lecture_3

February 27, 2017

1 Beginning Python Class -- Lecture 3

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

This week we will begin to talk about Functions. Functions are a way to compartmentalize code that performs a single job/function. Functions are one of the most useful concepts that we will cover in this class. We will spend this entire lecture examining functions and the application of functions in our code.

When we talk about functions we will be required to have a discussion about *Scope*. Scope is an important topic because it determines where/which variables are accessible to our code. Understanding the rules of *Scope* will help us avoid errors and trouble-shoot problems with our code.

Indentation and Syntax will be increasingly important this week and therefore I will leave the sections about `Style` and `Syntax` in this lecture for easy reference.

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)

1.2 Functions

A function is a named section of the program that performs a specific task. In general, functions take inputs then perform actions on them and return a new value to the part of the program that called the function. Functions allow us to write code in a way that can be written one time and execute it whenever we want using the inputs we desire. This allows us to write less code. We could use the same code with different inputs to generate different outputs.

The general form for defining a function is shown below:

```
def FUNCTION_NAME(INPUT1,INPUT2): # We can add as many inputs as we would like
    STATEMENTS
    STATEMENTS
    return VALUE # A return value is not required
```

Functions, just like variables, can be named almost anything that we would like. The keyword `def` is the word we use to start the declaration of a function. Let's look at an example.

```
In [1]: def multiply_3( x, y, z ): # Take 3 numbers and multiply them all together
            return x * y * z # return the solution.

        print( multiply_3( 2,3,4 ) )
        print( multiply_3( 9,2,4 ) )
```

```
24
72
```

In the example above we created a function named `multiply_3`. This function took three values: `x`, `y`, and `z`. In the function definition line `x`, `y`, and `z` are called *parameters* to the function `multiply_3`. The `2`, `3`, `4` and the `9`, `2`, `4` in the lines where we call the function are all called *arguments*. This is only a slight distinction but if we keep this nomenclature it will make explaining functions quite a bit easier. The *return value* of the function is what the function gives back to us. In our case the function gives us back the product of the 3 arguments that we pass to the function.

The `print` function is probably the function that we have used more than any other in the previous lectures. We have used it to allow us to see the state of variables in our code. We have kind of accepted that the way that we use the `print` function is by putting the variable/string we want to print inside parenthesis.

```
print("Hello World")
```

```
x = 42
print( "A good number is: " + str(x) )
```

What I didn't explain in previous lectures is that `print` is a function, and the way we invoke functions is by using the parenthesis attached to the function name. Another example that we used is the `str` function, which we have used to turn non-strings into their string representation. Both `str` and `print` are examples or built-in functions that are available to the user by using parenthesis and the name of the function to call/invoke a function.

1.2.1 Return Values

An explicit return value for a function is optional, but they are very common. Let me show you examples of functions that do and don't have explicit return values. It should be noted that functions that don't have explicit return values still return a value, that value is `None`. `None` is a special value that means empty data.

```
In [4]: def get_square_of_value( x ):
            return x ** 2 # ** means exponent or raised to.

        def square_value( x ):
            value = x ** 2

        def print_general_greeting():
            print( "A very special welcome to lecture_3 by a greeting function.")

        for y in [1,2,3,4,5,6,7]:
            print( get_square_of_value(y) )
            print( square_value(y) )

        print()

        print_general_greeting() # easier than typing out the greeting each time.
```

```
        print_general_greeting()
        print_general_greeting()
```

```
1
None
4
None
9
None
16
None
25
None
36
None
49
None

A very special welcome to lecture_3 by a greeting function.
A very special welcome to lecture_3 by a greeting function.
A very special welcome to lecture_3 by a greeting function.
```

There is one more piece of information that I would like to cover about return values. Only one data structure can be returned from a function. This could be a `number`, a `string`, a `list`, or any other data type. If you feel that you need to return more than one value you can wrap those values in a structure like a `list` or a `tuple`.

1.3 Rules of Scope

Scope is defined as the location of code in which variables are valid and accessible. Scope is quite a large topic but very important when dealing with functions and nested code.

```
In [3]: x = 30

        def new_func( y, z ):
            func_var = y*y + z*z + x # x still accessible here
            return func_var

        print( new_func( 10, 7) )
        print( new_func( 4, 9) )

        print( x )
        print( func_var )
```

```
179
127
30
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-3-0e86da238b59> in <module>()
  9
 10 print( x )
---> 11 print( func_var )


NameError: name 'func_var' is not defined
```

The variable `x` is considered to be in global scope. Global scope means that `x` is available to all functions and code below where `x` is defined. The function `new_func` has a variable `func_var` declared in side of it. This variable is only accessible to that function. If we try and use a function that is out of scope we get some nasty errors from python because it doesn't know where to look for that variable.

1.4 Built-in Functions

There are a number of very useful built-in functions in Python3 that you should become familiar with: https://docs.python.org/3/library/functions.html.

We have already been using some of these functions in our previous lectures without really understanding what they really were. For example, we have been working with the `print` function since the very first lecture.

Understanding the use of these functions will allow you to take advantage of someone else's programming experience. There is no need to ever recreate functions that do the same thing as these built-in functions because they are vetted very carefully by the python maintainers and are likely written in a manner that is more efficient than either you or I could write.

1.5 Practice Problem 1

Please write a function that greets users. This function will accept one argument, the name of the user, and will use the name provided in a more complex greeting.

Example definition:

```
def complex_greeting( name ):
```

Example output:

```
Hello ######, Welcome to the Micron Boise Campas!
```

1.6 Practice Problem 2

Write a function that takes three parameters: `input1`, `input2`, and `input3`. Return the value `input1*input2+input3`. Please test your funciton with many different combinations of number.

1.7 Practice Problem 3

Write a function that accepts three arguments and creates a list with the arguments in it. Then use the Built-in function `sorted()` to return the sorted version of the created list. Read about the `sorted()` function at the link provided in the Built-in Function section.

1.8 Practice Problem 4

Write a function that accepts a string and returns the number of unique characters in that string. Inside the function print out the unique characters in order as they appear in the string.

Example definition:

```
def find_unique_letters( string ):
```

Example Input:

```
'Dallin Marshall'
```

Example Output:

```
Dalin Mrsh
```

Example Return Value:

```
10
```

Remember we are counting the number of unique characters. A space is a character.