< Bryn Dunbar >

< 2020-Feb-29 >

< IT FDN 100: Foundations of Programming (Python) >

< Assignment 06 >

Modify an existing CD Inventory script; following the TODO’s for converting code to functions

# Introduction

The task of Assignment 06 is to modify an existing script by converting written code to functions and moving the functions to the appropriate Separation of Concerns section.

# Topic 1 – Module 06 Questions and Answers

1. What is a function?
   1. A function is a grouping of code that will likely be used on a repeated basis. For example, rather than writing code to ask a user a yes or no question, you can write a function once and then call to this function every time you need to ask a user a yes or no question. Calling to a function is easier than writing the code block of the function repeatedly.
2. What are parameters?
   1. Parameters are variables listed or defined in functions.
      1. Parameters can be positional, meaning that values are assigned by the order they are written in.
      2. Or parameters can be assigned default values, but default values can only be assigned at the end or after any positional arguments.
3. What are arguments?
   1. Arguments are what are assigned to the parameters within functions
      1. Arguments are the values for the parameters
4. What is the difference between parameters and arguments?
   1. A parameter is a variable or a container with a name, while arguments are the values that are being assigned or put into the variables
5. What are return values?
   1. Return values are values that are returned from functions. They allow functions to communicate other written code. Return values provide valuable information for the program.
6. What is the difference between a global and a local variable?
   1. A global variable is created and defined in the main body of the script, while a local variable is created and defined inside of a function.
   2. You cannot access local variables from the global scope
   3. A variable within a function can have the same name as a global variable in the script, but even though it has the same name, it is a different variable
7. What is shadowing?
   1. You can shadow a global variable from inside of a function
   2. Shadowing is when you give a variable inside of a function the same name as a variable in the global scope. With the same name, they look the same, but they are not. This shadowing concept can be deceiving and confusing, if the two are thought to be the same. It is best to not shadow global variables in your functions. One can prevent this by creating and defining unique variable names within functions.
8. How do you use functions to organize your code?
   1. I use functions to write small and manageable portions of code that complete a task. These functions often help me from writing repetitious code and often make my code more succinct.
9. What is the difference between a function and a class?
   1. A function is a smaller piece of code that performs or accomplishes a task that optionally returns a value or multiple values.
   2. A class is a grouping of functions or code that are all necessary to accomplish goal or similar goals, such as processing data, file processing, or input and output.
10. How do functions help you program using the “Separation of Concerns” pattern?
    1. Functions allow a program to be portioned out into smaller, well defined and well understood pieces. These functions can be written in any section of the program based on what they do. A function to add or delete data can be written in the ‘data processing’ portion of a script. A function to load data from a file or to save data to a file can be put in the ‘file processing’ section of code. Any time there is a task requiring user input or output, a function can be written and organized into the ‘IO’, or ‘User Input/Output’ section of a script. Then classes can be defined for each of these groupings in a script. Categorizing the script into pieces such as functions and classes makes it easier to comprehend and to read.

# Topic 2

The first TODO in this assignment was to add functions for processing. At first, I wasn’t sure what processing this meant. The class, for this TODO, is called **DataProcessor**, so I knew I needed to create or add functions for processing data. After further examination of the existing code, I realized that I needed to move snippets of code out of the while loop, in the main body, to the appropriate classes, while converting the sections of code to functions. What I needed to do became clear after I read the whole script. In section 3.3.2, I found a TODO referencing “processing code”. I moved the processing code for “adding an item to a table” to the **DataProcessor** class, by cutting and pasting this code:

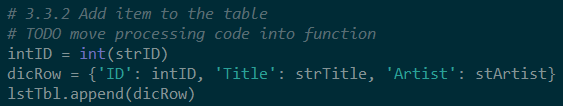


Figure – Existing Code that was Moved and Modified to the DataProcessor Class

After I pasted the code into the processing section of the script, I added ‘def’ above because this is a requirement to define a function. Then I created a name for my function, **add\_cd**. After this, I added parentheses for the function parameter. For the parameter argument, I defined a new local variable called ‘table’. I did not want to use the existing ‘lstTbl’ variable as my parameter value because it is a global variable. Defining a local variable inside the function, prevented me from shadowing the ‘lstTbl’ global variable. The ‘table’ variable holds the cd inventory information which comes from a list of dictionaries. Even though, return statements are optional, I also added a return statement in this section, to return the cd inventory back, with the new added cd entries. I also added a docstring section that can be quickly accessed in the help section of Spyder by typing ‘CTRL’ and ‘i’ keys on a keyboard.

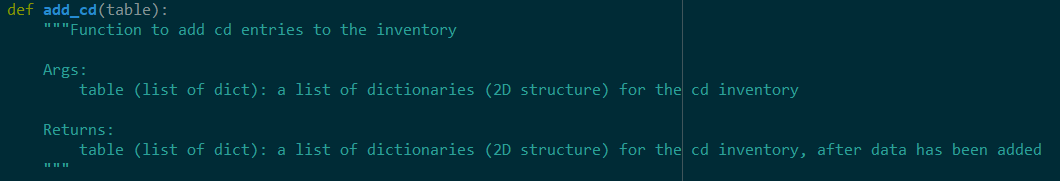


Figure – Defining a Function for Adding CDs to a CD Inventory

In order for my new function, to be a part of the DataProcessor class, I added ‘@staticmethod’ above the function definition.

Following another TODO, I moved the following code to the IO (input/output) class of the script.

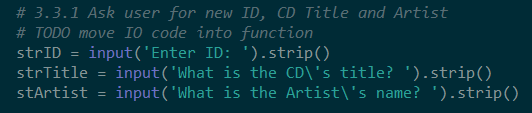


Figure – Existing Code that Was Moved and Modified to the IO Class

Similarly, I added ‘@staticmethod’ to make it a part of the IO class, and I added ‘def’, a unique name for the function, **get\_cd\_info**. I also added parenthese, but this time I did not put any arguments in the parameter of this function. I created a docstring to define the function, argument and return to the user. This function is for getting the required information for a cd from a user. I also defined new local variables in this function for the user input, because ‘strID’, ‘strTitle’, and ‘stArtist’ are in the previous **add\_cd** function. Once, I built the **get\_cd\_info** function, I went back to the **add\_cd** function and wrote code to call to it. I needed the user input in the **add\_cd** function. Considering the Separation of Concerns organization of my code though, I defined the input, for getting the cd information, in a separate section, but I could still call to it in the section and function that it is needed.



Figure – Calling to the GET\_CD\_INFO function from the IO Class

I also moved and modified the following code to the DataProcessor class.

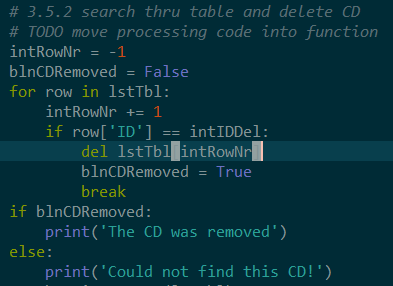


Figure – Existing Code to Remove CD that was Moved and Modified

To define this code as a function, I added a ‘def’ and named the function **remove\_cd**. I also added two parameters, filling them with two arguments or values for the cd inventory (table) and cd IDs (cd\_id). Of course, I added docstring text as well to help people know what this function is all about.

# Topic 3

When I moved the snippets of code to abide by the ‘Separation of Concerns’ organization in the script, I converted the code into functions. However, I cannot ignore the main body of the script, where I cut chunks of script out. After I moved the code to other parts of the script and defined appropriate functions, I had to write code to call to these functions.

I can use the global variables to provide information to the functions and I can use the returns of the functions and assign them to the programs cd inventory, ‘lstTbl’.



Figure – Call to the DataProcessor Class and Add\_CD Function



Figure – Call to the DataProcessor Class and the Remove\_CD Function

# Topic 4

Another necessary task of this assignment was to add code to the write\_file function. So, I took code from the ‘Save Inventory to File’ portion of the while loop.

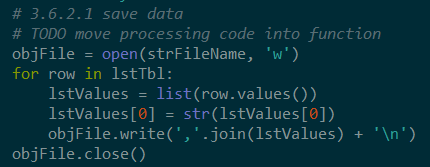


Figure – Existing Code for Saving the CD Inventory to a File that was Moved to the FileProcessor Class and Write\_File Function

Then I defined a new local variable for the cd inventory, to be independent of the global variable ‘lstTbl’. I used the local variable, ‘table’ as one of the arguments for the parameters of the function. I also used an argument (file\_name) in a parameter for the file to save the cd inventory to.

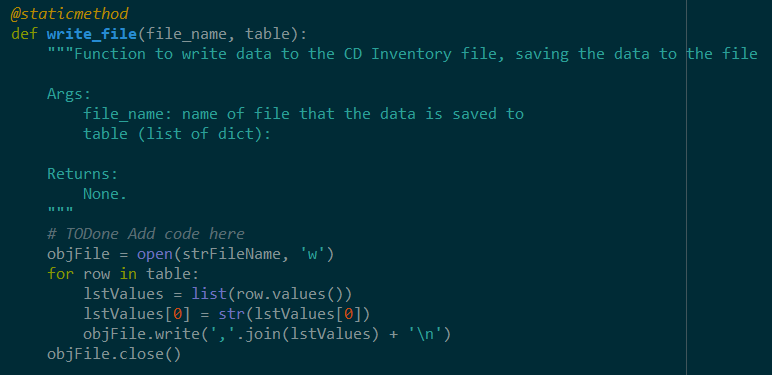


Figure – Code that I Modified, to Complete the Write\_File Function



Figure – Call to the FileProcessor Class and Write\_File Function

# Summary

The goal of this assignment was to modify existing code and to complete the TODO’s that were outlined within the script. The TODO’s consisted of moving written code to different sections (using the Separation of Concerns) of the script and creating functions for the moved snippets of code.

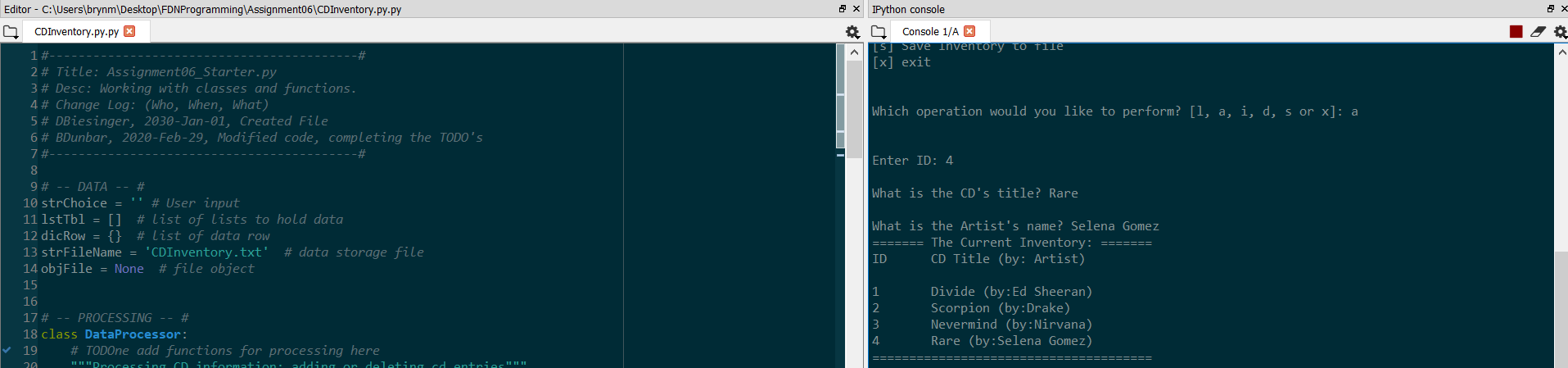


Figure – Image of Working Script in Spyder – Adding a CD to Inventory

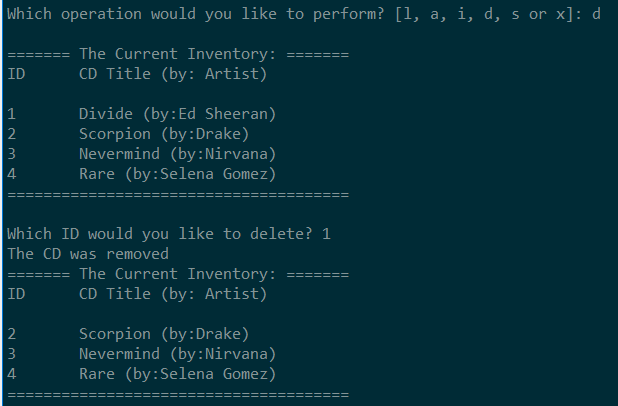


Figure – Another Image of Working Script in Spyder – Deleting a CD From Inventory

After I entered ‘s’ while running the script in Spyder, I typed ‘y’ and then ‘x’ to exit out of the program. Then I checked my file to confirm that the changes were applied (that my CD – ID 1 was removed). It successfully removed the cd that I chose to remove:

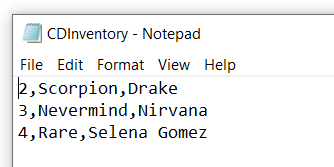


Figure – File that the CD Inventory was saved to by Running Program in Spyder

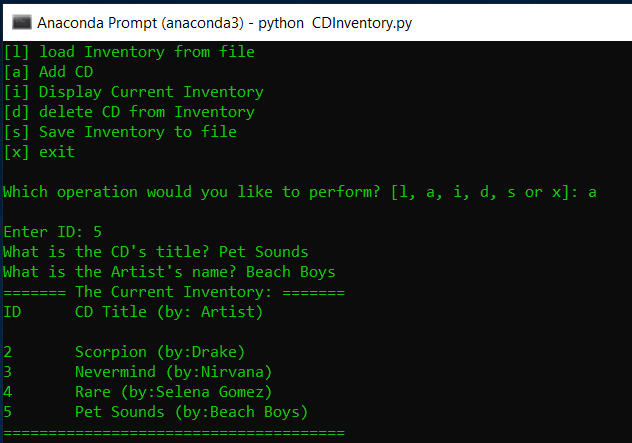


Figure – Image of Working Script in Anaconda Terminal – Adding a CD

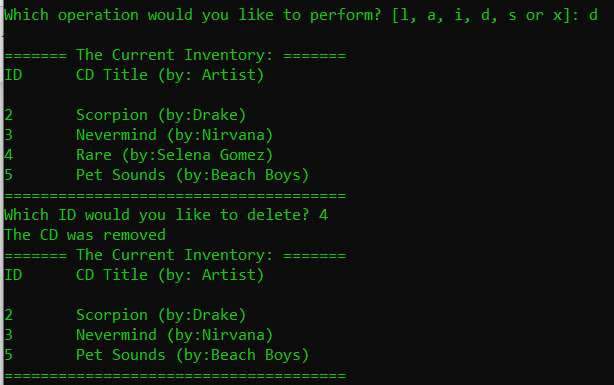


Figure - Image of Working Script in Anaconda Terminal – Removing a CD

# GitHub

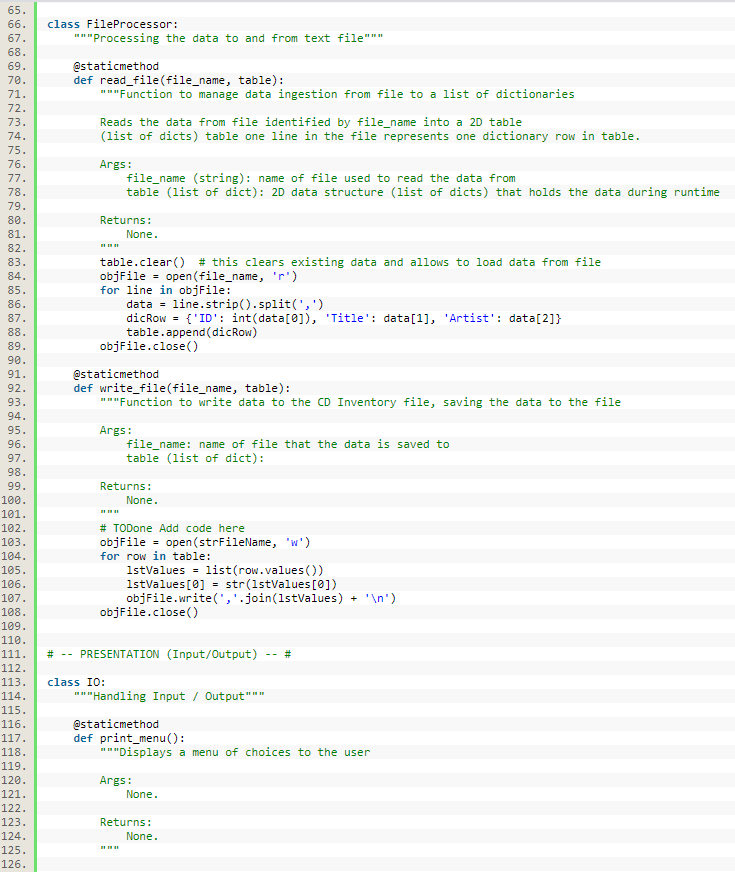
My assignment 06 files are uploaded to GitHub: <https://github.com/brynbar/Assignment_06>

# Appendix

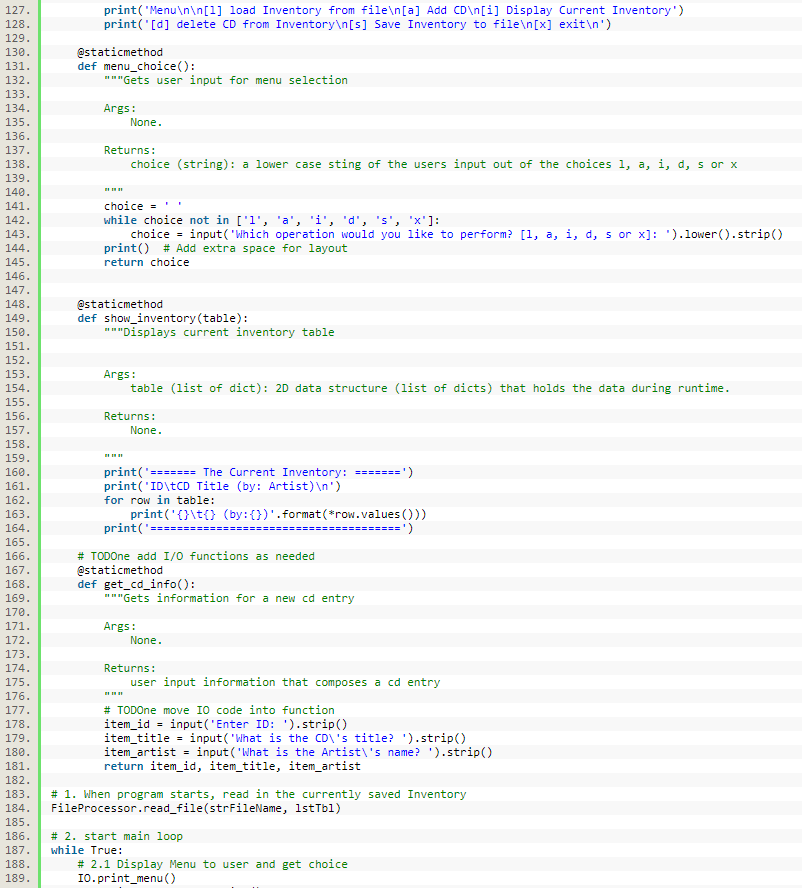
Using [PlanetB’s](http://www.planetb.ca/syntax-highlight-word) (external reference web page) [[1]](#footnote-1)



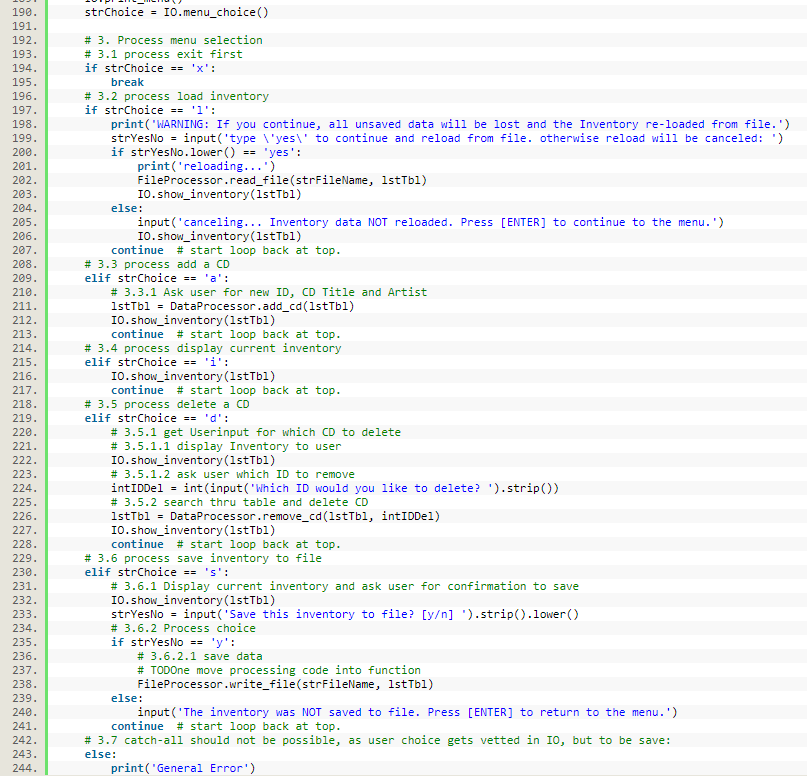
LISTING 1.0 – Source Code



LISTING 1.1 – Source Code



LISTING 1.2 – Source Code



Listing 1.3 - Source Code

1. <http://www.planetb.ca/projects/syntaxHighlighter/popup.php> - retrieved 24-Feb-20 [↑](#footnote-ref-1)