Foula Mitsiopoulos

March 06, 2022

Foundations of Programming (Python)

Assignment 06

CD Inventory: Functions and Classes

Introduction

For Assignment 06, I further modified the CD inventory we have been working on over the past few weeks using classes that contained defined functions within. These functions were then called in the while loop that executed the same options such as adding, deleting, and saving CDs into the inventory.

Concepts and Tools Used

This week’s assignment utilized functions that are defined by the programmer. To call a function, it first must be define; this is done by using ‘def’ followed by the function name and a set of parenthesis that can include parameters. Parameters, sometimes referred to as arguments, are variables that pass through the function. The function can end with a return value, which is what is returned once the function has been called and executed. LAB06\_A modifies the Basic Math program from an earlier module by defining functions:

![Text

Description automatically generated]()

***Figure 1:*** *LAB06\_A script.**Basic Math script modified with functions and return values*

![Text

Description automatically generated]()

***Figure 2:*** *LAB06\_A executed in Spyder*

As seen in Figure 1, 4 functions were defined that execute different procedures such as addition and subtraction. Each function contains the parameters ‘val1’ and ‘val2.’ These are placeholders for the arguments ‘intNumA’ and ‘intNumB’ that the user defines by entering in values when prompted. The functions are called by defining a variable that is equal to the function with the arguments implemented.

This same process can also contain tuples, like in LAB06\_B:

![Text

Description automatically generated]()

***Figure 3:*** *LAB06\_B script. Modifying LAB06\_A using tuples*

![A screenshot of a computer

Description automatically generated with medium confidence]()

***Figure 4:*** *LAB06\_B executed in Spyder*

In this example, there is only function defined that encompasses the 4 procedures that were individually defined in LAB06\_A. Then they were returned as a tuple. I then defined a set of four variables that were equal to the function, allOps that carried the arguments ‘intNumA’ and ‘intNumB’ that the user input values into. The executing looks exactly like that in LAB06\_A.

LAB06\_C modifies LAB06\_A as well, but it includes classes. Classes group functions and variables that are and includes more information about the contents. This creates a better organized script that is easier to read and modify in the future. The keyword used in this module to indicate the class object is @staticmethod and is put in before each function is defined.

![Text

Description automatically generated]()

***Figure 5:*** *LAB06\_C executed in Spyder*

![Text

Description automatically generated]()

***Figure 6:*** *LAB06\_C script. Math script organized with classes*

For this lab, I started with defining my class as SimpleMath() that has a description below it in that is encased with triple quotes. Before each function is defined, the @staticmethod is entered. After the functions are defined, triple quotes are used to give information about them containing a short description, the arguments involved, and the values returned. These are called docstrings. To call these functions later, the class must be inputted followed by the function, which is seen in lines 80-83 in Figure 6. Just as in LAB06\_B, the results after it’s executed is the same as LAB06\_A! It is valuable and interesting to see the different methods that can be used to produce the same output.

Final Code: CD Inventory using classes and functions

For the final code, I began with moving the code from the while loop into the classes that described their functions. The classes are:

* DataProcessor, which includes the functions that will later be processed. This includes the functions that create new dictionaries from user input (new\_dic)) and deleting entries(delete\_CD).
* FileProcessor, which is used for code that processes data to and from files. The functions included are for reading from files into a list of dictionaries (read\_file) and writing files which saves the inventory into the text file (write\_file).
* IO, which includes the functions that require user input.

Once these functions were defined in their respective classes, I was able to call the functions within the while loop. This created a much more organized and easier to read loop.

![Text

Description automatically generated]()

***Figure 7:*** *CDInventory.py lines 9-92*

***![Text

Description automatically generated]()***

***Figure 8:*** *CDInventory.py lines 94-178*

Text

Description automatically generated

***Figure 9:*** *CDInventory.py lines 180-247*

![Text

Description automatically generated]()Below are figures that show a couple menu options being executed on Spyder and Anaconda Prompt:

***Figure 10:*** *CDInventory run displaying inventory in Spyder*

Text

Description automatically generated

***Figure 10:*** *CDInventory run deleting an entry in Spyder*

Text

Description automatically generated

***Figure 11:*** *CDInventory run loading inventory in Anaconda prompt*

Text

Description automatically generated

***Figure 12:*** *CDInventory.txt*

Summary

This week’s module provided another opportunity to work with a starter code written by someone else. I find using classes to be extremely effective in creating an organized script that can be easily modified. I can see how functions created by the programming can become confusing for someone else accessing the script, so docstrings are another tool that can explain the logic of the function. I find while loops to be a bit challenging, especially when the code within contains more loops. So, using classes to break up the code and simply calling the function in the while loop to be easier for me to work with. Overall, I am finding the concepts I have learned throughout this course to be getting a little easier.