Functions

March 23, 2017

1 Functions, Modules, Classes, and Files

This section provides a quick overview of the following topics:

- Functions
- Modules
- Classes
- Reading from files and writing to files

1.1 Functions

Functions are reusable code blocks. Functions have specific names and can be called any number of times during a program. The following is an example of a function which adds two numbers.

```
In [1]: def add2(x,y):
            a = x + y
            return a

        b = add2(5,10)
        print(b)
        print(add2(20,10))
```

```
15
30
```

Functions are defined using the **def** keyword followed by the function name and paranthesis. In the above example, add2 is the function name. The paranthesis usually contains the arguments or parameters which are inputs for the code blocks inside the function. For the above example, there are two parameters x and y. The function also contains a return statement which passes an object back to the main program. In the above example, the function returns a which contains the sum of x and y. A function need not always return an object or can have multiple return statements as show below.

```
In [2]: def fact(x):
            if x == 0:
                return 1
            else:
```

```
            return x*fact(x-1)

        b = fact(5)
        print(b)
```

```
120
```

The above fact function is an example of recursive function where the same function is called within the function. Functions can be used to return objects other than variables. The following function returns a list.

```
In [3]: def listsort(x):
            x.sort()
            return x

        x = [10,9,8]
        print(listsort(x))
```

```
[8, 9, 10]
```

For a variable, changes made within the function will not affect its value outside of the function. In the example given below, even though we are doubling the value of x inside the function, the value of x outside the function remains the same.

```
In [4]: def doublex(x):
            x = x*2
            return x

        x = 10
        print(doublex(x))
        print(x)
```

```
20
10
```

Functions can also be used in the case where the number of arguments are not fixed.

```
In [5]: def mult(*x, **y):
            product = 1
            for i in x:
                product = product*i
            for i,j in y.items():
                product = product*j
            return product

        print(mult(1, 2, 3, austin=4, pittsburgh=5))
```

```
120
```

When we have an argument called *arg, then it is collected as a tuple. When we have an argument as **arg, then the corresponding values are colelcted as a dictionary.

2 Modules

A python module is a file containing python variables, statements, and functions. YOu can write your own python scrpit, save it as a filename.py, place it in the same directory as your current code and use it using the **import** function. Python has a Standard Library which is a collection of modules providing it with the various functionality. For example, the math module provides us access to the various math functions.

For example, the code below provides us access to the logarithm function from the math module.

```
In [6]: import math
        x = math.log(10)
        print(x)
```

```
2.302585092994046
```

If we want to avoid using math prefix, we can import everything in the math module as shown below.

```
In [7]: from math import *
        x = log(10)
        print(x)
```

```
2.302585092994046
```

The above approach is not recommended for large programs. Ideally you import only the functions you need.

```
In [8]: from math import log
        x = log(10)
        print(x)
```

```
2.302585092994046
```

Another way to import is shown below.

```
In [9]: import math as mt
        x = mt.log(10)
        print(x)
```

```
2.302585092994046
```

3 Classes

Classes can be thought of as a collection of variables, data structures, and functions. Classes can be used to design new objects. Classes help organize variables, data structures, and functions which operate on these variables and data structure in a compact manner

```
In [10]: class Student:
             studentCount = 0
             def __init__(self, name, exam1, exam2):
                 self.name = name
                 self.exam1 = exam1
                 self.exam2 = exam2
                 Student.studentCount = Student.studentCount + 1
             def displayStudent(self):
                 print("Name : ", self.name, ", Score in Exam 1: ", self.exam1, ",
             def displayStudentCount(self):
                 print("Total Number of Students: ", Student.studentCount)
             def examAverage(self):
                 return 0.5*(self.exam1 + self.exam2)

         a = Student("Tom", 80, 90)
         print("Student Count is ", a.studentCount)
         b = Student("Stacy", 90, 100)
         c = Student("Joe", 70, 90)

         print(a.name, a.examAverage())
         a.displayStudentCount()

         a.displayStudent()
         b.displayStudent()
         c.displayStudent()
```

```
Student Count is  1
Tom 85.0
Total Number of Students:  3
Name :  Tom , Score in Exam 1:  80 , Score in Exam 2:  90
Name :  Stacy , Score in Exam 1:  90 , Score in Exam 2:  100
Name :  Joe , Score in Exam 1:  70 , Score in Exam 2:  90
```

Classes are defined using the **class** keyword. In the above example **Student** is the name of the class. Note that Class names start with capital letters by convention. All the elements or members of the class are indented. In the first line we define a variable studentCount which is a member of the class and set it equal to 0.

The next line contains a function called **init**. This function is called a constructor which is called by default whenever you create an object of the class type. Constructors are used to initialize the variables or data structures defined in the class.

Note that the first argument or parameter in any function defined in a class is self. In the above program, an object of the type Student is created when you run the line **a = Student("Tom", 80,**

90). When an object is created, the constructur is called which initializes a.name to Tom, a.exam1 to 80, and a.exam2 to 90. The studentCount variable is increased to 1.

In the above code, we initialize three objects **a,b,** and **c**. The class members are accessed using the dot notation - **ObjectName.VariableName** or **ObjectName.FunctionName(Parameters)**.

In the above class, we have three functinos other than the constructor. The second function is to display information about the student. The third function displays the number of students and the fourth function returns the average score in two examinations.

4 Files

In many cases, you may have to read the input to your code from a file and write the output to a file. This section provides a quick overview of reading and writing to text files. Create a text file called "input.txt" with the following information and place it in the same directory as the python script or notebook:

From, To, Cost
1, 2, 3.0
1, 3, 4.0
2, 3, 3.0
3, 4, 5.0
2, 4, 4.0

```
In [11]: f = open("input.txt", "r")
         print(f.read())
         f.close()
```

```
From, To, Cost
1, 2, 3.0
1, 3, 4.0
2, 3, 3.0
3, 4, 5.0
2, 4, 4.0
```

The open command opens a file and returns a file object. In the example given above, the file object is stored in f. The first argument in the open command specifies the name of the file. The second argument specifies the mode. The common modes are:

- r - opening the file for reading
- w - opening the file for writing
- a - opening the file for appending, adding information to the end of the file
- r+ - opening the file for both reading and writing

Note that whenever you open a file for writing, it creates a completely new file. So if there was a file with information you dont want to be deleted, you want to open the file in append mode.

The f.read() reads the entire file and returns a string containing all the elements of the file. Once you open a file, do not forget to close the file using the close command.

You can also read the file, line by line using the readline() function.

```
In [12]: f = open("input.txt", "r")
         network = []
         line = f.readline() #This line reads the first line comprising of From,To,
         line = f.readline()
         while len(line):
             l = line.split(',')
             fromnode = int(l[0])
             to = int(l[1])
             cost = float(l[2])
             line = f.readline()
             network.append([fromnode, to, cost])
         f.close()
         print(network)
```

```
[[1, 2, 3.0], [1, 3, 4.0], [2, 3, 3.0], [3, 4, 5.0], [2, 4, 4.0]]
```

In the above code, we read line by line. Each line is split into components and stored in a list l. The elements of the list are then converted to integer and float values and stored in a multi-dimenstional list called network. Similarly we can use the open and write command to write stuff onto files.

```
In [13]: f = open("output.txt", "w")
         f.write("Testing output.\n")
         f.write("My attempt at writing to file in Python.")
         f.close()
```