

Numpy: Horizontal stack vectors (hstack)

```python
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.array([7, 8, 9])
print(a)
print(b)
print(c)
print()

d = np.hstack([a, b])
```

```
print(d)
print()

e = np.hstack([d, c])
print(e)
```

```
[1 2 3]
[4 5 6]
[7 8 9]

[1 2 3 4 5 6]

[1 2 3 4 5 6 7 8 9]
```

Numpy: Append or vertically stack vectors and matrices (vstack)

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.array([7, 8, 9])
print(a)
print(b)
print(c)
print()

m = np.vstack([a, b])
print(m)
print()

d3 = np.vstack([m, c])
print(d3)
```

```
[1 2 3]
[4 5 6]
[7 8 9]

[[1 2 3]
 [4 5 6]]
```

```

[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Numpy uint8

```python
import numpy as np

a = np.array([127], 'uint8')
print(a.dtype)    # uint8
print(a)          # [127]

a[0] += 1         # [128]
print(a)

a[0] -= 1         # [127]
print(a)

a[0] = 255
print(a)          # [255]

a[0] += 1
print(a)          # [0]
```

Numpy int8

```python
import numpy as np

a = np.array([127], 'int8')
print(a.dtype)    # int8
print(a)          # [127]

a[0] += 1         # [-128]
print(a)

a[0] -= 1         # [127]
print(a)

a[0] = 255
print(a)          # [-1]
```

```python

a[0] += 1
print(a)              # [0]
```

Pandas

Pandas

- Pandas Python Data Analysis Library
- Handle data sequences
- A Beginner's Guide to Optimizing Pandas Code for Speed

Planets

```
name,distance,mass
Mercury,0.4,0.055
Venus,0.7,0.815
Earth,1,1
Mars,1.5,0.107
Ceres,2.77,0.00015
Jupiter,5.2,318
Saturn,9.5,95
Uranus,19.6,14
Neptune,30,17
Pluto,39,0.00218
Charon,39,0.000254
```

Pandas Planets - Dataframes

```python
import pandas as pd

df = pd.read_csv('planets.csv', index_col='name')
print(type(df))   # <class 'pandas.core.frame.DataFrame'>
print(df)

df['dm'] = df['distance'] * df['mass']
print(df.head())

```

```
big = df[ df['mass'] > 20 ]
print(big)
```

```
            distance        mass
name
Mercury         0.40    0.055000
Venus           0.70    0.815000
Earth           1.00    1.000000
Mars            1.50    0.107000
Ceres           2.77    0.000150
Jupiter         5.20  318.000000
Saturn          9.50   95.000000
Uranus         19.60   14.000000
Neptune        30.00   17.000000
Pluto          39.00    0.002180
Charon         39.00    0.000254
```

```
            distance     mass        dm
name
Mercury         0.40  0.05500  0.022000
Venus           0.70  0.81500  0.570500
Earth           1.00  1.00000  1.000000
Mars            1.50  0.10700  0.160500
Ceres           2.77  0.00015  0.000415
```

```
            distance   mass      dm
name
Jupiter          5.2  318.0  1653.6
Saturn           9.5   95.0   902.5
```

Pandas Stocks

```
import pandas
import pandas_datareader.data as web
all_data = { ticker: web.get_data_yahoo(ticker) for ticker in ['AAPL', 'IBM', 'MSFT'\
, 'GOOG']}

print(all_data.keys())          # dict_keys(['MSFT', 'IBM', 'AAPL', 'GOOG'])
```

```
print(all_data['MSFT'].keys())  # Index(['Open', 'High', 'Low', 'Close', 'Volume', '\
Adj Close'], dtype='object')

price = pandas.DataFrame({ticker: data['Adj Close'] for ticker, data in all_data.ite\
ms()})

print(price.head())

volume = pandas.DataFrame({ticker: data['Volume'] for ticker, data in all_data.items\
()})

print(volume.tail())

returns = price.pct_change()  # change in percentage
print(returns.head())

# correlation
print(returns.MSFT.corr(returns.IBM))   # 0.49532932971
print(returns.MSFT.corr(returns.AAPL))  # 0.389551383559

# covariance
print(returns.MSFT.cov(returns.IBM))    # 8.50115754064e-05
print(returns.MSFT.cov(returns.AAPL))   # 9.15254855961e-05
```

Pandas Stocks

```
import pandas
prices = pandas.read_csv('stock_prices.csv')
print(prices)
```

Merge Dataframes

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

```

```python
# s = pd.Series([1,3,5,np.nan,6,8])
# dates = pd.date_range('20130101', periods=6)
# x = pd.date_range('20130101', periods=6, freq='3D')
# df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
# df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
# df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABC'))
# df2 = pd.DataFrame({ 'A' : 1.,
#                      'B' : pd.Timestamp('20130102'),
#                      'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
#                      'D' : np.array([3] * 4,dtype='int32'),
#                      'E' : pd.Categorical(["test","train","test","train"]),
#                      'F' : 'foo' })
a = pd.DataFrame({ 'A' : ['Joe', 'Jane', 'Foo', 'Bar'], 'B' : [1, 23, 12, 5] })
b = pd.DataFrame({ 'A' : ['Joe', 'Jane', 'Foo', 'Bar'], 'B' : [7, 10, 27, 1 ] })
#c = pd.DataFrame({ 'A' : ['Jane', 'Joe', 'Foo', 'Bar'], 'B' : [10, 7, 27, 1 ] })
c = b.sort_values(by = 'A')
print(a)
print(b)
print(c)
print('---')
#print(a+b)
x = pd.merge(a, b, on='A')
z = pd.DataFrame({ 'A' : x.A, 'B' : x.B_x + x.B_y })
print(z)



#sa = a.sort_values(by = 'A')
#sc = c.sort_values(by = 'A')
print('-----')
#print(sa)
#print(sc)
y = pd.merge(a, c, on='A')
#print(x)
q = pd.DataFrame({ 'A' : y.A, 'B' : y.B_x + y.B_y })
print(z)
```

Analyze Alerts

```
import pandas
alerts = pandas.read_csv('../../data/alerts.csv')
print(alerts.head())
#print(alerts.count())
```

Analyze IFMetrics

```
import pandas
data = pandas.read_csv('../../data/ifmetrics.csv', na_values=['(null)'])
data.fillna(0, inplace=True)
# , parse_dates=True )
# print(type(data))   # pandas.core.frame.DataFrame
print(data.columns)   # Index([ ... ],  dtype='object', length=135)

#print(data['Utilization In - Threshold Exception Rate'].head(3))

for col in ['Utilization In - Threshold Exception Rate', 'Overall Exception Rate']:
    dt = data[col]
    print(dt[dt != 0])


#print(data.head(1))
#print(data.get_values())
```

Create Excel file for experiment with random data

Input is an excel file with the following columns:

```
genome name, c1, c2, c3, c4, c5, c6
```

- c1-c3 are numbers of cond1

- c4-c6 are numbers of cond2

We would like to filter to the lines that fulfill the following equations:

```
log2(avg(1-3) / avg(4-6)) > limit
other_limit > p.value( )
```

```python
import numpy as np
import pandas as pd
import datetime
import sys

if len(sys.argv) < 2:
    exit("Need number of rows")

rows_num = int(sys.argv[1])
cols_num = 6

start = datetime.datetime.now()
x = np.random.rand(rows_num, cols_num)

genome_names = list(map(lambda i: f'g{i}', range(rows_num)))
column_names = list(map(lambda i: f'm{i}', range(cols_num)))

df = pd.DataFrame(x, index=genome_names, columns=column_names)
df.index.name = 'genome name'

print(df.head())
print(datetime.datetime.now() - start)
df.to_excel('raw_data.xlsx')
print(datetime.datetime.now() - start)
```

Calculate Genome metrics

```python
import pandas as pd
import numpy as np
import datetime
```

```python
import sys

if len(sys.argv) < 2:
    exit("Need filename")
filename = sys.argv[1]


def calculate_averages(row):
    v1 = row.iloc[0:3].mean()
    v2 = row.iloc[3:6].mean()
    return np.log2(v1/v2)

start = datetime.datetime.now()
df = pd.read_excel(filename, index_col='genome name')
print(df.head())
print(datetime.datetime.now() - start)

calculated_value = df.apply(calculate_averages, axis=1)
print(datetime.datetime.now() - start)

threshold = 0.2
filtered_df = df[calculated_value > threshold]
print(filtered_df.head())
print(datetime.datetime.now() - start)
```

Calculate Genome metrics - add columns

```python
import pandas as pd
import numpy as np
import datetime
import sys

if len(sys.argv) < 2:
    exit("Need filename")
filename = sys.argv[1]


def calculate_averages(row):
    v1 = row.iloc[0:3].mean()
    v2 = row.iloc[3:6].mean()
    return np.log2(v1/v2)

start = datetime.datetime.now()
```

```python
df = pd.read_excel(filename, index_col='genome name')
print(df.head())
print(datetime.datetime.now() - start)

# create a new column of the calculated value
df['calculated_value'] = df.apply(calculate_averages, axis=1)
print(datetime.datetime.now() - start)

threshold = 0.2
filtered_df = df[df['calculated_value'] > threshold]
print(filtered_df.head())
print(datetime.datetime.now() - start)
```

Calculate Genome metrics - vectorized

```python
import pandas as pd
import numpy as np
import datetime
import sys

if len(sys.argv) < 2:
    exit("Need filename")
filename = sys.argv[1]

def calculate_averages(df):
    v1 = df.iloc[:, 0:3].mean(axis=1)   # axis=1 -> calculate the mean row-wise
    v2 = df.iloc[:, 3:6].mean(axis=1)
    return np.log2(v1/v2)

start = datetime.datetime.now()
df = pd.read_excel(filename, index_col='genome name')
print(df.head())
print(datetime.datetime.now() - start)

calculated_value = calculate_averages(df)
print(datetime.datetime.now() - start)

threshold = 0.2
filtered_df = df[calculated_value > threshold]
print(filtered_df.head())
print(datetime.datetime.now() - start)
```

Calculate Genome metrics - vectorized numpy

```python
import pandas as pd
import numpy as np
import datetime
import sys

if len(sys.argv) < 2:
    exit("Need filename")
filename = sys.argv[1]

def calculate_averages(df_numpy):
    v1 = df_numpy[:, 0:3].mean(axis=1)
    v2 = df_numpy[:, 3:6].mean(axis=1)
    return np.log2(v1/v2)

start = datetime.datetime.now()
df = pd.read_excel(filename, index_col='genome name')
print(df.head())
print(datetime.datetime.now() - start)

# the .values attribute changes from Pandas to numpy array
# (no more iloc, no headers, no index)
calculated_value = calculate_averages(df.values)
print(datetime.datetime.now() - start)

threshold = 0.2
filtered_df = df[calculated_value > threshold]
print(filtered_df.head())
print(datetime.datetime.now() - start)
```

Genes using Jupyter

```
cd examples/pandas/
jupyter notebook genes.ipynb
```

Combine columns

```
fname,lname,age
Foo,Bar,100
Alma,Matter,78
Buzz,Lightyear,23
```

```python
import pandas as pd

filename = 'data.csv'
df = pd.read_csv(filename)
print(df)


def combine(row):
    return row['lname'] + '_' + row['fname']


df['combined'] = df.apply(combine, axis=1)
print(df)


def new_column(row):
    columns = ['lname', 'age', 'fname']
    return '_'.join(map(lambda name: str(row[name]), columns))

df['combined'] = df.apply(new_column, axis=1)
print(df)
```

```
  fname      lname  age
0   Foo        Bar  100
1  Alma     Matter   78
2  Buzz  Lightyear   23
  fname      lname  age        combined
0   Foo        Bar  100         Bar_Foo
1  Alma     Matter   78     Matter_Alma
2  Buzz  Lightyear   23  Lightyear_Buzz
  fname      lname  age           combined
0   Foo        Bar  100        Bar_100_Foo
1  Alma     Matter   78     Matter_78_Alma
2  Buzz  Lightyear   23  Lightyear_23_Buzz
```

Pandas more

```
df.iloc[:, 4:10].sum(axis=1)

# rearrange order of columns
cols = list(df.columns)
df = df[ cols[0:4], cols[-1], cols[4:20] ]

to_csv('file.csv', index=False)
to_excel()

read_csv(filename, delimiter='\t')
to_csv(filename, sep='\t')


# after filtering out some rows:
df = df.reset_index()
df.reset_index(drop=True, inplace=True)


fileter with
df.loc[ ~df['Name'].str.contains('substring') ]

can also have regex=True parameter

# replace values
df[ df['Name'] == 'old', 'Name' ] = 'new'
```

Pandas Series

```
import pandas

s = pandas.Series([1, 1, 2, 3, 5, 8])
print(s)

# 0    1
# 1    1
# 2    2
# 3    3
# 4    5
# 5    8
# dtype: int64
```

```

print(s.values)   # [1 1 2 3 5 8]
print(s.index)    # RangeIndex(start=0, stop=6, step=1)

print('----')
print(s.sum())       # 20
print(s.count())     # 6
print(s.mean())      # 3.33333333333
print(s.median())    # 2.5
print(s.std())       # 2.73252020426
print(s.cumsum())

# 0      1
# 1      2
# 2      4
# 3      7
# 4     12
# 5     20
# dtype: int64
```

Pandas Series with names

```
import pandas

planets        = ['Mercury', 'Venus', 'Earth', 'Mars']
distances_raw = [    0.4   ,    0.7   ,        1,   1.5   ]
masses_raw     = [    0.055,    0.815,        1,   0.107]

distance = pandas.Series(distances_raw, index = planets)
mass     = pandas.Series(masses_raw,    index = planets)

print(distance)

# Mercury      0.40
# Venus        0.70
# Earth        1.00
# Mars         1.50
# dtype: float64


print(distance.index)
# Index(['Mercury', 'Venus', 'Earth', 'Mars'],
dtype='object')
```

```
print(distance[distance < 0.8])
# Mercury    0.4
# Venus      0.7
# dtype: float64


print('------')
print(distance/mass)
# Mercury         7.272727
# Venus           0.858896
# Earth           1.000000
# Mars           14.018692
# dtype: float64
```

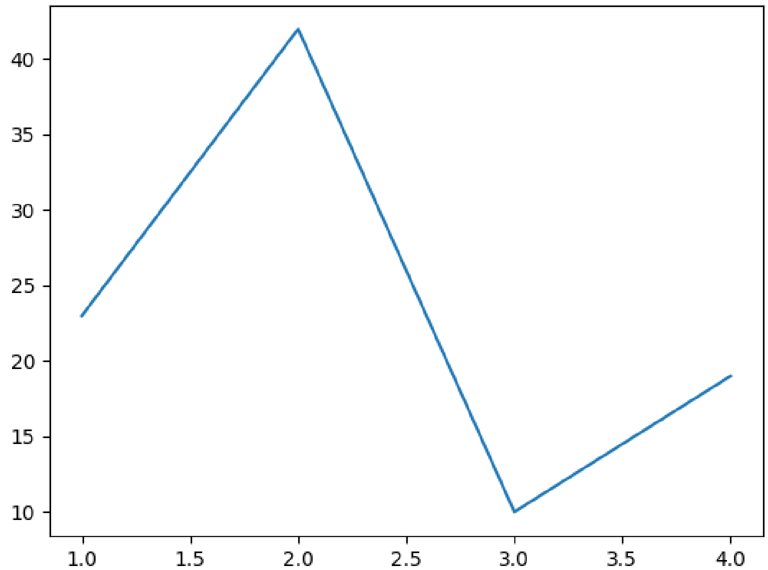
Matplotlib

About Matplotlib

- matplotlib

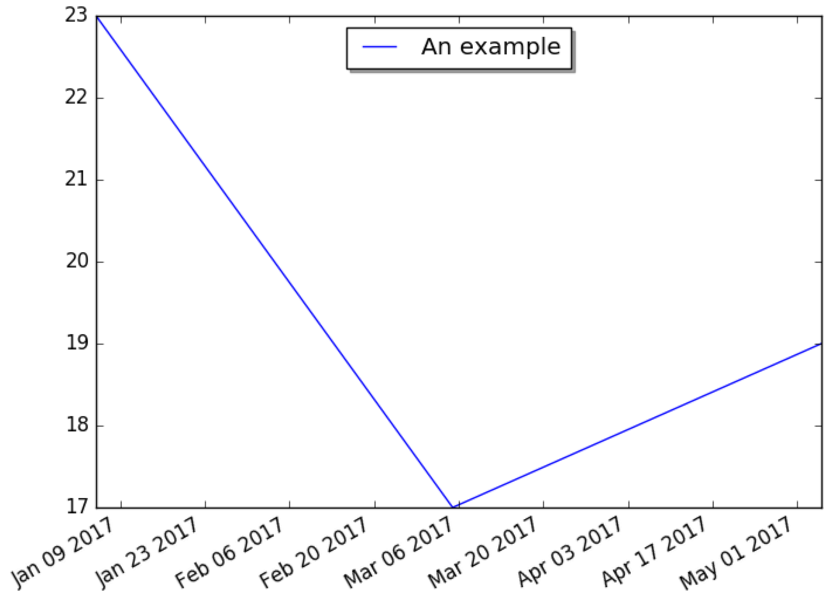
Matplotlib Line

```
import matplotlib.pyplot as plt

plt.plot([ 1,  2,   3,  4 ],[ 23, 42, 10, 19 ])
#fig, ax = plt.subplots()
#ax.plot(
#     [ 1,  2,   3,  4 ],
#     [ 23, 42, 10, 19 ],
#)
plt.show()
#plt.savefig('line.png')
```



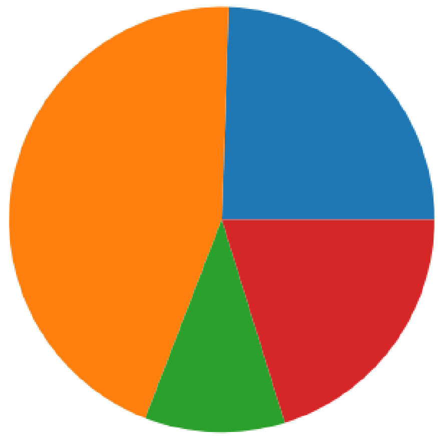
Matplotlib Line with dates

```
import datetime
import matplotlib.pyplot as plt

fig, subplots = plt.subplots()
subplots.plot(
    [datetime.date(2017, 1, 5), datetime.date(2017, 3, 5), datetime.date(2017, 5, 5)\
],
    [ 23, 17, 19 ],
    label='An example',
)
subplots.legend(loc='upper center', shadow=True)
fig.autofmt_xdate()
plt.show()
#plt.savefig('line_with_dates.png')
```



Matplotlib Simple Pie

```
import matplotlib.pyplot as plt

plt.pie([ 23, 42, 10, 19 ])

plt.show()
#plt.savefig('simple_pie.png')
```



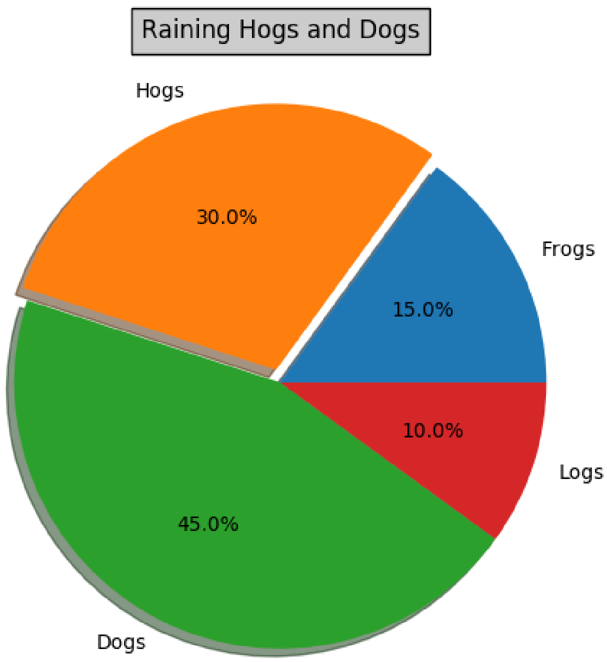
Matplotlib Simple Pie with params

```
import matplotlib.pyplot as plt

plt.pie(
    x = [ 23, 42, 10, 19 ],
    #explode = [0, 0, 0.1, 0.3],
    #labels = ["failure", "success", "maybe", "what"],
    #colors = ["red",  "green",  "blue", "#A395C1"],
    #shadow = True,
    #radius = 2,
)

plt.show()
```

- pyplot pie

Matplotlib Pie

```
import matplotlib.pyplot as plt


# Make a square figure and axes
plt.figure(1, figsize=(6, 6))
#ax = plt.axes([0.1, 0.1, 0.8, 0.8])

labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
fracs = [15, 30, 45, 10]

explode = (0, 0.05, 0, 0)
plt.pie(fracs,
    explode=explode,
    labels=labels,
    autopct='%1.1f%%',
    shadow=True)
plt.title('Raining Hogs and Dogs',
    bbox={'facecolor': '0.8', 'pad': 5})

plt.show()
#plt.savefig('pie.png')
#plt.savefig('pie.pdf')
```



Matplotlib Pie 2

```python
import matplotlib.pyplot as plt

cases = {
    'success': 38,
    'failure': 7,
    'skipped': 3,
    'xfailed': 8,
    'xpassed': 4,
}

explode = (0, 0.1, 0.1, 0.1, 0.1)
labels = cases.keys()
sizes = cases.values()

fig1, ax1 = plt.subplots()
```

```
ax1.pie(sizes, explode=explode, labels=labels,
autopct='%1.1f%%', shadow=True, start\
angle=90)
ax1.axis('equal')

plt.tight_layout()
plt.show()
```

Plot, scatter, histogram

- plot - line
- scatter - just the values
- histogram (to group the values into bins)
- plt.hist(data, bin=10)

Seaborn

Searborn use examples

seaborn

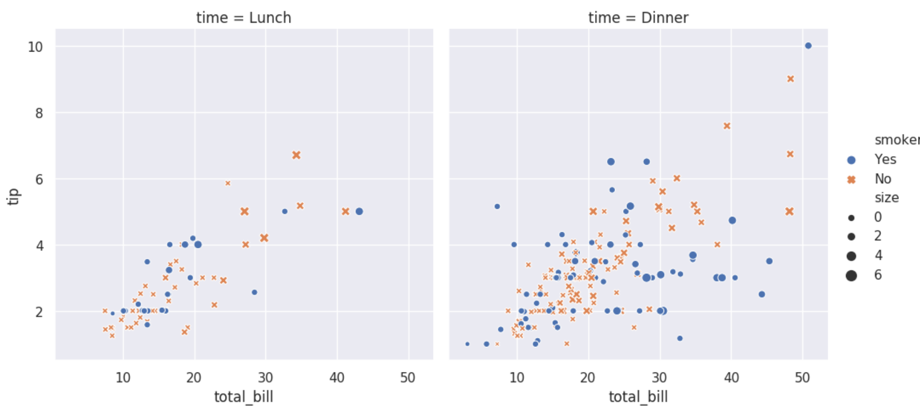
In Jupyter notebook type `%matplotlib` before writing the seaborn code.

In plain Python `import matplotlib`, then assign the result of the ploting function
to a variable, and call `matplotlib.pyplot.show(r)`.

Seaborn tip

```
"""
Source : https://seaborn.pydata.org/introduction.html
"""

import seaborn as sns

sns.set()   # Apply the default default seaborn theme, scaling, and color palette. Op\
tional.

tips = sns.load_dataset("tips")   # Load example dataset into Pandas DataFrame
#print(type(tips))

# print(tips)

plot = sns.relplot(
    x = "total_bill",
    y = "tip",
    col = "time",
    hue = "smoker",
```

```
    style = "smoker",
    size = "size",
    data = tips)

# print(type(plot))    # seaborn.axisgrid.FacetGrid
plot.savefig("tips.png")
```



Seaborn Anscombes Quartet

```
"""
Anscombe's quartet
==================

_thumb: .4, .4

Source: https://seaborn.pydata.org/examples/anscombes_quartet.html
"""
import seaborn as sns
import matplotlib
sns.set(style="ticks")

# Load the example dataset for Anscombe's quartet
df = sns.load_dataset("anscombe")

# Show the results of a linear regression within each dataset
r = sns.lmplot(
```

```
    x="x",
    y="y",
    col="dataset",
    hue="dataset",
    data=df,
    col_wrap=2,
    ci=None,
    palette="muted",
    height=4,
    scatter_kws={"s": 50, "alpha": 1})

matplotlib.pyplot.show(r)
```

Jupyter notebooks

Jupyter on Windows

On Windows install Anaconda
and then you'll be able to run Jupyter notebook from the start menu.

Jupyter on Linux and OSX

Install

For Linux and OSX I recommend using **virtualenv** and installing with **pip**.

```
virtualenv -p python3 ~/venv3
source ~/venv3/bin/activate
pip install jupyter
```

Run

```
cd examples/jupyter/
jupyter notebook
```

- Your browser should open. If not, there is a link in the terminal.

Jupyter add

- Open an existing notebook (ipynb file). e.g examples/jupyter/add.ipynb
- Create new notebook.
- File - Save As
- …
- Quit - shut down the notebook server.

```
def add(x, y):
    return x+y

add(2,3)
```

Planets

```
Planet name,Distance (AU),Mass
Mercury,0.4,0.055
Venus,0.7,0.815
Earth,1,1
Mars,1.5,0.107
Ceres,2.77,0.00015
Jupiter,5.2,318
Saturn,9.5,95
Uranus,19.6,14
Neptune,30,17
Pluto,39,0.00218
Charon,39,0.000254
```

Jupyter notebook Planets

```
%config IPCompleter.greedy=True
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


planets = pd.read_csv('planets.csv')
planets

planets.__class__.__name__
planets.columns
planets.dtypes
planets.index
planets.values
planets.describe()

#planets.sort_values('Mass', ascending=False)
planets.sort_values('Planet name', ascending=False)

planets.Mass
planets['Planet name']
planets[2:5]
planets.loc[3:6, ['Mass', 'Planet name']]
planets.Mass > 1

planets[planets.Mass > 1]
planets['Planet name'].isin(['Earth', 'Mars'])
planets[ planets['Planet name'].isin(['Earth', 'Mars']) ]

planets[(planets.Mass > 1) &amp; (planets.Mass < 100)]
# element-wise boolean and

center = 'Earth'
this = planets[ planets['Planet name'] == center ]
mass = this.iloc[0]['Mass']
dist = this.iloc[0]['Distance (AU)']

# gravitational force is F = G * (mass1*mass2) / D**2
G = 6
D = abs(dist - planets['Distance (AU)'])
D
```

```

forces = planets.copy()
forces

G * (planets.Mass * mass) / D**2
forces['F'] = G * (planets.Mass * mass) / D**2
forces.drop(columns = 'Mass', inplace=True)
forces.drop(columns = 'Distance (AU)', inplace=True)
forces
```

Jupyter StackOverflow

- Download the latest dataset from the survey.
- unzip the file. Feel free to remove the __MACOSX/ directory.

```
%config IPCompleter.greedy=True
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# The following might not work on your computer if it does not have enough free memo\
ry
df = pd.read_csv('survey_results_public.csv')
df

df.size # size in memory 7,555,055 it is too big if you only have 8gb memory

df.count()

df.info()

df.describe() # only few columns were identified to have numeric values

df.head(3)
```

Jupyter StackOverflow - selected columns

```
df = pd.read_csv('survey_results_public.csv', usecols=['Country', 'OpenSourcer', 'Co\
mpTotal'])
```

Jupyter processing chunks

```
for chunk in pd.read_csv('survey_results_public.csv', chunksize=chunksize):
    process(chunk)
```

Jupyter StackOverflow - selected rows

```
# Load only data from a specific country.

country_name = 'Israel'
df = None
for chunk in pd.read_csv('survey_results_public.csv', chunksize=10000):
    part = chunk[ chunk['Country'] == country_name ]
    if df is None:
        df = part.copy(deep = True)
    else:
        df = df.append(part.copy(deep = True), ignore_index = True)


df.count()
df.size
```

Jupyter StackOverflow - biggest countries (in terms of number of responses)

```
country_count = df['Country'].value_counts()
country_count

type(country_count) # pandas.core.series.Series
```

```
# country_count.__class__.__name__  # Series

# We can use it either as a dictionary or as a list
country_count['United States'] # 20949
# country_count[0] # 20949
# country_count['Israel']

# Take the top 20 countries
first20 = country_count.head(20)
first20
# type(first20) # Series

# first20 = country_count.iloc[0:20] # part of the Series
# first20
# type(first20) # Series

#first20 = country_count[0:20]
# first20
# type(first20) # Series

# Select rows of the "biggest" countries
first20.keys()
```

Jupyter StackOverflow - historgram

```
# Historgram of the top 20 countries
first20.hist(bins = 20)

# Plot using Seaborn
plot = sns.relplot(data = first20)
plot.set_xticklabels(rotation=90)
```

Jupyter StackOverflow - filter by country

```
df['Country'] == 'Israel'
df [ df['Country'] == 'Israel' ]

df[ df['Country'].isin( ['India', 'Israel'] ) ]
df[ df['Country'].isin( first20.keys() ) ]
```

Jupyter StackOverflow - OpenSourcer

```
df['OpenSourcer'].value_counts()

df['OpenSourcer'].unique()
```

Jupyter StackOverflow - cross tabulation

```
# Crosstabulation
first10 = country_count.head(10)
subset = df[ df['Country'].isin( first10.keys() ) ]
# subset.count()

# subset['OpenSourcer'].value_counts()
grouped = subset.groupby('Country')['OpenSourcer'].value_counts()
# grouped.plot.bar(figsize=(15,15))

pd.crosstab(subset['Country'], df['OpenSourcer'])

ct = pd.crosstab(subset['Country'], df['OpenSourcer']).apply(lambda r: 100 * r/r.sum\
(), axis=1)
ct

ct.transpose().hist(figsize=(15, 15))
```

Jupyter StackOverflow - salaries

```
# Try to show the average salary by country
grp = df.groupby('Country').mean().round({'CompTotal' : 0})
#grp['CompTotal']
pd.set_option('display.float_format', lambda x: '{:,}'.format(x))
grp.sort_values('CompTotal', ascending=False)
```

Jupyter StackOverflow - replace values

```
nd = df.replace({'OpenSourcer' : {
        'Never' : 0,
        'Less than once per year' : 1,
        'Less than once a month but more than once per year' : 2,
        'Once a month or more often' : 3,
       } })
nd
nd.describe()
nd.groupby('Country').mean().sort_values('OpenSourcer', ascending=False)
```

Jupyter StackOverflow - selected rows

```
# Distribution of responses among countries.
# Relation of Open Source contribution to experience.
# Open Source contribution by country.
# Look at the pdf file and create similar reports for a specific country
```

Jupyter notebook Intellisense (TAB completition)

```
%config IPCompleter.greedy=True
```

Jupyter examples

```
examples/jupyter/planets.csv

examples/jupyter/planets.ipynb

examples/jupyter/numpy_matrix.ipynb

examples/jupyter/seaborn_tips.ipynb
```

IPy Widgets

- Interact
- Widget list

Testing

Traditional Organizations

- Months of planning
- Many months of development
- Many months of testing / qa
- Release once every few months or once a year
- (Waterfall)

Quality Assurance

- Nightly build
- Testing new features
- Testing bug fixes
- Maybe testing critical features again and again…
- …or maybe not.
- Regression testing?
- Testing / qa has a huge boring repetative part.
- It is also very slow and expensive.

Web age Organizations

- Very frequent releases (20-30 / day!)
- Very little time for manual testing
- CI - Continuous Integration
- CD - Continuous Delivery
- CD - Continuous Deployment

TDD vs Testing as an Afterthought

- TDD - Test Driven Development.

*

- Testing as an afterthought:
- Exiting product
- Mostly works
- Hard to test

Why test?

- Business Value
- Avoid regression
- Better Software Design (TDD)
- Your Sanity

Testing Modes

- Functional testing
- Unit testing
- Integration testing
- Acceptance testing (BDD Behavior-driven development?)
- White box
- Black box
- Regression testing
- Usability testing
- Performance testing
- Load testing
- Security testing
- …

Testing Applications

- Web site
- Web application
- Web API / Microservice (JSON, XML)
- Mobile Application
- Desktop Application (GUI)
- Command-line tool (CLI)
- Batch process

Testing What to test?

- How would you check that they work as expected?
- What if they get invalid input?
- Edge cases? (e.g. 0, -1, 131314134141)
- A value that is too big or two small.
- Invalid or no response from third-party system.

Testing in Python

- Doctest
- Unittest
- Pytest
- Nose
- Nimoy
- Hypothesis
- Selenium
- Tox

Testing Environment

- Git (or other VCS)
- Virtualenv
- Docker
- …

Testing Setup - Fixture

- Web server
- Databases
- Other machines
- Devices
- External services

Testing Resources

- [AB Testing] Alan and Brent talk about Modern Testing

Testing with unittest

Use a module

We have a module called `mymath` that has two methods: `add` and `div`.

```
import mymath
print( mymath.add(2, 3) )
print( mymath.div(6, 2) )
```

```
import mymath
import sys

if len(sys.argv) != 4:
    exit("Usage: {} [add|div] INT INT".format(sys.argv[0]))

if sys.argv[1] == 'add':
    print(mymath.add(int(sys.argv[2]), int(sys.argv[3])))
if sys.argv[1] == 'div':
    print(mymath.div(int(sys.argv[2]), int(sys.argv[3])))
```

Test a module

```
import unittest
import mymath

class TestMath(unittest.TestCase):

    def test_match(self):
        self.assertEqual(mymath.add(2, 3), 5)
        self.assertEqual(mymath.div(6, 3), 2)
        self.assertEqual(mymath.div(42, 1), 42)
        self.assertEqual(mymath.add(-1, 1), 0)

```

```
if __name__ == '__main__':
    unittest.main()
```

The tested module

```
def add(x, y):
    """Adding two numbers

    >>> add(2, 3)
    5

    """
    return x + y

def div(x, y):
    """Dividing two numbers

    >>> div(8, 2)
    4
    >>> div(8, 0)
    Traceback (most recent call last):
    ...
    ZeroDivisionError: integer division or modulo by zero

    """
    return x / y


#print add(2, 3, 4)
```

Testing - skeleton

```
import unittest

def add(x, y):
    return x+y

class Something(unittest.TestCase):

    def setUp(self):
        pass
```

```python
        #print("setup")

    def tearDown(self):
        pass
        #print("teardown")

    def test_something(self):
        self.assertEqual(add(2, 3), 5)
        self.assertEqual(add(0, 3), 3)
        self.assertEqual(add(0, 3), 2)


    def test_other(self):
        self.assertEqual(add(-3, 3), 0)
        self.assertEqual(add(-3, 2), 7)
        self.assertEqual(add(-3, 2), 0)


if __name__ == '__main__':
    unittest.main()
```

Testing

```python
import unittest

class TestReg(unittest.TestCase):

    def setUp(self):
        self.str_number = "123"
        self.str_not_number = "12x"

    def test_match1(self):
        self.assertEqual(1, 1)
        self.assertRegexpMatches(self.str_number,
r'^\d+$')

    def test_match2(self):
        self.assertEqual(1, 1)
        self.assertRegexpMatches(self.str_not_number,
r'^\d+$')

if __name__ == '__main__':
```

```
    unittest.main()

```

Test examples

- pylev unittest
- weighted-levenshtein

Testing with PyTest

Pytest features

- Organize and run test per directory (test discovery)
- Run tests by name matching
- Run tests by mark (smoke, integration, db)
- Run tests in parallel with the xdist plugin.
- Create your own fixtures and distribute them.
- Create your own plugins and distribute them.

Pytest setup

Python 2

```
virtualenv venv2
source venv2/bin/activate
pip install pytest
```

Python 3

```
virtualenv venv3 -p python3
source venv3/bin/activate
pip install pytest
```

Python 3 Debian/Ubuntu

```
apt-get install python3-pytest
```

Python 3 RedHat/Centos

```
yum install python3-pytest
```

Testing with Pytest

A module called `mymath` with two functions: `add` and `div`.

```
def add(x, y):
    """Adding two numbers

    >>> add(2, 3)
    5

    """
    return x + y

def div(x, y):
    """Dividing two numbers

    >>> div(8, 2)
    4
    >>> div(8, 0)
    Traceback (most recent call last):
    ...
    ZeroDivisionError: integer division or modulo by zero

    """
    return x / y
```

Testing functions

```
import mymath

def test_math():
    assert mymath.add(2, 3)   == 5
    assert mymath.div(6, 3)   == 2
    assert mymath.div(42, 1)  == 42
    assert mymath.add(-1, 1)  == 0
```

Testing class and methods

```python
import mymath

class TestMath():
    def test_math(self):
        assert mymath.add(2, 3)  == 5
        assert mymath.div(6, 3)  == 2
        assert mymath.div(42, 1) == 42
        assert mymath.add(-1, 1) == 0
```

Pytest - execute

```
pytest test_mymath.py
```

```
============================= test session starts ==============================
platform darwin -- Python 3.6.3, pytest-3.3.0, py-1.5.2, pluggy-0.6.0
rootdir: /Users/gabor/work/training/python, inifile:
collected 1 item

examples/pytest/test_mymath.py . [100%]

=========================== 1 passed in 0.01 seconds ===========================
```

Pytest - execute

```
pytest
python -m pytest
```

Pytest simple module to be tested

An anagram is a pair of words containing the exact same letters in different order. For example:

- listen silent

- elvis lives

```python
def is_anagram(a_word, b_word):
    return sorted(a_word) == sorted(b_word)
```

Pytest simple tests - success

```python
from mymod_1 import is_anagram

def test_anagram():
    assert is_anagram("elvis", "lives")
    assert is_anagram("silent", "listen")
    assert not is_anagram("one", "two")
```

Pytest simple tests - success output

```
$ pytest test_mymod_1.py

===================== test session starts ======================
platform darwin -- Python 3.5.2, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
rootdir: /examples/python/pt, inifile:
collected 1 items

test_mymod_1.py .

=================== 1 passed in 0.03 seconds ===================
```

Pytest simple tests - failure

- Failure reported by user: **is_anagram("anagram", "nag a ram")** is expected to return true.
- We write a test case to reproduce the problem. It should fail now.

```python
from mymod_1 import is_anagram

def test_anagram():
    assert is_anagram("elvis", "lives")
    assert is_anagram("silent", "listen")
    assert not is_anagram("one", "two")

def test_multiword_anagram():
    assert is_anagram("ana gram", "naga ram")
    assert is_anagram("anagram", "nag a ram")
```

Pytest simple tests - failure output

```
$ pytest test_mymod_2.py

===================== test session starts ======================
platform darwin -- Python 3.5.2, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
rootdir: /examples/python/pt, inifile:
collected 2 items

test_mymod_2.py .F

============================ FAILURES ============================
____________________ test_multiword_anagram ____________________

    def test_multiword_anagram():
        assert is_anagram("ana gram", "naga ram")
>       assert is_anagram("anagram", "nag a ram")
E       AssertionError: assert False
E        +  where False = is_anagram('anagram', 'nag a ram')

test_mymod_2.py:10: AssertionError
============== 1 failed, 1 passed in 0.09 seconds ==============
```

Exercise: test math functions

- Test methods of the math module.
- ceil
- factorial
- gcd

Exercise: test this app

Write tests for the `swap` and `average` functions of the `app` module. Can you find a bug?

```
def swap(txt):
    '''
    >>> half("abcd"))
    cdab
    '''
    return txt[int(len(txt)/2):] + txt[:int(len(txt)/2)]

def average(*numbers):
    '''
    >>> average(2, 4, 6)
    4
    '''
    s = 0
    c = 0
    for n in numbers:
        s += n
        c += 1
    return s/c
```

Exercise: test the csv module

- [csv](csv)
- Create a CSV file, read it and check if the results are as expected!
- Test creating a CSV file?
- Test round trip?

Solution: Pytest test math functions

```python
import math

def test_gcd():
    assert math.gcd(6, 9) == 3
    assert math.gcd(17, 9) == 1

def test_ceil():
    assert math.ceil(0) == 0
    assert math.ceil(0.1) == 1
    assert math.ceil(-0.1) == 0

def test_factorial():
    assert math.factorial(0) == 1
    assert math.factorial(1) == 1
    assert math.factorial(2) == 2
    assert math.factorial(3) == 6
```

```python
import math
import pytest

def test_math():
    with pytest.raises(Exception) as exinfo:
        math.factorial(-1)
    assert exinfo.type == ValueError
    assert str(exinfo.value) == 'factorial() not defined for negative values'


    with pytest.raises(Exception) as exinfo:
        math.factorial(1.2)
    assert exinfo.type == ValueError
    assert str(exinfo.value) == 'factorial() only accepts integral values'
```

Solution: Pytest test this app

```python
import app

def test_swap():
```

```python
    assert app.swap("abcd") == "cdab"
    assert app.swap("abc") == "bca"
    assert app.swap("abcde") == "cdeab"
    assert app.swap("a") == "a"
    assert app.swap("") == ""

def test_average():
    assert app.average(2, 4) == 3
    assert app.average(2, 3) == 2.5
    assert app.average(42) == 42
    #assert app.average() == 0
```

Solution: test the csv module

```
Tudor;Vidor;10;Hapci
Szundi;Morgo;7;Szende
Kuka;"Hofeherke;
alma";100;Kiralyno
Boszorkany;Herceg;9;Meselo
```

```python
import csv


def test_csv():
    filename = '../../examples/csv/process_csv_file_newline.csv'
    with open(filename) as fh:
        rd = csv.reader(fh, delimiter=';')
        assert rd.__next__() == ['Tudor', 'Vidor', '10', 'Hapci']
        assert rd.__next__() == ['Szundi', 'Morgo', '7', 'Szende']
        assert rd.__next__() == ['Kuka', 'Hofeherke;\nalma', '100', 'Kiralyno']
        assert rd.__next__() == ['Boszorkany', 'Herceg', '9', 'Meselo']
```

PyTest bank deposit

```python
class NegativeDeposite(Exception):
    pass

class Bank:
    def __init__(self, start):
        self.balance = start

    def deposit(self, money):
        if money < 0:
            raise NegativeDeposite('Cannot deposit negative sum')
        self.balance += money
        return
```

PyTest expected exceptions (bank deposit)

```python
import pytest
from banks import Bank, NegativeDeposite


def test_negative_deposit():
    b = Bank(10)
    with pytest.raises(Exception) as exinfo:
        b.deposit(-1)
    assert exinfo.type == NegativeDeposite
    assert str(exinfo.value) == 'Cannot deposit negative sum'
```

```
pytest test_bank.py

test_bank.py .
```

PyTest expected exceptions (bank deposit) - no exception happens

Pytest properly reports that there was no exception where an exception was expected.

```python
class NegativeDeposite(Exception):
    pass

class Bank:
    def __init__(self, start):
        self.balance = start

    def deposit(self, money):
        #if money < 0:
        #    raise NegativeDeposite('Cannot deposit negative sum')
        self.balance += money
        return
```

```
    def test_negative_deposit():
        b = Bank(10)
        with pytest.raises(NegativeDeposite) as e:
>           b.deposit(-1)
E           Failed: DID NOT RAISE <class 'Exception'>
```

PyTest expected exceptions (bank deposit) - different exception is raised

```python
class NegativeDeposite(Exception):
    pass

class Bank:
    def __init__(self, start):
        self.balance = start

    def deposit(self, money):
        if money < 0:
            raise ValueError('Cannot deposit negative sum')
        self.balance += money
        return
```

```
    def test_negative_deposit():
        b = Bank(10)
        with pytest.raises(Exception) as exinfo:
```

```
            b.deposit(-1)
>       assert exinfo.type == NegativeDeposite
E       AssertionError: assert <class 'ValueError'> == NegativeDeposite
E        +  where <class 'ValueError'> = <ExceptionInfo ValueError tblen=2>.type
```

PyTest expected exceptions

```python
import pytest

def divide(a, b):
    if b == 0:
        raise ValueError('Cannot divide by Zero')
    return a / b

def test_zero_division():
    with pytest.raises(ValueError) as e:
        divide(1, 0)
    assert str(e.value) == 'Cannot divide by Zero'
```

PyTest expected exceptions output

```
$ pytest test_exceptions.py

test_exceptions.py .
```

PyTest expected exceptions (text changed)

```python
import pytest

def divide(a, b):
    if b == 0:
        raise ValueError('Cannot divide by Null')
    return a / b

def test_zero_division():
    with pytest.raises(ValueError) as e:
        divide(1, 0)
    assert str(e.value) == 'Cannot divide by Zero'
```

PyTest expected exceptions (text changed) output

```
$ pytest test_exceptions_text_changed.py


    def test_zero_division():
        with pytest.raises(ValueError) as e:
            divide(1, 0)
>       assert str(e.value) == 'Cannot divide by Zero'
E       AssertionError: assert 'Cannot divide by Null' == 'Cannot divide by Zero'
E         - Cannot divide by Null
E         ?                  ^^^^
E         + Cannot divide by Zero
E         ?                  ^^^^
```

PyTest expected exceptions (other exception)

```
import pytest

def divide(a, b):
#    if b == 0:
#        raise ValueError('Cannot divide by Zero')
    return a / b

def test_zero_division():
    with pytest.raises(ValueError) as e:
        divide(1, 0)
    assert str(e.value) == 'Cannot divide by Zero'
```

PyTest expected exceptions (other exception) output

```
    $ pytest test_exceptions_failing.py

```

```
    def test_zero_division():
        with pytest.raises(ValueError) as e:
>           divide(1, 0)

test_exceptions_failing.py:10:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

a = 1, b = 0

    def divide(a, b):
    #    if b == 0:
    #        raise ValueError('Cannot divide by Zero')
>       return a / b
E       ZeroDivisionError: division by zero
```

PyTest expected exceptions (no exception)

```python
import pytest

def divide(a, b):
    if b == 0:
        return None
    return a / b

def test_zero_division():
    with pytest.raises(ValueError) as e:
        divide(1, 0)
    assert str(e.value) == 'Cannot divide by Zero'
```

PyTest expected exceptions (no exception) output

```
    def test_zero_division():
        with pytest.raises(ValueError) as e:
>           divide(1, 0)
E           Failed: DID NOT RAISE <class 'ValueError'>
```

PyTest: Multiple Failures

```python
def test_one():
    assert True
    print('one')

def test_two():
    assert False
    print('two')

def test_three():
    assert True
    print('three')

def test_four():
    assert False
    print('four')

def test_five():
    assert True
    print('five')
```

PyTest: Multiple Failures output

```
test_failures.py .F.F.
```

```
$ pytest -v test_failures.py

test_failures.py::test_one PASSED
test_failures.py::test_two FAILED
test_failures.py::test_three PASSED
test_failures.py::test_four FAILED
test_failures.py::test_five PASSED
```

```
$ pytest -s test_failures.py

one
three
five
```

PyTest Selective running of test functions

```
pytest test_mymod_2.py::test_anagram

pytest test_mymod_2.py::test_multiword_anagram
```

PyTest: stop on first failure

```
pytest -x
pytest --maxfail 42
```

Pytest: expect a test to fail (xfail or TODO tests)

Use the `@pytest.mark.xfail` decorator to mark the test.

```python
from mymod_1 import is_anagram
import pytest

def test_anagram():
   assert is_anagram("abc", "acb")
   assert is_anagram("silent", "listen")
   assert not is_anagram("one", "two")

@pytest.mark.xfail(reason = "Bug #42")
def test_multiword_anagram():
   assert is_anagram("ana gram", "naga ram")
   assert is_anagram("anagram", "nag a ram")
```

Pytest: expect a test to fail (xfail or TODO tests)

```
$ pytest test_mymod_3.py
```

```
======= test session starts =======
platform darwin -- Python 3.5.2, pytest-3.0.7, py-1.4.33,
pluggy-0.4.0
Using --random-order-bucket=module
Using --random-order-seed=557111
```

```

rootdir:
/Users/gabor/work/training/python/examples/pytest, inifile:
plugins: xdist-1.16.0, random-order-0.5.4
collected 2 items

test_mymod_3.py .x

===== 1 passed, 1 xfailed in 0.08 seconds =====
```

PyTest: show xfailed tests with -rx

```
$ pytest -rx test_mymod_3.py
```

```
======= test session starts =======
platform darwin -- Python 3.5.2, pytest-3.0.7, py-1.4.33,
pluggy-0.4.0
Using --random-order-bucket=module
Using --random-order-seed=557111

rootdir:
/Users/gabor/work/training/python/examples/pytest, inifile:
plugins: xdist-1.16.0, random-order-0.5.4
collected 2 items

test_mymod_3.py .x

===== short test summary info =====
XFAIL test_mymod_3.py::test_multiword_anagram
  Bug #42

===== 1 passed, 1 xfailed in 0.08 seconds =====
```

Pytest: skipping tests

```python
import sys
import pytest

@pytest.mark.skipif(sys.platform != 'darwin', reason="Mac
tests")
```

```python
def test_mac():
    assert True

@pytest.mark.skipif(sys.platform != 'linux', reason="Linux tests")
def test_linux():
    assert True

@pytest.mark.skipif(sys.platform != 'win32', reason="Windows tests")
def test_windows():
    assert True

@pytest.mark.skip(reason="To show we can skip tests without any condition.")
def test_any():
    assert True
```

```
pytest test_on_condition.py
```

```
collected 4 items

test_on_condition.py ss.s

==== 1 passed, 3 skipped in 0.02 seconds ====
```

Pytest: show skipped tests woth -rs

```
$ pytest -rs test_on_condition.py
```

```
collected 4 items

test_on_condition.py s.ss

===== short test summary info =====
SKIP [1] test_on_condition.py:15: To show we can skip tests without any condition.
SKIP [1] test_on_condition.py:7: Linux tests
SKIP [1] test_on_condition.py:11: Windows tests
```

```
==== 1 passed, 3 skipped in 0.03 seconds ====
```

Pytest: show extra test summmary info with -r

- (f)ailed
- (E)error
- (s)skipped
- (x)failed
- (X)passed
- (p)passed
- (P)passed with output
- (a)all except pP

```
pytest -h
```

Pytest: skipping tests output in verbose mode

```
$ pytest -v test_on_condition.py

test_on_condition.py::test_mac PASSED
test_on_condition.py::test_any SKIPPED
test_on_condition.py::test_windows SKIPPED
test_on_condition.py::test_linux SKIPPED

==== 1 passed, 3 skipped in 0.01 seconds ======
```

Pytest verbose mode

```
$ pytest -v test_mymod_1.py

test_mymod_1.py::test_anagram PASSED
```

```
$ pytest -v test_mymod_2.py

test_mymod_2.py::test_anagram PASSED
test_mymod_2.py::test_multiword_anagram FAILED
```

Pytest quiet mode

```
$ pytest -q test_mymod_1.py
.
1 passed in 0.01 seconds
```

```
$ pytest -q test_mymod_2.py

.F
=========================== FAILURES
===========================
____________________ test_multiword_anagram
____________________

    def test_multiword_anagram():
        assert is_anagram("ana gram", "naga ram")
>       assert is_anagram("anagram", "nag a ram")
E       AssertionError: assert False
E        +  where False = is_anagram('anagram', 'nag a
ram')

test_mymod_2.py:10: AssertionError
1 failed, 1 passed in 0.09 seconds
```

PyTest print STDOUT and STDERR using -s

```python
import sys

def test_hello():
    print("hello testing")
    print("stderr during testing", file=sys.stderr)
    assert True
```

```
$ pytest -s -q test_stdout_stderr.py
hello testing
stderr during testing
.
1 passed in 0.01 seconds
```

PyTest failure reports

- Reporting success is boring
- Reporting failure can be interesting: assert + introspection

PyTest compare numbers

```
def double(n):
    #return 2*n
    return 2+n

def test_string_equal():
    assert double(2) == 4
    assert double(21) == 42
```

```
    $ pytest test_number_equal.py

    def test_string_equal():
        assert double(2) == 4
>       assert double(21) == 42
E       assert 23 == 42
E        +  where 23 = double(21)
```

PyTest compare numbers relatively

```
def get_number():
    return 23

def test_string_equal():
    assert get_number() < 0
```

```
$ pytest test_number_less_than.py
```

```
    def test_string_equal():
>       assert get_number() < 0
E       assert 23 < 0
E        +  where 23 = get_number()
```

PyTest compare strings

```
def get_string():
    return "abc"

def test_string_equal():
    assert get_string() == "abd"
```

```
$ pytest test_string_equal.py
```

```
    def test_string_equal():
>       assert get_string() == "abd"
E       AssertionError: assert 'abc' == 'abd'
E         - abc
E         + abd
```

PyTest compare long strings

```
import string

def get_string(s):
    return string.printable + s + string.printable

def test_long_strings():
    assert get_string('a') == get_string('b')
```

```
$ pytest test_long_strings.py
```

```
    def test_long_strings():
>       assert get_string('a') == get_string('b')
E       AssertionError: assert
'0123456789ab...t\n\r\x0b\x0c' == '0123456789abc...t\\
n\r\x0b\x0c'
E         Skipping 90 identical leading characters in
diff, use -v to show
E         Skipping 91 identical trailing characters in
diff, use -v to show
E           {|}~
E
E         - a012345678
E         ? ^
E         + b012345678
E         ? ^
```

PyTest is one string in another strings

Shows ~250 characters

```
import string

def get_string():
    return string.printable * 30

def test_long_strings():
    assert 'hello' in get_string()
```

```
    def test_long_strings():
>       assert 'hello' in get_string()
E       assert 'hello' in
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU\
VWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~
\t\n\r\x0b\x0c012345...x0b\x0c0123456789abcd\

efghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*
+,-./:;<=>?@[\\]^_`{|}~ \\
t\n\r\x0b\x0c'
E        +  where
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUV
WXYZ!"#\
$%&\'()*+,-./:;<=>?@[\\]^_`{|}~
```

```
\t\n\r\x0b\x0c012345...x0b\x0c0123456789abcdefghijkl\
mnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;
<=>?@[\\]^_`{|}~ \t\n\r\x0\
b\x0c' = get_string()
```

PyTest test any expression

```
def test_expression_equal():
    a = 3
    assert a % 2 == 0
```

```
$ pytest test_expression_equal.py

    def test_expression_equal():
        a = 3
>       assert a % 2 == 0
E       assert (3 % 2) == 0
```

PyTest element in list

```
def get_list():
    return ["monkey", "cat"]

def test_in_list():
    assert "dog" in get_list()
```

```
$ pytest test_in_list.py

    def test_in_list():
>       assert "dog" in get_list()
E       AssertionError: assert 'dog' in ['monkey', 'cat']
E        +  where ['monkey', 'cat'] = get_list()
```

PyTest compare lists

```
import string
import re
```

```

def get_list(s):
    return list(string.printable + s + string.printable)

def test_long_lists():
    assert get_list('a') == get_list('b')
```

```
$ pytest test_lists.py

    def test_long_lists():
>       assert get_list('a') == get_list('b')
E       AssertionError: assert ['0', '1', '2...'4', '5', ...]
              == ['0', '1', '2'...'4', '5', ...]
E         At index 100 diff: 'a' != 'b'
E         Use -v to get the full diff
```

PyTest compare short lists

```
import string
import re

def get_lista():
    return 'a', 'b', 'c'
def get_listx():
    return 'x', 'b', 'y'

def test_short_lists():
    assert get_lista() == get_listx()
```

```
$ pytest test_short_lists.py
```

```
    def test_short_lists():
>       assert get_lista() == get_listx()
E       AssertionError: assert ('a', 'b', 'c') == ('x', 'b', 'y')
E         At index 0 diff: 'a' != 'x'
E         Use -v to get the full diff
```

PyTest compare short lists - verbose output

```
$ pytest -v test_short_lists.py
```

```
    def test_short_lists():
>       assert get_lista() == get_listx()
E       AssertionError: assert ('a', 'b', 'c') == ('x', 'b', 'y')
E         At index 0 diff: 'a' != 'x'
E         Full diff:
E         - ('a', 'b', 'c')
E         ?   ^         ^
E         + ('x', 'b', 'y')
E         ?   ^         ^
```

PyTest compare dictionaries

```
import string
import re

def get_dictionary(k, v):
    d = dict([x, ord(x)] for x in  string.printable)
    d[k] = v
    return d

def test_big_dictionary_different_value():
    assert get_dictionary('a', 'def') == get_dictionary('a', 'abc')

def test_big_dictionary_differnt_keys():
    assert get_dictionary('abc', 1) == get_dictionary('def', 2)
```

PyTest compare dictionaries output

```
$ pytest test_dictionaries.py

______________ test_big_dictionary_different_value _______________

```

```
>   def test_big_dictionary_different_value():
>       assert get_dictionary('a', 'def') == get_dictionary('a', 'abc')
E       AssertionError: assert {'\t': 9, '\n...x0c': 12, ...}
          == {'\t': 9, '\n'...x0c': 12, ...}
E         Omitting 99 identical items, use -v to show
E         Differing items:
E         {'a': 'def'} != {'a': 'abc'}
E         Use -v to get the full diff

_______________ test_big_dictionary_differnt_keys _______________

    def test_big_dictionary_differnt_keys():
>       assert get_dictionary('abc', 1) == get_dictionary('def', 2)
E       AssertionError: assert {'\t': 9, '\n...x0c': 12, ...}
           == {'\t': 9, '\n'...x0c': 12, ...}
E         Omitting 100 identical items, use -v to show
E         Left contains more items:
E         {'abc': 1}
E         Right contains more items:
E         {'def': 2}
E         Use -v to get the full diff
```

PyTest Fixtures

- In generally we call test fixture the environment in which a test is expected to run.
- Pytest uses the same word for a more generic concept. All the techniques that make it easy to set up the environment and to tear it down after the tests.

PyTest Fixture setup and teardown

```
def setup_module():
    print("setup_module")

```

```
def teardown_module():
    print("teardown_module")


def setup_function():
    print("  setup_function")

def teardown_function():
    print("  teardown_function")


def test_one():
    print("    test_one")
    assert True
    print("    test_one after")

def test_two():
    print("    test_two")
    assert False
    print("    test_two after")

def test_three():
    print("    test_three")
    assert True
    print("    test_three after")
```

See next slide for the output.

PyTest Fixture setup and teardown output

```
test_fixture.py .F.
```

```
$ pytest test_fixture.py -s

setup_module

  setup_function
    test_one
    test_one after
  teardown_function

```

```
  setup_function
    test_two
  teardown_function

  setup_function
    test_three
    test_three after
  teardown_function

teardown_module
```

Note, the teardown_function is executed even after failed tests.

PyTest: Class setup and teardown

```python
class TestClass():
    def setup_class(self):
        print("setup_class called once for the class")

    def teardown_class(self):
        print("teardown_class called once for the class")


    def setup_method(self):
        print("setup_method called for every method")

    def teardown_method(self):
        print("teardown_method called for every method")


    def test_one(self):
        print("one")
        assert True
        print("one after")

    def test_two(self):
        print("two")
        assert False
        print("two after")

    def test_three(self):
        print("three")
```

```
        assert True
        print("three after")
```

PyTest: Class setup and teardown output

```
$ pytest -s test_class.py

setup_class called once for the class

setup_method called for every method
one
one after
teardown_method called for every method

setup_method called for every method
two
teardown_method called for every method

setup_method called for every method
three
three after
teardown_method called for every method

teardown_class called once for the class
```

Pytest Dependency injection

```
def function(thingy):
    pass
```

1. Find function.
2. Check parameters of the function.
3. Create the appropriate instances.
4. Call the function with the intsances.

Pytest fixture - tmpdir

```python
import os


def test_something(tmpdir):
    print(tmpdir)      # /private/var/folders/ry/z60xxmw0000gn/T/pytest-of-gabor/pyt\
est-14/test_read0

    d = tmpdir.mkdir("subdir")
    fh = d.join("config.ini")
    fh.write("Some text")

    filename = os.path.join( fh.dirname, fh.basename )

    temp_dir = str(tmpdir)

    # ...
```

Pytest capture STDOUT and STDERR with capsys

Captures everything that is printed to STDOUT and STDERR so we can compare that to the expected output and error.

```python
import sys

def greet(to_out, to_err=None):
    print(to_out)
    if to_err:
        print(to_err, file=sys.stderr)


def test_myoutput(capsys):
    greet("hello", "world")
    out, err = capsys.readouterr()
    assert out == "hello\n"
    assert err == "world\n"

    greet("next")
```

```python
    out, err = capsys.readouterr()
    assert out == "next\n"
```

Pytest Fixture - home made fixtures

```python
import pytest
import application


@pytest.fixture()
def getapp():
    print('getapp starts')
    app = application.App()

    yield app

    app.shutdown()
    print('getapp ends')

def test_add_user_foo(getapp):
    getapp.add_user("Foo")
    assert True

def test_add_user_bar(getapp):
    getapp.add_user("Bar")
    assert True
```

```python
class App:
    def __init__(self):
        self.pi = 3.14
        # .. set up database
        print("__init__ of App")


    def shutdown(self):
        print("shutdown of App cleaning up database")


    def add_user(self, name):
        print("Working on add_user({})".format(name))
```

```
$ pytest -s -q fixtures.py

getapp starts
__init__ of App
Working on add_user(Bar)
.shutdown of App cleaning up database
getapp ends

getapp starts
__init__ of App
Working on add_user(Foo)
.shutdown of App cleaning up database
getapp ends
```

More fixtures

```python
import pytest

@pytest.fixture(autouse = True, scope="module")
def fix_module():
    print("\nFix module setup")
    yield
    print("\nFix module teardown")


@pytest.fixture(autouse = True, scope="function")
def fix_function():
    print("\nFix function setup")
    yield
    print("\nFix function teardown")


@pytest.fixture()
def blue():
    print("\nFix blue setup")
    yield
    print("\nFix blue teardown")

@pytest.fixture()
def green():
    print("\nFix green setup")
    yield
    print("\nFix green teardown")
```

```python


def test_one(blue, green):
    print("Test one")


def test_two(green, blue):
    print("Test two")
```

```
========= test session starts =========
platform linux -- Python 3.7.3, pytest-5.1.1, py-1.8.0, pluggy-0.13.0 -- /home/gabor\
/venv3/bin/python3
cachedir: .pytest_cache
rootdir: /home/gabor/work/slides/python/examples/pytest
plugins: flake8-1.0.4
collecting ... collected 2 items

more_fixtures.py::test_one
Fix module setup

Fix function setup

Fix blue setup

Fix green setup
Test one
PASSED
Fix green teardown

Fix blue teardown

Fix function teardown

more_fixtures.py::test_two
Fix function setup

Fix green setup

Fix blue setup
Test two
PASSED
Fix blue teardown

```

```
Fix green teardown

Fix function teardown

Fix module teardown


======= 2 passed in 0.01s ====
```

- We can’t add fixtures to test_functions as decorators (as I think was the case in NoseTest), we need to use dependency injection.

Pytest: Mocking - why?

- Independent testing environment.
- Faster tests (mock remote calls, mock whole database)
- Fake some code/application/API that does not exist yet.
- Test error conditions in a system not under our control.

Pytest: Mocking - what?

- External dependency (e.g. an API)
- STDIN/STDOUT/STDERR
- Random values
- Methods of a database

Pytest: One dimensional spacefight

```python
import random

def play():
    debug = False
    move = False
    while True:
```

```
    print("\nWelcome to another Number Guessing game")
    hidden = random.randrange(1, 201)
    while True:
        if debug:
            print("Debug: ", hidden)

        if move:
            mv = random.randrange(-2, 3)
            hidden = hidden + mv

        user_input = input("Please enter your guess [x|s|d|m|n]: ")
        print(user_input)

        if user_input == 'x':
            print("Sad to see you leave early")
            return

        if user_input == 's':
            print("The hidden value is ", hidden)
            continue

        if user_input == 'd':
            debug = not debug
            continue

        if user_input == 'm':
            move = not move
            continue

        if user_input == 'n':
            print("Giving up, eh?")
            break

        guess = int(user_input)
        if guess == hidden:
            print("Hit!")
            break

        if guess < hidden:
            print("Your guess is too low")
        else:
            print("Your guess is too high")

```

```python

if __name__ == '__main__':
    play()
```

Pytest: Mocking input and output

```python
import game

def test_immediate_exit():
    input_values = ['x']
    output = []

    def mock_input(s):
       output.append(s)
       return input_values.pop(0)
    game.input = mock_input
    game.print = lambda s : output.append(s)

    game.play()

    assert output == [
        '\nWelcome to another Number Guessing game',
        'Please enter your guess [x|s|d|m|n]: ',
        'x',
        'Sad to see you leave early',
    ]
```

Pytest: Mocking random

```python
import game
import random

def test_immediate_exit():
    input_values = ['30', '50', '42', 'x']
    output = []

    def mock_input(s):
       output.append(s)
       return input_values.pop(0)
    game.input = mock_input
    game.print = lambda s : output.append(s)
```

```python
    random.randrange = lambda a, b : 42

    game.play()

    assert output == [
        '\nWelcome to another Number Guessing game',
        'Please enter your guess [x|s|d|m|n]: ',
        '30',
        'Your guess is too low',
        'Please enter your guess [x|s|d|m|n]: ',
        '50',
        'Your guess is too high',
        'Please enter your guess [x|s|d|m|n]: ',
        '42',
        'Hit!',
        '\nWelcome to another Number Guessing game',
        'Please enter your guess [x|s|d|m|n]: ',
        'x',
        'Sad to see you leave early',
    ]
```

Pytest: Flask echo

```python
from flask import Flask, request
eapp = Flask(__name__)

@eapp.route("/")
def hello():
    return '''
<form action="/echo" method="GET">
<input name="text">
<input type="submit" value="Echo">
</form>
'''

@eapp.route("/echo")
def echo():
    answer = request.args.get('text')
    if answer:
        return "You said: " + answer
    else:
        return "Nothing to say?"

```

```python

if __name__ == "__main__":
    eapp.run()
```

Pytest: testing Flask echo

```python
import flask_echo

class TestEcho:
    def setup_method(self):
        self.app = flask_echo.eapp.test_client()
        print("setup")

    def test_main(self):
        rv = self.app.get('/')
        assert rv.status == '200 OK'
        assert b'<form action="/echo" method="GET">' in rv.data

    def test_echo(self):
        rv = self.app.get('/echo?text=Hello')
        assert rv.status == '200 OK'
        assert b'You said: Hello' in rv.data

    def test_empty_echo(self):
        rv = self.app.get('/echo')
        assert rv.status == '200 OK'
        assert b'Nothing to say?' in rv.data
```

PyTest: Run tests in parallel with xdist

```
$ pip install pytest-xdist
$ pytest -n NUM
```

PyTest: Order of tests

Pytest runs the test in the same order as they are found in the test module:

```
def test_one():
    assert True

def test_two():
    assert True

def test_three():
    assert True
```

```
test_order.py::test_one PASSED
test_order.py::test_two PASSED
test_order.py::test_three PASSED
```

PyTest: Randomize Order of tests

Install [pytest-random-order]

```
pip install pytest-random-order
```

And from now on all the test will run in a random order.

PyTest: Force default order

If for some reason we would like to make sure the order remains the same,
we can add the following two lines of code.

```
import pytest
pytestmark = pytest.mark.random_order(disabled=True)
```

```
import pytest
pytestmark = pytest.mark.random_order(disabled=True)

def test_one():
    assert True

def test_two():
```

```
    assert True

def test_three():
    assert True
```

PyTest: no random order

```
pytest -p no:random-order -v
```

Anagram on the command line

```
from mymod_1 import is_anagram
import sys

if len(sys.argv) != 3:
    exit("Usage {} STR STR".format(sys.argv[0]))

print(is_anagram(sys.argv[1], sys.argv[2]))
```

PyTest testing CLI

```
import subprocess

def capture(command):
    proc = subprocess.Popen(command,
        stdout = subprocess.PIPE,
        stderr = subprocess.PIPE,
    )
    out,err = proc.communicate()
    return out, err, proc.returncode


def test_anagram_no_param():
    command = ["python3", "examples/pytest/anagram.py"]
    out, err, exitcode = capture(command)
    assert exitcode == 1
    assert out == b''
    assert err == b'Usage examples/pytest/anagram.py STR STR\n'

```

```
def test_anagram():
    command = ["python3", "examples/pytest/anagram.py", "abc", "cba"]
    out, err, exitcode = capture(command)
    assert exitcode == 0
    assert out == b'True\n'
    assert err == b''

def test_no_anagram():
    command = ["python3", "examples/pytest/anagram.py", "abc", "def"]
    out, err, exitcode = capture(command)
    assert exitcode == 0
    assert out == b'False\n'
    assert err == b''
```

PyTest test discovery

Running py.test will find test files and in the files test functions.

- test_*.py files
- *_test.py files
- test_* functions
- …

```
$ pytest
============================ test session starts =============================
platform darwin -- Python 2.7.5 -- py-1.4.20 -- pytest-2.5.2
collected 3 items

test_fibo.py F
test_fibonacci.py F
test_fibonacci_ok.py .

================================== FAILURES ==================================
_________________________________ test_fibo __________________________________

```

```
    def test_fibo():
        assert mymath.fibo(1) == [1]
        assert mymath.fibo(2) == [1, 1]
>       assert mymath.fibo(3) == [1, 1, 2]
E       assert [1, 1, 5] == [1, 1, 2]
E         At index 2 diff: 5 != 2

test_fibo.py:6: AssertionError
______________________________ test_fibonacci ______________________________

    def test_fibonacci():
        assert mymath.fibonacci(1) == 1
        assert mymath.fibonacci(2) == 1
>       assert mymath.fibonacci(3) == 2
E       assert 5 == 2
E        +  where 5 = <function fibonacci at 0x107f90488>(3)
E        +    where <function fibonacci at 0x107f90488> = mymath.fibonacci

test_fibonacci.py:6: AssertionError
==================== 2 failed, 1 passed in 0.04 seconds ====================
```

PyTest test discovery - ignore some tests

```
$ pytest


$ pytest --ignore venv3/
```

```
test_mymod_1.py .
test_mymod_2.py .F
```

- test_*.py files
- *_test.py files
- TestClasses
- test_* functions

- …

PyTest select tests by name

- –collect-only - only list the tests, don't run them yet.
- -k select by name

```python
def test_database_read():
    assert True

def test_database_write():
    assert True

def test_database_forget():
    assert True

def test_ui_access():
    assert True

def test_ui_forget():
    assert True
```

```
pytest --collect-only -k database test_by_name.py
    test_database_forget
    test_database_read
    test_database_write
```

```
pytest --collect-only -k ui test_by_name.py
    test_ui_access
    test_ui_forget
```

```
pytest --collect-only -k forget test_by_name.py
    test_database_forget
    test_ui_forget
```

```
pytest --collect-only -k "forget or read" test_by_name.py
    test_database_read
```

```
    test_database_forget
    test_ui_forget
```

PyTest select tests by marker

Use the @pytest.mark.name decorator to tag the tests.

```python
import pytest

@pytest.mark.smoke
def test_database_read():
    assert True

@pytest.mark.security
@pytest.mark.smoke
def test_database_write():
    assert True

@pytest.mark.security
def test_database_forget():
    assert True

@pytest.mark.smoke
def test_ui_access():
    assert True

@pytest.mark.security
def test_ui_forget():
    assert True
```

```
pytest --collect-only -m security test_by_marker.py
    test_ui_forget
    test_database_write
    test_database_forget
```

```
pytest --collect-only -m smoke test_by_marker.py
    test_database_read
    test_ui_access
    test_database_write
```

PyTest: Test Coverage

```
pip install pytest-cov

pytest --cov=my --cov-report html --cov-branch

Open htmlcov/index.html
```

Try werkzeug

```
pytest --cov=werkzeug --cov-report html --cov-branch
xdg-open htmlcov/index.html
```

Exercise: module

Pick one of the modules and write a test for it.

- algo
- editdistance Levenshtein distance implemented in C
- python-Levenshtein implemented in C
- pylev
- pyxdameraulevenshtein
- weighted-levenshtein
- OpenPyXL

Exercise: Open Source

- Visit the stats on PyDigger.com
- List the packages that have GitHub no Travis-CI.
- Pick one that sounds simple. Visit its GitHub page and check if it has tests.
- If it does not, wirte one.
- Send Pull Request

Pytest resources

- pytest.org
- Python Testing with pytest by Brian Okken (The Pragmatic Bookshelf)
- Python Testing by Brian Okken
- Talk Python to me by Michael Kennedy
- Python Bytes podcast by Brian Okken and Michael Kennedy

Pytest and tempdir

```python
import re

def parse_file(filename):
    data = {}
    with open(filename) as fh:
        for row in fh:
            row = row.rstrip("\n")
            if re.search(r'=', row):
                k, v = re.split(r'\s*=\s*', row)
                data[k] = v
            else:
                pass # error reporting?
    return data

def save_file(filename, data):
    with open(filename, 'w') as fh:
        for k in data:
            fh.write("{}={}\n".format(k, data[k]))

if __name__ == '__main__':
    print(parse_file('a.cfg'))
```

```
name=Foo Bar
email   =    foo@bar.com
```

```python
import mycfg
import os
```

```

class TestMe:
    def test_parse(self):
        data = mycfg.parse_file('a.cfg')
        assert data, {
            'name'  : 'Foo Bar',
            'email' : 'foo@bar.com',
        }

    def test_example(self, tmpdir):
        original = {
            'name'  : 'My Name',
            'email' : 'me@home.com',
            'home'  : '127.0.0.1',
        }
        filename = str(tmpdir.join('abc.cfg'))
        assert not os.path.exists(filename)
        mycfg.save_file(filename, original)
        assert os.path.exists(filename)
        new = mycfg.parse_file(filename)
        assert new == original
```

PyTest compare short lists - output

```
import configparser
import os


def test_read_ini(tmpdir):
    print(tmpdir)      #
/private/var/folders/ry/z60xxmw0000gn/T/pytest-of-
gabor/pyt\
est-14/test_read0
    d = tmpdir.mkdir("subdir")
    fh = d.join("config.ini")
    fh.write("""
[application]
user  =  foo
password = secret
""")

    print(fh.basename) # data.txt
    print(fh.dirname)  #
```

```
/private/var/folders/ry/z60xxmw0000gn/T/pytest-of-gabor/pyt\
est-14/test_read0/subdir
    filename = os.path.join( fh.dirname, fh.basename )

    config = configparser.ConfigParser()
    config.read(filename)

    assert config.sections() == ['application']
    assert config['application'], {
       "user" : "foo",
       "password" : "secret"
    }
```

PyTest with parameter

```
import pytest

@pytest.mark.parametrize("name", ["Foo", "Bar"])
def test_cases(name):
    print(f"name={name}")
    assert len(name) == 3
```

```
======== test session starts ========
platform linux -- Python 3.7.3, pytest-5.3.2, py-1.8.0, pluggy-0.13.0
rootdir: /home/gabor/work/slides/python-programming/examples/pytest
plugins: flake8-1.0.4
collected 2 items

test_with_param.py name=Foo
.name=Bar
.

=========== 2 passed in 0.00s ========
```

PyTest with parameters

```python
import pytest

@pytest.mark.parametrize("name,email", [
    ("Foo", "foo@email.com"),
    ("Bar", "bar@email.com"),
])
def test_cases(name, email):
    print(f"name={name}  email={email}")
    assert email.lower().startswith(name.lower())
```

```
========= test session starts =======
platform linux -- Python 3.7.3, pytest-5.3.2, py-1.8.0, pluggy-0.13.0
rootdir: /home/gabor/work/slides/python-programming/examples/pytest
plugins: flake8-1.0.4
collected 2 items

test_with_params.py name=Foo  email=foo@email.com
.name=Bar  email=bar@email.com
.

========== 2 passed in 0.01s =======
```

Pytest reporting in JUnit XML format

```
pytest --junitxml report.xml
```

- [pytest-json-report]()

```
pip install pytest-json-report

pytest --json-report --json-report-file=report.json
```

Recommended to also add

```
--json-report-omit=log
```

```
pytest -s --json-report --json-report-file=report.json --log-cli-level=INFO
```

```python
import logging

def add(x, y):
#     logger = logging.getLogger("mytest")
    logging.basicConfig(level = logging.INFO)
    logging.info("Just some info log")
    return x * y

def test_one():
    assert add(2, 2) == 4
```

No test selected

If you run pytest and it cannot find any tests, for example because you used some
selector and not test matched it, then Pytest will exit with exit code 5.

This is considered a failure by every tool, including Jenkins and other CI systems.

On the other hand you won't see any failed test reported. After all if no tests are run, then none of them fails.
This can be confusing.

Advancted functions

Variable scopes

- Local (inside a def)
- Enclosing (in the enclosing def, aka. nonlocal)
- Global (outside of all defs)

Name resolution order (LEGB)

1. Local
2. Enclosing
3. Global
4. Built-in

Scoping: global seen from fuction

```
a = 42
def f():
    print(a)

f()
```

42

Assignment creates local scope

```
a = 42
def f():
    a = 23
    print(a)
```

```

print('ok')
print(a)
f()
print(a)
```

```
ok
42
23
42
```

Local scope gone wrong

```
a = 42
def f():
    print(a)
    a = 23

print('ok')
print(a)
f()
print(a)
```

```
ok
42
Traceback (most recent call last):
  File "scoping_external_variable.py", line 8, in <module>
    f()
  File "scoping_external_variable.py", line 3, in f
    print(a)
UnboundLocalError: local variable 'a' referenced before assignment
```

Accessing a global variable inside a function works, but if I change it (make it refer to another piece of data),
then it is disallowed. If I only change the data inside (for mutable variables), that works, but is a bad practice.

Changing global variable from a function

```
a = 42
def f():
    global a
    print(a)
    a = 23

print(a)    # 42
f()         # 42
print(a)    # 23
```

Does not need to be created outside

```
def f():
    global a
    a = 23

f()
print(a)    # 23
```

Global variables mutable in functions

```
a = [2]

def f():
    print(a)        # [2]
    a.append(3)
    print(a)        # [2, 3]
    a[0] = 4

f()
print(a)            # [4, 3]
```

Scoping issues

```
text = ['aaaa', 'bb', 'ccc ccc']

length_1 = [ len(s) for s in text ]
```

```
print(length_1) # [4, 2, 7]


length_2 = [ len(s) for x in text ]
print(length_2)  # [7, 7, 7]
```

List comprehensions don’t create their own scope!

sub in sub

Functions can be defined inside functions.

```
def f():
    print("in f")
    def g():
       print("in g")
    g()

f()
#g()  # does not exist here
```

They are scoped locally

Scoping sub in sub (enclosing scope)

```
def external_func():
    the_answer = 42

    def func(args):
        print(args, "the_answer:", the_answer)

        # the_answer = 'what was the question?'
        # enabling this would give:
        # UnboundLocalError: local variable 'the_answer'
        #       referenced before assignment

    func("first")
    func("second")
```

```

external_func()
```

```
first the_answer: 42
second the_answer: 42
```

Function objects

```
The difference between
x = foo
y = foo()
```

```
c = 0

def foo():
    global c
    c += 1
    return c


print(foo())    # 1
print(foo())    # 2
x = foo            # assigning the function object
y = foo()          # assigning the return value of the
function
print(foo())    # 4
print(x())      # 5
print(y)        # 3
```

Functions are created at run time

def and class are run-time
Everything is runtime. Even compilation is runtime.

foo() will return a random value every time, but when
bar is defined it freezes the specific value that foo

returned when bar was created.

```
import random

def foo():
    return random.random()


print(foo())
print(foo())

def bar(a, b = foo()):
   return [a, b]

print(bar(1))
print(bar(2))
```

```
0.0756804810689
0.350692064113
[1, 0.7401995987184571]
[2, 0.7401995987184571]
```

Mutable default

The default list assigned to b is created when the f functions is defined.
After that, each call to f() (that does not get a “b” parameter) uses this
common list.

```
def f(a, b = []):
    b.append(a)
    return b

print(f(1))
print(f(2))
print(f(3))
```

```
[1]
[1, 2]
[1, 2, 3]
```

Use None instead:

Use None as default parameter

```
def f(a, b = None):
    if b == None:
        b = []
    b.append(a)
    return b

print(f(1))
print(f(2))
print(f(3))
```

```
[1]
[2]
[3]
```

Inner function created every time the outer function runs

> Also defined during run-time, but in every call of bar() the innter_func is redefined
> again and again.

```
import random

def foo():
    return random.random()

```

```
print(foo())
print(foo())

def bar(a, b = foo()):

    def inner_func(x, y = foo()):
        return [x, y]

    print('inner', inner_func(a))
    return [a, b]

print(bar(1))
print(bar(2))
```

```
0.821210904648
0.925337844251
inner [1, 0.9243163421154859]
[1, 0.38535850141949013]
inner [2, 0.5665772632462458]
[2, 0.38535850141949013]
```

Static variable

There are no function-level static variables in Python, but you can fake it quite easily

```
def counter():
    if 'cnt' not in counter.__dict__:
        counter.cnt = 0
    counter.cnt += 1
    return counter.cnt

print(counter())      # 1
print(counter())      # 2
print(counter())      # 3

print(counter.cnt)    # 3

counter.cnt = 6
print(counter())      # 7
```

Static variable in generated function

```python
def create():
    def func():
        func.cnt += 1
        return func.cnt
    func.cnt = 0
    return func

a = create()
b = create()
print(a())     # 1
print(a())     # 2
print(b())     # 1
print(a())     # 3

b.cnt = 7
print(a.cnt)   # 3
print(b.cnt)   # 7
```

Inspect

The inspect module provides introspection to Python runtime.
`inspect.stack` returns the stack-trace. Element 0 is the deepes (where we called inspect stack).
Each level has several values. A represantation of the frame, filename, linenumber, subroutine-name.

```python
import inspect
import sys

level = int(sys.argv[1])


def f():
    print("in f before g")
    g()
    print("in f after g")

def g():
    print("in g")
```

```python
    PrintFrame()


def PrintFrame():
  st = inspect.stack()

  frame = st[level][0]
  info = inspect.getframeinfo(frame)
  print('__file__:     ', info.filename)
  print('__line__:     ', info.lineno)
  print('__function__: ', info.function)

  print('* file', st[level][1])
  print('* line', st[level][2])
  print('* sub',  st[level][3])

f()
```

python caller.py 1

```
in f before g
in g
__file__:      caller.py
__line__:      15
__function__:  g
* file caller.py
* line 15
* sub g
in f after g
```

Variable number of function arguments

Python function arguments - a reminder

- Order of parameter
- Arguments with default values are optional (and come at the end of the definition)
- Number of arguments is know at the time of function definition. The only flexibility is provided by the optional arguments.

```
def f(a, b = 42):
    print(a)
    print(b)

f(23)
    # 23
    # 42

f(19, 11)
    # 19
    # 11

f(b=7, a=8)
    # 8
    # 7

# f()           # (runtime) TypeError: f() takes at least 1 argument (0 given)
# f(1, 2, 3)    # (runtime) TypeError: f() takes at most 2 arguments (3 given)
# f(b=10, 23)   # SyntaxError: non-keyword arg after keyword arg

```

```
# def g(a=23, b):
#     pass
#     SyntaxError: non-default argument follows default
argument
```

Functions with unknown number of argumerns

- sum(a, b, c, …)
- reduce(function, a, b, c, …)
- report (function, foo = 23, bar = 19, moo = 70, …)
- report (function, a, b, c, …, foo = 23, bar = 19, moo = 70, …)

Variable length argument list with * and **

```
def f(a, b=1, *args, **kwargs):
    print('a:     ', a)
    print('b:     ', b)
    print('args:  ', args)
    print('kwargs:', kwargs)
    return a + b

f(2, 3, 4, 5, c=6, d=7)
print()
f(2, c=5, d=6)
print()
f(10)
```

```
a:      2
b:      3
args:   (4, 5)
kwargs: {'c': 6, 'd': 7}

a:      2
b:      1
args:   ()
kwargs: {'c': 5, 'd': 6}

```

```
a:      10
b:      1
args:   ()
kwargs: {}
```

Passing arguments as they were received (but incorrectly)

What if we need to pass the list of individual arguments (or pairs) to another function?

```
def f(*args, **kwargs):
    print('f args:   ', args)
    print('f kwargs: ', kwargs)
    g(args, kwargs)

def g(*args, **kwargs):
    print('g args:   ', args)
    print('g kwargs: ', kwargs)

f(1, 2, a=3, b=4)
```

```
f args:    (1, 2)
f kwargs:  {'a': 3, 'b': 4}
g args:    ((1, 2), {'a': 3, 'b': 4})
g kwargs:  {}
```

g() received 2 individual parameters, the first was a tuple, the second a dictionary

Unpacking args before passing them on

```
def f(*args, **kwargs):
    print('f: ', args)
    print('f: ', kwargs)
    g(*args, **kwargs)

def g(*args, **kwargs):
```

```
    print('g: ', args)
    print('g: ', kwargs)

f(1, 2, a=3, b=4)
```

```
f:  (1, 2)
f:  {'a': 3, 'b': 4}
g:  (1, 2)
g:  {'a': 3, 'b': 4}
```

Exercise: implement the my_sum function

- my_sum should be able to accept any number of values and return their sum.
- my_sum() should return 0 or None. Decide yourself!
- my_sum(2, 3) should return 5. etc.

Solution: implement the my_sum function

```
def my_sum(*numbers):
    s = 0
    for n in numbers:
        s += n
    return s

print(my_sum())           # 0
print(my_sum(2, 3))       # 5
print(my_sum(-1, 2, -1,)) # 0
```

Exercise: implement the reduce function

```
my_reduce(function, a, b, c, ...)
```

- 'function' is expected to be a function that receives two arguments and returns a result.

- If only the function is given, return None.
- If only one value is given, return that value.
- Take the first two values, run the function on them. Then take the result and the next value and run the function on them. etc. When no more values are left, return the last result.

```
# print(my_reduce()) # TypeError: my_reduce() takes at least 1 argument (0 given)
print(my_reduce(lambda x,y: x+y))             # None
print(my_reduce(lambda x,y: x+y, 3))          # 3
print(my_reduce(lambda x,y: x+y, -1, 4, -2))  # 1

print(my_reduce(lambda x,y: x*y, -1, 4, -2))  # 8
```

Soluton: implement the reduce function

```
def my_reduce(f, *args):
    if len(args) == 0:
        return None
    result = args[0]
    for i in range(1, len(args)):
        result = f(result, args[i])
    return result

# print(my_reduce()) # TypeError: my_reduce() takes at least 1 argument (0 given)
print(my_reduce(lambda x,y: x+y))             # None
print(my_reduce(lambda x,y: x+y, 3))          # 3
print(my_reduce(lambda x,y: x+y, -1, 4, -2))  # 1

print(my_reduce(lambda x,y: x*y, -1, 4, -2))  # 8
```

Exercise: sort pairs

Create a function called sort_pairs, that would receive a sorting method, e.g.
the word ‘keys’ or the word ‘values’ and will receive an arbitrary

number of key-value pairs
and will return a list of tuples.

```
sort_pairs( 'keys', foo = 23, bar = 47)
[('bar', 47), ('foo', 23)]

sort_pairs( 'values', foo = 23, bar = 47)
[('foo', 23), ('bar', 47)]
```

Solution: sort pairs

```
def sort_pairs(how, **kwargs):
    if how == 'keys':
        sort_function = lambda s : s[0];
    elif how == 'values':
        sort_function = lambda s : s[1];
    else:
        raise Exception("Invalid sort function")
    return sorted(kwargs.items(), key=sort_function)



k = sort_pairs( 'keys', foo = 23, bar = 47)
print(k)
v = sort_pairs( 'values', foo = 23, bar = 47)
print(v)
```

Python Packages

Why Create package

> As a module gets larger and larger it will be more and more difficult to maintain.
>
> It might be eaier if we split it up into multiple files and put those files inside
> a directory. A ‘package’ is just that. A bunch of Python modules that belong together
> and are placed in a directory hierarchy. In order to tell Python that you really
> mean these files to be a package one must add a file called **init**.py in
> each directory of the project. In the most simple case the file can be empty.

- Code reuse
- Separation of concerns
- Easier distribution

Create package

```
mymath/
    __init__.py
    calc.py
    ...
    internal_use.py
```

```
def add(x, y):
    return x+y
```

```
# empty
```

Internal usage

```
import calc
print(calc.add(7, 8))   # 15

from calc import add
print(add(3, 5))        # 8
```

```
cd examples/package
python 1/mymath/internal_use.py
```

use module in package - relative path

```
import sys
import os

path = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), '1')
# print(path)   # /home/gabor/work/slides/python-programming/examples/package/1
sys.path.insert(0, path)

import mymath.calc
print(mymath.calc.add(2, 5))

from mymath.calc import add
print(add(2, 3))
```

```
7
5
```

use package (does not work)

```
import sys
import os

sys.path.insert(0, os.path.join(
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
,
    '1' ) )

import mymath
print(mymath.calc.add(4, 7))
```

```
Traceback (most recent call last):
  File "use_project/proj1_2.py", line 9, in <module>
    print(mymath.calc.add(4, 7))
AttributeError: module 'mymath' has no attribute 'calc'
```

If we import the main package name, it does not have access to the module inside.

package importing (and exporting) module

Put import (and thus re-export) in **init**.py

```
def add(x, y):
    return x+y
```

```
import mymath.calc
```

use package (module) with import

Still works…

```
import sys
import os
```

```

path = os.path.join(
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
, '2\
' )
# print(path)
sys.path.insert(0, path)

import mymath.calc
print(mymath.calc.add(2, 5))   # 7

from mymath.calc import add
print(add(2, 3))               # 5
```

use package with import

Now we can import the module from the package and use that.

```
import sys
import os

sys.path.insert(0, os.path.join(

os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
,
    '2' ) )

import mymath
print(mymath.calc.add(4, 7))   # 11

from mymath import calc
print(calc.add(5, 9))          # 14
```

Creating an installable Python package

The directory layout of a package:

```
├── mymath
│   ├── calc.py
│   └── __init__.py
└── setup.py
```

```python
from setuptools import setup





setup(name='mymath',
      version='0.1',
      description='The best math library',
      url='http://github.com/szabgab/mymath',
      author='Foo Bar',
      author_email='foo@bar.com',
      license='MIT',
      packages=['mymath'],
      zip_safe=False,
      )
```

Create tar.gz file

```
$ python setup.py sdist
```

- mymath.egg-info/
- dist/mymath-0.1.tar.gz

```
running sdist
running egg_info
creating mymath.egg-info
writing mymath.egg-info/PKG-INFO
writing top-level names to mymath.egg-info/top_level.txt
writing dependency_links to mymath.egg-info/dependency_links.txt
writing manifest file 'mymath.egg-info/SOURCES.txt'
reading manifest file 'mymath.egg-info/SOURCES.txt'
writing manifest file 'mymath.egg-info/SOURCES.txt'
warning: sdist: standard file not found: should have one of README, README.txt

creating mymath-0.1
creating mymath-0.1/mymath
creating mymath-0.1/mymath.egg-info
making hard links in mymath-0.1...
```

```
hard linking setup.py -> mymath-0.1
hard linking mymath/__init__.py -> mymath-0.1/mymath
hard linking mymath.egg-info/PKG-INFO -> mymath-0.1/mymath.egg-info
hard linking mymath.egg-info/SOURCES.txt -> mymath-0.1/mymath.egg-info
hard linking mymath.egg-info/dependency_links.txt -> mymath-0.1/mymath.egg-info
hard linking mymath.egg-info/not-zip-safe -> mymath-0.1/mymath.egg-info
hard linking mymath.egg-info/top_level.txt -> mymath-0.1/mymath.egg-info
Writing mymath-0.1/setup.cfg
creating dist
Creating tar archive
removing 'mymath-0.1' (and everything under it)
```

Install Package

```
$ pip install dist/mymath-0.1.tar.gz
```

```
$ easy_install --prefix ~/python/  dist/mymath-0.1.tar.gz
```

```
$ python setup.py install --prefix ~/python/
```

Upload to PyPi or distribute to your users.

Dependencies

```
requires=[
    'lawyerup',
],
```

To list them

```
$ python setup.py --requires
```

In the setup.py file we only need to change the version number and we
can release a new version of the package.

Add README file

```
.
├── bin
│   ├── runmymath.bat
│   └── runmymath.py
├── MANIFEST.in
├── mymath
│   └── test
│       ├── __init__.py
│       ├── test_all.py
│       └── test_calc.py
├── README.rst
└── setup.py
```

```
mymath
------

Super awesome Python module to compute the sum of numbers.

To use:

    import mymath
    mymath.sum(1, 2, 3)
```

```
include README.rst
```

Add README file (setup.py)

In the setup.py add the following function:

```
def readme():
    with open('README.rst') as f:
        return f.read()
```

and in the setup() call include the following parameter:

```
      long_description=readme(),
```

This will display the README file when called at

```
$ python setup.py --long-description
```

Include executables

```
root/
  setup.py
  README.rst
  MANIFEST.in
  bin/
    runmymath.py
    runmymath.bat
  mymath/
    __init__.py
    calc.py
```

```python
import mymath

def main():
    print("running")

main()
```

```
echo "hi"
```

setup.py will need to get

```
    scripts=['bin/runmymath.py', 'bin/runmymath.bat'],
```

Add tests

```
root/
  setup.py
  README.rst
  MANIFEST.in
  bin/
    runmymath.py
    runmymath.bat
  mymath/
    __init__.py
    calc.py
    test/
      __init__.py
      test_all.py
      test_calc.py
```

```
#empty (needed for unittest discover)
```

```
python mymath/test/test_calc.py
python mymath/test/test_all.py
```

```
python -m unittest discover
```

Add tests calc

```python
from os.path import dirname,abspath
import sys

sys.path.insert(0, dirname(dirname(dirname(abspath(__file__)))))
from mymath.calc import add
import unittest

class AddTest(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2, 3), 5)
        self.assertEqual(add(2, -2), 0)
        #self.assertEqual(add(1, 1), 1)
```

```python

if __name__ == '__main__':
    unittest.main()
```

Add tests all

```python
from os.path import dirname,abspath
import sys

sys.path.insert(0,
dirname(dirname(dirname(abspath(__file__)))))
from mymath.calc import *
import unittest

class AllTest(unittest.TestCase):
    def test_sum(self):
        self.assertEqual(add(2, 3), 5)
        #self.assertEqual(sum(1, 1), 2)
        #self.assertEqual(div(6, 2), 3)

if __name__ == '__main__':
    unittest.main()
```

setup.py

```python
from setuptools import setup

def readme():
    with open('README.rst') as f:
        return f.read()

setup(name='mymath',
      version='0.2',
      description='The best math library',
      url='http://github.com/szabgab/mymath',
      author='Foo Bar',
      author_email='foo@bar.com',
      license='MIT',
      packages=['mymath'],
      zip_safe=False,
      requires=[
```

```
            'lawyerup',
        ],
        long_description=readme(),
        scripts=['bin/runmymath.py', 'bin/runmymath.bat'],
        )
```

Run tests and create package

```
python setup.py test
python setup.py sdist
```

Packaging applications (creating executable binaries)

- py2exe on Windows (discontinued)
- Freeze on Linux
- py2app on Mac
- cx_Freeze cross-platform
- PyInstaller cross-platform
- Auto Py To Exe

Using PyInstaller

```
print("hello world")
```

```
pip install pyinstaller
pyinstaller myscript.py
pyinstaller --onefile hello_world.py
```

- See the results in dist/

Other PyInstaller examples

Use this to see where does the packaged version of our code look for modules:

```python
import sys

print(sys.path)
```

Use this to see how to pass command line parameters to the packaged exe:

```python
import sys

print(sys.argv)
```

Other

```
pyinstaller --onefile --windowed myscript.py
```

Py2app for Mac

```
pip install py2app
py2applet examples/basics/hello.py
```

Exercise: package

- Go to Pypi, find some interesting module and install it in a non-standard location (or in a virtualenv)
- Check if it was installed (try to import it in a python script).
- Take one of the previously created modules, and create a package for it.
- Install this new package in a non-standard location.
- Check if it works from some other place in your file-system.

- Take the mymath package, add another method, add tests and create the distubtable zip file.

Exercise: create executable

- Go over some of the examples in the course and package that.
- Package a script using some of your favorite modules.

Ctypes

ctypes - hello

```c
#include <stdio.h>

char * echo(char * what)
{
    return what;
}

int add_int(int a, int b)
{
    int sum = a+b;
    return sum;
}

int add_int(int a, int b)
{
    int sum = a+b;
    return sum;
}


int main(void)
{
    printf("hello\n");
    printf("%d\n", add_int(2, 3));
    printf("%s\n", echo("Foo"));
    return 0;
}
```

```
gcc -o hello hello.c
gcc -o hello.so -shared -fPIC hello.c
```

```python
from ctypes import cdll
from ctypes import c_char_p
```

```python

hello_lib = cdll.LoadLibrary("hello.so")

print(hello_lib.add_int(4, 5))          # 9

print(hello_lib.echo('Hello World'))    # 153977204


hello_lib.echo.restype = c_char_p
print(hello_lib.echo('Hello World'))    # Hello World
```

concat

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int len(char * s)
{
   return strlen(s);
}

char * concat(char * a, char * b)
{
   char * res;
   int leng = strlen(a) + strlen(b);
   res = (char *)malloc(leng);
   strcpy (res, a);
   strcat (res, b);
   return res;
}


int main(void)
{
    printf("concat\n");
    printf("%d\n", len("abc"));
    printf("%d\n", len(""));
    printf("%d\n", len("xxxxxxxxxx"));
    printf("%s\n", concat("Foo1", "Bar"));
    return 0;
}
```

```python
from ctypes import cdll
from ctypes import c_char_p

more_lib = cdll.LoadLibrary("more.so")

print(more_lib.len("abcd"))       # 4
print(more_lib.len(""))           # 0
print(more_lib.len("x" * 123))    # 123


more_lib.concat.restype = c_char_p
print(more_lib.concat("abc", "def"))
```

---

links

- ctypes
- Python Ctypes Tutorial

Advanced OOP

Class count instances

```python
class Person:
    count = 0
    def __init__(self, name):
        self.name = name
        #Person.count += 1
        #self.count += 1
        self.count = self.count + 1


print(Person.count)
joe = Person("Joe")
print(Person.count)
print(joe.count)

jane = Person("Jane")
print(Person.count)
print(jane.count)
```

```
0
0
1
0
1
```

Class Attributes

- Class attributes can be created inside a class.
- Assign to class attribute and fetch from it
- Class attributes can be also created from the outside.

```python
class Person:
    name = 'Joseph'

print(Person.name)     # Joseph

Person.name = 'Joe'
print(Person.name)     # Joe

Person.email = 'joe@foobar.com'
print(Person.email)    # joe@foobar.com
```

Class Attributes in Instances

```python
class Person:
    name = 'Joe'

# Class Attributes are inherited by object instances when accessing them.
x = Person()
print(x.name)          # Joe
y = Person()
print(y.name)          # Joe

# Changes to class attribute are reflected in existing instances as well
Person.name = 'Bar'
print(Person.name)     # Bar
print(x.name)          # Bar

# Setting the attribute via the instance will create an instance attribute that
# shadows the class attribute
x.name = 'Joseph'
print(x.name)          # Joseph
print(Person.name)     # Bar
# Nor does it impact the instance attribute of other instances:
print(y.name)          # Bar

# Both instance and class have a dictionary containing its members:
print(x.__dict__)        # {'name': 'Joseph'}
```

```
print(y.__dict__)         # {}
print(Person.__dict__)   # {..., 'name': 'Bar'}
```

Attributes with method access

- Use a method (show) to access it.

```
class Person():
    name = 'Joe'
    print(f'Hello {name}')

    def show(self):
        print(Person.name)

x = Person()             # Hello Joe
x.show()                 # Joe
print(x.name)            # Joe
print(Person.name)       # Joe

Person.name = 'Jane'
print(x.name)            # Jane
print(Person.name)       # Jane
x.show()                 # Jane

x.name = 'Hilda'         # creating and setting the instance attribute
print(x.name)            # Hilda
print(Person.name)       # Jane

x.show()                 # Jane
```

Instance Attribute

The attributes of the instance object can be set via 'self' from within the class.

```
class Person():
    name = 'Joseph'

    def __init__(self, given_name):
```

```python
        self.name = given_name

    def show_class(self):
        return Person.name

    def show_instance(self):
        return self.name

print(Person.name)          # Joseph

Person.name = 'Classy'
print(Person.name)      # Classy
# print(Person.show_class()) # TypeError: show_class() missing 1 required positional\
 argument: 'self'

x = Person('Joe')
print(x.name)               # Joe
print(Person.name)          # Classy
print(x.show_class())       # Classy
print(x.show_instance())    # Joe

Person.name = 'General'
print(x.name)               # Joe
print(Person.name)          # General
print(x.show_class())       # General
print(x.show_instance())    # Joe

x.name = 'Zorg'             # changing the instance attribute
print(x.name)               # Zorg
print(Person.name)          # General
print(x.show_class())       # General
print(x.show_instance())    # Zorg
```

Methods are class attributes

In this example we are going to replace the method in the class by a newly created function.
(monkey patching)

```python
class Person():
    def __init__(self, name):
        self.name = name

    def show(self):
        return self.name

y = Person('Jane')
print(y.show())          # Jane

def new_show(some_instance):
    print("Hello " + some_instance.name)
    return some_instance

Person.show = new_show
y.show()                 # Hello Jane
```

Monkey patching

```python
class Person():
    def __init__(self, name):
        self.name = name

    def show(self):
        return self.name

x = Person('Joe')
print(x.show())       # Joe

def patch(class_name):
    temp = class_name.show
    def debug(*args, **kwargs):
        print("in debug")
        return temp(*args, **kwargs)
    class_name.show = debug

patch(Person)

print(x.show())
     # in debug
     # Joe
```

Classes: instance method

Regular functions (methods) defined in a class are “instance methods”. They can only be called on “instance objects” and not on the “class object”
as see in the 3rd example.

The attributes created with “self.something = value” belong to the individual instance object.

```python
class Date:
    def __init__(self, Year, Month, Day):
        self.year  = Year
        self.month = Month
        self.day   = Day

    def __str__(self):
        return 'Date({}, {}, {})'.format(self.year, self.month, self.day)

    def set_date(self, y, m, d):
        self.year = y
        self.month = m
        self.day = d
```

```python
from mydate import Date

d = Date(2013, 11, 22)
print(d)

# We can call it on the instance
d.set_date(2014, 1, 27)
print(d)

```

```
# If we call it on the class, we need to pass an instance.
# Not what you would normally do.
Date.set_date(d, 2000, 2, 1)
print(d)


# If we call it on the class, we get an error
Date.set_date(1999, 2, 1)
```

`set_date` is an instance method. We cannot properly call it on a class.

```
Date(2013, 11, 22)
Date(2014, 1, 27)
Date(2000, 2, 1)
Traceback (most recent call last):
  File "run.py", line 17, in <module>
    Date.set_date(1999, 2, 1)
TypeError: set_date() missing 1 required positional argument: 'd'
```

Class methods and class attributes

“total” is an attribute that belongs to the class. We can access it using Date.total. We can create a `@classmethod` to access it, but actually we can access it from the outside even without the class method, just using the “class object”

```
class Date:
    total = 0

    def __init__(self, Year, Month, Day):
        self.year  = Year
        self.month = Month
        self.day   = Day
        Date.total += 1

    def __str__(self):
        return 'Date({}, {}, {})'.format(self.year,
```

```
self.month, self.day)

    def set_date(self, y, m, d):
        self.year = y
        self.month = m
        self.day = d

    @classmethod
    def get_total(class_object):
        print(class_object)
        return class_object.total
```

```
from mydate import Date

d1 = Date(2013, 11, 22)
print(d1)
print(Date.get_total())
print(Date.total)
print('')

d2 = Date(2014, 11, 22)
print(d2)
print(Date.get_total())
print(Date.total)
print('')

d1.total = 42
print(d1.total)
print(d2.total)
print(Date.get_total())
print(Date.total)
```

```
Date(2013, 11, 22)
<class 'mydate.Date'>
1
1

Date(2014, 11, 22)
<class 'mydate.Date'>
2
2

42
```

```
2
<class 'mydate.Date'>
2
2
```

Classes: constructor

- The "class" keyword creates a "class object". The default constructor of these classes are their own names.
- The actual code is implemented in the `__new__` method of the `object`.
- Calling the constructor will create an "instance object".

Class methods - alternative constructor

Class methods are used as Factory methods, they are usually good for alternative constructors. In order to be able to use a method as a class-method
(Calling Date.method(…) one needs to mark the method with the `@classmethod` decorator)

```python
class Date:
    def __init__(self, Year, Month, Day):
        self.year  = Year
        self.month = Month
        self.day   = Day

    def __str__(self):
        return 'Date({}, {}, {})'.format(self.year, self.month, self.day)

    def set_date(self, y, m, d):
        self.year = y
        self.month = m
        self.day = d

    @classmethod
    def from_str(class_object, date_str):
```

```
        '''Call as
           d = Date.from_str('2013-12-30')
        '''
        print(class_object)
        year, month, day = map(int, date_str.split('-'))
        return class_object(year, month, day)
```

```
from mydate import Date

d = Date(2013, 11, 22)
print(d)

d.set_date(2014, 1, 27)
print(d)

print('')
dd = Date.from_str('2013-10-20')
print(dd)

print('')
z = d.from_str('2012-10-20')
print(d)
print(z)
```

```
Date(2013, 11, 22)
Date(2014, 1, 27)

<class 'mydate.Date'>
Date(2013, 10, 20)

<class 'mydate.Date'>
Date(2014, 1, 27)
Date(2012, 10, 20)
```

Abstract Base Class

- Create a class object that cannot be used to create an instance object. (It must be subclassed)

- The subclass must implement certain methods required by the base-class.

```python
class NotImplementedError(Exception):
    pass

class Base():
    def foo(self):
        raise NotImplementedError()

    def bar(self):
        raise NotImplementedError()

class Real(Base):
    def foo(self):
        print('foo in Real')
    def bar(self):
        print('bar in Real')
    def other(self):
        pass

class Fake(Base):
    def foo(self):
        print('foo in Fake')

r = Real()
r.foo()
r.bar()
f = Fake()
f.foo()
f.bar()
```

```
foo in Real
bar in Real
foo in Fake
Traceback (most recent call last):
  File "no_abc.py", line 28, in <module>
    f.bar()     # NotImplementedError
  File "no_abc.py", line 9, in bar
    raise NotImplementedError()
__main__.NotImplementedError
```

Abstract Base Class with abc

- [abc]()

```python
from abc import ABC, abstractmethod

class Base(ABC):
    def __init__(self, name):
        self.name = name

    @abstractmethod
    def foo(self):
        pass

    @abstractmethod
    def bar(self):
        pass
```

ABC working example

```python
from with_abc3 import Base

class Real(Base):
    def foo(self):
        print('foo in Real')

    def bar(self):
        print('bar in Real')

    def other(self):
        pass

r = Real('Jane')
print(r.name)       # Jane
```

```
Jane
```

ABC - cannot instantiate the base-class

```
from with_abc3 import Base

b = Base('Boss')
```

```
Traceback (most recent call last):
  File "with_abc3_base.py", line 3, in <module>
    b = Base('Boss')
TypeError: Can't instantiate abstract class Base with abstract methods bar, foo
```

ABC - must implement methods

```
from with_abc3 import Base

class Fake(Base):
    def foo(self):
        print('foo in Fake')

f = Fake('Joe')
```

```
Traceback (most recent call last):
  File "with_abc3_fake.py", line 7, in <module>
    f = Fake('Joe')
TypeError: Can't instantiate abstract class Fake with abstract methods bar
```

Use Python @propery to fix bad interface (the bad interface)

When we created the class the first time we wanted to have a field representing the age of
a person. (For simplicity of the example we onlys store the years.)

```
class Person():
    def __init__(self, age):
        self.age = age

```

```
p = Person(19)
print(p.age)        # 19

p.age = p.age + 1
print(p.age)        # 20
```

Only after releasing it to the public have we noticed the problem. Age changes.

We would have been better off storing birthdate and if necessary calculating the age.

How can we fix this?

Use Python @propery to fix bad interface (first attempt)

This might have been a good solution, but now we cannot use this as a “fix” as this
would change the public interface from `p.age` to `p.age()`

```python
from datetime import datetime
class Person():
    def __init__(self, years):
        self.set_birthyear(years)

    def get_birthyear(self):
        return datetime.now().year - self._birthyear

    def set_birthyear(self, years):
        self._birthyear = datetime.now().year - years

    def age(self, years=None):
        if (years):
            self.set_birthyear(years)
        else:
            return self.get_birthyear()

```

```
p = Person(19)
print(p.age())        # 19

p.age(p.age() + 1)
print(p.age())        # 20
```

Use Python @propery to fix bad API

```
property(fget=None, fset=None, fdel=None, doc=None)
```

```
from datetime import datetime
class Person():
    def __init__(self, years):
        self.age =  years

    def get_birthyear(self):
        return datetime.now().year - self.birthyear

    def set_birthyear(self, years):
        self.birthyear = datetime.now().year - years

    age = property(get_birthyear, set_birthyear)

p = Person(19)
print(p.age)        # 19

p.age = p.age + 1
print(p.age)        # 20

p.birthyear = 1992
print(p.age)        # 28
   # warning: this will be different if you run the example in a year different from\
 2020 :)
```

Use Python @propery decorator to fix bad API

```python
from datetime import datetime
class Person():
    def __init__(self, years):
        self.age =  years

    # creates "getter"
    @property
    def age(self):
        return datetime.now().year - self.birthyear

    # creates "setter"
    @age.setter
    def age(self, years):
        self.birthyear = datetime.now().year - years

p = Person(19)
print(p.age)        # 19

p.age = p.age + 1
print(p.age)        # 20


p.birthyear = 1992
print(p.age)        # 28
   # warning: this will be different if you run the example in a year different from\
 2020 :)
```

- property article
- property docs

Use Python @propery for value validation

```python
from datetime import datetime
class Person():
    def __init__(self, years):
        self.age =  years

    @property
    def age(self):
        return datetime.now().year - self.birthyear
```

```

    @age.setter
    def age(self, years):
        if years < 0:
            raise ValueError("Age cannot be negative")
        self.birthyear = datetime.now().year - years
```

```
from person5 import Person

p = Person(19)
print(p.age)        # 19

p.age = p.age + 1
print(p.age)        # 20

p.birthyear = 1992
print(p.age)        # 28
    # warning: this will be different if you run the
example in a year different from\
 2020 :)
```

```
from person5 import Person

print("Hello")

p = Person(-1)
```

```
Hello
Traceback (most recent call last):
  File "person5_bad_init.py", line 5, in <module>
    p = Person(-1)
  File "/home/gabor/work/slides/python-
programming/examples/classes/person/person5.p\
y", line 4, in __init__
    self.age =  years
  File "/home/gabor/work/slides/python-
programming/examples/classes/person/person5.p\
y", line 13, in age
    raise ValueError("Age cannot be negative")
ValueError: Age cannot be negative
```

```
Hello
10
Traceback (most recent call last):
  File "person5_bad_setter.py", line 7, in <module>
    p.age = -1
  File "/home/gabor/work/slides/python-
programming/examples/classes/person/person5.p\
y", line 13, in age
    raise ValueError("Age cannot be negative")
ValueError: Age cannot be negative
```

class and static methods

Static methods are used when no “class-object” and no “instance-object” is required.
They are called on the class-object, but they don’t receive it as a parameter.

They might be better off placed in a module, like the other_method.

```python
def other_method(val):
    print(f"other_method: {val}")

class Date(object):
    def __init__(self, Year, Month, Day):
        self.year  = Year
        self.month = Month
        self.day   = Day

    def __str__(self):
        return 'Date({}, {}, {})'.format(self.year,
self.month, self.day)

    @classmethod
    def from_str(class_object, date_str):
        '''Call as
           d = Date.from_str('2013-12-30')
        '''
        print(f"from_str: {class_object}")
        year, month, day = map(int, date_str.split('-'))

```

```python
        other_method(43)

        if class_object.is_valid_date(year, month, day):
            return class_object(year, month, day)
        else:
            raise Exception("Invalid date")

    @staticmethod
    def is_valid_date(year, month, day):
        if 0 <= year <= 3000 and  1 <= month <= 12 and 1 <= day <= 31:
            return True
        else:
            return False
```

```python
import mydate

dd = mydate.Date.from_str('2013-10-20')
print(dd)

print('')
print(mydate.Date.is_valid_date(2013, 10, 20))
print(mydate.Date.is_valid_date(2013, 10, 32))
print('')

x = mydate.Date.from_str('2013-10-32')
```

```
from_str: <class 'mydate.Date'>
other_method: 43
Date(2013, 10, 20)

True
False

from_str: <class 'mydate.Date'>
other_method: 43
Traceback (most recent call last):
  File "run.py", line 11, in <module>
    x = mydate.Date.from_str('2013-10-32')
  File "/home/gabor/work/slides/python-programming/examples/classes/mydate4/mydate.py", line 26, in from_str
```

```
    raise Exception("Invalid date")
Exception: Invalid date
```

Destructor: del

```python
class Person:
    def __init__(self):
        print('__init__')
    def __del__(self):
        print('__del__')

def main():
    a = Person()
    print('in main - after')

main()
print('after main')
```

```
__init__
in main - after
__del__
after main
```

Destructor delayed

Becasue the object has a reference to itself. (Python uses both reference count and garbage collection.)

```python
class Person:
    def __init__(self, name):
        self.name = name
        print(f'__init__ {name}')

    def __del__(self):
        print(f'__del__ {self.name}')

def main():
    a = Person('A')
    b = Person('B')
    a.partner = a
```

```
    print('in main - after')

main()
print('after main')
```

```
__init__ A
__init__ B
in main - after
__del__ B
after main
__del__ A
```

Destructor delayed for both

Because the instances reference each other

```
class Person:
    def __init__(self, name):
        self.name = name
        print(f'__init__ for {self.name}')
    def __del__(self):
        print(f'__del__ for {self.name}')

def main():
    a = Person('Joe')
    b = Person('Jane')
    a.partner = b
    b.partner = a
    print('in main - after')

main()
print('after main')
```

```
__init__ for Joe
__init__ for Jane
in main - after
after main
__del__ for Joe
__del__ for Jane
```

Opearator overloading

```python
import copy

class Rect:
    def __init__(self, w, h):
        self.width  = w
        self.height = h

    def __str__(self):
        return 'Rect[{}, {}]'.format(self.width, self.height)

    def __mul__(self, other):
        o = int(other)
        new = copy.deepcopy(self)
        new.height *= o
        return new
```

```python
import shapes

r = shapes.Rect(10, 20)
print(r)
print(r * 3)
print(r)

print(4 * r)
```

```
Rect[10, 20]
Rect[10, 60]
Rect[10, 20]
Traceback (most recent call last):
  File "rect.py", line 8, in <module>
    print(4 * r)
TypeError: unsupported operand type(s) for *: 'int' and 'Rect'
```

In order to make the multiplication work in the other direction, one needs to implement the **rmul** method.

Operator overloading methods

```
*     __mul__,  __rmul__
+     __add__, __radd__
+=    __iadd__
<     __lt__
<=    __le__
...
```

- [see all of them in datamodel]()

Exercise: rectangular

Take the Rect class in the shapes module. Implement **rmul**, but in that case multiply the width of the rectangular.

Implement the addition of two rectangulars. I think this should be defined only if one of the sides is the same,
but if you have an idea how to add two rectangualars of different sides, then go ahead, implement that.

Also implement all the comparision operators when comparing two rectangulars, compare the area of the two. (like less-than)
Do you need to implement all of them?

Exercise: SNMP numbers

- SNMP numbers are strings consisting a series of integers separated by dots: 1.5.2, 3.7.11.2
- Create a class that can hold such an snmp number. Make sure we can compare them with less-than (the comparision is pair-wise for each number until we find two numbers that are

different. If one SNMP number is the prefix is the other then the shorter is “smaller”).

- Add a class-method, that can tell us how many SNMP numbers have been created.
- Write a separate file to add unit-tests

Exercise: Implement a Gene inheritance model combining DNA

- A class representing a person. It has an attribute called “genes” which is string of letters. Each character is a gene.
- Implement the `+` operator on genes that will create a new “Person” and for the gene will select one randomly from each parent.

```
a = Person('ABC')
b = Person('DEF')

c = a + b
print(c.gene) # ABF
```

Exercise: imaginary numbers - complex numbers

Create a class that will represent imaginary numbers `(x, y*i)` and has methods to add and multiply two imaginary numbers.

```
The math:

z1 = (x1 + y1*i)
z2 = (x2 + y2*i)
z1+z2 = (x1 + x2  + (y1 + y2)*i)

z1*z2 = x1*y1 + x2*y2*i*i + x1*y2*i + x2*y1*i
```

Add operator overloading so we can really write code like:

```
z1 = Z(2, 3)
z2 = Z(4, 7)

zz = z1*z2
```

- See cmath

```
z = complex(2, 3)
print(z)
print(z.real)
print(z.imag)

imag = (-1) ** 0.5
print(imag)

i = complex(0, 1)
print(i)
print(i ** 2)
```

```
(2+3j)
2.0
3.0
(6.123233995736766e-17+1j)
1j
(-1+0j)
```

Solution: Rectangular

```
import copy
import shapes

class Rectangular(shapes.Rect):

    def __rmul__(self, other):
        o = int(other)
        new = copy.deepcopy(self)
        new.width *= o
        return new
```

```python

    def area(self):
        return self.width * self.height

    def __eq__(self, other):
        return self.area() == other.area()

    def __add__(self, other):
        new = copy.deepcopy(self)
        if self.width == other.width:
            new.height += other.height
        elif self.height == other.height:
            new.width += other.width
        else:
            raise Exception('None of the sides are equal')
        return new
```

```python
import shape2
import unittest

class TestRect(unittest.TestCase):

    def assertEqualSides(self, left, right):
        if isinstance(right, tuple):
            right = shape2.Rectangular(*right)

        if left.width != right.width:
            raise AssertionError('widths are different')
        if left.height != right.height:
            raise AssertionError('heights are different')

    def setUp(self):
        self.a = shape2.Rectangular(4, 10)
        self.b = shape2.Rectangular(2, 20)
        self.c = shape2.Rectangular(1, 30)
        self.d = shape2.Rectangular(4, 10)

    def test_sanity(self):
        self.assertEqualSides(self.a, self.a)
        self.assertEqualSides(self.a, self.d)
        try:
            self.assertEqualSides(self.a, self.b)
        except AssertionError as e:
```

```
            self.assertEqual(e.args[0], 'widths are different')

        try:
            self.assertEqualSides(self.a, shape2.Rectangular(4, 20))
        except AssertionError as e:
            self.assertEqual(e.args[0], 'heights are different')

        self.assertEqualSides(self.a, (4, 10))

    def test_str(self):
        self.assertEqual(str(self.a), 'Rect[4, 10]')
        self.assertEqual(str(self.b), 'Rect[2, 20]')
        self.assertEqual(str(self.c), 'Rect[1, 30]')

    def test_mul(self):
        self.assertEqual(str(self.a * 3), 'Rect[4, 30]')
        self.assertEqual(str(self.b * 7), 'Rect[2, 140]')

    def test_rmul(self):
        self.assertEqual(str(3 * self.a), 'Rect[12, 10]')
        self.assertEqualSides(3 * self.a, (12, 10))

    def test_area(self):
        self.assertEqual(self.a.area(), 40)
        self.assertEqual(self.b.area(), 40)
        self.assertEqual(self.c.area(), 30)

    def test_equal(self):
        self.assertEqual(self.a, self.d)
        self.assertEqual(self.a, self.b)

    def test_add(self):
        self.assertEqualSides(self.a + shape2.Rectangular(4, 20), (4, 30))




if __name__ == '__main__':
    unittest.main()
```

Solution: Implement a Gene inheritance model combining DNA

```python
import random

class Person(object):
    def __init__(self, DNA):
        self.DNA = DNA

    def gene(self):
        return list(self.DNA)

    def print_genes(self):
        print(list(self.DNA))

    def __add__(self, other):
        DNA_father = self.gene()
        DNA_mother = other.gene()
        if len(DNA_father) != len(DNA_mother):
            raise Exception("Incompatible couple")

        DNA_childPosible_sequence = DNA_father + DNA_mother
        DNA_child = ""
        for i in range(len(self.gene())):
            DNA_child += random.choice([DNA_father[i], DNA_mother[i]])

        return Person(DNA_child)


a = Person("ABCD")
b = Person("1234")
c = a + b
print(c.DNA)
```

Instance counter

```python
class Bike:
    count = 0
    def __init__(self):
        Bike.count += 1
```

```python

    def __del__(self):
        Bike.count -= 1

def bike_trip():
    print(Bike.count)    # 0
    a = Bike()
    print(Bike.count)    # 1
    b = Bike()
    print(Bike.count)    # 2
    c = Bike()
    print(Bike.count)    # 3
    b = None
    print(Bike.count)    # 2


bike_trip()
print(Bike.count)        # 0
```

2to3

Convertig from Python 2 to Python 3

from **future** import …

division

```
print 3/2   # 1
```

```
from __future__ import division

print 3/2   # 1.5
```

print in Python 2

```
fname = 'Foo'
lname = 'Bar'
print("Name: %s %s" % (fname, lname))
print("Name: {} {}".format(fname, lname))
print(fname, lname)
print fname, lname
```

```
Name: Foo Bar
Name: Foo Bar
('Foo', 'Bar')
Foo Bar
```

print in Python 3

print now requires print()

```
from __future__ import print_function

fname = 'Foo'
lname = 'Bar'
print("Name: %s %s" % (fname, lname))
print("Name: {} {}".format(fname, lname))
print(fname, lname)
```

```
Name: Foo Bar
Name: Foo Bar
Foo Bar
```

input and raw_input

`raw_input()` was renamed to `input()`

In Python 2 `raw_input()` returned the raw string. input(), on the other hand ran eval(raw_input())
which meant it tried to execute the input string as a piece of Python code. This was dangerous and was not really used.

In Python 3 raw_input() is gone. input() behaves as the old raw_input() returning the raw string. If you would like to get the old,
and dangerous, behavior of input() you can call eval(input()).

Code that works on both 2 and 3

```
import platform

def my_input(text):
    if platform.python_version_tuple()[0] == 3:
        return input(text)
    else:
        return raw_input(text)
```

Compare different types

```
x = 3
y = '3'

                           # Python 2        Python 3
print( x > y )             # False           TypeError:
unorderable types: int() > str()
print( x < y )             # True            TypeError:
unorderable types: int() < str()
print( x == y )            # False           False
```

Octal numbers

Octal numbers in 2.x was `011` in 3.x is: `0o11`

2to3 Resources

- python3porting book
- wiki
- Dive into Python 3
- The **future** module
- The third-party future module
- The six module
- docs of 2to3

Design Patterns

What are Design Patterns?

Not all the Design Patterns discussed for Java or C++ are interesting, relevant or even needed in Python.
Design Patterns are formal descriptions of how people do things, and not how you should do things.
The formal description makes it easy to talk about them.

Some of the DPs exists to overcome problems in that specific language.
Oher DPs are more general, solving classes of problem that are generic.

Don’t replace built-in objects

```
import sys

print = 'hello'
sys.stdout.write(print)
sys.stdout.write('\n')
```

```
pip install flake8-builtins
flake8 --ignore=   replace_print.py

replace_print.py:3:1: A001 "print" is a python builtin
and is being shadowed, consid\
er renaming the variable
```

Facade - simple interface to complex system

> Facade, a structural design pattern. - Provide a simple interface (maybe a single class with few methods) to some complex system behind it.
> This gives flexibility for the implementation of the complex system while users gain simplicity in using
> it in certain subsets of operations.

```
os.path.basename, os.path.dirname are faced for os.path.split + indexing in the list
os.path.basename  = os.path.split()[-1]
os.path.split = split with os.sep
os.path.join(names) = os.sep.join(names)
os.path.isdir(path) = stat.S_ISDIR(os.stat(path))
```

- [](http://docs.python.org/library/os.path.html)
- [](http://docs.python.org/library/os.html)
- [](http://docs.python.org/library/stat.html)

Monkey Patching

```python
import real_class
class faker(object): pass
fake = faker
real_class.time = fake
fake.sleep =
fake.time =
```

- handy in emergencies
- easily abused for NON-emergencies - gives dynamic languages a bad name

- subtle hidden “communication” via secret obscure pathways (explicit is better)

```
class Monkey:

    def __init__(self, count):
        self.bananas = count

    def is_hungry(self):
        hungry = True
        if hungry:
            self.eat()

    def eat(self):
        self.bananas -= 1


m = Monkey(10)
print(m.bananas)        # 10
print(m.is_hungry())    # None
print(m.bananas)        # 9

Monkey.eat = lambda self: True

om = Monkey(10)
print(om.bananas)       # 10
print(om.is_hungry())   # None
print(om.bananas)       # 10
```

Creation DPs “Just One”

we want just one instance to exist

- Singleton - subclassing can never be really smooth
- Use a module instead of a class (no inheritance, no special methods)
- make just one instance (self discipline, no enforcement), need to decide to “when” (in which part if the code) to make it
- monostate (borg)

Singleton

```
class Singleton(object):
def __new__(cls, *a, **kw):
    if not hasattr(cls, '_inst'):
        cls._inst = super(Singleton, cls).__new__(*a, **kw)
    return cls._inst
```

the problem

```
class Foo(Singleton): pass
class Bar(Foo): pass
f = Foo()
b = Bar()
# what class is b now? is that a Bar or a Foo instance?
```

Monostate (Borg)

- [Monostate Pattern]()

```
class Monostate(object):
    _shared_state = {}
    def __new__(cls, *a, **kw):
        obj = super(Monostate, cls).__new__(*a, **kw)
        obj.__dict__ = _shared_state
        return obj

class Foo(Monostate) pass
class Bar(Foo) pass
f = Foo()
b = Bar()
```

Better than singleton, data overriding to the rescue:
But what if two calls to the constructor provide different initial data?

Dispatch table

```
calls = []
calls.append( lambda x: x+1 )
calls.append( lambda x: x*2 )

others = [
    lambda x: x-1,
    lambda x: 0
]

def do_something( call_list ):
    for c in call_list:
        print(c(3))


do_something( calls )
do_something( others )
```

Parallel

Types of Problems

- CPU intensive application - use more of the cores to reduce the wallclock time.
- IO intensive applications - don't waste the CPU and wallclock time while waiting for the IO process.
- Interactive applications - make sure they are responsive during long operations.

Types of solutions

- Number of processes (forking on Unix or spawning)
- Number of threads (Single threaded vs Multi-threaded)
- Asynchronous, non-blocking or synchronous vs blocking (aka "normal") Cooperative Multitasking

How many parallels to use?

* First of all, I call them "parallels" as this applies to forks, threads, spawns, and even to async code.

- Overhead of creating new parallel.
- Overhead of communication (sending job input to parallel, receiving results).
- Total number of items to process.
- Time it takes to process an item.

- Distribution of processing times. (e.g. one long and many short jobs.)
- Number of cores (CPUs).

Dividing jobs

- N items to process
- K in parallel

- Divide the items in K groups of size int(N/K) and int(N/K)+1.
- Create K parallels with one item each. When it is done, give it another item.
- Create K parallels with one item each. When done let it stop and create a new parallel.

Performance Monitoring

- Linux, OSX: htop
- Windows: Performance Monitor

Threads

Python Threading docs

- threading
- Real Python
- Wikibooks

Threaded counters

```
import threading
import sys

class ThreadedCount(threading.Thread):
    def run(self):
        for cnt in range(6):
            print(f"{cnt} {threading.current_thread().name}")
        return

a = ThreadedCount()
b = ThreadedCount()
c = ThreadedCount()

a.start()
b.start()
c.start()
print('main - Running {} threads'.format(threading.active_count()))

a.join()
b.join()
c.join()
print("main - thread is done")
```

```
0 Thread-1
1 Thread-1
0 Thread-2
2 Thread-1
1 Thread-2
0 Thread-3
3 Thread-1
2 Thread-2
main - Running 4 threads
3 Thread-2
1 Thread-3
4 Thread-2
2 Thread-3
5 Thread-2
3 Thread-3
4 Thread-1
4 Thread-3
5 Thread-1
5 Thread-3
main - thread is done
```

Simple threaded counters

```python
import threading
import sys

class ThreadedCount(threading.Thread):
    def run(self):
        thread = threading.current_thread()
        print('{} - start'.format(thread.name))
        for c in range(10):
            print('{} - count {}'.format(thread.name, c))
        print('{} - end'.format(thread.name))
        return

a = ThreadedCount()
b = ThreadedCount()
c = ThreadedCount()
a.start()
b.start()
c.start()

print('main - running {}
```

```
threads'.format(threading.active_count()))

a.join()
b.join()
c.join()
print("main - thread is done")
```

```
Thread-1 - start
Thread-1 - count 0
Thread-1 - count 1
Thread-2 - start
Thread-1 - count 2
Thread-2 - count 0
Thread-1 - count 3
Thread-3 - start
main - running 4 threads
Thread-2 - count 1
Thread-1 - count 4
Thread-2 - count 2
Thread-1 - count 5
Thread-2 - count 3
Thread-1 - count 6
Thread-2 - count 4
Thread-1 - count 7
Thread-2 - count 5
Thread-1 - count 8
Thread-2 - count 6
Thread-1 - count 9
Thread-2 - count 7
Thread-1 - end
Thread-2 - count 8
Thread-2 - count 9
Thread-2 - end
Thread-3 - count 0
Thread-3 - count 1
Thread-3 - count 2
Thread-3 - count 3
Thread-3 - count 4
Thread-3 - count 5
Thread-3 - count 6
Thread-3 - count 7
Thread-3 - count 8
Thread-3 - count 9
```

```
Thread-3 - end
main - thread is done
```

Simple threaded counters (parameterized)

The same as the previous one, but with parameters controlling the numbers
of threads and the range of the counter.

```python
import threading
import sys

num_threads, count_till = 3, 5

class ThreadedCount(threading.Thread):
    def run(self):
        thread = threading.current_thread()
        print(f'{thread.name} - start')
        for cnt in range(count_till):
            print(f'{thread.name} - count {cnt}')
        print(f'{thread.name} - end')
        return

threads = []
for ix in range(num_threads):
    threads.append(ThreadedCount())

for th in threads:
    th.start()

print('main - running {} threads'.format(threading.active_count()))

for th in threads:
    th.join()
print("main - thread is done")
```

```
Thread-1 - start
Thread-1 - count 0
Thread-1 - count 1
Thread-1 - count 2
```

```
Thread-1 - count 3
Thread-1 - count 4
Thread-1 - end
Thread-2 - start
Thread-2 - count 0
Thread-2 - count 1
Thread-2 - count 2
Thread-2 - count 3
Thread-2 - count 4
Thread-2 - end
Thread-3 - start
Thread-3 - count 0
Thread-3 - count 1
Thread-3 - count 2
Thread-3 - count 3
Thread-3 - count 4
Thread-3 - end
main - running 1 threads
main - thread is done
```

Pass parameters to threads - Counter with attributes

```python
import threading
import sys

class ThreadedCount(threading.Thread):
    def __init__(self, name, start, stop):
        super().__init__()
        self.name = name
        self.counter = start
        self.limit = stop
        print('__init__ of {} in {}'.format(self.name, threading.current_thread()))

    def run(self):
        print('start run of {} in {}'.format(self.name, threading.current_thread()))
        while self.counter < self.limit:
            print('count {} of {}'.format(self.name, self.counter))
            self.counter += 1
```

```
        print('end run   of {} in {}'
            .format(self.name, threading.current_thread()))
        return

foo = ThreadedCount("Foo", 1, 11)
bar = ThreadedCount("Bar", 1, 11)
foo.start()
bar.start()
print('main - running {} threads'.format(threading.active_count()))
foo.join()
bar.join()
print("main - thread is done")
```

```
__init__  of Foo in <_MainThread(MainThread, started 139645405484864)>
__init__  of Bar in <_MainThread(MainThread, started 139645405484864)>
start run of Foo in <ThreadedCount(Foo, started 139645391374080)>
count Foo of 1
count Foo of 2
start run of Bar in <ThreadedCount(Bar, started 139645382981376)>
count Bar of 1
main - running 3 threads
count Foo of 3
count Bar of 2
count Foo of 4
count Bar of 3
count Foo of 5
count Bar of 4
count Foo of 6
count Bar of 5
count Foo of 7
count Bar of 6
count Foo of 8
count Bar of 7
count Foo of 9
count Bar of 8
count Foo of 10
count Bar of 9
end run   of Foo in <ThreadedCount(Foo, started
```

```
139645391374080)>
count Bar of 10
end run    of Bar in <ThreadedCount(Bar, started 139645382981376)>
main - thread is done
```

Create a central counter

```python
import threading
import sys
import time

cnt = 0
num = 30
limit = 100000

class ThreadedCount(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.counter = 0

    def run(self):
        global cnt
        while self.counter < limit:
            self.counter += 1
            cnt += 1
        return

start = time.time()
threads = [ ThreadedCount() for n in range(num) ]
[ t.start() for t in threads ]
[ t.join() for t in threads ]
end = time.time()

print("Expected: {}".format(num * limit))
print("Received: {}".format(cnt))
print("Elapsed: {}".format(end-start))

# Expected: 3000000
# Received: 2659032
# Elapsed: 0.437514066696167
```

Lock - acquire - release

```python
import threading
import sys
import time

cnt = 0
num = 30
limit = 100000

locker = threading.Lock()

class ThreadedCount(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.counter = 0
    def run(self):
        global cnt
        while self.counter < limit:
            self.counter += 1
            locker.acquire()
            cnt += 1
            locker.release()
        return

start = time.time()
threads = [ ThreadedCount() for n in range(num) ]
[ t.start() for t in threads ]
[ t.join() for t in threads ]
end = time.time()

print("Expected: {}".format(num * limit))
print("Received: {}".format(cnt))
print("Elapsed: {}".format(end-start))

# Expected: 3000000
# Received: 3000000
# Elapsed: 12.333643198013306
```

Counter - plain

```python
import sys
import time

cnt = 0
num = 30
limit = 100000

class Count():
    def __init__(self):
        self.counter = 0
    def run(self):
        global cnt
        while self.counter < limit:
            self.counter += 1
            cnt += 1
        return

start = time.time()
for _ in range(num):
    c = Count()
    c.run()
end = time.time()

print("Expected: {}".format(num * limit))
print("Received: {}".format(cnt))
print("Elapsed: {}".format(end-start))

# Expected: 3000000
# Received: 3000000
# Elapsed: 0.4130408763885498
```

GIL - Global Interpreter Lock

- Solves the problem introduced by having reference count.
- Not going away any time soon.

- [GIL wiki]()
- [GIL realpython]()

Thread load

```python
import threading
import sys
import time
import random


results = []
locker = threading.Lock()

class ThreadedCount(threading.Thread):
    def __init__(self, n):
        threading.Thread.__init__(self)
        self.n = n

    def run(self):
        count = 0
        total = 0
        while count < 40000000 / self.n:
            rnd = random.random()
            total += rnd
            count += 1

        locker.acquire()
        results.append({'count': count, 'total': total})
        locker.release()
        return

def main():
    if len(sys.argv) != 2:
        exit("Usage: {} POOL_SIZE")
    size = int(sys.argv[1])
    start = time.time()
    threads = [ ThreadedCount(n=size) for i in range(size) ]
    [ t.start() for t in threads ]
    [ t.join() for t in threads ]
    print("Results: {}".format(results))
    totals = map(lambda r: r['total'], results)
    print("Total: {}".format(sum(totals)))
    end = time.time()
    print(end - start)

```

```
if __name__ == '__main__':
    main()
```

```
$ time python thread_load.py 1
Results: [{'count': 40000000, 'total':
19996878.531261113}]
Total: 19996878.531261113
6.478948354721069

real    0m6.539s
user    0m6.491s
sys     0m0.012s
```

```
$ time python thread_load.py 4
Results: [{'count': 10000000, 'total':
5000680.7382364655}, {'count': 10000000, 'tot\
al': 5000496.15077697}, {'count': 10000000, 'total':
5000225.747780174}, {'count': 1\
0000000, 'total': 4999503.803068357}]
Total: 20000906.43986197
6.180345296859741

real    0m6.241s
user    0m6.283s
sys     0m0.029s
```

Exercise: thread files

- Get a list of files (from the current directory or from all the files in the "slides" repository.
- Process each file:
- 1. get size of file
- 2. count how many times each character appear in the file.
- The script should accept the number of threads to use.

Exercise: thread URL requests.

In the following script we fetch the URLs listed in a file:

```
https://google.com/
https://youtube.com/
https://facebook.com/
https://baidu.com/
https://twitter.com/
https://instagram.com/
https://wikipedia.com/
https://www.amazon.com/
https://yahoo.com/
https://yandex.ru/
https://vk.com/
https://live.com/
https://naver.com/
https://yahoo.co.jp/
https://google.com.br/
https://netflix.com/
https://reddit.com/
https://ok.ru/
https://mail.ru/
https://ebay.com/
https://linkedin.com/
https://qq.com/
https://pinterest.com/
https://bing.com/
https://whatsapp.com/
https://office.com/
https://amazon.de/
https://aliexpress.com/
https://amazon.co.jp/
https://msn.com/
https://google.de/
https://paypal.com/
https://rakuten.co.jp/
https://amazon.co.uk/
https://daum.net/
https://google.co.jp/
https://taobao.com/
https://bilbili.com/
https://imdb.com/
https://booking.com/
https://roblox.com/
https://9apps.com/
```

```
https://globo.com/
https://duckduckgo.com/
https://www.nttdocomo.co.jp/
```

It takes about 1.5-2 sec / URL from home. (It depends on a lot of factors including your network connection.)

```python
import time
import requests
import sys
from bs4 import BeautifulSoup

def get_urls(limit):
    with open('urls.txt') as fh:
        urls = list(map(lambda line: line.rstrip("\n"), fh))
    if len(urls) > limit:
        urls = urls[:limit]

    return urls

def get_title(url):
    try:
        resp = requests.get(url)
        if resp.status_code != 200:
            return None, f"Incorrect status_code {resp.status_code} for {url}"
    except Exception as err:
        return None, f"Error: {err} for {url}"

    soup = BeautifulSoup(resp.content, 'html.parser')
    return soup.title.string, None

def main():
    if len(sys.argv) < 2:
        exit(f"Usage: {sys.argv[0]} LIMIT")
    limit = int(sys.argv[1])
    urls = get_urls(limit)
    print(urls)
    start = time.time()

    titles = []
    for url in urls:
```

```
        #print(f"Processing {url}")
        title, err = get_title(url)
        if err:
            print(err)
        else:
            print(title)
        titles.append({
            "url": url,
            "title": title,
            "err": err,
        })
    end = time.time()
    print("Elapsed time: {} for {} pages.".format(end-start, len(urls)))
    print(titles)


if __name__ == '__main__':
    main()
```

Create a version of the above script that can use K threads.

Exercise: thread queue

Write an application that handles a queue of jobs in N=5 threads. Each job contains a number between 0-5.
Each thread takes the next element from the queue and sleeps for the given amount
of second (as an imitation of actual work it should be doing). When finished it checks
for another job. If there are no more jobs in the queue, the thread can close itself.

```
import threading
import random
import sys

thread_count = 5

```

```
counter = 0
queue = map(lambda x: ('main', random.randrange(5)), range(20))
print(queue)
```

If that's done, change the code so that each thread will generate a random number between 0-5 (for sleep-time) and in 33% of the cases it will add it to the central queue as a new job.

Another extension to this exercise is to change the code to limit the number of jobs each thread can execute in its lifetime. When the thread has finished that many jobs it will quit and the main thread will create a new worker thread.

Solution: thread queue

```
import threading
import random
import sys
import time

thread_count = 5

counter = 0
queue = list(map(lambda x: ('main', random.randrange(5)), range(20)))
#print(queue)

locker = threading.Lock()

class ThreadedCount(threading.Thread):
    def run(self):
        global counter
        my_counter = 0
        thread = threading.current_thread()
```

```python
        print('{} - start thread'.format(thread.name))
        while (True):
            locker.acquire()
            job = None
            if len(queue) > 0:
                counter += 1
                my_counter += 1
                job = queue[0]
                queue[0:1] = []
            locker.release()
            if job == None:
                print('{} - no more jobs'.format(thread.name))
                break

            print('{} - working on job {} ({}) from {} sleep for {}'
                .format(thread.name, counter, my_counter, job[0], job[1]))
            time.sleep(job[1])

        return

threads = []
for i in range(thread_count):
    threads.append(ThreadedCount())
for t in threads:
    t.start()
for t in threads:
    t.join()
```

Solution: thread URL requests.

```python
import time
import threading
import requests
import sys
from bs4 import BeautifulSoup

from fetch_urls import get_urls, get_title

titles = []
locker = threading.Lock()
```

```
class GetURLs(threading.Thread):
    def __init__(self, urls):
        threading.Thread.__init__(self)
        self.urls = urls

    def run(self):
        my_titles = []
        for url in self.urls:
            title, err = get_title(url)
            my_titles.append({
                'url': url,
                'title': title,
                'err': err,
            })
        locker.acquire()
        titles.extend(my_titles)
        locker.release()
        return

def main():
    if len(sys.argv) < 3:
        exit(f"Usage: {sys.argv[0]} LIMIT THREADS")
    limit = int(sys.argv[1])
    threads_count = int(sys.argv[2])

    urls = get_urls(limit)
    print(urls)
    start_time = time.time()
    batch_size = int(limit/threads_count)
    left_over = limit % threads_count
    batches = []
    end = 0
    for ix in range(threads_count):
        start = end
        end = start + batch_size
        if ix < left_over:
            end += 1
        batches.append(urls[start:end])

    threads = [ GetURLs(batches[ix]) for ix in range(threads_count) ]
    [ t.start() for t in threads ]
    [ t.join() for t in threads ]
```

```
    end_time = time.time()
    print("Elapsed time: {} for {} pages.".format(end_time-start_time, len(urls)))
    print(titles)


if __name__ == '__main__':
    main()
```

Forking

Fork

- fork

```python
import os
import time

print('{} - start running'.format(os.getpid()))

pid = os.fork()
if not pid:
    print('{} - in child. Parent is {}'.format(os.getpid(), os.getppid()))
    time.sleep(1)
    exit(3)

print('{} - in parent (child pid is {})'.format(os.getpid(), pid))

child_pid, exit_code = os.wait()
print('{} - Child with pid {} exited. Exit code {}'.format(os.getpid(), child_pid, e\
xit_code))
print('Real exit code {}'.format(int(exit_code/256)))   # The upper byte
print('Also known as {}'.format(exit_code >> 8))   # Right shift 8 bits
```

```
10278 - start running
10279 - in child. Parent is 10278
10278 - start running
10278 - in parent (child pid is 10279)
10278 - Child with pid 10279 exited. Exit code 768
Real exit code 3
Also known as 3
```

Forking

```python
import os
import time

name = "common"

def child():
    print("In Child of {}".format(name))
    print("In Child PID: {} PPID: {}".format(os.getpid(), os.getppid()))

    time.sleep(5)
    exit(3)

def parent(child_pid):
    print("In Parent ({}) The child is: {}".format(name, child_pid))
    print("In Parent PID: {} PPID: {}".format(os.getpid(), os.getppid()))
    r = os.wait()
    print(r)

pid = os.fork()
print(pid)
if pid == 0:
    child()
else:
    parent(pid)
```

```
0
In Child of common
In Child PID: 11212 PPID: 11211
11212
In Parent (common) The child is: 11212
In Parent PID: 11211 PPID: 4195
(11212, 768)
```

Fork skeleton

```python
import os
import glob

files = glob.glob("*.py")
# print(files)
count = len(files)
print(f"Number of items to process: {count}")

parallel = 4    # How many in parallel

batch = int(count/parallel)
leftover = count % parallel
print(f"batch size: {batch}  leftover: {leftover}")

def parent(pid):
    print(f"parent {pid}")

def child(files):
    print(f"{os.getpid()}  {files}")
    exit()

end = 0
for ix in range(parallel):
    start = end
    end   = start + batch
    if ix < leftover:
        end += 1
    print(f"start={start} end={end}")

    pid = os.fork()
    if pid:
        parent(pid)
    else:
        child(files[start:end])

print(f"In parent {os.getpid()}")
for ix in range(parallel):
    r = os.wait()
    print(r)
```

Fork with load

```python
import os
import random
import sys

if len(sys.argv) != 2:
    exit("Usage: {} N".format(sys.argv[0]))
n = int(sys.argv[1])
for p in range(0, n):
    pid = os.fork()
    if not pid:
        print('In Child')
        i = 0
        while i < 40000000/n:
            x = random.random()
            y = random.random()
            z = x+y
            i += 1
        exit(3)
    print('In Parent of', pid)

for p in range(0, n):
    r = os.wait()
    print(r)
```

Fork load results

```
$ time python fork_load.py 1
```

```
In Parent of 96355
In Child
(96355, 768)

real	0m26.391s
user	0m25.893s
sys	0m0.190s
```

```
$ time python fork_load.py 8
```

```
In Parent of 96372
In Parent of 96373
```

```
In Parent of 96374
In Child
In Child
In Parent of 96375
In Child
In Child
In Parent of 96376
In Child
In Parent of 96377
In Child
In Child
In Parent of 96378
In Parent of 96379
In Child
(96374, 768)
(96372, 768)
(96375, 768)
(96373, 768)
(96376, 768)
(96377, 768)
(96378, 768)
(96379, 768)

real    0m12.754s
user    0m45.196s
sys 0m0.164s
```

Marshalling / Serialization

Marshalling (or serialization) is the operation when we take an arbitrary
data structure and convert it into a string in a way that we can convert
the string back to the same data structure.

Marshalling can be used to save data persistent between execution of the same
script, to transfer data between processes, or even between machines.

In some cases it can be used to communicate between two processes written in different programming languages.

The marshal module provides such features but it is not recommended as it was built for internal object serialization for python.

The pickle module was designed for this task.

The json module can be used too.

Fork with random

When the **random** module is loaded it automatically calls `random.seed()` to initialize the random generator. When we create a fork this is not called again and thus all the processes will return the same random numbers. We can fix this by calling `random.seed()` manually.

```python
import os, time, random

print('{} - start running'.format(os.getpid()))

pid = os.fork()
if not pid:
    #random.seed()
    print('{} - in child'.format(os.getpid()))
    print(random.random())
    time.sleep(1)
    exit(3)

print('{} - in parent (child pid is {})'.format(os.getpid(), pid))
print(random.random())

```

```
done = os.wait()
print('{} - Child exited {}'.format(os.getpid(), done))
```

Exercise: fork return data

Create a script that will go over a list of numbers and does some computation on each number.

```
import sys
import time
from mymodule import calc

def main(n):
    results = {}
    print(f"do 1-{n}")
    for ix in range(1, n):
        results[ix] = calc(ix)
    return results

if __name__ == '__main__':
    if len(sys.argv) < 2:
        exit(f"Usage: {sys.argv[0]} NUMBER")

    start = time.time()
    results = main(1+int(sys.argv[1]))
    end = time.time()
    total = sum(results.values())
    print(f"Total: {total}")
    print("Elapsed time: {}".format(end-start))
```

Allow the child process to return data to the parent process. Before exiting from the child process, serialize the data-structure you want to send back and save
in a file that corresponds to the parent process and the child process. (eg. created from the PID of the paraent process and the PID of the child process)
In the parent process, when one of the children exits, check if there

is a file corresponding to this child process, read the file and de-serialize it.

Solution: fork return data

```python
import sys
import os
import json
import time
from mymodule import calc

def child(start, end):
    results = {}
    for ix in range(start, end):
        results[ix] = calc(ix)
    filename = str(os.getpid()) + '.json'
    with open(filename, 'w') as fh:
        json.dump(results, fh)
    exit()

def main(total_number, parallels):
    results = {}

    processes = []
    a_range = int(total_number / parallels)
    for cnt in range(parallels):
        start = 1 + cnt * a_range
        end =  start + a_range
        if cnt == parallels - 1:
            end = total_number + 1
        print(f"do: {start}-{end}")
        pid = os.fork()
        if pid:
            processes.append(pid) # parent
        else:
            child(start, end)
    for _ in range(len(processes)):
        pid, exit_code = os.wait()
        #print(pid, exit_code)
        filename = str(pid) + '.json'
        with open(filename) as fh:
            res = json.load(fh)
            print(f"{pid}: {res}")
```

```python
                results.update(res)
            os.unlink(filename)
    return results

if __name__ == '__main__':
    if len(sys.argv) < 3:
        exit(f"Usage: {sys.argv[0]} NUMBER PARALLEL")

    start = time.time()
    results = main(int(sys.argv[1]), int(sys.argv[2]))
    print(f"results: {results}")
    end = time.time()
    total = sum(results.values())
    print(f"Total: {total}")
    print("Elapsed time: {}".format(end-start))
```

Asyncronus programming with AsyncIO

Sync chores

We have a number of household chores to do. Each takes a couple of seconds for a machine to do
while we have time to do something else. We also have one task, cleaning potatoes, that requires
our full attention. It is a CPU-intensive process.

We also have two processes depending each other. We can turn on the dryer only after the
washing machine has finished.

```python
import time

def boil_water(sec):
    print(f"Start boiling water for {sec} seconds")
    time.sleep(sec)
    print(f"End boiling water for {sec} seconds")

def washing_machine(sec):
    print("Start washing machine")
    time.sleep(sec)
    print("End washing machine")

def dryer(sec):
    print("Start dryer")
    time.sleep(sec)
    print("End dryer")

def dishwasher(sec):
    print("Start dishwasher")
    time.sleep(sec)
```

```python
    print("End dishwasher")

def clean_potatoes(pieces):
    print("Start cleaning potatoes")
    for ix in range(pieces):
        print(f"Cleaning potato {ix}")
        time.sleep(0.5)
    print("End cleaning potatoes")

def main():
    dishwasher(3)
    washing_machine(3)
    dryer(3)
    boil_water(4)
    clean_potatoes(14)

start = time.time()
main()
end = time.time()
print(f"Elapsed {end-start}")
```

```
Start dishwasher
End dishwasher
Start washing machine
End washing machine
Start dryer
End dryer
Start boiling water for 4 seconds
End boiling water for 4 seconds
Start cleaning potatoes
Cleaning potato 0
Cleaning potato 1
Cleaning potato 2
Cleaning potato 3
Cleaning potato 4
Cleaning potato 5
Cleaning potato 6
Cleaning potato 7
Cleaning potato 8
Cleaning potato 9
Cleaning potato 10
Cleaning potato 11
Cleaning potato 12
Cleaning potato 13
```

```
End cleaning potatoes
Elapsed 20.017353534698486
```

Async chores

```python
import time
import asyncio

async def boil_water(sec):
    print(f"Start boiling water for {sec} seconds")
    await asyncio.sleep(sec)
    print(f"End boiling water for {sec} seconds")

async def washing_machine(sec):
    print(f"Start washing machine for {sec} seconds")
    await asyncio.sleep(sec)
    print(f"End washing machine for {sec} seconds")
    await dryer(3)

async def dryer(sec):
    print(f"Start dryer for {sec} seconds")
    await asyncio.sleep(sec)
    print(f"End dryer for {sec} seconds")

async def dishwasher(sec):
    print(f"Start dishwasher for {sec} seconds")
    await asyncio.sleep(sec)
    print(f"End dishwasher for {sec} seconds")

async def clean_potatoes(pieces):
    print(f"Start cleaning potatoes for {pieces} pieces")
    for ix in range(pieces):
        print(f"Cleaning potato {ix}")
        time.sleep(0.5)
        #await asyncio.sleep(0.0001)
    print(f"End cleaning potatoes for {pieces} pieces")

async def main():
    await asyncio.gather(dishwasher(3),
washing_machine(3), boil_water(4), clean_pot\
atoes(14))

start = time.time()
```

```
asyncio.run(main())
end = time.time()
print(f"Elapsed {end-start}")
```

From the output you can see that we noticed that the washing machine has finished only after we
have finished all the potatoes. That’s becasue our potato cleaning process was a long-running
CPU-intensive process. This means the dryer only starts working after the potatoes are clean.

```
Start dishwasher for 3 seconds
Start washing machine for 3 seconds
Start boiling water for 4 seconds
Start cleaning potatoes for 14 pieces
Cleaning potato 0
Cleaning potato 1
Cleaning potato 2
Cleaning potato 3
Cleaning potato 4
Cleaning potato 5
Cleaning potato 6
Cleaning potato 7
Cleaning potato 8
Cleaning potato 9
Cleaning potato 10
Cleaning potato 11
Cleaning potato 12
Cleaning potato 13
End cleaning potatoes for 14 pieces
End dishwasher for 3 seconds
End washing machine for 3 seconds
Start dryer for 3 seconds
End boiling water for 4 seconds
End dryer for 3 seconds
Elapsed 10.01340126991272
```

If after cleaning each potato we look up for a fraction of a second, if we let the main loop run,

then we can notice that the washing machine has ended and we can turn on the dryer before continuing
with the next potato. This will allow the dryer to work while we are still cleaning the potatoes.

```
Start dishwasher for 3 seconds
Start washing machine for 3 seconds
Start boiling water for 4 seconds
Start cleaning potatoes for 14 pieces
Cleaning potato 0
Cleaning potato 1
Cleaning potato 2
Cleaning potato 3
Cleaning potato 4
Cleaning potato 5
End dishwasher for 3 seconds
End washing machine for 3 seconds
Start dryer for 3 seconds
Cleaning potato 6
Cleaning potato 7
End boiling water for 4 seconds
Cleaning potato 8
Cleaning potato 9
Cleaning potato 10
Cleaning potato 11
End dryer for 3 seconds
Cleaning potato 12
Cleaning potato 13
End cleaning potatoes for 14 pieces
Elapsed 7.02296781539917
```

Explanation

- Single thread
- Single process
- The feeling of parallelism
- Coroutines

* async/await

* event loop

* Cooperative Multitasking

- Asynchronous
- non-blocking or synchronous vs blocking (aka “normal”)

Coroutines

* Functions that can be suspended mid-way and allow other functions to run (a generator)

- `async def` is a native coroutine or asynchronous generator
- `async with`
- `async for`

More about asyncio

- AsyncIO in Real Python
- asyncio
- aiohttp

Async files

```
import aiohttp
import asyncio

async def fetch(session, url):
    async with session.get(url) as response:
        return await response.text()

async def main():
    async with aiohttp.ClientSession() as session:
        html = await fetch(session, 'http://python.org')
        print(html)
        print("OK")
```

```python

asyncio.run(main())
```

```python
import aiofiles
```

Asynchronus programming with Twisted

About Twisted

- [Twisted]()

Echo

```python
from twisted.internet import protocol,reactor

port = 8000

class Echo(protocol.Protocol):
    def dataReceived(self, data):
        text = data.decode('utf8')
        print(f"Received: {text}")
        self.transport.write("You said: {}".format(text).encode('utf8'))

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

print(f"Listening on port {port}")
reactor.listenTCP(port, EchoFactory())
reactor.run()
```

```python
from twisted.internet import reactor,protocol
import sys

if len(sys.argv) < 2:
    exit("Usage: {sys.argv[0]} TEXT")

message = sys.argv[1]
```

```python
port = 8000

class EchoClient(protocol.Protocol):
    def connectionMade(self):
        self.transport.write(message.encode('utf8'))

    def dataReceived(self, data):
        print(f"Server said: {data}")
        self.transport.loseConnection()

class EchoFactory(protocol.ClientFactory):
    def buildProtocol(self, addr):
        return EchoClient()

    def clientConnectionFailed(self, connector, reason):
        print("connection failed")
        reactor.stop()

    def clientConnectionLost(self, connector, reason):
        print("connection lost")
        reactor.stop()

reactor.connectTCP("localhost", port, EchoFactory())
reactor.run()
```

Echo with log

```python
from twisted.internet import protocol,reactor

port = 8000

class Echo(protocol.Protocol):
    def dataReceived(self, data):
        print("Received: {}".format(data))
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        print(f"Contection established with {addr}")
        return Echo()

print(f"Started to listen on port {port}")
```

```
reactor.listenTCP(port, EchoFactory())
reactor.run()
```

Simple web client

The code behind this example was deprecated. Need to be fixed.

- getPage() returns a “deferred”
- addCallbacks(on_success, on_failure)
- addBoth(on_both) adds callbock to both success and failure callback chain

---

```
from twisted.internet import reactor
from twisted.web.client import getPage
import sys

def printPage(result):
    print("Page")
    print('Size of the returned page is {}'.format(len(result)))

def printError(error):
    print("Error")
    print(f"Error: {error}")
    #sys.stderr.write(error)

def stop(result):
    print('stop')
    reactor.stop()

if (len(sys.argv) != 2):
    sys.stderr.write("Usage: python " + sys.argv[0] + " <URL>\n")
    exit(1)

d = getPage(sys.argv[1])
d.addCallbacks(printPage, printError)
d.addBoth(stop)

reactor.run()

```

```
# getPage(sys.argv[1], method='POST', postdata="My test data").
```

Web client

```
from twisted.internet import reactor
from twisted.web.client import getPage
import sys
import re
import time

queue = [
  'http://docs.python.org/3/',
  'http://docs.python.org/3/whatsnew/3.3.html',
  'http://docs.python.org/3/tutorial/index.html',
  'http://docs.python.org/3/library/index.html',
  'http://docs.python.org/3/reference/index.html'
  'http://docs.python.org/3/howto/index.html',
  'http://docs.python.org/3/howto/pyporting.html',
  'http://docs.python.org/3/howto/cporting.html',
  'http://docs.python.org/3/howto/curses.html',
  'http://docs.python.org/3/howto/descriptor.html',
  'http://docs.python.org/3/howto/functional.html',
  'http://docs.python.org/3/howto/logging.html',
  'http://docs.python.org/3/howto/logging-cookbook.html',
  'http://docs.python.org/3/howto/regex.html',
  'http://docs.python.org/3/howto/sockets.html',
  'http://docs.python.org/3/howto/sorting.html',
  'http://docs.python.org/3/howto/unicode.html',
  'http://docs.python.org/3/howto/urllib2.html',
  'http://docs.python.org/3/howto/webservers.html',
  'http://docs.python.org/3/howto/argparse.html',
  'http://docs.python.org/3/howto/ipaddress.html',
]

max_parallel = 3
current_parallel = 0
if len(sys.argv) == 2:
  max_parallel = int(sys.argv[1])

def printPage(result):
  print("page size: ", len(result))
  global current_parallel
```

```
  current_parallel -= 1
  print("current_parallel: ", current_parallel)
  #urls = re.findall(r'href="([^"]+)"', result)
  #for u in urls:
  #  queue.append(u)
  #queue.extend(urls)
  process_queue()

def printError(error):
  print("Error: ", error)
  global current_parallel
  current_parallel -= 1
  process_queue()


def stop(result):
  reactor.stop()

def process_queue():
  global current_parallel, max_parallel,queue
  print("process_queue cs: {} max: {}".format(current_parallel, max_parallel))
  while True:
    if current_parallel >= max_parallel:
      print("No empty slot")
      return
    if len(queue) == 0:
      print("queue is empty")
      if current_parallel == 0:
        reactor.stop()
      return
    url = queue[0] + '?' + str(time.time())
    queue[0:1] = []
    current_parallel += 1
    d = getPage(url)
    d.addCallbacks(printPage, printError)

process_queue()
reactor.run()
print("----done ---")
```

Multiprocess

Multiprocess CPU count

- multiprocessing

```
import multiprocessing as mp
print(mp.cpu_count())
```

Multiprocess Process

```
import multiprocessing as mp
print(mp.cpu_count())
```

Multiprocess N files: Pool

Analyze N files in parallel.

```
from multiprocessing import Pool
import os
import sys
import re

def analyze(filename):
    print("Process {:>5} analyzing {}".format(os.getpid(), filename))
    digits = 0
    spaces = 0
    total  = 0
    with open(filename) as fh:
        for line in fh:
            for char in line:
                total += 1
                if re.search(r'^\d$', char):
                    digits += 1
```

```python
                if char == ' ':
                    spaces += 1
    return {
        'filename': filename,
        'total': total,
        'digits': digits,
        'spaces': spaces,
    }

def main():
    if len(sys.argv) < 3:
        exit("Usage: {} POOL_SIZE FILEs")
    size  = int(sys.argv[1])
    files = sys.argv[2:]

    with Pool(size) as p:
        results = p.map(analyze, files)
    for res in results:
        print(res)

if __name__ == '__main__':
    main()
```

```
$ python multiprocess_files.py 3 multiprocess_*

Process 22688 analyzing multiprocess_files.py
Process 22689 analyzing multiprocess_load.py
Process 22690 analyzing multiprocess_pool_async.py
Process 22688 analyzing multiprocess_pool.py
{'filename': 'multiprocess_files.py', 'total': 833, 'digits': 10, 'spaces': 275}
{'filename': 'multiprocess_load.py', 'total': 694, 'digits': 14, 'spaces': 163}
{'filename': 'multiprocess_pool_async.py', 'total': 695, 'digits': 8, 'spaces': 161}
{'filename': 'multiprocess_pool.py', 'total': 397, 'digits': 3, 'spaces': 80}
```

We asked it to use 3 processes, so looking at the process ID you can see one of them worked twice.

The returned results can be any Python datastructure. A dictionary is usually a good idea.

Multiprocess load

```python
import random
import multiprocessing
import time
import sys
# Works only in Python 3

def calc(n):
    count = 0
    total = 0
    while count < 40000000 / n:
        rnd = random.random()
        total += rnd
        count += 1
    return {'count': count, 'total': total}

def main():
    if len(sys.argv) != 2:
        exit("Usage: {} POOL_SIZE")

    start = time.time()
    size = int(sys.argv[1])
    with multiprocessing.Pool(size) as pool:
        results = pool.map(calc, [size] *  size)
        print("Results: {}".format(results))
        totals = map(lambda r: r['total'], results)
        print("Total: {}".format(sum(totals)))
    end = time.time()
    print(end - start)

if __name__ == '__main__':
    main()
```

Multiprocess: Pool

`Pool(3)` creates 3 child-processes and let's them compute the values. `map`
returns the results in the same order as the input came in.

```
from multiprocessing import Pool
import os
import sys

def f(x):
    print("Input {} in process {}".format(x,
os.getpid()))
    #print(x)
    return x*x

def main():
    if len(sys.argv) != 3:
        exit("Usage: {} NUMBERS POOL_SIZE")
    numbers = int(sys.argv[1])
    size    = int(sys.argv[2])

    with Pool(size) as p:
        results = p.map(f, range(numbers))
    print(results)

if __name__ == '__main__':
    main()
```

```
python multiprocess_pool.py  11 3
python multiprocess_pool.py  100 5
```

Multiprocess load async

```
from multiprocessing import Pool
import os


def f(x):
    print("Input {} in process {}".format(x,
os.getpid()))
    return x*x

```

```python
def prt(z):
    print(z)

def main():
    with Pool(5) as p:
        results = p.imap(f, range(11)) # <multiprocessing.pool.IMapIterator object
        print(results)
        print('--')
        for r in results:
            print(r)

        #results = p.map_async(f, range(11))   # <multiprocessing.pool.MapResult obje\
ct>, not iterable

        #results = []
        #p.map_async(f, range(11))   # <multiprocessing.pool.MapResult object>, not i\
terable
        #print(results)
        #for r in results:
        #    print(r)


if __name__ == '__main__':
    main()
```

Multiprocess and logging

Tested on Windows

```python
from multiprocessing import Pool
import os
import logging
import logging.handlers

count = 0
def f(x):
   global count
   count += 1
   #print("Input {} in process {}".format(x, os.getpid()))
```

```
    logger = logging.getLogger("app")
    logger.info("f({}) count {} in PID {}".format(x, count, os.getpid()))
    return x*x


def prt(z):
    print(z)

def setup_logger():
    level = logging.DEBUG
    logger = logging.getLogger("app")
    logger.setLevel(level)
    log_file = 'try.log'
    formatter = logging.Formatter('%(asctime)s - %(levelname)-8s - %(filename)-20s:%(\
lineno)-5d - %(funcName)-22s - %(message)s')
    ch = logging.FileHandler(log_file)
    #ch = logging.handlers.TimedRotatingFileHandler(log_file, when='D', backupCount=2)
    ch.setLevel(level)
    ch.setFormatter(formatter)
    logger.addHandler(ch)
    logger.info("Setup logger in PID {}".format(os.getpid()))

def main():
    logger = logging.getLogger('app')
    logger.info("main")

    with Pool(5) as p:
        results = p.imap(f, range(110)) # <multiprocessing.pool.IMapIterator object
        print(results)
        print('--')
        for r in results:
            print(r)

setup_logger()
if __name__ == '__main__':
    main()
```

Exercise: Process N files in parallel

Create N=100 files 1.txt - N.txt
In each file put L random strings of up to X characters

Write a script that will read all the files for each file and count how many times each digit appears. Then provide a combined report.
First write the script in a single process way.
Then convert it to be able to work with multiprocess.

Exercise: Process N Excel files in parallel

- Create N Excel files with random 10 random numbers in the first row of each file.
- Write a process that reads the N Excel files and sums up the numbers in each one of them and then sums up the numbers of all the files.

Exercise: Fetch URLs in parallel

- top-websites
- Given a file with a list of URLs, collect the title of each site.

```
https://google.com/
https://youtube.com/
https://facebook.com/
https://baidu.com/
https://twitter.com/
https://instagram.com/
https://wikipedia.com/
https://www.amazon.com/
https://yahoo.com/
https://yandex.ru/
https://vk.com/
https://live.com/
https://naver.com/
```

```
https://yahoo.co.jp/
https://google.com.br/
https://netflix.com/
https://reddit.com/
https://ok.ru/
https://mail.ru/
https://ebay.com/
https://linkedin.com/
https://qq.com/
https://pinterest.com/
https://bing.com/
https://whatsapp.com/
https://office.com/
https://amazon.de/
https://aliexpress.com/
https://amazon.co.jp/
https://msn.com/
https://google.de/
https://paypal.com/
https://rakuten.co.jp/
https://amazon.co.uk/
https://daum.net/
https://google.co.jp/
https://taobao.com/
https://bilbili.com/
https://imdb.com/
https://booking.com/
https://roblox.com/
https://9apps.com/
https://globo.com/
https://duckduckgo.com/
https://www.nttdocomo.co.jp/
```

```
import time
import requests
import sys
from bs4 import BeautifulSoup

def get_urls(limit):
    with open('urls.txt') as fh:
        urls = list(map(lambda line: line.rstrip("\n"),
fh))
    if len(urls) > limit:
        urls = urls[:limit]
```

```python

    return urls

def get_title(url):
    try:
        resp = requests.get(url)
        if resp.status_code != 200:
            return None, f"Incorrect status_code {resp.status_code} for {url}"
    except Exception as err:
        return None, f"Error: {err} for {url}"

    soup = BeautifulSoup(resp.content, 'html.parser')
    return soup.title.string, None

def main():
    if len(sys.argv) < 2:
        exit(f"Usage: {sys.argv[0]} LIMIT")
    limit = int(sys.argv[1])
    urls = get_urls(limit)
    print(urls)
    start = time.time()

    titles = []
    for url in urls:
        #print(f"Processing {url}")
        title, err = get_title(url)
        if err:
            print(err)
        else:
            print(title)
        titles.append({
            "url": url,
            "title": title,
            "err": err,
        })
    end = time.time()
    print("Elapsed time: {} for {} pages.".format(end-start, len(urls)))
    print(titles)


if __name__ == '__main__':
    main()
```

Exercise: Fetch URLs from one site.

Download the sitemap or the other sitemap file
and fetch the first N URLs from there. Collecting the titles.

```python
import time
import requests
import xml.etree.ElementTree as ET
from bs4 import BeautifulSoup

def get_urls(content):
    urls = []
    root = ET.fromstring(content)
    for child in root:
        for ch in child:
            if ch.tag.endswith('loc'):
                urls.append(ch.text)
    #print(len(urls)) # 2653
    MAX = 20
    if len(urls) > MAX:
        urls = urls[:MAX]

    return urls

def main():
    start = time.time()
    url = 'https://code-maven.com/slides/sitemap.xml'
    resp = requests.get(url)
    if resp.status_code != 200:
        exit(f"Incorrect status_code {resp.status_code}")

    urls = get_urls(resp.content)

    titles = []
    for url in urls:
        resp = requests.get(url)
        if resp.status_code != 200:
            print(f"Incorrect status_code {resp.status_code} for {url}")
            continue

        soup = BeautifulSoup(resp.content, 'html.parser')
        print(soup.title.string)
        titles.append(soup.title.string)
```

```python
    end = time.time()
    print("Elapsed time: {} for {} pages.".format(end-start, len(urls)))
    print(titles)


if __name__ == '__main__':
    main()
```

Solution: Fetch URLs in parallel

- First create function and use regular map.
- Deal with encoding.
- Replace continue by return, include None in results.
- It has some 2 sec overhead, but then 20 items reduced from 18 sec to 7 sec using pool of 5.

```python
import time
import requests
import xml.etree.ElementTree as ET
from bs4 import BeautifulSoup
from multiprocessing import Pool
import os


def get_urls(content):
    urls = []
    root = ET.fromstring(content)
    for child in root:
        for ch in child:
            if ch.tag.endswith('loc'):
                urls.append(ch.text)

    #print(len(urls)) # 2653
    MAX = 20
    if len(urls) > MAX:
        urls = urls[:MAX]

    return urls

def get_title(url):
```

```
    resp = requests.get(url)
    if resp.status_code != 200:
        print(f"Incorrect status_code {resp.status_code} for {url}")
        return

    soup = BeautifulSoup(resp.content, 'html.parser')
    print(soup.title.string)
    return soup.title.string.encode('utf-8')


def main():
    start = time.time()
    url = 'https://code-maven.com/slides/sitemap.xml'
    resp = requests.get(url)
    if resp.status_code != 200:
        exit(f"Incorrect status_code {resp.status_code}")

    urls = get_urls(resp.content)

    titles = []
#     for url in urls:
#         titles.append(get_title(url))
#     titles = list(map(get_title, urls))
    with Pool(5) as pool:
        results = pool.map(get_title, urls)
    for r in results:
        titles.append(r)
    end = time.time()
    print("Elapsed time: {} for {} pages.".format(end-start, len(urls)))
    print(list(titles))
    print("DONE")


if __name__ == '__main__':
   main()
```

Multitasking

What is Multitasking?

- Multitasking
- A wrapper around threading and os.fork by Ran Aroussi

```
pip install multitasking
```

Multitasking example

```python
import multitasking
import time
import random

multitasking.set_max_threads(2)

@multitasking.task
def work(ix, sec):
    print(f"Start {ix} sleeping for {sec}s")
    time.sleep(sec)
    print(f"Finish {ix}")

if __name__ == "__main__":
    tasks = (6, 0.7, 0.8, 0.3, 0.4, 3, 0.1)
    for ix, sec in enumerate(tasks):
        work(ix+1, sec)

    print("do some work after all the jobs are done")
```

```
Start 1 sleeping for 6s
Start 2 sleeping for 0.7s
do some work after all the jobs are done
Finish 2
Start 3 sleeping for 0.8s
```

```
Finish 3
Start 4 sleeping for 0.3s
Finish 4
Start 5 sleeping for 0.4s
Finish 5
Start 6 sleeping for 3s
Finish 6
Start 7 sleeping for 0.1s
Finish 7
Finish 1
```

Multitasking example with wait

```python
import multitasking
import time
import random

multitasking.set_max_threads(2)

@multitasking.task
def work(ix, sec):
    print(f"Start {ix} sleeping for {sec}s")
    time.sleep(sec)
    print(f"Finish {ix}")

if __name__ == "__main__":
    tasks = (6, 0.7, 0.8, 0.3, 0.4, 3, 0.1)
    for ix, sec in enumerate(tasks):
        work(ix+1, sec)
    multitasking.wait_for_tasks()

    print("do some work after all the jobs are done")
```

```
Start 1 sleeping for 6s
Start 2 sleeping for 0.7s
Finish 2
Start 3 sleeping for 0.8s
Finish 3
Start 4 sleeping for 0.3s
Finish 4
Start 5 sleeping for 0.4s
Finish 5
```

```
Start 6 sleeping for 3s
Finish 6
Start 7 sleeping for 0.1s
Finish 7
Finish 1
do some work after all the jobs are done
```

Multitaksing - second loop waits for first one

```python
import multitasking
import time
import random

@multitasking.task
def first(count):
    sleep = random.randint(1,10)/2
    if count == 10:
        sleep = 10
    print("Start First {} (sleeping for {}s)".format(count, sleep))
    time.sleep(sleep)
    print("finish First {} (after for {}s)".format(count, sleep))

@multitasking.task
def second(count):
    sleep = random.randint(1,10)/2
    print("Start Second {} (sleeping for {}s)".format(count, sleep))
    time.sleep(sleep)
    print("finish Second {} (after for {}s)".format(count, sleep))

if __name__ == "__main__":
    for i in range(0, 10):
        first(i+1)
    multitasking.wait_for_tasks()
    print('first done')

    for i in range(0, 10):
        second(i+1)

```

```python
    multitasking.wait_for_tasks()
    print('second done')
```

Multitasking counter

```python
import multitasking
import time


multitasking.set_max_threads(10)
counter = 0


@multitasking.task
def count(n):
    global counter
    for _ in range(n):
        counter += 1


if __name__ == "__main__":
    start = time.time()
    k = 10
    n = 1000000
    for _ in range(k):
        count(n)
    multitasking.wait_for_tasks()
    end = time.time()
    expected = k * n
    print(f'done actual: {counter} expected: {expected}. Missing: {expected-counter}\
')
    print(f'Elapsed time {end-start}')
```

```
done actual: 3198547 expected: 10000000. Missing: 6801453
Elapsed time 0.5210244655609131
```

Multitasking counter with thread locking

```python
import multitasking
import time
import threading


multitasking.set_max_threads(10)
counter = 0


locker = threading.Lock()


@multitasking.task
def count(n):
    global counter
    for _ in range(n):
        locker.acquire()
        counter += 1
        locker.release()


if __name__ == "__main__":
    start = time.time()
    k = 10
    n = 1000000
    for _ in range(k):
        count(n)
    multitasking.wait_for_tasks()
    end = time.time()
    expected = k * n
    print(f'done actual: {counter} expected: {expected}. Missing: {expected-counter}\
')
    print(f'Elapsed time {end-start}')
```

```
done actual: 10000000 expected: 10000000. Missing: 0
Elapsed time 37.231414556503296
```

Improving Performance - Optimizing code

Problems

- Speed
- Memory usage
- I/O (disk, network, database)

Optimization strategy

The 3 rules of optimization

- Don’t do it!
- Don’t do it!
- Don’t do it yet!

Premature optimization is the root of all evil ~ Donald Knuth

Locate the source of the problem

- I/O is expensive! Database access, file access, GUI update
- If memory is full swapping starts - speed decreases

Optimizing tactics

- Choose the Right Data Structure (Dictionary?, Set?, List?)

- Sorting: Decorate Sort Undecorate (DSU) aka. Schwartzian Transform.
- String Concatenation: avoid extensive concatenation.
- Loops: for, list comprehension: use generators and iterators.
- Delay expanding range, map, filter, etc. iterables.
- Caching results, memoizing.

Read more performance tips

DSU: Decorate Sort Undecorate

In Perl it is called Schwartzian transform

```
animals = ['chicken', 'cow', 'snail', 'elephant']
print(sorted(animals))
print(sorted(animals, key=len))

decorated = [(len(w), w) for w in animals]
print(decorated)

decorated.sort()
result = [ d[1] for d in decorated]
print(result)

# at once
print( [ d[1] for d in sorted( [(len(w), w) for w in animals] ) ] )
```

```
['chicken', 'cow', 'elephant', 'snail']
['cow', 'snail', 'chicken', 'elephant']
[(7, 'chicken'), (3, 'cow'), (5, 'snail'), (8, 'elephant')]
['cow', 'snail', 'chicken', 'elephant']
['cow', 'snail', 'chicken', 'elephant']
```

Profile code

Always profile before starting to optimize!

- [profile]

Slow example

> This code does some stuff which was deemed to be "too slow" by some client.
> The actual content is not that interesting.

```
import random

def f():
    n = 0
    for i in range(30):
        n += random.random()
    return n

def g():
    return random.random() * 30


def main(n):
    text = get_str(n)

    #print(str)
    text_sorted = sort(text)
    return text_sorted

def sort(s):
    chars = list(s)
    for i in reversed(range(len(chars))):
        a = f()
        b = g()
        for j in range(i, len(chars)-1):
            swap(chars, j)

    return ''.join(chars)

def get_str(n):
    text = ''
```

```python
    for i in range(1, n):
        text += chr(65 + random.randrange(0, 26))
    return text

def swap(lst, loc):
    if lst[loc] > lst[loc + 1]:
        lst[loc], lst[loc + 1] = lst[loc + 1], lst[loc]

if __name__ == '__main__':
    print(main(1000))
```

profile slow code

```python
import slow
import profile

profile.run('slow.main(1000)')
```

```
         537471 function calls in 3.078 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      999    0.003    0.000    0.003    0.000 :0(chr)
        1    0.000    0.000    0.000    0.000 :0(join)
     1000    0.003    0.000    0.003    0.000 :0(len)
    31968    0.083    0.000    0.083    0.000 :0(random)
     1999    0.009    0.000    0.009    0.000 :0(range)
        1    0.001    0.001    0.001    0.001 :0(setprofile)
        1    0.000    0.000    3.076    3.076 <string>:1(<module>)
        0    0.000             0.000          profile:0(profiler)
        1    0.000    0.000    3.078    3.078 profile:0(slow.main(1000))
      999    0.009    0.000    0.012    0.000 random.py:173(randrange)
      999    0.005    0.000    0.008    0.000 slow.py:10(g)
        1    0.000    0.000    3.076    3.076
```

```
slow.py:14(main)
          1    1.410    1.410    3.053    3.053
slow.py:21(sort)
          1    0.008    0.008    0.023    0.023
slow.py:31(get_str)
     498501    1.456    0.000    1.456    0.000
slow.py:37(swap)
        999    0.090    0.000    0.171    0.000
slow.py:4(f)
```

cProfile slow code

```python
import slow
import cProfile

cProfile.run('slow.main(1000)')
```

```
      537470 function calls in 0.325 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
     1    0.000    0.000    0.325    0.325
<string>:1(<module>)
   999    0.002    0.000    0.002    0.000
random.py:173(randrange)
   999    0.000    0.000    0.000    0.000 slow.py:10(g)
     1    0.000    0.000    0.325    0.325
slow.py:14(main)
     1    0.119    0.119    0.322    0.322
slow.py:21(sort)
     1    0.001    0.001    0.003    0.003
slow.py:31(get_str)
498501    0.189    0.000    0.189    0.000
slow.py:37(swap)
   999    0.008    0.000    0.010    0.000 slow.py:4(f)
   999    0.000    0.000    0.000    0.000 {chr}
  1000    0.000    0.000    0.000    0.000 {len}
     1    0.000    0.000    0.000    0.000 {method
'disable' of '_lsprof.Profiler' o\
bjects}
```

```
       1    0.000    0.000    0.000    0.000 {method 'join' of 'str' objects}
   31968    0.003    0.000    0.003    0.000 {method 'random' of '_random.Random' obje\
cts}
    1999    0.003    0.000    0.003    0.000 {range}
```

Benchmarking

- benchmark

```python
import timeit
from functools import reduce
import random

chars = []
for i in range(200):
    chars.append(chr(65 + random.randrange(0, 26)))

print(timeit.timeit('string = "".join(chars)',
    setup="from __main__ import chars", number=10000))

print(timeit.timeit('reduce(lambda x, y: x+y, chars)',
    setup="from __main__ import chars, reduce", number=10000))
```

```
0.01576369699614588
0.15464225399773568
```

Benchmarking subs

```python
import timeit

def one_by_one():
    import random
    text = ""
    for i in range(200):
        text += chr(65 + random.randrange(0, 26))
    return text

```

```python
def at_once():
    import random
    chars = []
    for i in range(200):
        chars.append(chr(65 + random.randrange(0, 26)))
    text = ''.join(chars)
    return text

print(timeit.timeit('one_by_one()',
    setup="from __main__ import one_by_one", number=10000))

print(timeit.timeit('at_once()',
    setup="from __main__ import at_once", number=10000))
```

```
1.5248507579963189
1.5566942970035598
```

Levenshtein distance

- editdistance Levenshtein distance implemented in C
- python-Levenshtein implemented in C
- pylev
- pyxdameraulevenshtein
- weighted-levenshtein

Generate words

```python
import sys
import random
import string

# TODO: set min, max word length
# TODO: set filename
# TODO: set character types
# TODO: allow spaces?

def main():
    filename = "words.txt"
```

```python
    min_len   = 6
    max_len   = 6

    if len(sys.argv) != 2:
        exit(f"Usage: {sys.argv[0]} WORD_COUNT")
    count = int(sys.argv[1])
    with open(filename, 'w') as fh:
        for _ in range(count):
            word = ''
            length = random.randrange(min_len, max_len+1)
            for _ in range(length):
                word +=
random.choice(string.ascii_lowercase)
            fh.write(word + "\n")

main()
```

Levenshtein - pylev

```python
import sys
import pylev

def main():
    if len(sys.argv) != 2:
        exit(f"Usage: {sys.argv[0]} filename")
    filename = sys.argv[1]
    outfile = 'out.txt'

    rows = []
    with open(filename) as fh:
        for row in fh:
            rows.append(row.rstrip("\n"))
    with open(outfile, 'w') as fh:
        for a in rows:
            for b in rows:
                dist = pylev.levenshtein(a, b)
                fh.write(f"{a},{b},{dist}\n")

main()
```

Levenshtein - edittidtance

```python
import sys
import editdistance

def main():
    if len(sys.argv) != 2:
        exit(f"Usage: {sys.argv[0]} filename")
    filename = sys.argv[1]
    outfile = 'out.txt'

    rows = []
    with open(filename) as fh:
        for row in fh:
            rows.append(row.rstrip("\n"))
    with open(outfile, 'w') as fh:
        for a in rows:
            for b in rows:
                dist = editdistance.eval(a, b)
                fh.write(f"{a},{b},{dist}\n")

main()
```

Editdistance benchmark

- editdistance

A Tool to Generate text files

```python
import sys
import string
import random
import argparse
import os

# Generate n file of size S with random letters

def get_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--dir',
help="Directory where to create the fil\
es", default=".")
    parser.add_argument('--files', type=int, help="Number
```

```
of files to create", defau\
lt=1)
    parser.add_argument('--size',  type=int, help="Size
of files",             defau\
lt=10)
    args = parser.parse_args()
    return args

def main():
    args = get_args()
    chars = list(string.ascii_lowercase) + [' '] * 5 +
['\n']

    for ix in range(args.files):
        all_chars = []
        for _ in range(args.size):
            all_chars.extend(random.sample(chars, 1))
        #print(len(all_chars))

        #print(all_chars)
        filename = os.path.join(args.dir, str(ix) +
'.txt')
        with open(filename, 'w') as fh:
            fh.write(''.join(all_chars))


def old_main():
    if len(sys.argv) < 2:
        exit(f"Usage: {sys.argv[0]} NUMBER_OF_ROWS")

    row_count = int(sys.argv[1])
    min_width = 30
    max_width = 50
    filename  = 'data.log'

    chars = list(string.ascii_lowercase) + [' '] * 5
    all_chars = chars * max_width

    with open(filename, 'w') as fh:
        for i in range(row_count):
            width = random.randrange(min_width,
max_width+1)
            row = ''.join(random.sample(all_chars,
width))
            fh.write(row + "\n")
```

```
main()
```

Count characters

```
# changes chars and counter
def add_char(chars, counter, ch, cnt=1):
    for ix in range(len(chars)):
        if chars[ix] == ch:
            counter[ix] += cnt
            break
    else:
        chars.append(ch)
        counter.append(cnt)


def count_in_file(filename):
    #print(filename)
    chars   = []
    counter = []
    with open(filename) as fh:
        for row in fh:
            for ch in row:
                #print(ch)
                if ch == ' ':
                    continue
                if ch == '\n':
                    continue
                add_char(chars, counter, ch)

    #print(chars)
    #print(counter)
    return chars, counter

def merge(chars1, counter1, chars2, counter2):
    chars = []
    counter = []
    for ix in range(len(chars1)):
        add_char(chars, counter, chars1[ix], cnt=counter1[ix])
    for ix in range(len(chars2)):
        add_char(chars, counter, chars2[ix], cnt=counter2[ix])
```

```python
    return chars, counter


def print_results(chars, counter):
    print("Results")
    for ix in range(len(chars)):
        print("{}  {}".format(chars[ix], counter[ix]))

def count_in(filenames):
    total_chars = []
    total_counter = []
    for filename in filenames:
        chars, counter = count_in_file(filename)
        total_chars, total_counter = merge(total_chars,
total_counter, chars, counte\
r)

    return total_chars, total_counter


if __name__ == '__main__':
    import sys
    chars, counter = count_in(sys.argv[1:])
    print_results(chars, counter)
```

```python
import count_characters as count
import cProfile
import sys

cProfile.run('chars, counter =
count.count_in(sys.argv[1:])')
```

Memory leak

```python
import random

def alloc():
    a = {
        'data': str(random.random()) + "a" * 10000000,
    }
    b = {
        'data': str(random.random()) + "b" * 10000000,
```

```
    }
    a['other'] = b
    b['other'] = a
```

```
import sys
from mymem import alloc

if len(sys.argv) < 2:
    exit(f"Usage: {sys.argv[0]} N")

count = int(sys.argv[1])

for _ in range(count):
    alloc()
input("End the script")
```

Garbage collection

- gc

```
import sys
from mymem import alloc
import gc

if len(sys.argv) < 2:
    exit(f"Usage: {sys.argv[0]} N")

count = int(sys.argv[1])

for _ in range(count):
    alloc()
input("Run gc")

gc.collect()
input("End the script")
```

Weak reference

- weakref

```python
import random
import weakref

def alloc():
    a = {
        'data': str(random.random()) + "a" * 10000000,
    }
    b = {
        'data': str(random.random()) + "b" * 10000000,
    }
    #a['other'] = weakref.WeakKeyDictionary(b)
    z = weakref.ref(b)
    #a['other'] =
    #weakref.ref(a['other'])
    #b['other'] = a
    #weakref.ref(b['other'])
```

```python
import sys
from weakmymem import alloc

if len(sys.argv) < 2:
    exit(f"Usage: {sys.argv[0]} N")

count = int(sys.argv[1])

for _ in range(count):
    alloc()
input("End the script")
```

Exercise: benchmark list-comprehension, map, for

- Create several functions that accept a list of numbers from 1 to 1000 and calculate their square:
- A function with a `for`-loop.
- A function that uses `map`.
- A function that uses list-comprehension.

* Feel free to have any other calucaltion and measure that.
* Send me the code and the results!

Exercise: Benchmark Levenshtein

* Take the implementation of the Levenshtein distance calculations and check which one is faster.

Exercise: sort files

Write a script that given a path to a directory will print the files sorted by date.
If you don't have one large folder, then use `os.walk` to get the path to the files of a whole directory tree.

* Write a simple solution.
* Profile.
* Use [DSU](https://en.wikipedia.org/wiki/Schwartzian_transform).

Exercise: compare split words:

We have three ways of splitting a string into words. Using `split`, using `re.split` and by going over it character-by-charcter.
Which one is the fastest?

```python
import sys
import re

def split_to_words_by_regex(text):
    return re.split(' ', text)

def split_to_words_by_split(text):
    return text.split()

def split_to_words_by_chars(text):
    words = []
```

```python
    word = ''
    for ch in text:
        if ch == ' ':
            words.append(word)
            word = ''
        else:
            word += ch
    if word:
        words.append(word)
    return words


if __name__ == '__main__':
    if len(sys.argv) < 2:
        exit(f"Usage: {sys.argv[0]} FILENAME")

    filename = sys.argv[1]
    with open(filename) as fh:
        text = fh.read()
    res1 = split_to_words_by_split(text)
    res2 = split_to_words_by_chars(text)
    res3 = split_to_words_by_regex(text)
    #print(res1)
    #print(res2)
    assert res1 == res2
    assert res1 == res3
```

Exercise: count words

Given a file count how many times each word appears.
Have two implementations. One using two list and one using a dictionary.
Profile the code and benchmark the two solutions.

See `examples/lists/count_words_two_lists.py` and `examples/dictionary/count_words.py`

GUI with Python/Tk

Sample Tk app

```python
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import os


def scary_action():
    messagebox.showerror(title="Scary", message="Deleting hard disk. Please wait...")


def run_code():
    text = ""
    text += "Name: {}\n".format(name.get())
    text += "Password: {}\n".format(password.get())
    text += "Animal: {}\n".format(animal.get())
    text += "Country: {}\n".format(country.get())
    text += "Colors: "
    for ix in range(len(colors)):
        if colors[ix].get():
            text += color_names[ix] + " "
    text += "\n"

    selected = list_box.curselection()  # returns a tuple
    text += "Animals: "
    text += ', '.join([list_box.get(idx) for idx in selected])
    text += "\n"

    text += "Filename: {}\n".format(os.path.basename(filename_entry.get()))

    resp = messagebox.askquestion(title="Running with", message=f"Shall I start runn\
ing with the following values?\n\n{text}")
```

```
    if resp == 'yes':
        output_window['state'] = 'normal'  # allow editing of the Text widget
        output_window.insert('end', f"{text}\n--------\n")
        output_window['state'] = 'disabled'  # disable editing
        output_window.see('end')  # scroll to the end as we make progress
        app.update()


def close_app():
    app.destroy()


app = tk.Tk()
app.title('Simple App')

menubar = tk.Menu(app)
app.config(menu=menubar)

menu1 = tk.Menu(menubar, tearoff=0)
menubar.add_cascade(label="File", underline=0, menu=menu1)
menu1.add_separator()
menu1.add_command(label="Exit", underline=1, command=close_app)

top_frame = tk.Frame(app)
top_frame.pack(side="top")
pw_frame = tk.Frame(app)
pw_frame.pack(side="top")

# Simple Label widget:
name_title = tk.Label(top_frame, text=" Name:", width=10, anchor="w")
name_title.pack({"side": "left"})

# Simple Entry widget:
name = tk.Entry(top_frame)
name.pack({"side": "left"})
# name.insert(0, "Your name")

# Simple Label widget:
```

```python
password_title = tk.Label(pw_frame, text=" Password:", width=10, anchor="w")
password_title.pack({"side": "left"})

# In order to hide the text as it is typed (e.g. for Passwords)
# set the "show" parameter:
password = tk.Entry(pw_frame)
password["show"] = "*"
password.pack({"side": "left"})

radios = tk.Frame(app)
radios.pack()
animal = tk.StringVar()
animal.set("Red")
my_radio = []
animals = ["Cow", "Mouse", "Dog", "Car", "Snake"]
for animal_name in animals:
    radio = tk.Radiobutton(radios, text=animal_name, variable=animal, value=animal_n\
ame)
    radio.pack({"side": "left"})
    my_radio.append(radio)


checkboxes = tk.Frame(app)
checkboxes.pack()
colors = []
my_checkbox = []
color_names = ["Red", "Blue", "Green"]
for color_name in color_names:
    color_var = tk.BooleanVar()
    colors.append(color_var)
    checkbox = tk.Checkbutton(checkboxes, text=color_name, variable=color_var)
    checkbox.pack({"side": "left"})
    my_checkbox.append(checkbox)

countries = ["Japan", "Korea", "Vietnam", "China"]

def country_change(event):
    pass
    #selection = country.current()
    #print(selection)
    #print(countries[selection])
```

```
def country_clicked():
    pass
    #print(country.get())

country = ttk.Combobox(app,  values=countries)
country.pack()
country.bind("<<ComboboxSelected>>", country_change)




list_box = tk.Listbox(app, selectmode=tk.MULTIPLE, height=4)
animal_names = ['Snake', 'Mouse', 'Elephant', 'Dog', 'Cat', 'Zebra', 'Camel', 'Spide\
r']
for val in animal_names:
    list_box.insert(tk.END, val)
list_box.pack()

def open_filename_selector():
    file_path = filedialog.askopenfilename(filetypes=(("Any file", "*"),))
    filename_entry.delete(0, tk.END)
    filename_entry.insert(0, file_path)


filename_frame = tk.Frame(app)
filename_frame.pack()
filename_label = tk.Label(filename_frame, text="Filename:", width=10)
filename_label.pack({"side": "left"})
filename_entry = tk.Entry(filename_frame, width=60)
filename_entry.pack({"side": "left"})
filename_button = tk.Button(filename_frame, text="Select file", command=open_filenam\
e_selector)
filename_button.pack({"side": "left"})

output_frame = tk.Frame(app)
output_frame.pack()
output_window = tk.Text(output_frame, state='disabled')
output_window.pack()
```

```python

buttons = tk.Frame(app)
buttons.pack()

scary_button = tk.Button(buttons, text="Don't click here!", fg="red", command=scary_\
action)
scary_button.pack({"side": "left"})

action_button = tk.Button(buttons, text="Run", command=run_code)
action_button.pack()

app.mainloop()

# TODO: key binding?
# TODO: Option Menu
# TODO: Scale
# TODO: Progressbar (after the deleting hard disk pop-up)
# TODO: Frame (with border?)
```

GUI Toolkits

When creating an application there are several ways to interact with the user. You can accept command line parameters.
You can interact on the Standard Output / Standard Input runnin in a Unix Shell or in the Command Prompt of Windows.

Many people, especially those who are using MS Windows, will frown upon both of those. They expect a Graphical User Interface (GUI)
or maybe a web interface via their browser. In this chapter we are going to look at the possibility to create a desktop GUI.

There are plenty of ways to create a GUI in Python. The major ones were listed here, but there are many more. See the additional links.

In this chapter we are going to use the Tk Toolkit.

- Tk
- GTK
- Qt
- wxWidgets

- GUI FAQ
- GUI Programming

Installation

Tk in Python is actually a wrapper arount the implementation in Tcl.

Tcl/Tk usually comes installed with Python. All we need is basically the Tkinter Python module.
In some Python installations (e.g. Anaconda), Tkinter is already installed. In other cases you might
need to install it yourself. For examples on Ubuntu you can use `apt` to install it.

```
sudo apt-get install python3-tk
```

Python Tk Documentation

The documentation of Tk in Python does not cover all the aspects of Tk. If you are creating a complex GUI application you might need to dig in the documentation written for Tcl/Tk.

- Tk
- The Tk Command of Tcl 8.6
- Python GUI Geeks for Geeks

In the Unix world where Tk came from the various parts of a GUI application are called widgets. In the MS Windows world they are usually called controls. There are several commonly used Widgets. For example, Label, Button, Entry, Radiobutton, Checkbox.
First we are going to see small examples with each one of these Widgets. Then we'll see how to combine them.

Python Tk Button

- Button

```
import tkinter as tk

app = tk.Tk()
app.title('Single Button')

button = tk.Button(app, text='Close', width=25, command=app.destroy)
button.pack()

app.mainloop()
```

Python Tk Button with action

```
import tkinter as tk

def run_action():
    print("clicked")

app = tk.Tk()
app.title('Single Button')

action_button = tk.Button(app, text='Action', width=25, command=run_action)
action_button.pack()
#action_button.pack(side="left")

exit_button = tk.Button(app, text='Close', width=25, command=app.destroy)
exit_button.pack()

app.mainloop()
```

Python Tk Label

- Label

```
import tkinter as tk

app = tk.Tk()
#app.title('Simple Label')

label = tk.Label(app, text='Some fixed text')
label.pack()

app.mainloop()
```

Python Tk Label - font size and color

```python
import tkinter as tk

app = tk.Tk()
app.title('Label with font')

label = tk.Label(app, text='Some text with larger letters')
label.pack()
label.config(font=("Courier", 44))
label.config(fg="#0000FF")
label.config(bg="yellow")

app.mainloop()
```

Python Tk Keybinding

```python
import tkinter as tk

app = tk.Tk()
app.title('Key binding')

label = tk.Label(app, text='Use the keyboard: (a, Ctr-b, Alt-c, F1, Alt-F4)')
label.config(font=("Courier", 44))
label.pack()

def pressed_a(event):
    print("pressed a")

def pressed_control_b(event):
    print("pressed Ctr-b")

def pressed_alt_c(event):
    print("pressed Alt-c")

def pressed_f1(event):
    print("pressed F1")

app.bind("<a>", pressed_a)
app.bind("<Control-b>", pressed_control_b)
app.bind("<Alt-c>", pressed_alt_c)
app.bind("<F1>", pressed_f1)

```

```python

app.mainloop()
```

Python Tk Entry (one-line text entry)

- Entry

```python
import tkinter as tk

app = tk.Tk()
app.title('Text Entry')

entry = tk.Entry(app)
entry.pack()

def clicked():
    print(entry.get())

button = tk.Button(app, text='Show', width=25,
command=clicked)
button.pack()

exit_button = tk.Button(app, text='Close', width=25,
command=app.destroy)
exit_button.pack()

app.mainloop()
```

Python Tk Entry for passwords and other secrets (hidden text)

```python
import tkinter as tk

app = tk.Tk()
app.title('Text Entry')

entry = tk.Entry(app)
entry['show'] = '*'
entry.pack()

```

```
def clicked():
    print(entry.get())

button = tk.Button(app, text='Show', width=25, command=clicked)
button.pack()

exit_button = tk.Button(app, text='Close', width=25, command=app.destroy)
exit_button.pack()

app.mainloop()
```

Python Tk Checkbox

```
import tkinter as tk

app = tk.Tk()
app.title('Checkbox')

var1 = tk.BooleanVar()
cb1 = tk.Checkbutton(app, text='male', variable=var1)
cb1.pack()

var2 = tk.BooleanVar()
cb2 = tk.Checkbutton(app, text='female', variable=var2)
cb2.pack()

def clicked():
    print(var1.get())
    print(var2.get())

button = tk.Button(app, text='Show', width=25, command=clicked)
button.pack()

exit_button = tk.Button(app, text='Close', width=25, command=app.destroy)
exit_button.pack()

app.mainloop()
```

- [Variables]

Python Tk Radiobutton

```
import tkinter as tk

def run_action():
    print("clicked")
    print(count.get())

app = tk.Tk()
app.title('Radio button')

count = tk.IntVar()
#count.set(2)

my_radios = []
values = [(1, "One"), (2, "Two"), (3, "Three")]
for ix in range(len(values)):
    my_radios.append(tk.Radiobutton(app, text=values[ix][1], variable=count, value=v\
alues[ix][0]))
    my_radios[ix].pack()

action_button = tk.Button(app, text='Action', width=25, command=run_action)
action_button.pack()

exit_button = tk.Button(app, text='Close', width=25, command=app.destroy)
exit_button.pack()

app.mainloop()
```

Python Tk Listbox

```
import tkinter as tk

app = tk.Tk()
app.title('List box')


```

```python
def clicked():
    print("clicked")
    selected = box.curselection()  # returns a tuple
    if selected:
        first = selected[0]
        color = box.get(first)
        print(color)

box = tk.Listbox(app)
values = ['Red', 'Green', 'Blue', 'Purple']
for val in values:
    box.insert(tk.END, val)
box.pack()

button = tk.Button(app, text='Show', width=25, command=clicked)
button.pack()

exit_button = tk.Button(app, text='Close', width=25, command=app.destroy)
exit_button.pack()

app.mainloop()
```

Python Tk Listbox Multiple

```python
import tkinter as tk

app = tk.Tk()
app.title('List box')


def clicked():
    print("clicked")
    selected = box.curselection()  # returns a tuple
    for idx in selected:
        print(box.get(idx))

box = tk.Listbox(app, selectmode=tk.MULTIPLE, height=4)
values = ['Red', 'Green', 'Blue', 'Purple', 'Yellow', 'Orange', 'Black', 'White']
for val in values:
    box.insert(tk.END, val)
```

```
box.pack()

button = tk.Button(app, text='Show', width=25,
command=clicked)
button.pack()

exit_button = tk.Button(app, text='Close', width=25,
command=app.destroy)
exit_button.pack()

app.mainloop()
```

Python Tk Menubar

- Menubar
- Menu

- `underline` sets the hot-key.
- `tearoff=` (the default) allows floating menu by clicking on the dashed line.
- enable/disable menu items.
- Set actions via `command` on the menu items.

```
import tkinter as tk

app = tk.Tk()
app.title('Menu')

def run_new():
    print("new")

def run_exit():
    print("exit")
    app.destroy()

def enable_languages():
    menu2.entryconfig("Klingon", state="normal")
def disable_languages():
    menu2.entryconfig("Klingon", state="disabled")

```

```python
def set_language(lang):
    print(lang)


menubar = tk.Menu(app)

menu1 = tk.Menu(menubar, tearoff=0)
menu1.add_command(label="New", command=run_new)
menu1.add_command(label="Enable language", command=enable_languages)
menu1.add_command(label="Disable language", command=disable_languages)
menu1.add_separator()
menu1.add_command(label="Exit", underline=1, command=run_exit)

menubar.add_cascade(label="File", underline=0, menu=menu1)

menu2 = tk.Menu(menubar, tearoff=1)
menu2.add_command(label="English")
menu2.add_command(label="Hebrew")
menu2.add_command(label="Spanish")
menu2.add_command(label="Klingon", state="disabled", command=lambda : set_language('\
Klingon'))
menu2.add_command(label="Hungarian")

menubar.add_cascade(label="Language", menu=menu2)

app.config(menu=menubar)

app.mainloop()
```

Python Tk Text

```python
import tkinter as tk

app = tk.Tk()
app.title('Text Editor')

text = tk.Text(app)
text.pack({"side": "bottom"})
```

```
app.mainloop()
```

- `text.delete(1.0, tk.END)`
- `text.insert('end', content)`
- `content = text.get(1.0, tk.END)`

- tk text

Python Tk Dialogs

- Dialogs
- Filedialogs
- Message boxes

Python Tk Filedialog

- file dialogs
- dialog

- askopenfilename - returns path to file
- asksaveasfilename - returns path to file
- askopenfile - returns filehandle opened for reading
- asksaveasfile - retutns filehandle opened for writing

- Allow the listing of file-extension filters.

```
import tkinter as tk
from tkinter import filedialog

input_file_path = None
output_file_path = None

def run_process():
    print("Parameters:")
```

```
    print(f"in: {input_file_path}")
    print(f"out: {output_file_path}")

def close_app():
    print("Bye")
    app.destroy()

def select_input_file():
    global input_file_path
    input_file_path = filedialog.askopenfilename(filetypes=(("Excel files", "*.xlsx"\
), ("CSV files", "*.csv"), ("Any file", "*")))
    print(input_file_path)

def select_output_file():
    global output_file_path
    output_file_path = filedialog.asksaveasfilename(filetypes=(("Excel files", "*.xl\
sx"), ("CSV files", "*.csv"), ("Any file", "*")))
    print(output_file_path)

app = tk.Tk()
app.title('Convert file')

input_button = tk.Button(app, text='Select input file', command=select_input_file)
input_button.pack()

output_button = tk.Button(app, text='Select output file', command=select_output_file)
output_button.pack()

process_button = tk.Button(app, text='Process', width=25, command=run_process)
process_button.pack()

exit_button = tk.Button(app, text='Close', width=25, command=close_app)
exit_button.pack()

app.mainloop()
```

Python Tk messagebox

```python
import tkinter as tk
from tkinter import messagebox

app = tk.Tk()
app.title('Menu')

def run_show_info():
    messagebox.showinfo(title = "Title", message = "Show info text")

def run_show_warning():
    messagebox.showwarning(title = "Title", message = "Show warning text")

def run_show_error():
    messagebox.showerror(title = "Title", message = "Show error text")

def run_ask_question():
    resp = messagebox.askquestion(title = "Title", message = "Can I ask you a questi\
on?")
    print(resp)   # "yes" / "no" (default "no")

def run_ask_okcancel():
    resp = messagebox.askokcancel(title = "Title", message = "Shall I do it?")
    print(resp)   # True / False (default = False)

def run_ask_retrycancel():
    resp = messagebox.askretrycancel(title = "Title", message = "Shall retry it?")
    print(resp)   # True / False (default = False)

def run_ask_yesno():
    resp = messagebox.askyesno(title = "Title", message = "Yes or No?")
    print(resp)   # True / False (default = False)

def run_ask_yesnocancel():
    resp = messagebox.askyesnocancel(title = "Title", message = "Yes, No, or Cancel?\
```

```python
")
    print(resp)   # True / False / None (default = None)

def run_exit():
    app.destroy()


menubar = tk.Menu(app)

menu1 = tk.Menu(menubar, tearoff=0)
menu1.add_command(label="Info",    underline=0, command=run_show_info)
menu1.add_command(label="Warning", underline=0, command=run_show_warning)
menu1.add_command(label="Error",   underline=0, command=run_show_error)
menu1.add_separator()
menu1.add_command(label="Exit", underline=1, command=run_exit)

menubar.add_cascade(label="Show", underline=0, menu=menu1)

menu2 = tk.Menu(menubar, tearoff=0)
menu2.add_command(label="Question", underline=0, command=run_ask_question)
menu2.add_command(label="OK Cancel", underline=0, command=run_ask_okcancel)
menu2.add_command(label="Retry Cancel", underline=0, command=run_ask_retrycanc\
el)
menu2.add_command(label="Yes or No", underline=0, command=run_ask_yesno)
menu2.add_command(label="Yes, No, or Cancel", underline=5, command=run_ask_yesnocanc\
el)

menubar.add_cascade(label="Ask", underline=0, menu=menu2)

app.config(menu=menubar)

app.mainloop()
```

- [Tk messagebox]()

Python Tk Combobox

```python
import tkinter as tk
from tkinter import ttk

countries = ["Japan", "Korea", "Vietnam", "China"]

app = tk.Tk()
app.title('Combo box')


def change(event):
    # VirtualEvent
    print("change")
    selection = country.current()
    print(selection)
    print(countries[selection])

def clicked():
    print("clicked")
    print(country.get())

country = ttk.Combobox(app, values=countries)
country.pack()
country.bind("<<ComboboxSelected>>", change)

button = tk.Button(app, text='Run', width=25, command=clicked)
button.pack()


app.mainloop()
```

Python Tk OptionMenu

```python
import tkinter as tk

def run_action():
    color = color_var.get()
    print(color)

    size = size_var.get()
```

```
    print(size)

app = tk.Tk()
app.title('Option Menu')

color_var = tk.StringVar(app)
color_selector = tk.OptionMenu(app, color_var, "Red", "Green", "Blue")
color_selector.pack()

sizes = ("Small", "Medium", "Large")
size_var = tk.StringVar(app)
size_selector = tk.OptionMenu(app, size_var, *sizes)
size_selector.pack()

action_button = tk.Button(app, text='Action', width=25, command=run_action)
action_button.pack()

app.mainloop()
```

Python Tk Scale

```
import tkinter as tk

def run_action():
    h = scale_h.get()
    print(h)

    v = scale_v.get()
    print(v)

app = tk.Tk()
app.title('Scale')

scale_h = tk.Scale(app, from_=0, to=42, orient=tk.HORIZONTAL)
scale_h.pack()

scale_v = tk.Scale(app, from_=1, to=100, orient=tk.VERTICAL)
scale_v.pack()
scale_v.set(23)
```

```
action_button = tk.Button(app, text='Action', width=25, command=run_action)
action_button.pack()

app.mainloop()
```

Python Tk Progressbar

```
import tkinter as tk
from tkinter import ttk

app = tk.Tk()
app.title('Single Button')

progressbar = ttk.Progressbar(app)
progressbar.pack()

def stop():
    progressbar.stop()

def start():
    app.after(10000, stop)
    progressbar.start(100)


button = tk.Button(app, text='Start', width=25, command=start)
button.pack()

exit_button = tk.Button(app, text='Close', width=25, command=app.destroy)
exit_button.pack()

app.mainloop()
```

Python Tk Frame

```
import tkinter as tk

def close():
```

```python
    app.destroy()

def clicked(val):
    entry.insert(tk.END, val)

app = tk.Tk()
app.title('Frame')

entry = tk.Entry(app)
entry.pack()

frames = {}
frames[1] = tk.Frame(app)
frames[1].pack(side="top")
frames[2] = tk.Frame(app)
frames[2].pack(side="top")
frames[3] = tk.Frame(app)
frames[3].pack(side="top")

btn = {}

btn["a"] = tk.Button(frames[1], text="a", width=25, command=lambda : clicked("a"))
btn["a"].pack(side="left")

btn["b"] = tk.Button(frames[1], text="b", width=25, command=lambda : clicked("b"))
btn["b"].pack(side="left")

btn["c"] = tk.Button(frames[2], text="c", width=25, command=lambda : clicked("c"))
btn["c"].pack(side="left")

btn["d"] = tk.Button(frames[2], text="d", width=25, command=lambda : clicked("d"))
btn["d"].pack(side="left")

close_btn = tk.Button(frames[3], text='Close', width=25, command=close)
close_btn.pack(side="right", expand=0)

app.mainloop()
```

- width

- side: left, right, top, bottom

Not so Simple Tk app with class

```python
from tkinter import Tk, Frame, BOTH


class Example(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent, background="white")
        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Simple")
        self.pack(fill=BOTH, expand=1)


def main():
    root = Tk()
    root.geometry("250x150+300+300")
    app = Example(parent=root)

    # move the window to the front (needed on Mac only?)
    root.lift()
    root.call('wm', 'attributes', '.', '-topmost', True)
    root.after_idle(root.call, 'wm', 'attributes', '.',
'-topmost', False)

    root.mainloop()

main()
```

Tk: Hello World

```python
import tkinter as tk

class Example(tk.Frame):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.pack()
        self.createWidgets()
```

```
    def createWidgets(self):
        # Simple Label widget:
        self.name_title = tk.Label(self, text="Hello World!")
        self.name_title.pack({"side": "left"})

def main():
    root = tk.Tk()
    app = Example(parent=root)
    app.mainloop()

main()
```

Tk: Quit button

```
import tkinter as tk

class Example(tk.Frame):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.pack()
        self.createWidgets()

    def createWidgets(self):
        self.QUIT = tk.Button(self)
        self.QUIT["text"] = "QUIT"
        self.QUIT["fg"]   = "red"
        self.QUIT["command"] =  self.quit
        self.QUIT.pack({"side": "left"})

def main():
    root = tk.Tk()
    app = Example(parent=root)

    app.mainloop()

main()
```

Tk: File selector

```python
import tkinter as tk
from tkinter import filedialog

class Example(tk.Frame):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.pack()
        self.createWidgets()

    def get_file(self):
        file_path = filedialog.askopenfilename()
        print(file_path)
        self.filename.delete(0, tk.END)
        self.filename.insert(0, file_path)

    def run_process(self):
        print("Running a process on file {}".format(self.filename.get()))

    def createWidgets(self):
        self.QUIT = tk.Button(self)
        self.QUIT["text"] = "QUIT"
        self.QUIT["fg"]   = "red"
        self.QUIT["command"] =  self.quit
        self.QUIT.pack({"side": "right"})

        # Simple Label widget:
        self.filename_title = tk.Label(self, text="Fileame:")
        self.filename_title.pack({"side": "left"})

        # Simple Entry widget:
        self.filename = tk.Entry(self, width=120)
        self.filename.pack({"side": "left"})
        self.filename.delete(0, tk.END)

        self.selector = tk.Button(self)
        self.selector["text"] = "Select",
        self.selector["command"] = self.get_file
        self.selector.pack({"side": "left"})

        self.process = tk.Button(self)
        self.process["text"] = "Process",
        self.process["command"] = self.run_process
        self.process.pack({"side": "left"})
```

```python
def main():
    root = tk.Tk()
    app = Example(parent=root)

    root.lift()
    root.call('wm', 'attributes', '.', '-topmost', True)
    root.after_idle(root.call, 'wm', 'attributes', '.', '-topmost', False)

    app.mainloop()

main()
```

Tk: Checkbox

```python
import tkinter as tk

class Example(tk.Frame):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.pack()
        self.createWidgets()

    def show_values(self):
        print("show values")
        for v in self.vars:
            print(v.get())

    def createWidgets(self):
        self.QUIT = tk.Button(self)
        self.QUIT["text"] = "QUIT"
        self.QUIT["fg"]   = "red"
        self.QUIT["command"] =  self.quit
        self.QUIT.pack({"side": "left"})


        self.vars = []
        self.cbs = []
        self.vars.append(tk.IntVar())
        cb = tk.Checkbutton(text="Blue", variable=self.vars[-1])
```

```
        cb.pack({"side": "left"})
        self.cbs.append(cb)

        self.vars.append(tk.IntVar())
        cb = tk.Checkbutton(text="Yellow",
variable=self.vars[-1])
        cb.pack({"side": "left"})
        self.cbs.append(cb)

        self.show = tk.Button(self)
        self.show["text"] = "Show",
        self.show["command"] = self.show_values
        self.show.pack({"side": "left"})

def main():
    root = tk.Tk()
    app = Example(parent=root)

    root.lift()
    root.call('wm', 'attributes', '.', '-topmost', True)
    root.after_idle(root.call, 'wm', 'attributes', '.',
'-topmost', False)

    app.mainloop()

main()
```

Tk: Runner

```
import tkinter as tk
import time

# TODO: async or threading to run long-running other
processes


class RunnerApp(tk.Frame):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.pack()

        # Capture event when someone closes the window
with the X on the top-right c\
```

```
orner of the window
        parent.protocol("WM_DELETE_WINDOW", self.close_app)

        self.QUIT = tk.Button(self)
        self.QUIT["text"] = "QUIT"
        self.QUIT["fg"] = "red"
        self.QUIT["command"] = self.close_app
        self.QUIT.pack({"side": "left"})

        self.start_button = tk.Button(self)
        self.start_button["text"] = "Start"
        self.start_button["command"] = self.start
        self.start_button.pack({"side": "left"})

        self.stop_button = tk.Button(self)
        self.stop_button["text"] = "Stop"
        self.stop_button["command"] = self.stop
        self.stop_button.pack({"side": "left"})

        self.text = tk.Text(self, state='disabled')
        self.text.pack({"side": "bottom"})

        self.stop_process = False

    def close_app(self):
        print("close")
        self.stop_process = True
        self.quit()

    def stop(self):
        print("stop")
        self.stop_process = True
        self.add_line('stop')

    def start(self):
        self.stop_process = False
        for i in range(100):
            if self.stop_process:
                break
            self.add_line(str(i))
            time.sleep(0.1)

    def add_line(self, line):
        self.text['state'] = 'normal'  # allow editing of
```

```
the Text widget
            self.text.insert('end', line + "\n")
            self.text['state'] = 'disabled'   # disable
editing
            self.text.see('end')   # scroll to the end as we
make progress
            self.update()   # update the content and allow
other events (e.g. from stop a\
nd quit buttons) to take place


def main():
    tk_root = tk.Tk()
    app = RunnerApp(parent=tk_root)

    tk_root.lift()
    tk_root.call('wm', 'attributes', '.', '-topmost',
True)
    tk_root.after_idle(tk_root.call, 'wm', 'attributes',
'.', '-topmost', False)

    app.mainloop()


main()
```

Tk: Runner with threads

```
import tkinter as tk
import time
import threading
import queue
import ctypes

class MyStopButton(Exception):
    pass

class ThreadedJob(threading.Thread):
    def __init__(self, que):
        self.que = que
        threading.Thread.__init__(self)
    def run(self):
        thread = threading.current_thread()
```

```python
        print("Start thread {}".format(thread.name))
        try:
            for i in range(10):
                print(i)
                self.que.put(str(i))
                time.sleep(1)
        except Exception as err:
            print(f"Exception in {thread.name}: {err} {err.__class__.__name__}")



    def raise_exception(self):
        thread = threading.current_thread()
        print(f"Raise exception in {thread.name}")
        thread_id = self.native_id
        res = ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id, ctypes.py_object\
(MyStopButton))
        if res > 1:
            ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id, 0)
            print('Exception raise failure')
        print("DONE")

class RunnerApp(tk.Frame):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.pack()

        # Capture event when someone closes the window with the X on the top-right c\
orner of the window
        parent.protocol("WM_DELETE_WINDOW", self.close_app)

        self.QUIT = tk.Button(self)
        self.QUIT["text"] = "QUIT"
        self.QUIT["fg"] = "red"
        self.QUIT["command"] = self.close_app
        self.QUIT.pack({"side": "left"})

        self.start_button = tk.Button(self)
        self.start_button["text"] = "Start"
```

```
        self.start_button["command"] = self.start
        self.start_button.pack({"side": "left"})

        self.stop_button = tk.Button(self)
        self.stop_button["text"] = "Stop"
        self.stop_button["command"] = self.stop
        self.stop_button.pack({"side": "left"})

        self.text = tk.Text(self, state='disabled')
        self.text.pack({"side": "bottom"})

        self.stop_process = False

    def close_app(self):
        print("close")
        self.stop_process = True
        self.quit()

    def stop(self):
        print("stop")
        print(self.job.name)
        self.job.raise_exception()
        #self.stop_process = True
        self.add_line('stop')


    def start(self):
        self.stop_process = False
        self.start_button['state'] = 'disabled'
        self.que = queue.Queue()
        self.job = ThreadedJob(self.que)
        self.job.start()
        self.master.after(100, self.process_queue)

    def process_queue(self):
        print("process " + str(time.time()))
        if not self.job.is_alive():
            self.job.join()
            self.job = None
            self.stop_process = True
            self.start_button['state'] = 'normal'
            print("finished")
            return

        try:
```

```
                msg = self.que.get(0)
                self.add_line(msg)
        except queue.Empty:
            pass
        finally:
            if not self.stop_process:
                self.master.after(100, self.process_queue)

    def add_line(self, line):
        self.text['state'] = 'normal'  # allow editing of the Text widget
        self.text.insert('end', line + "\n")
        self.text['state'] = 'disabled'  # disable editing
        self.text.see('end')  # scroll to the end as we make progress
        self.update()  # update the content and allow other events (e.g. from stop a\
nd quit buttons) to take place


def main():
    tk_root = tk.Tk()
    app = RunnerApp(parent=tk_root)

    tk_root.lift()
    tk_root.call('wm', 'attributes', '.', '-topmost', True)
    tk_root.after_idle(tk_root.call, 'wm', 'attributes', '.', '-topmost', False)

    app.mainloop()


main()
```

Getting started with Tk

```
import tkinter as tk

class Example(tk.Frame):
    def __init__(self, parent=None):
```

```
        super().__init__(parent)
        self.pack()
        self.createWidgets()

    def say_hi(self):
        print("hi there, everyone! ")
        print("Name: {}".format(self.name.get()))
        print("Password: {}".format(self.password.get()))
        print("count: {}".format(self.count.get()))
        self.password.delete(0, 'end')


    def createWidgets(self):
        self.QUIT = tk.Button(self)
        self.QUIT["text"] = "QUIT"
        self.QUIT["fg"]   = "red"
        self.QUIT["command"] =  self.quit
        self.QUIT.pack({"side": "left"})

        # Simple Label widget:
        self.name_title = tk.Label(self, text="Name:")
        self.name_title.pack({"side": "left"})

        # Simple Entry widget:
        self.name = tk.Entry(self)
        self.name.pack({"side": "left"})
        self.name.insert(0, "Your name")

        # Simple Label widget:
        self.password_title = tk.Label(self, text="Password:")
        self.password_title.pack({"side": "left"})

        self.count = tk.IntVar()
        self.count.set(2)
        self.my_radio = []
        radio = [(1, "One"), (2, "Two"), (3, "Three")]
        for ix in range(len(radio)):
            self.my_radio.append(tk.Radiobutton(self, text=radio[ix][1], variable=se\
lf.count, value=radio[ix][0]))
            self.my_radio[ix].pack({"side": "bottom"})

        # In order to hide the text as it is typed (e.g. for Passwords)
```

```python
        # set the "show" parameter:
        self.password = tk.Entry(self)
        self.password["show"] = "*"
        self.password.pack({"side": "left"})

        self.hi_there = tk.Button(self)
        self.hi_there["text"] = "Hello",
        self.hi_there["command"] = self.say_hi

        self.hi_there.pack({"side": "left"})

def main():
    root = tk.Tk()
    app = Example(parent=root)

    root.lift()
    root.call('wm', 'attributes', '.', '-topmost', True)
    root.after_idle(root.call, 'wm', 'attributes', '.', '-topmost', False)

    app.mainloop()

main()
```

Exercise: Tk - Calculator one line

Write a Tk application that behaves like a one-line calculator.
It has an entry box where one can enter an expression like “2 + 3” and a button.
When the button is pressed the expression is calculated.

There is another button called “Quit” that will close the application.

Exercise: Tk Shopping list

Create a Tk application that allows you to create a shopping list.

Exercise: Tk TODO list

- Create a Tk application to handle your TODO items.
- A Menu to be able to exit the application
- A List of current tasks.
- A way to add a new task. For a start each task has a title and a status. The status can be “todo” or “done”. (default is “todo”)
- A way to edit a task. (Primarily to change its title).
- A way to mark an item as “done” or mark it as “todo”.
- A way to move items up and down in the list.
- The application should automatically save the items in their most up-to-date state in a “database”. The database can be a JSON file or and SQLite database or anything else you feel fit.

Exercise: Tk Notepad

- Create a Notepad like text editor.
- It needs to have a menu called File with item: New/Open/Save/Save As/Exit
- It needs to have an area where it can show the content of a file. Let you edit it.

- Create a menu called About that displays an about box containing the names of the authors of the app.
- Menu item to Search for text.

Exercise: Tk Copy files

An application that allows you to type in, or select an existing file and another filename
for which the file does not exists.
Then copy the old file to the new name.

Exercise: Tk

- Application that accepts a “title” - line of text, a file selected, a new filename (that probably does not exist) and then runs.

Solution: Tk - Calculator one line

```python
import tkinter as tk

app = tk.Tk()
app.title('Calculator')

entry = tk.Entry(app)
entry.pack()

def calc():
    print("clicked")
    inp = entry.get()
    print(inp)
    out = eval(inp)
    entry.delete(0, tk.END)
    entry.insert(0, out)

def close():
    app.destroy()

calc_btn = tk.Button(app, text='Calculate', width=25, command=calc)
calc_btn.pack()


close_btn = tk.Button(app, text='Close', width=25, command=close)
close_btn.pack()

app.mainloop()
```

```python
import tkinter as tk

# This solutions is not ready yet

app = tk.Tk()
app.title('Calculator')

```

```python
entry = tk.Entry(app)
entry.pack()

def calc():
    print("clicked")
    inp = entry.get()
    print(inp)
    out = eval(inp)
    entry.delete(0, tk.END)
    entry.insert(0, out)

def close():
    app.destroy()
    exit()

def enter(num):
    entry.insert(tk.END, num)

def add_button(num, frame):
    btn = tk.Button(frame, text=num, width=25, command=lambda : enter(num))
    btn.pack(side="left")
    buttons[num] = btn

numbers_frame = tk.Frame(app)
numbers_frame.pack()
numbers_row = {}
numbers_row[1] = tk.Frame(numbers_frame)
numbers_row[1].pack(side="top")
numbers_row[2] = tk.Frame(numbers_frame)
numbers_row[2].pack(side="top")
numbers_row[3] = tk.Frame(numbers_frame)
numbers_row[3].pack(side="top")
ops_row = tk.Frame(numbers_frame)
ops_row.pack(side="top")

buttons = {}

add_button(1, numbers_row[1])
add_button(2, numbers_row[1])
add_button(3, numbers_row[1])
add_button(4, numbers_row[2])
add_button(5, numbers_row[2])
add_button(6, numbers_row[2])
add_button(7, numbers_row[3])
```

```python
add_button(8, numbers_row[3])
add_button(9, numbers_row[3])


for op in ['+', '-', '*', '/']:
    add_button(op, ops_row)


calc_btn = tk.Button(app, text='Calculate', width=25, command=calc)
calc_btn.pack()


close_btn = tk.Button(app, text='Close', width=25, command=close)
close_btn.pack()

app.mainloop()
```

Solution: Tk

```python
import tkinter as tk
from tkinter import filedialog

def run_process():
    print("---- Start processing ----")
    title = title_entry.get()
    print(title)
    filename = input_file.get()
    print(filename)

    app.destroy()

def select_input_file():
    file_path = filedialog.askopenfilename()
    filedialog.asksaveasfile()
    print(file_path)
    input_file.set(file_path)

app = tk.Tk()
app.title('Convert file')

input_file = tk.StringVar()
```

```python

title_label = tk.Label(app, text='Title')
title_label.pack()
title_entry = tk.Entry(app)
title_entry.pack()

input_button = tk.Button(app, text='Input file', command=select_input_file)
input_button.pack()
input_label = tk.Label(app, textvariable=input_file)
input_label.pack()


button = tk.Button(app, text='Process', width=25, command=run_process)
button.pack()

app.mainloop()
```

Solution: Tk Notepad

```python
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import os

file_path = None

app = tk.Tk()
app.title('Menu')

def run_new():
    global file_path
    file_path = None
    text.delete(1.0, tk.END)

def run_open():
    global file_path
    file_path = filedialog.askopenfilename(filetypes=(("Any file", "*"),))
    if file_path and os.path.isfile(file_path):
        with open(file_path) as fh:
            content = fh.read()
        text.delete(1.0, tk.END)
```

```
        text.insert('end', content)

def run_save():
    global file_path
    if file_path is None:
        file_path = filedialog.asksaveasfilename(filetypes=(("Any file", "*"),))
        if not file_path:
            file_path = None
            return
    #print(f"'{file_path}'")
    content = text.get(1.0, tk.END)
    with open(file_path, 'w') as fh:
        fh.write(content)

def run_exit():
    print("exit")
    app.destroy()

def run_about():
    #print(dir(simpledialog))
    #answer = simpledialog.Dialog(app, "The title")
    messagebox.showinfo(title = "About", message = "This simple text editor was crea\
ted as a solution for the exercise.\n\nCopyright: Gabor Szabo")

menubar = tk.Menu(app)

menu1 = tk.Menu(menubar, tearoff=0)
menu1.add_command(label="New",  underline=0, command=run_new)
menu1.add_command(label="Open", underline=0, command=run_open)
menu1.add_command(label="Save", underline=0, command=run_save)
menu1.add_separator()
menu1.add_command(label="Exit", underline=1, command=run_exit)
menubar.add_cascade(label="File", underline=0, menu=menu1)

menubar.add_command(label="About", underline=0, command=run_about)
```

```

app.config(menu=menubar)

text = tk.Text(app)
text.pack({"side": "bottom"})

app.mainloop()

# TODO: Show the name of the file somewhere? Maybe at the bottom in a status bar?
# TODO: Indicate if the file has been changed since the last save?
# TODO: Ask before exiting or before replacing the content if the file has not been \
saved yet.
# TODO: Undo/Redo?
# TODO: Search?
# TODO: Search and Replace?
```

Simple file dialog

```
from tkinter import filedialog

input_file_path = filedialog.askopenfilename(filetypes=(("Excel files", "*.xlsx"), (\
"CSV files", "*.csv"), ("Any file", "*")))
print(input_file_path)

input("Press ENTER to end the script...")
```

Python Pitfalls

Reuse of existing module name

```
import random

print(random.random())
```

```
$ python examples/pitfalls/random.py
```

```
Traceback (most recent call last):
  File "examples/pitfalls/random.py", line 1, in <module>
    import random
  File ".../examples/pitfalls/random.py", line 3, in
<module>
    print(random.random())
TypeError: 'module' object is not callable
```

- Write an example to use random number and call your example **number.py**
- Same with any other module name.
- Lack of multi-level namespaces
- Solution: user longer names. Maybe with project specific names.

Use the same name more than once

```
class Corp(object):
    people = []
    def add(self, name, salary):
        Corp.people.append({ 'name': name, 'salary' :
salary})
```

```

    def total(self):
        self.total = 0
        for n in Corp.people:
            self.total += n['salary']
        return self.total

c = Corp()
c.add("Foo", 19)
print(c.total())

c.add("Bar", 23)
print(c.total())
```

```
$ python examples/pitfalls/corp.py
```

```
19
Traceback (most recent call last):
  File "examples/pitfalls/corp.py", line 19, in <module>
    print(c.total())
TypeError: 'int' object is not callable
```

Compare string and number

```
x = 2
y = "2"

print(x > y)
print(x < y)
```

Python 2 - compares them based on the type of values (wat?)

```
$ python examples/pitfalls/compare.py
```

```
False
True
```

Python 3 - throws exception as expected.

```
$ python3 examples/pitfalls/compare.py
```

```
Traceback (most recent call last):
  File "examples/pitfalls/compare.py", line 4, in <module>
    print(x > y)
TypeError: unorderable types: int() > str()
```

Compare different types

```
x = 2
y = "2"

print(x == y)

with open(__file__) as fh:
    print(fh == x)
```

In both Python 2 and Pyhton 3 these return **False**

```
import sys

hidden = 42    # would be random

if sys.version_info.major < 3:
    guess = raw_input('Your guess: ')
else:
    guess = input('Your guess: ')

if hidden == guess:
    print("Match!")
```

Will never match. Even if user types in 42. - Hard to debug and understand as there is no error.

Sort mixed data

```python
from __future__ import print_function

mixed = [10, '1 foo', 42, '4 bar']
print(mixed)  # [100, 'foo', 42, 'bar']
mixed.sort()
print(mixed)  # [42, 100, 'bar', 'foo']
```

In Python 2 it “works” is some strange way.

```
$ python examples/pitfalls/sort.py
```

```
[10, '1 foo', 42, '4 bar']
[10, 42, '1 foo', '4 bar']
```

In Python 3 in **correctly** throws an exception.

```
air:python gabor$ python3  examples/pitfalls/sort.py
```

```
[10, '1 foo', 42, '4 bar']
Traceback (most recent call last):
  File "examples/pitfalls/sort.py", line 5, in <module>
    mixed.sort()
TypeError: unorderable types: str() < int()
```

Linters

Static Code Analyzis - Linters

- PEP8
- Flake8
- Pylint

PEP8

```
pip install pep8
```

- pep8
- pep8

F811 - redefinition of unused

```
import subprocess
import datetime
import sys
from datetime import datetime
```

```
$ flake8 importer.py
importer.py:4:1: F811 redefinition of unused 'datetime'
from line 2
```

Warn when Redefining functions

```
sum = 42

def len(thing):
```

```
    print(f"Use {thing}.__len__() instead!")

len("abc")
```

```
************* Module redef
redef.py:1:0: C0111: Missing module docstring (missing-docstring)
redef.py:2:0: W0622: Redefining built-in 'sum' (redefined-builtin)
redef.py:4:0: W0622: Redefining built-in 'len' (redefined-builtin)
redef.py:2:0: C0103: Constant name "sum" doesn't conform to UPPER_CASE naming style \
(invalid-name)
redef.py:4:0: C0111: Missing function docstring (missing-docstring)

------------------------------------------------------------------------
Your code has been rated at -2.50/10 (previous run: -2.50/10, +0.00)
```

Python .NET

IronPython

Python running on the DLR
that is on top of the CLR of Microsoft.

- [https://ironpython.net/
- GitHub
- Only supports Python 2

- Iron Python 3
- Not ready for production

Use .NET libraries from Python

- pythonnet
- pythonnet source code

```
pip install pythonnet
```

The latest Visual Studio is supposed to include Nuget, but if you don't have it, you can download it from Nuget downloads

Make sure `nuget.exe` is somewhere in your PATH:

For example I've created `C:\Bin`, put the nuget.exe in this directory and added `C:\Bin` to the `PATH`.

Then install the compilers using nuget install Microsoft.Net.Compilers as suggested on [Roslyn]
This created the `Microsoft.Net.Compilers.3.4.0` directory in my home directory

Make sure `csc.exe` is somewhere in your PATH or use the full path to it:

“UsersGabor SzaboMicrosoft.Net.Compilers.3.4.0\tools\csc.exe” /t:library MyMath.cs

Python and .NET console

```
import clr
from System import Console

Console.WriteLine("Hello My World!")
```

```
python net_console.py
```

Python and .NET examples

```
namespace MyMath
{
    public static class MyMathClass
    {
        public static int addInts(int a, int b)
        {
            return a+b;
        }

        public static double addDouble(double a, double
b)
        {
            return a+b;
        }

```

```
        public static string addString(string a, string b)
        {
            return a+" "+b;
        }

        public static bool andBool(bool a, bool b)
        {
            return a && b;
        }

        public static string str_by_index(string[] a, int b)
        {
            return a[b];
        }
        public static int int_by_index(int[] a, int b)
        {
            return a[b];
        }

    }
}
```

```python
import clr
dll = clr.FindAssembly('MyMath')  # returns path to dll
assembly = clr.AddReference('MyMath')
#print(type(assembly)) # <class 'System.Reflection.RuntimeAssembly'>
#print(dir(assembly))
from MyMath import MyMathClass
from MyMath import MyMathClass as My


assert My.addInts(2, 3)          == 5
assert My.addInts(2.7, 7.8)      == 9
assert My.addDouble(11.2, 23.3) == 34.5
assert My.addString("hello", "world") == "hello world"

assert My.andBool(1, 1) is True
assert My.andBool(1, 0) is False
assert My.andBool(True, True) is True
assert My.andBool(False, True) is False

```

```python
assert My.str_by_index(["apple", "banana", "peach"], 0) == "apple"
assert My.str_by_index(["apple", "banana", "peach"], 1) == "banana"
assert My.int_by_index([17, 19, 42], 1) == 19
# Mixed list cannot be passed

# tuple can be passed
assert My.int_by_index((17, 21, 42), 2) == 42

# TODO: string, char, float
# TODO strings, lists, dicts,
# TODO complex data structures in C#
# TODO Async
```

```
csc /t:library MyMath.cs
python myapp.py
```

C:WindowsMicrosoft.NETFramework\v4.0.30319\
C:Program Files\dotnet\

Exercise Python and .NET

- Take a .NET class that you would like to use, try that.

Python and Java

Jython

- Jython
- See separate chapter

Calling Java from Python

- Pyjnius/Jnius - GitHub
- JCC
- javabridge
- Jpype - GitHub
- Py4j

Jython - Python running on the JVM

Jython Installation

- Jython
- java -jar jython-installer-2.7.0.jar
- ~/jython2.7.0/

Jython Installation

```
java -jar ~/jython2.7.0/jython.jar

java -jar ~/jython2.7.0/jython.jar some.py
```

Jython load Java class

```
cd examples/mymath/
java -jar ~/jython2.7.0/jython.jar
Jython 2.7.0 (default:9987c746f838, Apr 29 2015, 02:25:11)
[Java HotSpot(TM) 64-Bit Server VM (Oracle Corporation)] on java1.8.0_60
Type "help", "copyright", "credits" or "license" for more information.
>>> import Calculator
>>> Calculator.add(2, 3)
5
>>> Calculator.add(10, 3)
10
>>>
```

Jython load Java class in code

```java
public class Calculator {
    public static Integer add(Integer a, Integer b) {
        if (a == 10) {
            return 10;
        }
        return a+b;
    }

}
```

```python
# use only with Jython

import Calculator
print(Calculator.add(4, 8))
print(Calculator.add(10, 8))
```

```
cd examples/jython/mymath/
java -jar ~/jython2.7.0/jython.jar calc.py
```

Jython test Java class

```python
import unittest
import Calculator

class TestAdd(unittest.TestCase):

    def test_add(self):
        self.assertEqual(Calculator.add(4, 8), 12)
        self.assertEqual(Calculator.add(10, 8), 18)
        self.assertEqual(Calculator.add(-1, 1), 0)

if __name__ == '__main__':
    unittest.main()

```

```
java -jar ~/jython2.7.0/jython.jar calc.py
java -jar ~/jython2.7.0/jython.jar -m unittest discover
```

PIL - Pillow

Install Pillow

- [Pillow]
- [Pillow on PyPI]
- [GitHub]

```
pip install pillow
```

Create First Image

```python
from PIL import Image

img = Image.new('RGB', size=(100, 60), color='#eb8634')
img.save('first.png')
img.show()      # Using ImageMagic on Linux
```

- Color can be one of the well-known names e.g. “red”
- Color can be RGB in decimal or hex. (RGB=Red Green Blue)

Write Text on Image

```python
from PIL import Image, ImageDraw

img = Image.new('RGB', size=(100, 60), color='#eb8634')

draw = ImageDraw.Draw(img)
draw.text(
    text="Some text",
    xy=(10, 20),
)

```

```python
img.save('first.png')
img.show()
```

Select font for Text on Image

```python
from PIL import Image, ImageDraw, ImageFont

img = Image.new(mode='RGB', size=(300, 60), color='#eb8634')
font = ImageFont.truetype('Pillow/Tests/fonts/FreeMono.ttf', 20)
#font = ImageFont.truetype(f'c:\Windows\Fonts\Candara.ttf', 30)
#font = ImageFont.truetype(f'c:\Windows\Fonts\Candarab.ttf', 30)
#font = ImageFont.truetype(f'c:\Windows\Fonts\david.ttf', 30)


draw = ImageDraw.Draw(img)
draw.text(
    text="Some text",
    xy=(10, 20),
    font=font,
)

img.save('first.png')
img.show()
```

Font directories

```
Linux: /usr/share/fonts/
Max OS: /Library/Fonts/
Windows: C:\Windows\fonts
```

Get size of an Image

```python
from PIL import Image
import sys
if len(sys.argv) !=2:
```

```
    exit(f"Usage: {sys.argv[0]} FILENAME")

in_file = sys.argv[1]

img = Image.open(in_file)
print(img.size)    # a tuple
print(img.size[0]) # width
print(img.size[1]) # height
```

Get size of text

```
    font = ImageFont.truetype(
        'path/to/font.ttf', size
    )
    size = font.getsize(text)
```

Resize an existing Image

```
from PIL import Image

in_file = 'in.png'
out_file = 'new.png'

img = Image.open(in_file)

size = (img.size[0] / 2, img.size[1] / 2)
img.thumbnail(size)

img.save(out_file)
```

Crop an existing Image

```
from PIL import Image

in_file = 'in.png'
out_file = 'out.png'

img = Image.open(in_file)
width, height = img.size
width, height = img.size
```

```

# crop
# 10 pixels from the left
# 20 pixels from the top
# 30 pixels from the right
# 40 pixels from the bottom

cropped = img.crop((10, 20, width - 30, height - 40))
cropped.save(out_file)
cropped.show()
```

Combine two images

- Load one image from file
- Create a plain background
- Put the loaded image on the background
- Save the combined image

Rotated text

```
from PIL import Image, ImageDraw, ImageFont, ImageOps

img = Image.new(mode='RGB', size=(400, 200), color='#eb8634')

font = ImageFont.truetype('Pillow/Tests/fonts/FreeSansBold.ttf', 30)

text_layer = Image.new('L', (330, 50))
draw = ImageDraw.Draw(text_layer)
draw.text( (30, 0), "Text slightly rotated",  font=font, fill=255)

rotated_text_layer = text_layer.rotate(10.0, expand=1)
img.paste( ImageOps.colorize(rotated_text_layer, (0,0,0), (10, 10,10)), (42,60),  ro\
tated_text_layer)
img.show()
```

Rotated text in top-right corner

TODO: fix this

```
from PIL import Image, ImageDraw, ImageFont, ImageOps

width = 400
height = 200
start = 100
end   = 50

img = Image.new(mode='RGB', size=(width, height), color='#FAFAFA')

stripe_color = "#eb8634"
draw = ImageDraw.Draw(img)
draw.polygon([(width-start, 0), (width-end, 0), (width, end), (width, start) ], fill\
=stripe_color)


font = ImageFont.truetype('Pillow/Tests/fonts/FreeSansBold.ttf', 30)
text_layer = Image.new('RGB', size=(100, 100), color=stripe_color)

draw = ImageDraw.Draw(text_layer)
text = "Free"
size = draw.textsize(text=text, font=font)
# print(size)
draw.text( xy=(20, 0), text=text,  font=font, fill=1)
#
rotated_text_layer = text_layer.rotate(-45.0, expand=0)
rotated_text_layer.show()
#img.paste( ImageOps.colorize(rotated_text_layer, (0,0,0), (10, 10,10)), (42,60),  r\
otated_text_layer)
#img.paste(im = rotated_text_layer, box=(300, 0))
#img.paste(im = text_layer, box=(300, 0))
#img.show()
```

Embed image (put one image on another one)

```
from PIL import Image

in_file = 'python.png'

width = 600
height = 300
background = Image.new(mode='RGB', size=(width, height),
color='#AAFAFA')

img = Image.open(in_file)
(emb_width, emb_height) = img.size
print(emb_width)
print(emb_height)

# slightly off the lower right corner of the background
image
# using the image as the mask makes its background
transparent
background.paste(im = img, box=(width-emb_width-10,
height-emb_height-10), mask=img)

background.show()
```

Draw a triangle

```
from PIL import Image, ImageDraw

img = Image.new(mode='RGB', size=(800, 450),
color='#eb8634')

draw = ImageDraw.Draw(img)
draw.polygon([(800, 275), (800, 450), (300, 450) ])

img.save('first.png')
img.show()
```

Draw a triangle and write text in it

```python
from PIL import Image, ImageDraw, ImageFont

img = Image.new(mode='RGB', size=(800, 450), color='#eb8634')

draw = ImageDraw.Draw(img)
draw.polygon([(800, 275), (800, 450), (300, 450) ], fill = (255, 255, 255))

font = ImageFont.truetype('Pillow/Tests/fonts/FreeSansBold.ttf', 30)

draw.text((500, 400), 'Hello from Python', (0, 0, 0), font=font)


img.save('first.png')
img.show()
```

Draw a triangle and write rotated text in it

```python
from PIL import Image, ImageDraw, ImageFont, ImageOps

img = Image.new(mode='RGB', size=(400, 200), color='#eb8634')

# #draw = ImageDraw.Draw(img)
# #draw.polygon([(800, 275), (800, 450), (300, 450) ], fill = (255, 255, 255))
#
#
#font = ImageFont.load_default()
font = ImageFont.truetype('Pillow/Tests/fonts/FreeSansBold.ttf', 30)
# txt = Image.new('L', (500, 500))
# d = ImageDraw.Draw(txt)
# d.text((300, 400), 'Hello from Python', font=font, color="white")
# w=txt.rotate(17.5,  expand=1)
#
# #img.paste(txt)
```

```
# img.paste( ImageOps.colorize(w, (0,0,0), (255,255,84)), (242,60),  w)
# # img.save('first.png')
# img.show()
#

text_layer = Image.new('L', (300, 50))
draw = ImageDraw.Draw(text_layer)
draw.text( (30, 0), "Text slightly rotated",  font=font, fill=255)

rotated_text_layer = text_layer.rotate(10.0, expand=1)
img.paste( ImageOps.colorize(rotated_text_layer, (0,0,0), (10, 10,10)), (42,60),  ro\
tated_text_layer)
img.show()
```

Draw a rectangular

```
from PIL import Image, ImageDraw

img = Image.new(mode='RGB', size=(800, 450), color='#eb8634')

draw = ImageDraw.Draw(img)
draw.polygon([(400, 200), (400, 300), (200, 300), (200, 200) ])

img.save('first.png')
img.show()
```

Draw a rectangle

```
from PIL import Image, ImageDraw

img = Image.new('RGB', size=(100, 100))

draw = ImageDraw.Draw(img)
draw.rectangle((10, 10, 90, 90), fill="yellow", outline="red")
img.show()
```

Draw circle

```
from PIL import Image, ImageDraw

img = Image.new('RGB', (200, 200))

draw = ImageDraw.Draw(img)
draw.ellipse((50, 50, 150, 150), fill="#F00F4F")
img.show()
```

Draw heart

```
from PIL import Image, ImageDraw

def heart(size, fill):
    width, height = size
    img = Image.new('RGB', size, (0, 0, 0, 0))
    draw = ImageDraw.Draw(img)
    polygon = [
        (width / 10, height / 3),
        (width / 10, 81 * height / 120),
        (width / 2, height),
        (width - width / 10, 81 * height / 120),
        (width - width / 10, height / 3),
    ]
    draw.polygon(polygon, fill=fill)
    #img.show()

    draw.ellipse((0, 0,  width / 2, 3 * height / 4), fill=fill)
    draw.ellipse((width / 2, 0,  width, 3 * height / 4), fill=fill)
    return img

img = heart((50, 40), "red")
img.show()
```

Some samples, including this one, originally by [Nadia Alramli](#)

Rectangle with rounded corners

```python
from PIL import Image, ImageDraw


def round_corner(radius, fill):
    """Draw a round corner"""
    corner = Image.new('RGB', (radius, radius), (0, 0, 0, 0))
    draw = ImageDraw.Draw(corner)
    draw.pieslice((0, 0, radius * 2, radius * 2), 180, 270, fill=fill)
    return corner


def round_rectangle(size, radius, fill):
    """Draw a rounded rectangle"""
    width, height = size
    rectangle = Image.new('RGB', size, fill)
    corner = round_corner(radius, fill)
    rectangle.paste(corner, (0, 0))
    rectangle.paste(corner.rotate(90), (0, height - radius))  # Rotate the corner an\
d paste it
    rectangle.paste(corner.rotate(180), (width - radius, height - radius))
    rectangle.paste(corner.rotate(270), (width - radius, 0))
    return rectangle


img = round_rectangle((50, 50), 10, "yellow")

img.show()
```

Some samples, including this one, originally by [Nadia Alramli]

TODO

http://web.archive.org/web/20130115175340/http://nadiana.com/pil-tutorial-basic-advanced-drawing

- Make the background color change from top to bottom

- Add straight lines to existing images
- Blur image
- Add rectangular to area on existing image
- Draw other simple images

FAQ

How not to name example scirpts?

Don't - by mistake - call one of your files the same as a module you will be loading.
For example `random.py` is a bad idea if you will `import random`.
Your code will try to locate random.py to load, but will find itself and not the one that comes with Python.

Python will also create a random.pyc file - a compiled file - and it will take time till you recall this
and delete that too.
Till then the whole thing will seem to be broken.

Platform independent code

In general Python is platform independent, but still needs some care to make sure
you don't step on some aspects of Operating System or the file system that works differently
on other OS-es.

- Filenames are case sensitive on some OS-es (e.g. Windows). They used to be restricted to 8.3. Make sure you are within the restriction of every OS you might want to use.
- Directory path: (slash or backslash or something else?) use the os.path methods.
- os.path.expanduser('~') works on both Linux and Windows, but the root of a Linux/Unix file system starts with a slash (/)

and on Windows it is c:\ and d:\ etc.
- On Linux/Unix you have user 'root' and on Windows 'Administrator'
- File permissions are different on Linux and Windows.
- Stay away from OS specific calls, but as a last resort use os.name or sys.platform to figure out which os is this. os.name is 'posix' on Linux and 'nt' on Windows.
- For GUI use wxWindows that has a native look on Windows and Gnome look on Linux.
- Pay attention to any 32/64 bit issues. Big/Little Endian issues.
- Some modules might be OS specific. Check the documentation.
- Pay attention to the use of os.system and subsystem modules.

How to profile a python code to find causes of slowness?

Use one of these modules:

- cProfile is in C. It is faster and preferable.
- profile

pdb = Python Debugger

Include the following code in your script at any point, and run the script as you'd do normally.
It will stop at the given point and enter the debugger.

```
import pdb; pdb.set_trace()
```

pdb

Avoid Redefining functions

Can I tell python to stop compilation when someone is redefining a function?
Or at least give me a warning?

Use `pylint` for that

Appendix

print_function

```
from __future__ import print_function

print(23)
```

Dividers (no break or continue)

We will see how break and continue work, but first let's see a loop to find all the dividers on a number n.

```
i = 2
n = 3*5*7
while i < n:
    if (n / i) * i == n:
        print('{:2}  divides {}'.format(i, n))
    i = i + 1
```

```
 3  divides 105
 5  divides 105
 7  divides 105
15  divides 105
21  divides 105
35  divides 105
```

Lambdas

```
a = lambda x: True
b = lambda x: False
c = lambda x: x
#c = lambda x: return
#c = lambda x: pass
```

```python
d = lambda x: c(x)+c(x)

print(a(1))
print(b(1))
print(c(42))
print(d(21))
```

Abstract Class

```python
import abc

class Port():
    __metaclass__ = abc.ABCMeta

    @abc.abstractmethod
    def num(self):
        pass

class HTTPPort(Port):
    def num(self):
        return 80

class FTPPort(Port):
    def num(self):
        return 21

class ZorgPort(Port):
    def nonum(self):
        return 'zorg'

f = FTPPort()
print(f.num())
h = HTTPPort()
print(h.num())
z = ZorgPort()
# Traceback (most recent call last):
#   File "abstract.py", line 26, in <module>
#     z = ZorgPort()
# TypeError: Can't instantiate abstract class ZorgPort with abstract methods num


print(z.num())
```

Remove file

os.remove or
os.unlink

Modules: more

- sys.modules to list loaded modules
- imp.reload to reload module (Just reload before 3.3)

```
import __builtin__

def xx(name):
    print("hello")
__builtin__.__import__ = xx;

print('body')
def f():
    print("in f")
```

```
import sys

print('mod' in sys.modules) # False

import mod
print('mod' in sys.modules)  # True
print(sys.modules['mod'])
     # <module 'mod' from '/stuff/python/examples/modules/mod.py'>

print(sys.modules["sys"])    # <module 'sys' (built-in)>
```

import hooks

Python resources

- Central Python site

* [Python documentation]
* [Learning Python the Hard way]
* [Python Weekly]
* [PyCoder's Weekly]

Progress bar

```
# http://stackoverflow.com/questions/3173320/text-progress-bar-in-the-console
import time, sys

for i in range(10):
    sys.stdout.write('\r' +  '=' * i)
    sys.stdout.flush()
    time.sleep(1)
```

from future

```
from __future__ import print_function
from __future__ import division
```

or

```
from __future__ import print_function, division
```

See also **future**

> We cannot import everything that is in **future**, because we don't know what will be in **future** in the future….
> and we don't want to blindly change the behaviour of Python.

Variable scope

- There are two scopes: outside of all functions and inside of a function.
- The first assignment to a variable defines it.
- Variables that were declared outside all functions can be seen inside, but cannot be changed.
- One can connect the outside name to an inside name using the ‘global’ keyword.
- if and for blocks don’t provide scoping.

```
a = 23

def main():
    global b
    b = 17
    c = 42
    print('a:', a)    # a: 23
    print('b:', b)    # b: 17
    print('c:', c)    # c: 42

    if True:
        print('a:', a)    # a: 23
        print('b:', b)    # b: 17
        b = 99
        print('b:', b)    # b: 99
        print('c:', c)    # c: 42

    print('a:', a)    # a: 23
    print('b:', b)    # b: 99
    print('c:', c)    # c: 42


main()

print('a:', a)  # a: 23
print('b:', b)  # b: 99
print('c:', c)  # c:
# Traceback (most recent call last):
#  File "examples\basics\scope.py", line 27, in <module>
```

```
#    print 'c:', c # c:
# NameError: name 'c' is not defined
```

global scope

scope

```
# x is global

x = 1
print(x, "- before sub")

def f():
    #print(x, "- inside before declaration")  #
UnboundLocalError
    x = 2
    print(x, "- inside sub")

print(x, "- after sub declaration")

f()

print(x, "- after calling sub")

# 1 - before sub
# 1 - after sub declaration
# 2 - inside sub
# 1 - after calling sub
```

```
# x is global

def f():
    #print(x, "- inside before declaration")  #
UnboundLocalError
    x = 2
    print(x, "- inside sub")

x = 1
```

```
print(x, "- before calling sub")

print(x, "- after sub declaration")

f()

print(x, "- after calling sub")

# 1 - before calling sub
# 1 - after sub declaration
# 2 - inside sub
# 1 - after calling sub
```

If we declare a variable outside of all the subroutines, it does not matter if we do it before the sub declaration, or after it. In neither case has the global variable any presence inside the sub.

```
def f():
    x = 2
    print(x, "- inside sub")

# print(x, " - after sub declaration")  # NameError

f()

# print(x, " - after calling sub")   # NameError

# 2 - inside sub
```

A name declared inside a subroutine is not visible outside.

```
def f():
    global x
```

```
    # print(x)   # NameError
    x = 2
    print(x, "- inside sub")

# print(x, " - after sub declaration")   # NameError

f()

print(x, "- after calling sub")

# 2 - inside sub
# 2 - after calling sub
```

Unless it was marked using the global word.

type

```
x = 2
y = '2'
z = [2, '2']
d = {}

def f():
    pass
l = lambda q: q

class Cold():
    pass
cold = Cold()

class Cnew(object):
    pass
cnew = Cnew()

# r = xrange(10)   # Python 3 does not have xrange

print(type(x))   # <type 'int'>
print(type(y))   # <type 'str'>
print(type(z))   # <type 'list'>
```

```
print(type(d))   # <type 'dict'>
print(type(f))   # <type 'function'>
print(type(l))   # <type 'function'>
print(type(Cold))  # <type 'classobj'>
print(type(cold))  # <type 'instance'>
print(type(Cnew))  # <type 'type'>
print(type(cnew))  # <class '__main__.Cnew'>
#print(type(r))  # <type 'xrange'>

print(type(x).__name__)  # int
print(type(y).__name__)  # str
print(type(z).__name__)  # list
```

Look deeper in a list

```
x = ['abcd', 'efgh']
print(x)          # ['abcd', 'efgh']

print(x[0:1])     # ['abcd']
print(x[0])       # 'abcd'

print(x[0][0])    # a
print(x[0][1])    # b
print(x[0][0:2])  # ab
```

Exercise: iterators - count

- Reimplement the count functions of itertools using iterator class.

(We have this as one of the example)

Simple function (before generators)

TODO: probably not that interesting

```
def number():
    return 42
```

```

print(number())    # 42
print(number())    # 42
print(number())    # 42
```

```
def number():
    return 42
    return 19
    return 23

print(number()) # 42
print(number()) # 42
print(number()) # 42
```

Other slides

Other slides

Some slides that used to be part of the material and they might return to be there, but for now they were parked here.

Atom for Python

> Some details about the Atom editor. You can freely skip this part. Personally I don't use it now.

- Atom

Autocomplete

- apm install autocomplete-python

Autocomplete

- easy_install jedi
- apm install autocomplete-plus-python-jedi

Linter

- easy_install flake8
- easy_install flake8-docstrings
- apm install linter

- apm install linter-flake8

source

IDLE - Integrated DeveLopment Environment

- Python shell
- Better editing
- Limited debugger
- `c:\Python27\Lib\idlelib\idle.bat`
- `C:\Users\Gabor\AppData\Local\Programs\Python\Python35\Lib\idlelib\idle.bat`

sh-bang - executable on Linux/Apple

```
#!/usr/bin/env python

print("Hello World")
```

- The first line staring with # is needed if you want to have a file that can be executed without explicitly typing in python as well.
- Make your file executable: **chmod u+x hello_ex.py**
- Run like: **./hello_ex.py**
- In order to run it as **hello_ex.py** in needs to be located in one of the directories listed in the **PATH** environment variable.

Strings as Comments

marks single line comments.

There are no real multi-line comments in Python, but we can use triple-quots to
create multi-line strings and if they are not part of another statement, they will be
disregarded by the Python interpreter. Effectively creating multi-line comments.

```
print("hello")

'A string which is disregarded'

print(42)

'''
  Using three single-quotes on both ends (a triple-quoted string)
  can be used as a multi-line comment.
'''

print("world")
```

pydoc

If you really want it, you can also read some of the documentation on the command line, but unless you are locked up some place without Internet connection,
I don’t recommend this.

Type `pydoc`. On Windows, you might need to create the following file and put it in a directory in your PATH. (see `echo %PATH%`)

```
@python c:\Python27\Lib\pydoc.py %*
```

How can I check if a string can be converted to a number?

There is no is_int, we just need to try to convert and catch the exception, if there is one.

```
def is_float(val):
    try:
        num = float(val)
    except ValueError:
        return False
    return True

def is_int(val):
    try:
        num = int(val)
    except ValueError:
        return False
    return True

print( is_float("23") )       # True
print( is_float("23.2") )     # True
print( is_float("23x") )      # False
print( '-----' )              # -----
print( is_int("23") )         # True
print( is_int("23.2") )       # False
print( is_int("23x") )        # False
```

Spyder Intro

- iPython console (bottom right)
- Spyder-Py2 / Preferences / Console / Advanced Settings
- Save the file (Ctrl-S / Command-S)
- Run/Run (F5)
- F9 - execute selected text (e.g. we can eecute a function definition after we’ve changed it)
- TAB for autocomple names of already existing variables.

```
print("abc")
"abc".           shows the available methods.
"abc".center     Command-I will explain what is "center"
```

Interactive Debugging

```python
def f(a, b):
    c = a + b
    d = a * b
    return c+d

def run():
    print(f(2, 3))

    import code
    code.interact(local=locals())

    print(f(19, 23))

run()
```

Parameter passing

```python
def hello(name):
    msg = name + '!!!!'
    print('Hello ' + msg)

hello('Foo')
hello('Bar')
```

```
Hello Foo!!!!
```

Command line arguments and main

```python
import sys

def hello(name):
    msg = name + '!!!!'
```

```
    print('Hello ' + msg)

def main():
    hello(sys.argv[1])

main()
```

Run as **python argv.py Foo**

Later we'll see the `argparse` module that can handle command line arguments in a better way.

Infinite loop

```
i = 0
while True:
    i += 1
    print(i)

print("done")
```

break

```
i = 0
while True:
    print(i)
    i += 1
    if i >= 7:
        break

print("done")
```

```
0
1
2
3
4
5
```

```
6
done
```

continue

```
i = 0
while True:
    i += 1

    if i > 3 and i < 8:
        continue

    if i > 10:
        break
    print(i)
```

```
1
2
3
8
9
10
```

While with many conditions

```
while (not found_error) and (not found_warning) and (not found_exit):
    do_the_real_stuff()

while True:
    line = get_next_line()

    if found_error:
        break

    if found_warning:
        break

    if found_exit:
        break
```

```
    do_the_real_stuff()
```

while loop with many conditions

```
while True:
   line = get_next_line()

   if last_line:
       break

   if line is empty:
      continue

   if line_has_a_hash: # at the  beginning:
      continue

   if line_has_two_slashes: // at the beginning:
      continue

   do_the_real_stuff()
```

Format with conversion (stringifiation with str or repr)

Adding !s or !r in the place-holder we tell it to cal the str or repr method of the object, respectively.

- repr (**repr**) Its goal is to be unambiguous
- str (**str**) Its goal is to be readable
- The default implementation of both are useless
- Suggestion
- Difference between **str** and **repr**

```
class Point:
    def __init__(self, a, b):
        self.x = a
        self.y = b
```

```python

p = Point(2, 3)
print(p)                    # <__main__.Point object at 0x10369d750>
print("{}".format(p))       # <__main__.Point object at 0x10369d750>
print("{!s}".format(p))     # <__main__.Point object at 0x10369d750>
print("{!r}".format(p))     # <__main__.Point object at 0x10369d750>
```

```python
class Point:
    def __init__(self, a, b):
        self.x = a
        self.y = b
    def __format__(self, spec):
        #print(spec) // empty string
        return("{{'x':{}, 'y':{}}}".format(self.x, self.y))
    def __str__(self):
        return("({},{})".format(self.x, self.y))
    def __repr__(self):
        return("Point({}, {})".format(self.x, self.y))

p = Point(2, 3)
print(p)                    # (2,3)
print("{}".format(p))       # {'x':2, 'y':3}
print("{!s}".format(p))     # (2,3)
print("{!r}".format(p))     # Point(2, 3)
```

Name of the current function in Python

```python
import inspect

def first():
    print(inspect.currentframe().f_code.co_name)
    print(inspect.stack()[0][3])
    second()


def second():
    print(inspect.currentframe().f_code.co_name)
```

```python
    print(inspect.stack()[0][3])

def main():
    first()

main()
```

Name of the caller function in Python

```python
import inspect

def first():
    print("in first")
    print("Called by", inspect.stack()[1][3])
    second()

def second():
    print("in second")
    print("Called by", inspect.stack()[1][3])

def main():
    first()

main()
```

Stack trace in Python using inspect

```python
import inspect

def first():
    second()


def second():
    for info in inspect.stack():
        #print(info)
        #FrameInfo(
        #    frame=<frame at 0x1c18b18, file 'stack_trace.py', line 9, code second>,
        #    filename='stack_trace.py',
        #    lineno=8,
```

```python
        #     function='second',
        #     code_context=['    for level in inspect.stack():\n'],
        #     index=0)

        #print(info.frame)
        print(info.filename)
        print(info.lineno)
        print(info.function)
        print(info.code_context)
        print('')

def main():
    first()


if __name__ == '__main__':
    main()
```

```
stack_trace.py
8
second
['    for info in inspect.stack():\n']

stack_trace.py
4
first
['    second()\n']

stack_trace.py
26
main
['    first()\n']

stack_trace.py
30
<module>
['    main()\n']
```

Module Fibonacci

```
def fibonacci_number(n):
    if n==1:
        return 1
    if n==2:
        return 1
    if n==3:
        return 5

    return 'unimplemented'

def fibonacci_list(n):
    if n == 1:
        return [1]
    if n == 2:
        return [1, 1]
    if n == 3:
        return [1, 1, 5]
    raise Exception('unimplemented')
```

PyTest - assertion

```
import mymath

def test_fibonacci():
    assert mymath.fibonacci(1) == 1
```

```
$ py.test test_fibonacci_ok.py
============================= test session starts ==============================
platform darwin -- Python 2.7.5 -- py-1.4.20 -- pytest-2.5.2
collected 1 items

test_fibonacci_ok.py .

=========================== 1 passed in 0.01 seconds ===========================
```

PyTest - failure

```python
import mymath

def test_fibonacci():
    assert mymath.fibonacci(1) == 1
    assert mymath.fibonacci(2) == 1
    assert mymath.fibonacci(3) == 2
```

```
$ py.test test_fibonacci.py
============================== test session starts ==============================
platform darwin -- Python 2.7.5 -- py-1.4.20 -- pytest-2.5.2
collected 1 items

test_fibonacci.py F

=================================== FAILURES ====================================
________________________________ test_fibonacci _________________________________

    def test_fibonacci():
        assert mymath.fibonacci(1) == 1
        assert mymath.fibonacci(2) == 1
>       assert mymath.fibonacci(3) == 2
E       assert 5 == 2
E        +  where 5 = <function fibonacci at 0x10a024500>(3)
E        +    where <function fibonacci at 0x10a024500> = mymath.fibonacci

test_fibonacci.py:6: AssertionError
=========================== 1 failed in 0.02 seconds ============================
```

PyTest - list

```python
import fibo

def test_fibonacci_number():
    assert fibo.fibonacci_number(1) == 1
    assert fibo.fibonacci_number(2) == 1
```

```python
    assert fibo.fibonacci_number(3) == 2
    assert fibo.fibonacci_number(4) == 2

def test_fibo():
    assert fibo.fibonacci_list(1) == [1]
    assert fibo.fibonacci_list(2) == [1, 1]
    assert fibo.fibonacci_list(3) == [1, 1, 2]
```

```
$ py.test test_fibo.py
========================== test session starts ===========================
platform darwin -- Python 2.7.5 -- py-1.4.20 -- pytest-2.5.2
collected 1 items

test_fibo.py F

================================ FAILURES ================================
_______________________________ test_fibo ________________________________

    def test_fibo():
        assert mymath.fibo(1) == [1]
        assert mymath.fibo(2) == [1, 1]
>       assert mymath.fibo(3) == [1, 1, 2]
E       assert [1, 1, 5] == [1, 1, 2]
E         At index 2 diff: 5 != 2

test_fibo.py:6: AssertionError
======================== 1 failed in 0.01 seconds ========================
```

SAX with coroutine

```python
import xml.sax

file = 'examples/xml/data.xml'

class EventHandler(xml.sax.ContentHandler):
    def __init__(self,target):
        self.target = target
```

```
    def startElement(self,name,attrs):
        self.target.send(('start',(name,attrs._attrs)))
    def characters(self,text):
        self.target.send(('text',text))
    def endElement(self,name):
        self.target.send(('end',name))

def printer():
    def start(*args,**kwargs):
        cr = func(*args,**kwargs)
        cr.next()
        return cr
    return start

# example use
if __name__ == '__main__':
    @coroutine
    def printer():
        while True:
            event = (yield)
            print(event)

    xml.sax.parse(file, EventHandler(printer()))
```

---

copied from Stack Overflow
based on coroutines

```
import xml.sax

file = 'examples/xml/data.xml'

class EventHandler(xml.sax.ContentHandler):
    def __init__(self,target):
        self.target = target
    def startElement(self,name,attrs):
        self.target.send(('start',(name,attrs._attrs)))
    def characters(self,text):
        self.target.send(('text',text))
    def endElement(self,name):
        self.target.send(('end',name))

def coroutine(func):
    def start(*args,**kwargs):
```

```python
        cr = func(*args,**kwargs)
        cr.next()
        return cr
    return start

# example use
if __name__ == '__main__':
    @coroutine
    def printer():
        while True:
            event = (yield)
            print(event)

    xml.sax.parse(file, EventHandler(printer()))
```

Getting the class name of an object

How to find out which class an object (instance) belongs to?

```python
import re

a = 2
b = "3"
c = 2.3

m = re.search(r'\d', str(c))

print(a.__class__)    # <type 'int'>
print(b.__class__)    # <type 'str'>
print(c.__class__)    # <type 'float'>

print(type(a))    # <type 'int'>
print(type(b))    # <type 'str'>
print(type(c))    # <type 'float'>


print(a.__class__.__name__)    # int
print(b.__class__.__name__)    # str
print(c.__class__.__name__)    # float

print(re.__class__.__name__)  # module
print(m.__class__.__name__)    # SRE_Match or Match
```

Inheritance - super

We can also call super() passing a different class name

```
class Point():
    def __init__(self, x, y):
        print('__init__ of point')
        self.x = x
        self.y = y

class Circle(Point):
    def __init__(self, x, y, r):
        print('__init__ of circle')
        super().__init__(x, y)
        self.r = r

class Ball(Circle):
    def __init__(self, x, y, r, z):
        print('__init__ of ball')
        #super(Circle, self).__init__(x, y) # r
        Point.__init__(self, x, y) # r
        self.z = z


b = Ball(2, 3, 10, 7)
print(b)

# __init__ of ball
# __init__ of point
# <__main__.Ball object at 0x10a26f190>
```

Inheritance - super - other class

We cannot pass any class name to super()

```python
class Point:
    def __init__(self, x, y):
        print('__init__ of point')
        self.x = x
        self.y = y

class Circle(Point):
    def __init__(self, x, y, r):
        print('__init__ of circle')
        super(Circle, self).__init__(x, y)
        self.r = r

class Ball(Circle):
    def __init__(self, x, y, r, z):
        print('__init__ of ball')
        super(Zero, self).__init__(x, y)
        self.z = z

class Zero:
    def __init__(self, x, y):
        print('really?')
    pass


b = Ball(2, 3, 10, 7)
print(b)

# __init__ of circle
# Traceback (most recent call last):
#   File "bad_shapes.py", line 25, in <module>
#     b = Ball(2, 3, 10, 7)
#   File "bad_shapes.py", line 16, in __init__
#     super(Zero, self).__init__(x, y)
# TypeError: super(type, obj): obj must be an instance or subtype of type
```

iterator - pairwise

```python
def pairwise(iterable):
    "s -> (s0,s1), (s2,s3), (s4, s5), ..."
    i = 0
    while i+1 < len(iterable):
        t = (iterable[i], iterable[i+1])
```

```
        i += 2
        yield t

l = [1, 2, 3, 4, 5, 6]
for x, y in pairwise(l):
   print(f"{x} + {y} = {x + y}")
```

iterator - grouped

```
def grouped(iterable, n):
    """s -> (s0,s1,s2,...sn-1),
           (sn,sn+1,sn+2,...s2n-1),
           (s2n,s2n+1,s2n+2,...s3n-1), ..."""

    i = 0
    while i+n-1 < len(iterable):
        t = tuple(iterable[i:i+n])
        i += n
        yield t

l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for x, y, z in grouped(l, 3):
    print("{} + {} + {} = {}".format(x, y, z, x + y + z))
```

```
1 + 2 + 3 = 6
4 + 5 + 6 = 15
7 + 8 + 9 = 24
```

itertools - groupby

Group elements

```
from itertools import groupby

def groupby_even_odd(items):
    f = lambda x: 'even' if x % 2 == 0 else 'odd'
    gb = groupby(items, f)
    print(gb)
    for k, items in gb:
        print('{}: {}'.format(k, ','.join(map(str,
```

```
items))))

groupby_even_odd([1, 3, 4, 5, 6, 8, 9, 11])
```

Circular references

circular references are cleaned up the by the garbage collector but maybe not all the memory is given back to the OS, and it can take some time to clean them up.

```
import time


def create_pair():
    a = {'name' : 'Foo'}
    b = {'name' : 'Bar'}
    a['pair'] = b
    b['pair'] = a
    #print(a)


for i in range(1, 30000000):
    create_pair()

print("let's sleep now a bit")
time.sleep(20)
```

but weakref might expedite the cleanup. See also the gc module and if I can show it http://stackoverflow.com/questions/2428301/should-i-worry-about-circular-references-in-python

Context managers: with (file) experiments

```
with open('out.txt', 'w') as h:
    h.write("hello\n")

```

```
h = open('out.txt')
print(h.read())
```

```
f = open('out.txt', 'w')
f.write("hello\n")
f.close()

# for line in open("myfile.txt"):
#     print line,
# the file is closed only when script ends
```

itertools - izip

Python 3 does not need this any more as the built-in zip is already an iterator.

Combine two unbounded lists

```
from itertools import izip, count

for t in izip(count(start=1, step=1), count(start=10,
step=-1)):
    print("{:3} + {:3} = {}".format(t[0], t[1],
t[0]+t[1]))
    if t[0] > 20:
        break
#   1 +  10 = 11
#   2 +   9 = 11
#   3 +   8 = 11
#   4 +   7 = 11
#   ...
#  20 +  -9 = 11
#  21 + -10 = 11
```

mixing iterators

Combine three unbounded lists

```python
from itertools import izip, count
from my_iterators import fibo, alter

mixer = izip(count(), fibo(), alter())

for mix in mixer:
    print("{:3}  {:3}  {:3}".format(*mix))
    if mix[0] >= 8: break

  # 0    1    1
  # 1    1   -2
  # 2    2    3
  # 3    3   -4
  # 4    5    5
  # 5    8   -6
  # 6   13    7
  # 7   21   -8
  # 8   34    9
```

mixing iterators

```python
def fibo():
    a, b = 1, 1
    while True:
        yield a
        a, b = b, a+b

def alter():
    n = 1
    while True:
       yield n
       if n < 0:
           n -= 1
       else:
           n += 1
       n *= -1
```

itertools - pairwise

```python
from itertools import izip

```

```python
def pairwise(iterable):
    "s -> (s0,s1), (s2,s3), (s4, s5), ..."
    a = iter(iterable)
    return izip(a, a)

l = [1, 2, 3, 4, 5, 6, 7]
for x, y in pairwise(l):
   print("{} + {} = {}".format(x, y, x + y))

# 1 + 2 = 3
# 3 + 4 = 7
# 5 + 6 = 11
```

Every 2 element from a list. We are using the exact same iterator object in both places of the izip() call,
so very time izip() wants to return a tuple, it will fetch two elements from the same iterator.

[Iterating over every two elements in a list]

itertools - grouped

Every N element from a list

```python
from itertools import izip

def grouped(iterable, n):
    '''s -> (s0,s1,s2,...sn-1),
          (sn,sn+1,sn+2,...s2n-1),
          (s2n,s2n+1,s2n+2,...s3n-1), ...'''
    a = iter(iterable)
    iterators = [a] * n
    return izip(*iterators)

l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for x, y, z in grouped(l, 3):
   print("{} + {} + {} = {}".format(x, y, z, x + y + z))

# 1 + 2 + 3 = 6
```

```
# 4 + 5 + 6 = 15
# 7 + 8 + 9 = 24
```

range vs xrange in Python

```
from __future__ import print_function
import sys

r = range(1000)
x = xrange(1000)

for v in r:    # 0..999
    pass
for v in x:    # 0..999
    pass

print(sys.getsizeof(r))   # 8072
print(sys.getsizeof(x))   # 40
```

In Python 2 `range` creates a list of values `range(from, to, step)` and `xrnage` creates and iterator.
In Python 3 `range` creates the iterator and if really necesary then `list(range())` can create the list.

[range vs. xrange in Python]()

profile (with hotshot) slow code

It was experimental and dropped from Python 3

- [](https://docs.python.org/2/library/hotshot.html)

```
import slow
import os
import hotshot, hotshot.stats

prof = hotshot.Profile("slow.prof")
prof.runcall(slow.main, 1000)
```

```
prof.close()
stats = hotshot.stats.load("slow.prof")
stats.strip_dirs()
stats.sort_stats('time', 'calls')
stats.print_stats(20)

os.remove("slow.prof")
```

```
         501501 function calls in 0.337 seconds

   Ordered by: internal time, call count

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   498501    0.192    0.000    0.192    0.000 slow.py:37(swap)
        1    0.136    0.136    0.335    0.335 slow.py:21(sort)
      999    0.006    0.000    0.006    0.000 slow.py:4(f)
      999    0.002    0.000    0.002    0.000 random.py:173(randrange)
        1    0.001    0.001    0.003    0.003 slow.py:31(get_str)
      999    0.000    0.000    0.000    0.000 slow.py:10(g)
        1    0.000    0.000    0.337    0.337 slow.py:14(main)
        0    0.000             0.000          profile:0(profiler)
```

Abstract Base Class without abc

Only works in Python 2?

```
import inspect

class Base():
    def __init__(self, *args, **kwargs):
        if self.__class__.__name__ == 'Base':
            raise Exception('You are required to subclass the {} class'
```

```
                    .format('Base'))

        methods = set([ x[0] for x in
            inspect.getmembers(self.__class__, predicate=inspect.ismethod)])
        required = set(['foo', 'bar'])
        if not required.issubset( methods ):
            missing = required - methods
            raise Exception("Requried method '{}' is not implemented in '{}'"
                    .format(', '.join(missing), self.__class__.__name__))


class Real(Base):
    def foo(self):
        print('foo in Real')
    def bar(self):
        print('bar in Real')
    def other(self):
        pass

class Fake(Base):
# user can hide the __init__ method of the parent class:
#    def __init__(self):
#        pass
    def foo(self):
        print('foo in Fake')

r = Real()
#b = Base()  # You are required to subclass the Base class
#f = Fake()  # Requried method 'bar' is not implemented in class 'Fake'
```

Abstract Base Class with abc Python 2 ?

```
from abc import ABCMeta, abstractmethod

#class Base(metaclass = ABCMet):
class Base():
    __metaclass__ = ABCMeta

```

```python
    @abstractmethod
    def foo(self):
        pass

    @abstractmethod
    def bar(self):
        pass


class Real(Base):
    def foo(self):
        print('foo in Real')
    def bar(self):
        print('bar in Real')
    def other(self):
        pass

class Fake(Base):
    def foo(self):
        print('foo in Fake')

r = Real()
f = Fake()
   # TypeError: Can't instantiate abstract class Fake with abstract methods bar
```

- [Abstract Base Classes in Python]()
- [abc]()

Abstract Base Class with metaclass

```python
import inspect
class MyABC(type):
    def __init__(class_object, *args):
        #print('Meta.__init__')
        #print(class_object)
        #print(args)
            # ('Base',
            # (<type 'object'>,),
            # {
            #   '__required_methods__': ['foo', 'bar'],
            #   '__module__': '__main__',
```

```python
            #   '__metaclass__': <class '__main__.MyABC'>
            # })
#        attr = dict(args)
        if not '__metaclass__' in args[2]:
            return

        if not '__required_methods__' in args[2]:
             raise Exception("No __required_methods__")
        name = args[0]
        required_methods = set(args[2]['__required_methods__'])
        def my_init(self, *args, **kwargs):
            if self.__class__.__name__ == name:
                raise Exception("You are required to subclass the '{}' class"
                    .format(name))

            #print("my_init")
            methods = set([ x[0] for x in
                inspect.getmembers(self.__class__, predicate=inspect.ismethod)])
            if not required_methods.issubset( methods ):
                missing = required_methods - methods
                raise Exception("Requried method '{}' is not implemented in '{}'"
                    .format(', '.join(missing), self.__class__.__name__))

        class_object.__init__ = my_init


class Base(object):
    __metaclass__ = MyABC
    __required_methods__ = ['foo', 'bar']

# b = Base() # Exception: You are required to subclass the 'Base' class

class Real(Base):
    def foo():
        pass
    def bar():
        pass

r = Real()
```

```python

class Fake(Base):
    def foo():
        pass

#f = Fake() # Exception: Requried method 'bar' is not implemented in class 'Fake'

class UnFake(Fake):
    def bar():
        pass

uf = UnFake()
```

Create class with metaclass

```python
class M(type):
    pass

class A(object):
    pass

class B(object):
    __metaclass__ = M

a = A()
print(type(a))
b = B()
print(type(b))



class Meta(type):
    def __init__(self, *args, **kwargs):
        print('Meta.__init__')
        print(self)    # <class '__main__.C'>
        print(args)    # ('C', (<type 'object'>,),
                       # {'__module__': '__main__',
                       #  '__metaclass__': <class '__main__.Meta'>})
        print(kwargs) # {}

class C(object):
```

```
    __metaclass__ = Meta

c = C()
print(type(c))

class MyABC(type):
    def __init__(self, *args):
        print('Meta.__init__')
        print(args)    # ('C', (<type 'object'>,),
                       # {'__module__': '__main__',
                       # '__metaclass__': <class '__main__.Meta'>})

class Base(object):
    __metaclass__ = MyABC
```

```
# http://stackoverflow.com/questions/100003/what-is-a-metaclass-in-python

# Create a new-style class
class A(object):
    pass
print(type(A))          # <type 'type'>
a = A()
print(type(a))          # <class '__main__.A'>

B = type('B', (), {})
print(type(B))          # <type 'type'>
b = B()
print(type(b))          # <class '__main__.B'>

# old style
class C():
    pass
print(type(C))         # <type 'classobj'>
c = C()
print(type(c))         # <type 'instance'>

# Have attributes in the class
class AA(object):
    name = 'Foo'
print(AA.name)    # Foo
aa = AA()
print(aa.name)    # Foo
```

```
BB = type('BB', (), {'name' : 'Bar'})
print(BB.name)    # Bar
bb = BB()
print(bb.name)    # Bar


# Inherit from a class
class AAA(AA):
    pass
print(AAA.name) # Foo
aaa = AAA()
print(aaa.name) # Foo

BBB = type('BBB', (BB,), {})
print(BB.name)   # Bar
bbb = BBB()
print(bbb.name) # Bar


def f(self):
    print(self.name)

class AAAA(object):
    name = 'AAAA-Foo'
    def show(self):
        print(self.name)

aaaa = AAAA()
aaaa.show() # AAAA-Foo

BBBB = type('BBBB', (), { 'name': 'BBBB-Bar', 'show' : f})
bbbb = BBBB()
bbbb.show() # BBBB-Bar
```

- [what is a metaclass]()

Python Descriptors

A more manual way to implement the property() functionality we have just seen.
Use cases:

- Implement type-checking and/or value checking for attribute setters ()

- [Descriptors]()
- [Descriptor HowTo Guide]()

alter iterator

Is this interesting at all ?

```python
from my_iterators import alter

for a in alter():
    print(a)
    if a >= 6:
        break

# 1
# -2
# 3
# -4
# 5
# -6
# 7
```

Create a counter queue

```python
import threading
import Queue

class ThreadedCount(threading.Thread):
    def __init__(self, name, start, stop):
        threading.Thread.__init__(self)
        self.name = name
```

```
        self.counter = start
        self.limit = stop
    def run(self):
        while self.counter < self.limit:
            self.counter += 1
            print(self.name, self.counter)

        print(self.name , "finished")
        return

queue = Queue()
foo = ThreadedCount("Foo", 1, 10)
bar = ThreadedCount("Bar", 1, 10)
foo.start()
bar.start()
print("main - running")

foo.join()
bar.join()
print("main - thread is done")
```

A Queue of tasks

```
from queue import Queue
from threading import Thread

def source():
    """Returning the list of tasks"""
    return range(1, 10)

def do_work(item):
    print("Working on item " + str(item) + "\n", end="")
# print("Working on item ", str(item))
# would show the output intermingled as the separate items of the print statement
# (even the trailing newline) might be printed only after context switch


def worker():
    while True:
        item = q.get()
        do_work(item)
```

```
        q.task_done()

def main():
    for i in range(num_worker_threads):
        t = Thread(target=worker)
        t.daemon = True
        t.start()

    for item in source():
        q.put(item)

    q.join()       # block until all tasks are done

num_worker_threads = 3
q = Queue()
main()
```

Filtered Fibonacci with ifilter

```
from series import fibonacci
from itertools import ifilter

even = ifilter(  lambda f: f % 2 == 0, fibonacci()  )
for e in even:
    print(e)
    if e > 200:
        break
```

Python from .NET

TODO and add to dotnet

TODO: example with async call in .NET getting back to python