May 5, 2017

Excelsior - Deep Dive Topics

Python

- An interpreted, object oriented, high-level programming language
- Python has high level data structures (lists or arrays, dictionaries, tuples)
- Dynamic typing
- Supports modules and packages to encourage modularity (with import statement)
- Extensive libraries and open source
- Most Linux distributions & Mac OS systems have python installed by default:

```
$ python
```

Will start up the python interpreter. The interpreter is a fun way to test out some code. What version do you have?

Variables

Python is dynamically typed so a variable can be assigned two different values (not at the same time of course).

```
>>> x = 10
>>> x = “Excelsior Academy”
```

Both of these are legal and python won’t complain that at first x was an integer and then you assigned it to be a string. (The 3 <<< means you are in the interpreter)

Variable names usually have words separated by an underscore (python convention):

```
>>> first_name = “Joe”
>>> last_name = “Schmo”
```

Quit the interpreter by typing quit() or exit():

```
>>> quit()
```

Data Types

Python has many datatypes like booleans, strings, integers, floats, etc which you are probably familiar with from Javascript or other languages but the more exciting data types are tuples, lists, and dictionaries.

Boolean

This data type equates to simply `True` or `False`:

```
>>> is_hungry = True
>>> is_tired = False
```

Strings & String operations

```
>>> name = “Sampson”
>>> print len(name)
>>> max(name)
```

What does max(string) do?

Concatenating Strings with +

```
>>> print ( name + “ the Bulldog”)
```

```
>>> print 12 + “ makes a dozen”
```

What happened?

To fix we need to convert the integer to a string with str():

```
>>> print str(12) + “ Makes a dozen”
```

Using **in** to check for characters in a string:

```
>>> vowels = “aeiou”
>>> print “d” in vowels
```

Arrays / Lists

List allows you add, delete or process elements in very simple ways. List is very similar to arrays.

```
>>> my_hobbies = [“running”,”reading”, “gaming”]
```

Each element in the list has an index # beginning with 0.

0: running	1: reading	2: gaming

where `my_hobbies[0] = “running”, my_hobbies[1] = “reading”, my_hobbies[2] = “gaming”`. You can access any element in the list by referencing it’s index position:

```
>>> my_hobbies[0]
```

Lists don’t need to contain the same data type:

```
>>>  items = [“car”, 12, “chickens”]
```

You can also use **in** or **not in** to check for an item in a list:

```
>>> print “car” in items
>>> print “Steve” not in items
```

Other list operations

```
>>> items.append(“monkeys”)
>>> items.remove(“chickens”)
>>> items.reverse()
```

What does items look like here?

```
>>> items.sort()
```

Be careful! `reverse()` and `sort()` will change the list permanently! Use `sorted(list)` to just return a new sorted list but not change the old one.

```
>>>  items.reverse()
>>> sorted( items )
```

Tuples

In Python Tuples are very similar to list but once a tuple is created, you cannot add, delete, replace, reorder elements. (They will say “tuples are immutable” i.e. cannot be altered). Creating a tuple:

```
>>> t = ()
>>> t = (12,13,14)
```

operations:

```
>>> min( t )
>>> sum( t )
>>> len( t )
```

Tuple assignment:

```
>>> (name, age) = (“Steve”, 17)
>>> print name
```

Dictionaries

Dictionary is a python data type that is used to store key value pairs. It enables you to quickly retrieve, add, remove, modify, values using key. Dictionary is very similar to what we call associative array or hash on other languages.

Empty dictionary:

```
>>> my_iTunes = {}
```

Add entries:

```
>>> my_iTunes[‘Bruno Mars’] = ‘Thats what I like’
>>> my_iTunes[‘Ed Sheeren’] = ‘Shape of you’

>>> print my_iTunes
>>> print my_iTunes[‘Bruno Mars’]
```

Dictionaries can store lists to make them like a database:

```
>>> my_iTunes["Playlist one"] = { "Bruno Mars" : [ “Bruno song
1", “Bruno song 2"], "Ed Sheeren": ["Ed song1","ed song2”]}
>>> my_iTunes[“Playlist one”][“Bruno Mars”]
```

To remove something from the dictionary:

```
>>> del my_iTunes[“Playlist one”]
```

Python Loops

The most common loop is the for loop. Python separates blocks of code by using indentation (4 spaces is the convention). A for loop typically traverses a data type (list, tuple, dictionary) but can also be done a specific number of times:

```
>>> alphabet = “abcdefghijklmnopqrstuvwxyz”
>>> for alpha in alphabet:
```

block statements end in a colon (“:”) and each line in that block is indented 4 spaces

```
        print alpha


>>> my_list = [“apples”,”oranges”,”pears”]
>>> for item in my_list:
        print item
```

Print a range of numbers (from 0 through 9):

```
>>> for i in range(0,9):
      print i
```

Control Statements

It is very common for programs to execute statements based on some conditions. In this section we will learn about python `if .. else ...` statement. But before that we need to learn about relational operators. Relational operators allows us to compare two objects.

SYMBOL / DESCRIPTION

`<=` smaller/less than or equal to
`<` smaller/less than
`>` greater than
`>=` greater than or equal to
`==` equal to
`!=` not equal to

The result of comparison will always be a boolean value i.e `True` or `False` . Remember `True` and `False` are python keyword for denoting boolean values.

```
>>> x = 12.0
>>> y = 12
>>> print x == y
```

Question: What is the answer?

If / Else

```
>>> is_bored = False
>>> if is_bored:
      print “I am bored”
```

```
else:
  print “Python rocks!”
```

File Input / Output

Use curl to download the Declaration of Independence file from my github account:

```
$ curl https://raw.githubusercontent.com/PTech-ScubaTeam/linux-training/master/declaration.txt > declaration.txt
```

Opening a file

Before reading/writing you first need to open the file. Syntax of opening a file is.

```
f = open(filename, mode)
```

`open()` function accepts two arguments `filename` and `mode` . `filename` is a string argument which specify `filename` along with it's path and mode is also a string argument which is used to specify how file will be used i.e for reading or writing. And `f` is a file handler object also known as file pointer.

`modes` are:
`"r"` - Open a file for read only
`"w"` - Open a file for writing. If file already exists its data will be cleared before opening. Otherwise new file will be created
`"a"` - Opens a file in append mode i.e to write a data to the end of the file
`"wb"` - Open a file to write in binary mode
`"rb"` - Open a file to read in binary mode

Closing a file

After you have finished reading/writing to the file you need to close the file using `close()` method like this,

```
f.close()  # where f is a file pointer
```

The # symbol is a comment in python!

Writing to a file:

```
>>> f = open('myfile.txt', 'w')     # open file for writing mode = ‘w’
>>> f.write('this first line\n')    # write a line to the file
>>> f.write('this second line\n')   # write one more line to the file
```

```
>>> f.close()                          # close the file
```

Reading data from a file

To read data from a file you need one of these three methods.

```
read([number]) : Return specified number of characters from the
file. if omitted it will read the entire contents of the file.

readline() : Return the next line of the file.

readlines() : Read all the lines as a list of strings in the file
```

```
>>> f = open('declaration.txt', 'r')
>>> declaration_of_independence = f.read()
>>> f.close()
>>> print declaration_of_independence

>>> f = open('declaration.txt', 'r')
>>> declaration_array = f.readlines()
>>> f.close()
>>> print declaration_array
```

Question: What was the difference?

Putting it all together into a python program

Up until now we've been doing all of our work in the python interpreter. Let's put all of these tasks together into a python program (file with a .py extension).

Exercise: Let's say we want to create a wordle using the Declaration of Independence. How would we do this?

Pseudocode (i.e, in english):

- Read the file Declaration of Independence
- Break the file into words
- Remove any stop words (and, or, it , they, etc)
- Count up the occurrence of each word

* Create a file called *wordle.py*
(extra credit for the linux way: `touch wordle.py`)

```
#Import the string package - some nice functions to remove punctuation
import string

#List of stopwords
stopwords =
['ah','about','since','has','if','way','into','it','on','for','to','is','was'
,'any','so','of','as','and','or','but','are','the','a','that','they','in','ou
r','then','their','am','also','yet','not','very','we','after','do','this','ha
ve','been','he','know','some','him','her','what','can','just','an','why','whe
re','with','it\'s','oh','here','which','other','did','than','even','ever','ve
ry','get','like','there','by','from','one','even','though','no','had','be','o
k','because','might','much','too']

#Word List
words = []

#Word frequency dictionary ( will have word and # of times it appears)
frequency_dictionary = {}

#Open the file in READ mode 'r'
f =  open('declaration.txt','r')

#Read each line one at a time
for line in f.readlines():
    #strip the line (remove trailing newlines) and
    #split it up into words which are separated by spaces usually
    #parssed_words is now a list of words ["each","man"]
```

```
    parsed_words = line.strip().split(' ')

    #now we need to normalize (remove punctuation & lowercase)
    #and remove them if they are stopwords

    for word in parsed_words:
        lowercase_word = word.lower().strip( string.punctuation)

         #Add to the list of words now
        if lowercase_word not in stopwords and len(lowercase_word) > 0:
            words.append( lowercase_word)

words.sort()

#Go through the list and add them to the dictionary.
for w in words:
    if w not in frequency_dictionary:
        frequency_dictionary[w] = 1
    else:
        frequency_dictionary[w] = frequency_dictionary[w] + 1

f.close()
print frequency_dictionary

#We could have also only printed out words that appear more than 2 times:
for word,freq in frequency_dictionary.items():
    if freq > 2:
        print word + " [" + str(freq) + "]"
```

To run:

```
$ python wordle.py
```

Takeaways From Today

Python …

… is an interpreted, object oriented, high-level programming language
… is very powerful, fast, and has thousands of libraries
… commands do NOT end with semi-colons
… block commands end with colons
… indentation (4 spaces) delineates block commands
… uses the # to mark comments