**Everything is from this website:** <http://www.learnpython.org/>

**Notes:**

* It seems that running “***python filename.py”*** actually compiles AND runs it, so you don’t need to compile and run separately
* Within a class, you need to define all your methods BEFORE you make use of them
  + Or define them in another class and instantiate a variable of that class I guess
* ***ord(‘a’)*** can be used to convert single characters into their Unicode value!
  + ***char(number)*** does the opposite
* You can sort a list by doing ***list.sort()*** OR by doing ***sorted(list)***
* The ***this*** operator in java is basically the ***self*** operator in Python

**Learnpython.org**

**Learn the Basics:**

**Hello, World!**

* There are two major versions of Python: Python 2 and Python 3
  + This tutorial uses Python 2 which is more widely used and supported
  + Python 3 is more semantically correct and supports newer features
* Print stuff with: ***print “the text you want to print”***
* You don’t declare the type of your variables: ***x = 1*** is fine
* Also, you don’t add a semicolon at the end
* You comment things out with #
* **If statements:**
  + You don’t use curly braces, use spaces or tabs instead
  + Multiple lines with the same spacing inside an if statement will all be run
  + End your if and else statements with a colon
  + An if statement will proceed if there is the Boolean true or just a non-empty object
    - Empty objects: “”, [], 0, False
* **Enhanced for loops:**
  + ***for x in myList: print x***
  + myList is a list, obviously

**Variables and Types:**

* Python is object oriented and not “statically typed”: you do not need to declare variables before using them
* Every variable in python is an object
* There are the following types:
  + **Numbers:** There are integers and floating point numbers and complex numbers
    - **Integers:** Just write it out: ***anumber = 8***
    - **Floating point number:** Two ways to do it:
      * ***anumber = 8.0***
      * ***anumber = float(8)***
  + **Strings:** You can define them with either a single or a double quote. You would use double quotes instead of single ones if you needed to include apostrophes because they would terminate the string with single quotes
    - ***aString = ‘hello’***
    - ***aString = “hello”***
    - You can also use a backslash within your quotes to add one that would be ignored: ***‘doesn\’t’*** gives “doesn’t”
    - But if you add double quotes of whatever around the word with your backslash, then it keeps it
    - You concatenate strings the same way you do in java, just + them
    - BUT YOU CAN’T COCATENATE OBJECTS OF DIFFERENT TYPES IN PYTHON LIKE YOU DO IN JAVA (because java just toString them, python doesn’t)
    - You can do str(123) to make “123”
  + It seems that, if you want to add stuff in your string, you do: “***blablabla %s” %stringName*** (similar to java)
    - ***%s:***for Strings or any object that has a string representation (toString()), like lists. It seems you can also actually do it for floating points.
    - ***%d:*** For integers
    - ***%f:*** Floating point numbers
    - ***%.<number of digits>f***: Floating point numbers with fixed amount of digits after dot
    - ***%x/%X:*** Integers in hex representation (lowercase/uppercase)
    - If you’re adding multiple variables, pur parentheses around them after the % outside the quotes
* You can assign multiple variables at once: ***“a, b = 3, 4”***
  + **print a** will print “3”, **print b** will print “4”
  + **print a + b** would print “7”
  + **print a, b** would print “3, 4”
* You can’t just add variables of different types, but you can ***print var1, var2, var3*, etc.** even if they’re different types
* You can use **==** to test the content for strings, ints and floating points
* There is also a ***isinstance(variable, type)*** method

**Lists:**

* Lists are like arrays, they can contain any type of variable and as many of them as you wish.
* They can be iterated over simply
* Can be created in two ways:
  + ***myList = []*** and on the next lines: ***myList.append(1) myList.append(2)* etc.**
  + ***myList = [1, 2, 3]***
* Access data like in arrays: ***myList[1]***
* You can just print lists: ***print(myList)***
* Test the length of a list: ***len(myList)***
* Count the number of times variable x is in the list: ***myList.count(x)***

**Basic Operators:**

* Works almost the exact same way as in java, even the modulo operator is the same (%)
* To raise something to a power, do \*\*: ***7 \*\* 2***  = 49
* You can multiply a string by an integer to make it repeat!!!:
  + **var1 = “hello” \* 4** will make var1 equal to hellohellohellohello
* Adding lists jut concatenates them
* Multiplying a list by an integer concatenates that list with itself (like with strings)

**Basic String Operations:**

* You can also use the len() function on strings
* You can do ***aString.index(“o”)***and it will return the index where it first occurs
  + You can do strings in the parentheses
* You can also use .count(something)
* ***aString[3:5]***  returns the things in aString from indices 3 to 5
  + Excludes the 3rd and includes the 5th index
  + Leave out the first and second number but leave the colon to get the string from the start to number or number to end
  + ***[start:stop:skip]*** is the format if you want specific parts
    - You need to do this backwards to print a string in reverse
    - E.g. Reverse the entire string with ***[::-1]***
* There are ***.upper()*** and ***.lower()*** functions
* There are ***.startswith(“string”)***  and ***.endswith(“string”)***
* There is a ***.split(“splitter”)*** function that returns a list with all the words split by the splitter
* You can just get the character at an index using square brackets ***srt[2]***

**Conditions:**

* Not equals is also ***!=***
* **The in operator!!!:**
  + You can use ***in*** to test if specific values or variables are in an iterable object container: ***if name in [“John”, “Rick”]***
* **The is operator!:**
  + == tests that values inside the variables (.equals() in java, unless there’s no subclass implementation, in which case .equals() is the same as ==)
  + ***is*** tests the instances themselves (== in java)

**Loops:**

* **For loops:**
  + We already saw how the loops iterate over a given sequence (first page, enhanced for loop)
  + To make loops iterate over a sequence of numbers, use ***range(int)*** or ***xrange(int)***
    - range(int) returns a list with numbers of the specified range
    - xrange(int) returns an iterator, it is more efficient
  + E.g. ***for x in xrange(5)*** will run through it 5 times, where x will be 0, 1, 2, 3 and 4
  + E.g. ***for x in xrange(3, 6)*** will run through it 3 times, x will be 3, 4 and 5
  + E.g. ***for x in xrange(3,8, 2)*** x will be 3, 5 and 7
* **While loops:**
  + Basically what you’d expect**: *while count < 5 …*** if you’ve initialized count before hand
  + Then ***count += 1*** within your while loop
  + DON’T FORGET THE SEMICOLONS AT THE END OF YOUR WHILE LINE
* **The “break” and “continue” statements**
  + ***break*** will break out of the while and if loops it’s in
  + ***continue*** will skip the rest of that code block (“continue without the rest”)
  + else statements will still be executed if you continue before them (but not if you break before them)

**Functions:**

* Functions are a key way to define interfaces so programmers can share their code
* Functions in python are defined with the block keyword ***def*** followed with the function’s name
  + E.g. ***def my\_function(): #does stuff***
  + E.g. ***def my\_function\_with\_args(username, greeting): #does stuff***
* You can also use the ***return*** keyword (without : or anything, just return whatever)
* You call a function the way you would expect, with just ***my\_function()*** for example
* **pass** is a statement that is just skipped, it’s like it wasn’t there
  + Usually used to fill in for blocks of code that haven’t been written yet

**Classes and Objects:**

* You start a class like this: ***class MyClass: #does stuff***
  + There’s no need to close it at the end or anything
* You put classes in objects here: ***myObjectx = MyClass()***
  + And then myObjectx holds an object of the class MyClass that contains the variable and the function defined within MyClass
* You can access variables within these objects: ***myObjectx.variable***
  + So it’s basically exactly the same as in java: you can make a class with instance variables and methods and just create an instance of that class to access them
* If you define multiple objects of the same class, they will have independent instance variables (just like in java, unless you make it static)
* You access a function inside a variable the same way: ***myObjectx.function()***

**Dictionaries:**

* I guess they’re like sets and maps in Java
* Dictionaries are similar to arrays but work with keys and values instead of indexes
  + The key can be any type of object
* You can define a dictionary like this (the example is a phonebook): ***phonebook = {}***
  + And then you add values with: ***phonebook[“John”] = 938477566***
* Or you can just do it all in one go: ***phonebook = { “John”: 938477566, “Matthieu”: 6784676771 }***
  + To retrieve an item: ***phonebook[“John”]***, which will return 938477566
* You can iterate over dictionaries: ***for name, number in phonebook.iteritems(): #do stuff***
  + Kind of like how you make an iterator in java
* You can remove an item in two ways:
  + ***del phonebook[“John”]***
  + ***phonebook.pop(“John”)***
* You can actually have keys of different types and it will work just fine
* You can just do ***if “John” in phonebook: #do stuff***
* ***phonebook.clear()*** to remove all items
* ***del phonebook*** to delete the entire dictionary

**Modules and Packages:**

* Modules are just python files with the py extension
* Modules are imported from each other using ***import***
* The first time you load a module into your Python script it is initialized by executing the code one
  + If another module in your code imports the same madule, it will still only be loaded once
* WHEN YOU IMPORT, YOU ARE ONLY IMPORTING THE MODULE, NOT THE CLASSES INSIDE
  + SO, IF YOU JUST IMPORT THE MODULE, YOU INSTANTIATE LIKE: ***var = module.class()***
  + OR YOU CAN DO ***from module import class***
    - You can replace class with \* to just import everything
  + OR **import module.class**
    - NOT QUITE THOUGH, JUST DON”T USE THAT ONE
* **dir(classInModule)** Will give you an array with all the methods, variables and whatever in the class, dir(Module) would not work
* **help(classInModule.thing)** will give you info about whatever “thing” is. If could be a variable, method, class, module, etc?

Packages

* Packages are namespaces which contain multiple packages and modules themselves
* Packages are basically directories but they need to contain a file called ***\_\_init\_\_.py*** which means that the directory is a Python package and can be imported
  + It can be empty
* **Note: If you import a package and all it’s modules, you still have to import the module’s classes to be able to use them!!!**
* I think you can also change which modules are exported using the \_\_init\_\_.py file: ***\_\_all\_\_ = [“bar”]***

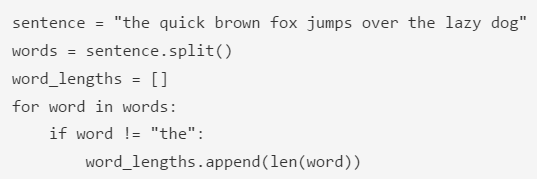
**Advanced Tutorials**

**Generators:**

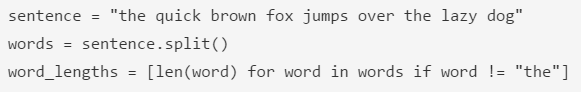
* Used to create iterators but with a different approach
* Generators are functions that return an iterabe set of items, one at a time, in a special way
* When you use a for loop over a set of items, the generator begins to run
  + It won’t do anything unless it reaches a ***yield*** statement

**List Comprehensions:**

* When you create a new list based on another list using a single, readable line
* E.g. Say we need to create a list of integers that describe the lengths of each word in a sentence if the word is not “the”



* You can do that all in one line



**Multiple Function Arguments:**

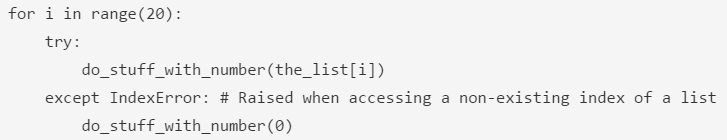
* Basically when you write a function that can take in a varying number of arguments (like varargs in java)
* ***def method(first, second, third, \*others): #do stuff***
  + Note: in java, \*others would be type…others
* You can also make functions that take in arguments: ***def function(first, second, third, \*\*options):***
  + ***Function(1, 2, 3, var1 = “hi”, var2 = “there”)***
* If you do this second thing, you get the name of the **content** of the variables with ***options.get(“var1”)***
  + I think it assumes you already know the name of var1 and var 2 though

**Regular Expressions:**

* Sometimes called regexp, regex or re
* Use them as last resort, they get fairly complex
* Okay I skipped this because it’s pretty hard
  + I’ll get back to it when I’m a lot better at Python

**Exception Handling:**

* You have *try-except* blocks (instead of try-catch)
* Say you want to iterate over a list of 20 numbers but you have fewer than that in the list:



* An exception clause may name multiple errors: ***except (RuntimeError, TypeError, NameError):***
* You can also ***raise*** (throw) an exception, it will have that name

**Sets:**

* It seems you can just do ***set(#Some list or string)*** and it will return a set
* A set is a list with no duplicate entries
* E.g. ***set(“my name is Matthieu and Matthieu is my name”.split())*** will return a list containing “my”, “name”, “is”, “Matthieu”, “and”.
* You can find out which variables are in two lists: ***a.intersection(b)***
  + Which will return a set with the variables that are in both a and b
* You also find out which variables are in only one of the lists: ***a.symmetric\_difference(b)***
* You can find out which events attended only the list before the dot: ***a.difference(b)***
* You can get a list of all variables in both lists: ***a.union(b)***
* Note: YOU CAN ONLY PERFOR ALL THESE OPERATIONS ON SETS, NOT ON LISTS

**Serialization:**

* Python has built-in JSON libraries to encode and decode JSON
* ***import json***
* Skipped the rest, idk JavaScript anyway

**Partial Functions:**

* There’s a tool to make partial functions: ***from functools import partial***
* Partial functions are used to create a function with fewer parameters from one with more parameters
* So, basically, say you have a function that needs 4 variables: ***def func(u, v, w, x): #does stuff***
  + You can define a function ***p*** with fewer variables: ***p = partial(func, 5, 6, 7)***
  + Note that the variables you fix fill up from left to right, so u = 5, v = 6, w = 7
  + And then you can call: ***p(8)*** for example! And x = 8!

**Code Introspection:**

* Basically just using functions to find out more about other functions or classes
* You can find out if certain objects have specific attributes or callable methods, etc.
  + ***help()***
  + ***dir()***
  + ***hasattr()***
  + ***id()***
  + ***type()***
  + ***repr()***
  + ***callable()***
  + ***issubclass()***
  + ***isinstance()***
  + ***\_\_doc\_\_***
  + ***\_\_name\_\_***

**Closures:**

* **Skipped**

**Decorators:**

* **Skipped**