Patrick Danielson

02/21/2021

Foundations of Programming (Python)

Module 06 – Assignment 06

Functions and Classes

# Introduction

This document summarizes the work performed for Module 06 of the Foundations of Programming course. The purpose of this module was to familiarize myself with the use of class objects to organize functions with a Python program. Additionally, I became comfortable using docstrings as a means of documentation of user defined functions and practiced organizing my programs through applying the separation of concerns philosophy. The problem addressed in this assignment was to take a starter code for the CDInventory program with an already defined class structure and create additional functions that could be called by the main program to integrate the remaining program functionality as it pertained to input/output processing, data processing, and file processing.

# Adding a CD Entry

In order to follow the principle of separation of concerns when adding a new entry to the table, I create one function within the IO class to gather the entry information and another within the DataProcessor class to append the data to the table. The function get\_new\_entry() would then prompt the user for 3 pieces of information and return them as a list to the main runtime. I chose to return the values as a List rather than the default Tuple object because it would make it easier to use as an argument for another function.[[1]](#footnote-1)

For the data processing, I created a function add\_entry() with 2 arguments, the list of strings returned from the get\_new\_entry() function and the list table (2D list of dictionaries) that contained the CD inventory data. Both of these arguments were reference type and therefore, when the dictionary row was created and appended to the table within the function, the values of the list table itself were modified. Therefore, the function did not have to return the local table variable as it was a reference to the global lstTbl variable.[[2]](#footnote-2)

# Remove Entry

In order to remove an entry from the table, the user would again be prompted for an input, however, as this was only a single integer value, I did not believe this command warranted its own function call. This would have added unnecessary complexity to the program where a function was used to return a single value. I left that portion of the code untouched and simply used the integer input by the user for the entry deletion as an argument for the function remove\_entry() that I created. This function took 2 arguments, one being the previously mentioned integer and the second being the list table reference. Like the function add\_entry(), the data processing code was added to remove the specified entry from the table if it existed and did not have to return any objects.

# File Processor

When I first ran the program, I immediately received an error message saying that the text file was unable to be read in. Since no inventory file existed yet the program crashed. To solve this, I added a section of code in the data setup section to check if the text file existed. This utilized the os.path.exists() function and if the CDInventory.txt file was not found, it was created. [[3]](#footnote-3) This is shown in Figure 1 below.

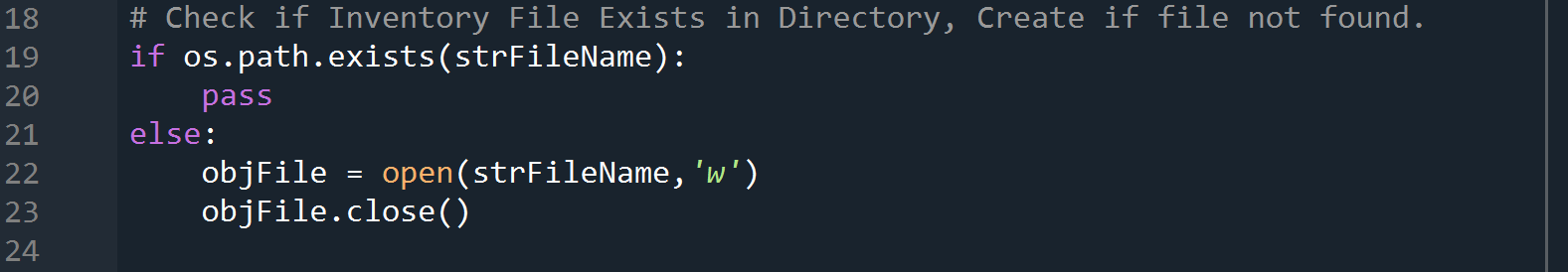


Figure 1: Check for Existing CDInventory File

In order to write the list table to the text file, I added the code block to the from the starter code to the write\_file() function. This function took 2 argument, the first being the string of the file name the program would write to and the second being the list table reference.

# Executing the Program

The execution of the program is summarized in this section. To begin with, I started in Spyder with CDInventory.txt file in the directory containing 2 entries. I then executed the program to remove the first entry and add another, shown in Figure 2. I then reloaded the initial data and displayed the inventory, shown in Figure 3. The final step was to then add another entry and save the data to the text file, shown in Figure 4. This confirmed all functionality within Spyder. When running the program in the terminal window, I initially loaded in the file and removed the second entry, shown in Figure 5. I then added a final entry and saved the file, shown in Figure 6. The final text file is shown in Figure 7. The CDInventory.txt file shown in Figure 7 is submitted for this assignment.

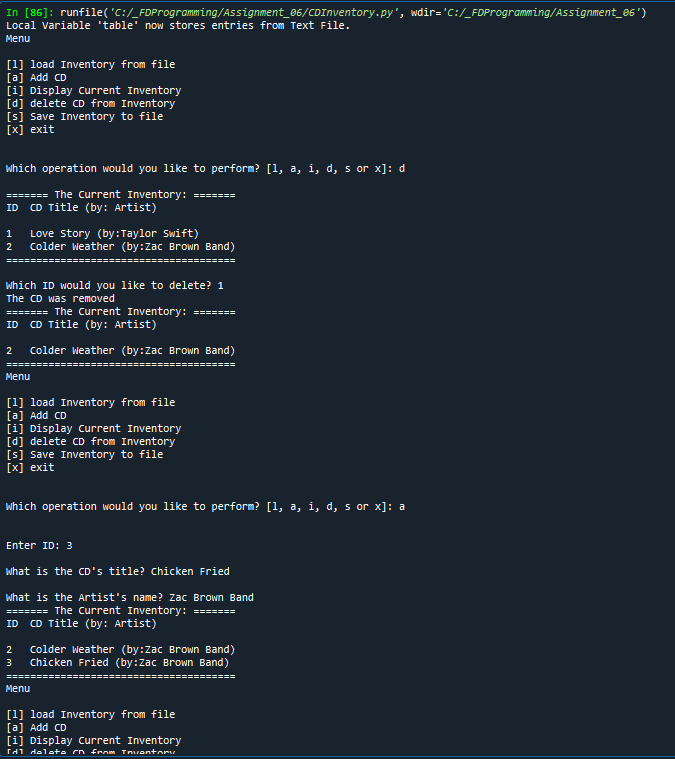


Figure 2: Spyder Execution – Remove and Add Entry

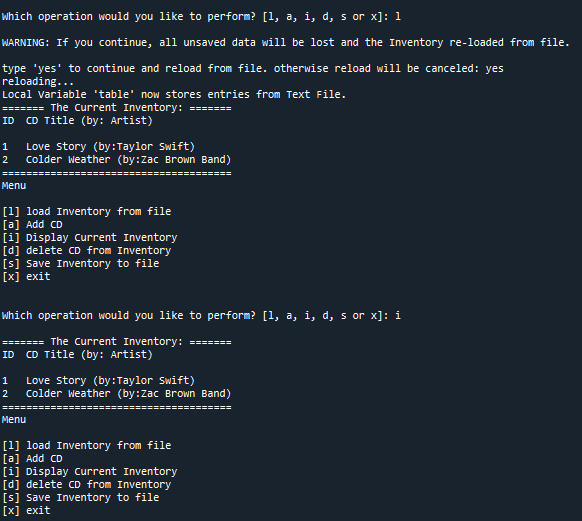


Figure 3: Spyder Execution – Reload Data and Display Inventory

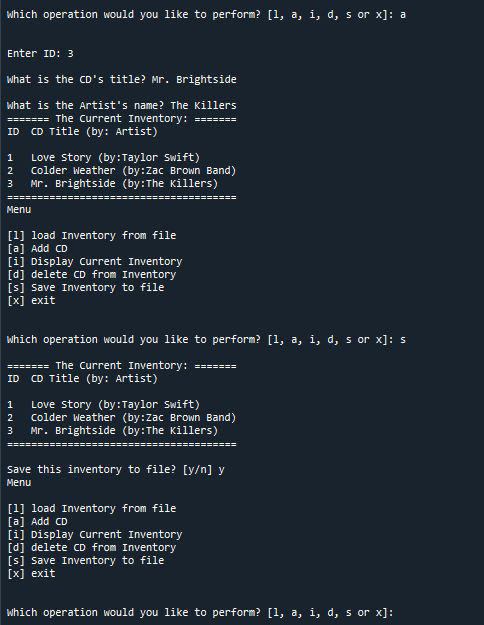


Figure 4: Spyder Execution – Add Entry and Save File

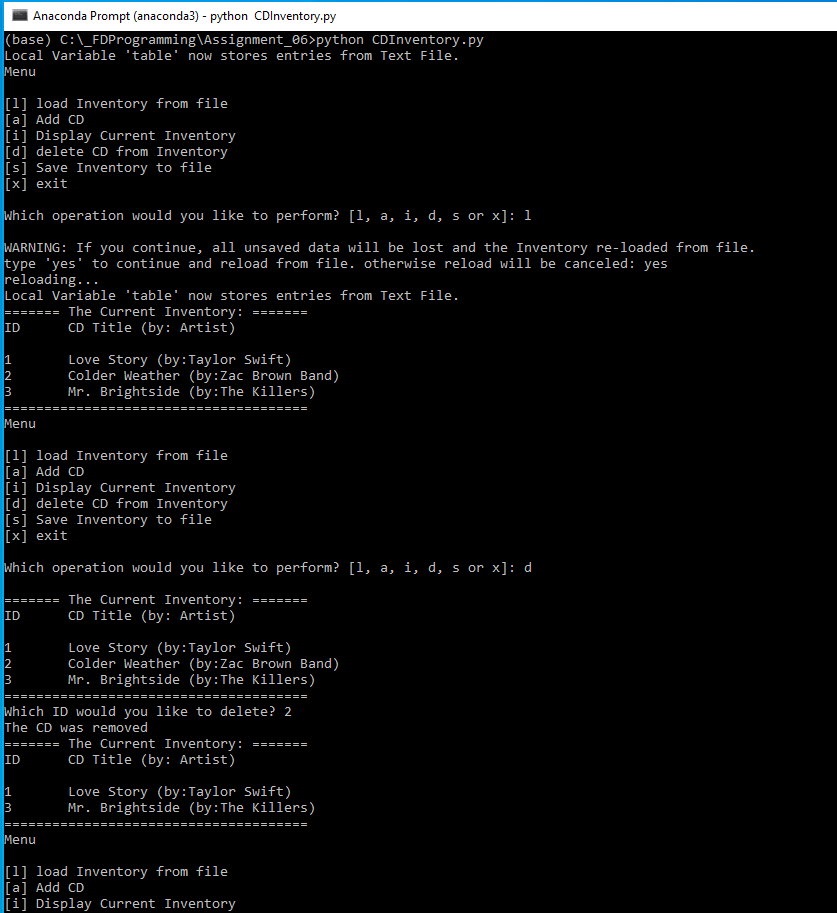


Figure 5: Terminal Execution – Load Data and Remove Entry

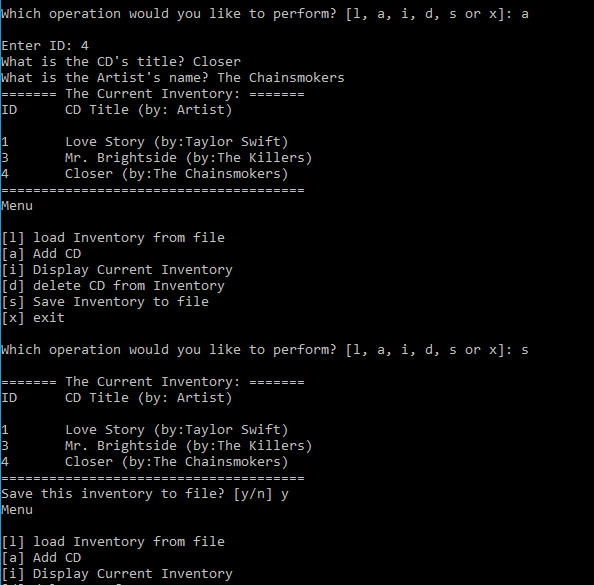


Figure 6: Terminal Execution – Add Entry and Save File

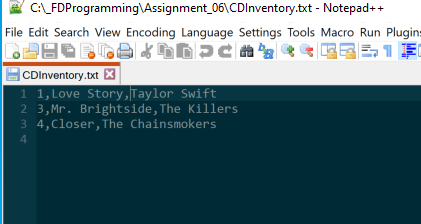


Figure 7: Final CDInventory.txt File

# Summary

In this assignment, I was able to successfully take the starter script and create additional program functions and integrate them into a class structure that followed the separation of concerns principle. The program maintained the same functionality, but the main body of code was simplified by using function calls to process either user inputs, data, or files. By doing so, I have learned how to work with classes and function parameters to process various inputs. Additionally, I have become more familiar with use of docstrings to help document user defined function and have a better understanding of variable scope. The Python script and CDInventory.txt file used in this assignment, along with this document, are posted on GIT[[4]](#footnote-4) under the Assignment\_06 repository.

1. FDN\_Py\_Module\_06.pdf, Biesinger, Dirk, P. 6 [↑](#footnote-ref-1)
2. FDN\_Py\_Module\_06.pdf, Biesinger, Dirk, P. 15 [↑](#footnote-ref-2)
3. <https://docs.python.org/3/library/os.path.html> [↑](#footnote-ref-3)
4. Patrick Danielson GIT Repositories: <https://github.com/pdaniel441?tab=repositories> [↑](#footnote-ref-4)