lecture_5

March 14, 2017

1 Beginning Python Class -- Lecture 5

1.1 Expected Understanding

At this point in the class it is very important that you have a good understanding of the topics in Lecture_1 and Lecture_2. The topics today will build directly on the previous topics such as:

```
- Basic Data Types
- Conditionals
- Loops
- Functions
- User Defined Datatypes -- Classes
```

In this lecture I am going to go over a few common topics that you will encounter when looking/writing python and I would like you to know what is happening. These topics will feel more disjointed than in previous lectures but I hope they will help you understand what is happening when you encounter them outside of class. Some of these topics might be difficult to replicate in the lab because of running the code in a web browser (from my testing at home it seems that everything works in the web browser).

1.1.1 Running Python in the Lab

Not everyone has a dedicated Windows machine that they can remote into, and not everyone has their own Linux account. Therefore we are going to use an online Python Interpreter to run our Python code for this class. This way we can practice in the lab and students can practice at home without having to install python on their PC. I will run python from my machine during the lecture so that the process of running python from the command line is understood.

Here is the link to the online Python Interpreter -- `https://repl.it/languages/python3`

1.1.2 PEP8 Style Guide

PEP8 style guide should be used as a reference for styling decisions. This document (`http://legacy.python.org/dev/peps/pep-0008/`) covers topics like: - Indentation - Comments - Line Length These are a set of standards that allow Python developers to easily work on each others code and avoid nasty syntax bugs caused by white space inconsistencies.

1.1.3 Syntax

Syntax in the context of computer code is the set of rules that defines the combinations of symbols that are considered to be a correctly structured document or fragment in that language.

Python was designed to have a very clean look to the code. The compromise is that to correctly interpret the code Python elected to use white space as a syntactical character. The last language to use syntactical white space was Fortran which was created by IBM in 1957.

Languages such as Perl and C ignore white space and rely on other special characters that often mean different things in different contexts. This allowed the programmer to develop their own style independent of the format of the code.

A quick example is shown below:

```
foreach my $a (@list_of_numbers){
    print "value of a: $a\n";
}
```

The exact same Perl code could be written like:

```
foreach my $a (@list_of_numbers){ print "value of a: $a\n"; }
```

This is because one of the characters that Perl uses to group instructions are "{}" (brackets).

Python avoids this type of problem by requiring white space syntax to group sequences of instruction:

```
for a in list_of_numbers:
    print( "Value of a: " + str(a))
```

Since the 'print' statement is executed during the loop it is nested/grouped inside the loop with 4 spaces. (More on loops later in Lecture 2)

2 New Topics

2.1 Ternary Operator

The Ternary Operator is an operator that takes 3 arguments. This operator is often used during the assignment process. Assignment is what it is called when we take a variable and set it = to some value. We have done this many times without putting a name to it.

```
In [1]: name = 'Dallin' # Assignment
        print(name)

        is_big = True
        size = 'Big' if is_big else 'Small'
        print(size)

        is_big = False
        size = 'Big' if is_big else 'Small'
        print(size)
```

```
Dallin
Big
Small
```

In the example above we assign `name` equal to the string `'dallin'` with regular assignment. A little lower in the code we use the Ternary Operator to assign `size` to either the string `'Big'` or `'Small'` based on the condition `is_big`. The Ternary Operator is short-hand for the code below.

```
In [2]: is_big = True
        size = ''
        if is_big:
            size = 'Big'
        else:
            size = 'Small'
        print(size)

        is_big = False
        size = ''
        if is_big:
            size = 'Big'
        else:
            size = 'Small'
        print(size)
```

```
Big
Small
```

2.2 List Comprehension and Range Function

List Comprehension is a really handy tool when creating lists of data. They are very quick and are used all over the place in python code. Let's jump right in.

The range function is a function that was changed between python2 and python3. In python2, the `range` function returned a `list`, but in python3 the `range` function returns an iterator. Iterators are outside the scope of this class but you can read up on them here: https://www.programiz.com/python-programming/iterator. Therefore, the proper way to create a `list` from a `range` is using the `list()` function or using the `list` comprehension. See the first example below.

```
In [3]: x = [y for y in range(100)]
        print(x)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 2
```

We can selectively add numbers to a `list` using the `if` in the comprehension. If the conditional after the `if` is `True` then the value is added to the list otherwise it is ignored. See the example below where we are looking for prime numbers.

```
In [4]: def is_prime(x):
            if x <=1:
                return False
            for y in range(2,int(x/2)+1):
                if x % y == 0:
                    return False
            return True

        list_of_primes = [y for y in range(100) if is_prime(y)]
        print(list_of_primes)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

We can also pass each value to a function and then add it to the list. In the example below we are going to find 2 to the `n`th power and pair it in a tuple with `n`.

```
In [5]: z = [(n, 2**n) for n in range(64)]
        print(z)
```

```
[(0, 1), (1, 2), (2, 4), (3, 8), (4, 16), (5, 32), (6, 64), (7, 128), (8, 256), (9, 512), (10, 1
```

For more information on List Comprehension please see the following links: - https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions - http://python-3-patterns-idioms-test.readthedocs.io/en/latest/Comprehensions.html - http://www.python-course.eu/python3_list_comprehension.php

2.3 Modules

Modules are a collection of Python code that work together. We can `import` modules in order to get extra functionality for free. There are two standard ways of importing libraries. One only uses the `import` keywork and the other way also uses the `from` keyword.

```
In [6]: import math
        from random import *

        print( math.pi )
        print( randint(1,100) )
```

```
3.141592653589793
3
```

Notice that since we only used the `import` command we have to tell python exactly which library the `pi` variable is in by typing `math.pi`. The `from` keyword is used to tell python to `import` everything (`*`) in the `random` library directly into the current scope. That is why we can access `randint` without typing `random.randint(1,100)`. We can also `import` a specific attribute from a library so that we only get what we need.

```
In [7]: from math import pi
        print( pi )
```

```
3.141592653589793
```

The last thing that I want to bring up about modules is that you can write your own. As long as they are files with the .py extention then they can be imported.

2.4 Files

Often we need to read data from files or store data in files for later. It is possible to read and write from the same file but in most cases we do either one or the other. There are many styles for working with files so I will show a standard method of working with fils. For extra info about file manipulation use the following link: https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files.

```
In [8]: with open( 'temp.txt', 'w' ) as fout:
            fout.write( "Test Output " )

        with open( 'temp.txt', 'r' ) as fin:
            print( fin.read() )
```

```
Test Output
```

The `with` keyword is used to give us a little extra error handling protection if the file doesn't open or if the file doesn't exist.

2.5 Executing External Shell Commands

Often we like to use other scripts and other built-in Linux commands in our scripts. Running external commands can be a little different depending on the version of Python you are using. The easiest way to access external commands is to use the `subprocess` module.

```
In [9]: import subprocess

        response = subprocess.check_output('pwd', shell=True)
        print( response )

        response = subprocess.check_output('ls', shell=True)
        print( response )

        response = subprocess.check_output('whoami', shell=True)
        print( response )
```

```
b'/Users/dallin/projects/python_class/lecture_5\n'
b'lecture_5.ipynb\ntemp.txt\n'
b'dallin\n'
```

3 Practice Problems

There are a lot of things in Python to practice and many things that I did not try to show/teach in these 5 lectures. Python/Programming is something that you will continue to master for years to come. I find that often the best was to learn a new language is to pick a problem and start trying to solve it with that language. Along the way you will learn a lot of the ins and outs of the new language.

Another option is something like CodeWars. CodeWars is a site that has many practice problems that are posted by other users. After you come up with a solution your code is tested to see if it gives the correct output. The really cool thing about CodeWars is that after you complete a problem you can look at how other people solved the same problem and learn new little pieces of Python from the other solutions. `https://www.codewars.com`

This week for practice please practice anything you would like. Try to solve a problem that you think up or try to solve some of the practice problems on CodeWars. Good Luck