CoderZ Summer Assignment #4

Shaurya Singh

Contents

Hello everyone! This is the fourth and last assignment you will receive this summer. By this point, all of you should have received and submitted Assignment #1, #2, and Assignment #3, which had you learn the basics of python and variables as well as loops, if/else statements, and operations If you still need help, feel free to email me at shaunsingh0207@gmail.com.

In Assignment #4, you will learn about functions and python modules

1 Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

1.1 Defining a function

Functions begin with the keyword `def` followed by the function name and parenthesis. Any input parameters or arguments should be placed within these parenthesis.

The first statement of a function can be an optional statement - the documentation string of the function or docstring.

The code block within every function starts with a colon (:) and is indented.

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

1.2 Calling a function

Defining a function gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

1.3 Example:

```
#!/usr/bin/python3

# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print (str)
   return

# Now you can call printme function
printme("This is first call to the user defined function!")
printme("Again second call to the same function")
```

```
This is first call to the user defined function!
```

```
Again second call to the same function
```

In the example, we input a string, but you can ask it to input anything. In the future we will likely use integers.

1.4 Global vs Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope

2 Modules

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

2.1 Example

The Python code for a module named aname normally resides in a file namedaname.py. Here is an example of a simple module, support.py –

```
def print_func( par ):
   print "Hello : ", par
   return
```

2.2 The import Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The import has the following syntax –

```
import module1[, module2[, ... moduleN]
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module hello.py, you need to put the following command at the top of the script –

```
#!/usr/bin/python3

# Import module support
import support

# Now you can call defined function that module as follows
support.print_func("Shaurya")
```

When the above code is executed, it produces the following result –

```
Hello : Shaurya
```

A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening repeatedly, if multiple imports occur.

2.3 The from...import Statement

Python's from statement lets you import specific attributes from a module into the current namespace. The from...import has the following syntax –

```
from modname import name1[, name2[, ... nameN]]
```

For example, to import the function fibonacci from the module fib, use the following statement –

```
#!/usr/bin/python3

# Fibonacci numbers module

def fib(n): # return Fibonacci series up to n
```

```
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a + b
    return result
```

Now in a python interpreter, run the following

```
>>> from fib import fib
>>> fib(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

This statement does not import the entire module fib into the current namespace; it just introduces the item fibonacci from the module fib into the global symbol table of the importing module.

2.4 The from...import * Statement

It is also possible to import all the names from a module into the current namespace by using the `from ... import *` statement

2.5 From modname import *

This provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly.

3 Assignment

This week, your assignment is to create a module which has a function to print a string, then import that module in another file and call the function