# **Python - Things to Remember**

* Anything you just write in a python file, even if it’s not a class will start executing (except things inside functions themselves)
* You can define functions in a python file even if it isn’t a class (I’m calling those scripts)
* The functions must be defined before they’re used to work, for both classes and scripts

Corey Schafer Tutorials

# Programming Terms Tutorials

## Programming Terms: First Class Functions

* Important to understand things like Higher – order functions, currying?, and closures
* From the Wikipedia definitions, we can deduce that a First Class Function is one that is treated as a “**First class Citizen**”, meaning it can be passed as an argument, returned from a function, and assigned to a variable
* Example:



* Functions can also be passed in as arguments and returned from other functions
  + **Those are called higher order functions (if they take in or return functions)**



* You can also return functions from functions:



## Programming Terms: Closures – How to Use Them and Why They Are Useful

* The last example in the First Class Functions tutorial (in .py file but not in doc is a good example of closure)
* A closure seems to be that a variable defined in an outer function but used in an inner function is accessible to the inner function even AFTER it has been returned
  + So if your outer function returns the inner function, and that inner function uses a variable defined in the outer function, then the fact that the inner function can still use it makes this a closure
* The .py file also contains a more complex example that creates a log file every time the returned function is called

# Python Tutorials

# CS – 3600 Assignment 1

* “’<string>’.upper()”
  + Capitalizes every letter
* “‘<string>’.lower()”
* “dir(<variable>)”
  + In interpreter mode, this will give you all the functions available for variables of that type
  + Would be useful to learn some of these!
* “help(<variable>.<functionWithoutParenthesis>)”
* Use the + operator to concatenate lists and strings
* Negative indexing is allowed for strings (e.g. aList[-1] returns the last item in the list)
* You can plut out multiple elements from a list with: aList[start:end]
* “<listName>.pop()”
  + Important! Removes last element in the list
* You can distribute out a list! If “a = [1, 35, 53]”, then “x, y, z = a” will distribute the three values to x, y, and z
  + Also works with tuples

**Sets**

* A set is a list with no repeating element, you can declare it from a set
* “<varName> = set(<list>)”
* You can do addition, subtraction, &, |, etc. operations on sets to find their unions, intersections, etc.

**Dictionaries**

* Can have keys of various types
* “*del dicoName[‘<key>’]”* to delete an entry
* A value can be a list, tuple, etc.
* Don’t forget to do “*dico.items()*” if you’re going to iterate through a dictionary

**Map and filter:**

* This could be very useful:

****

* In python, you have to import a file to be able to use it to declare instances of any of the classes inside it “*import <fileNameWithout.py>*
* REMEMBER THAT THE FIRST ARGUMENT OF ALL FUNCTIONS IS “*self”*
* In a file, a variable will be static if you define it in a class but outside of a function
  + You can’t define a variable outside of a class in a file and refer to it in the class
  + INTERESTING: look at person\_class.py, notice that to change the static variable, they do: *“<ClassName>.<StaticVarName>”* whereas, to edit a variable specific to an instance, they do: *“self.<VarName>*”
* If you’re in the interpreter and edited a file, you can reload it to account for the changes: “*reload(<fileWithout.py>”*
* You can do “global <variable name>” inside a function to make it accessible outside of the function
* “del <variable name>” will delete that variable
* When you define a function with input’s, you can set default values (e.g. “def myfunc(a = 4, c = None, num, b = [‘1’, ‘a’])”
  + Then you can call it and provide 1-4 inputs (in this case) and replace the default value or not and input parameters in reverse order even (e.g. “myfunc(num = 4, a = 2)”)
* You can also use \* in your definition to have a variable number of arguments
* Conditional in python is something like
  + Variable = “something” if (condition) else “something else”
  + E.g. st = “x is less than y” if (x < y) else “x is greater than or equal to y”
* Use “break” to terminate the function
* Use “continue” to skip the rest of the statement
* In a for loop, if you want a counter variable:
  + “for i, d in enumerate(<list>):”
* Classes
  + “class myClass():”
  + If you define a method within a class, it needs to take “self” as first variable
  + Instantiate with “c = myClass()”
* Inheritance: you must define a class with the class you’re inheriting from as variable “class anotherClass(myClass):”
  + Then it works the same way as in java (if you don’t define the method but call it, it will look in the parent class for that method)
  + (you can override a method by giving it the same name
* You can import libraries to get date and time
  + “from datetime import date”
  + “from datetime import time”
  + “from datetime import datetime”
  + Make a variable to get what you need: “today = date.today()”
    - You can get the specifics “today.day”, “today.month”, “today.year”
  + “datetime.now()” gives the date and time
  + There’s also a “timedelta” which is a period in time, you can add the current date to timedelta(days = …, months = …, years = …, hours = …, minutes = …) to find out when the time (after addition) will be
    - Or you can subtract
* String formatting
  + %

**Working with files**

* “f = open(“textfile.txt”, “w+”)”
  + First input is file name
  + Second input is permissions you want on file
  + “w” means write, + means create file if it doesn’t exist
    - “f.write(“…”)”
  + “a” means append (you only add stuff after what’s already there)
  + “r” means read
    - “contents = f.read()”
* You can check what mode it’s opened in with “f.mode”, which returns a string “r”, “a”, or “w”
* Don’t forget to “f.close()”
* You can actually find out what the path of a file is:
  + “from os import path”
  + “path.realpath(<filename>)” gives you the path
  + Cast it as a string: “str(path.realpath(“<filename>”)”
  + You can also do things like “path.exists(“<filename”>)”, “path.isfile(…)” or “path.isdir(…)” to see if the file exists, if it’s a file or if it’s a directory
  + Use “path.split(“<filename>”)” to separate the path from the filename: “head, tail = path.split(“<filename>”)”
* You can combine this with the time operations we saw earlier
  + You can find when it was last modified “path.getmtime(“<filename>”)”
  + But you have to convert that to a time using the .ctime function of the time import: “time.ctime(path.getmtime(“<filename>”))”
  + …
* You can use shutil to copy files to another location: “shutil.copy(src, dst)” where src and dst are strings of a source file and a destination directory
  + But IMPORT SHUTIL FIRST: “import shutil”
  + .copystat should copy all the content as well?
* You can rename files: “os.rename(“originalName.extension”, “newName.extension”)”
* You can put things into a ZIP archive “shutil.make\_archive(“<name>”, “zip”, <directory>)”
  + I think you need to “from shutil import make\_archive”
  + <name> is what you want to name this zip file
  + You can replace zip with tar to make tar files in unix
  + The directory is the one with everything that will be archived
* You can include specific files to a zip archive instead of everything in a directory:
  + “from zipfile import Zipfile”
  + “with ZipFile(“archiveName.zip”, “w”) as newzip:”
  + “newzip.write(“<file1>”)”
  + “newzip.write(“<file2>”)”

**Working with Web data**

* “import urllib2”
  + A module to connect to web addresses and recover data from them
* You must declare a variable: “variable = urllib2.urlopen(“<url>”)”
* You should check that it worked correctly by getting the code for that variable “webUrl.getcode())” – should be equal to 200 if everything worked out right
* …