Python Help Session

CSCI1820

February 9, 2017

Contents

1 Introduction **2**

2 Commenting **2**

3 Using the Sys library **2**
3.1 Explanation . . . 2
3.1.1 Example . . . 2

4 File I/O **3**
4.1 Printing to the Screen . . . 3
4.2 Opening and Closing Files . . . 3
4.2.1 The Open Function . . . 3
4.2.2 The Close Function . . . 3
4.2.3 The Write Function . . . 4

5 String Manipulation **4**
5.1 What is a string? . . . 4
5.2 Python Builtin Functions . . . 4
5.2.1 Creation . . . 4
5.2.2 Accessing . . . 4
5.2.3 Length . . . 4
5.2.4 Slicing . . . 5
5.2.5 Upper and Lower Casing . . . 5
5.2.6 Splitting . . . 5
5.2.7 Concatenation . . . 5

6 Dictionaries **6**
6.1 Overview of Useful Dictionary Features . . . 6

7 Looping **7**
7.1 For Loop . . . 7
7.2 Break . . . 7
7.3 Continue . . . 7
7.4 While Loop . . . 7
7.5 Nested Loop . . . 7

8 Introduction to Functions **8**

1 Introduction

This help session is a quick overview of Python! Hopefully by the end of this handout, you should have most, if not all, of the tools needed to complete our projects in CSCI1820. We will go over the following topics here:

a) Commenting in Python

b) using the sys library to use terminal inputs as parameters

c) file I/O which includes opening, reading, and outputting files

d) string manipulation

e) using dictionaries

f) basic looping

g) introduction to functions

2 Commenting

In code, comments are text in your program that are never executed but are there to help anyone who reads your code understand what it is doing. Think of code comments as annotations in a text. In Python, there are two types of comments:

- One-line comments: these are done using the "#" character. Everything on the line after the hashtag character is commented out:

```
# This is a comment
print 3  # This is also comment
```

- Block comments: these are done by surrounding all the text you want to comment out in triple quotation marks, like so:

```
"""All of this will be commented out in Python
even though it is on multiple lines, but
the following line won't be"""
print "Hi"
```

3 Using the Sys library

3.1 Explanation

This library is a collection of system-specific parameters and functions. That is, this module provides access to variables and information stored by the interpreter as well as to functions that closely interact with the interpreter. For our purposes, we will be using the variable **sys.argv**.

sys.argv gives you access to argv which is a list of command line arguments passed to a Python script. If you are familiar with other object oriented languages like Java, note that the Python equivalent of an array is a list. Almost everything in Python is stored using lists (as you will later see in the strings section). Now in order to understand how argv stores the command line, we will observe an example.

3.1.1 Example

Let's say that we have a program that has three parameters: two numbers and an operation. For example a command line might look like this:

```
~ terminal: python math.py 2 10 +
```

This command line calls upon the file *math.py* and then lists the parameters 2, 10, and +. From this command line the argv will be the following

Index	0	1	2	3
Value	'math.py'	'2'	'10'	'+'

There are a couple of things to note from this argv.

i. First we have that the first element of the list is not the parameters as we would intuitively think it is. Note that in the command line from the computers perspective, the file name, too, is an argument. **Thus, for our purposes, any parameters that we use start to get stored starting from argv[1].** For this example then, we have that the two numbers are argv[1] and argv[2], and the operation is argv[3].

ii. **Second, notice that all of the elements within the list are stored as strings or char lists.** What this means for us is that is we want to use the actual value of our parameters, we must first convert our parameters into the form we want them to be in. For example, to add 2 and 10, we must first convert the chars '2' and '10' to the integers 2 and 10.

4 File I/O

4.1 Printing to the Screen

The simplest way to print to the screen is to use the **print** statement. You can pass in none to multiple arguments separated by a comma to print out the arguments to standard output.

```
print "Sorin is really smart." , "he's fun too"
```

This line will produce the following into standard output once run

```
Sorin is really smart. he's fun too
```

4.2 Opening and Closing Files

4.2.1 The Open Function

For our purposes, this will most likely be the method you use the most from this section (other than print). Before you can read or write the file you need to use the Python built-in function, open(), to open the file. This function will create a **file** object which can be used to call upon many useful methods that will help you parse or manipulate the opened file. Here is the syntax

```
file_object = open(file_name [, access_mode])
```

Note that the file_name parameter is the name of the file that we are opening. Often times one of the command line inputs that we got from argv is the name of the file that we want to open. Now the access_mode is not a required parameter. That is you can still call this function without defining the access_mode parameter. What this parameter does, though, is that is sets the mode in which the file has been opened. Here are the list of some commonly used modes

a) r - opens the file for read-only

b) r+ - opens the file for both reading and writing

c) w - opens the file for writing only

4.2.2 The Close Function

The Python builtin function close() is not always necessary; however, it is always good practice to call a close for every open that you have. More specifically, at the end of your program after you have done all of the file manipulation, you should close all the files. What this does is that it prevents any other factors from the computer or the program from modifying the file. Here is the syntax

```
# Created a file called filename
filename = open(file)

# At the end of your program, before it terminates closes the file, filename
filename.close()
```

4.2.3 The Write Function

The Python builtin function write() writes any string to an open file. Also note that the write() function does not automatically add a newline character ("\n"). Here is the syntax

```
# Open a file
example = open("Sorin_Compliments.txt")
example.write("CSCI1820 is going to be great!")

# Close
example.close()
```

This will write into Sorin_Compliments.txt the following

```
CSCI1820 is going to be great!
```

5 String Manipulation

5.1 What is a string?

Note that unlike ava, Python stores its strings as lists of characters. More specifically, the strings are essentially arrays where each character has its own index. This makes Python a language where manipulating the string is very easy! Python recognizes that something is a string if it has either ' ' or " " surrounding it.

5.2 Python Builtin Functions

Note that Python has many more builtin functions for strings than the ones listed here. When doing projects, it may be useful to look at the documentation for some builtin commands you may need.

5.2.1 Creation

Note that you can simply create strings like all variables in Python - you simply declare it. For example

```
# Creating Word
word = "Hello World"
```

5.2.2 Accessing

Since a string is a list in python, we can use [] to access certain characters of the string. For example if we want the first letter of the word we would simply call on word[0]. For the above example then, we have that word[0] = H and word[6] = W.

5.2.3 Length

We use the builtin function **len** to find the length of the string (as well as any list).

```
word = "Hello World"
length_of_word = len(word)
# Print the length
print length_of_word
```

Once this program runs we will get

```
11
```

5.2.4 Slicing

Note that this also applies to any list. Before going into slicing, let me make a note about Python list indexing. A useful feature of Python is that all the lists also have back indexing. That is just like how the first element in a list has index 0, the last character of the list has index -1. Then the second to last element has index -2 and so on. This allows you to easily call upon the last elements without knowing the length of the whole list. Now for slicing, you can use [# : #] to get a set of elements from the list

1) word[start:end] - the elements starting from the start index to the end index -1

2) word[start:] - the elements starting from the start index to the end of the list

3) word[:end] - the elements from the beginning of the list to the end index -1

4) word[:] - a copy of the whole list

Here is an example to show this for the word = "Hello World"

```
word[1:3]  #"el"
word[4:]  #"o World"
word[:-1]  #"Hello Worl"
word[-3:]  #"rld"
word[:]  #"Hello World"
```

5.2.5 Upper and Lower Casing

Python has builtin commands to upper and lower case any of the characters in a string. They are **upper()** and **lower()**. Here is an example

```
word = "Computational Biology"

word.upper()  # results in "COMPUTATIONAL BIOLOGY"
word.lower()  # results in "computational biology"
```

5.2.6 Splitting

Often times we need to use the **split()** method to parse our arguments. Here is an example that illustrates how this method works

```
word = "I am a TA"

# Now we will split this string by ' ', or a space
word.split(' ') # results in ['I', 'am', 'a', 'TA']
```

Notice how we now have a list of strings that were separated by the delimiter which was a space in this case. The delimiter is a parameter of split() and can be anything.

5.2.7 Concatenation

Another very useful properties of strings is that we can concatenate them by simply using +. Here is an example

```
"hello" + " " + "world" + " " + str(2017) # results in "hello world 2017"
```

Note that the method str() is a useful method to convert integers to strings so that we can concatenate them like so. Also note that you can concatenate spaces as well.

6 Dictionaries

A dictionary is another structure in Python that is used to store data. Let us first look at a list. Note that a list can be viewed as a mapping between integers and the elements in the list. What I mean by this is that the integer indexes, 0, 1, 2,..., all map to the elements within the list. A dictionary on the other hand is a mapping between any object to the elements in the dictionary. The more technical term to describe this would be that a dictionary is a mapping between key and value pairs. Note that this is type of data structure is often called a hash map. Let us look at a few examples

```
# Create a dictionary
stuff = {'name': 'Jonathan', 'age' : 21, 'height' : 6*12}
print stuff['name'] # Jonathan
print stuff['age']  # 21
print stuff['height'] # 72

# Add a key/value pair to stuff
stuff['school'] = 'Brown University'
print stuff['school'] # Brown University
```

Note that the above example illustrates a few things.

1. We first have that in order to initialize a dictionary we use {} instead of [].
2. If we want to define mappings within the brackets we use the following syntax: {key: value, key: value, ...}.
3. If you want to add an element into the dictionary then you do: dictionary[key] = value
4. If you want to access values in the dictionary you call on it by its key: dictionary[key]

6.1 Overview of Useful Dictionary Features

```
# Key/value pair declaration
dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}

# Get all the keys
dict.keys()   # ['key1', 'key2', 'key3']

# Get all the values
dict.values()   # ['value1', 'value2', 'value3']

# Modifying the dictionary
dict['key2'] = 'value8'

# Empty Declaration and filling in the dictionary
emptyDict = {}
emptyDict['key1'] = 'value1'

# Size of the dictionary
len(dict)

# Deleting an entry and Clearing the dictionary
del dict['key1']
dict.clear()

# Looping through a dictionary
for key in dict.keys():
        print dict[key]
```

7 Looping

Looping is a crucial part of coding in any language and any style of coding. Here are the loops used most often in Python as well as some commands that will help with loop functionality.

7.1 For Loop

The for loop is used to iterate over elements of a sequence. It is often used when you have a piece of code which you want to repeat a certain number of times. Note that the builtin function range() is used often in the construction of for loops. Here is an example of the syntax of for loops

```
# Prints numbers 1 to 9
for i in range(1, 10):
    print i

# Print elements in a list
elements = [1, 10, 20, 30, 40]
for num in elements:
    print num
```

7.2 Break

To break out of a loop when you want it to before it reaches completion, you can use the command break

```
# Prints 1 and 2
for i in range(1, 10):
    if i == 3:
        break
    print i
```

7.3 Continue

The continue statement is used to skip over the rest of the statements in the current loop block and to move on to the next iteration.

```
# Prints numbers 1 through 9 except for the number 3
for i in range(1, 10):
    if i == 3:
       continue
    print i
```

7.4 While Loop

The while loop as the name suggests, is a loop that will continue to execute as long as the defined conditions are met. Here is an example of a while loop.

```
computer_brands = ["Apple", "Asus", "Dell", "Samsung"]
i = 0
while i < len(computer_brands):
    print computer_brands(i)
    i = i + 1
```

Notice in the above example, that there is a counter i that keeps track of how many times we execute the loop. For loops like while loops, we can often count the exact number of loops by creating a counter like this.

7.5 Nested Loop

Note that you may also nest loops or put loops within loops. You can play around with this in some exercises later.

8 Introduction to Functions

Functions are a way to partition blocks of code that have a certain functionality. By doing this, we can use these functions throughout the code. Here is a simple example where I will create a function that will get the average of a list of numbers.

```
# Define the function
def getAverage(list):
    total_sum = sum(list)
    length = len(list)
    print total_sum/length

# Use the defined function
print (getAverage([1, 2, 3, 4, 5])) # Prints out 3
print (getAverage([2, 10, 40, 100])) # Prints out 38
```