Makena Mezistrano

March 1, 2020

Foundations of Programming, Python

Assignment 06

CD Inventory with Functions

Introduction

In this assignment I will explain the steps I took to modify a pre existing solution to the CD inventory program to incorporate functions and docstrings. The program will allow the user to view current inventory from in memory, load data from a text file, add new data to memory, save data to a text file, and quit the program. By using classes and functions, this script is clearly organized according to Sequence of Concerns (SoC), thereby illustrating the benefit of classes and functions. This script was also saved to Github.[1]

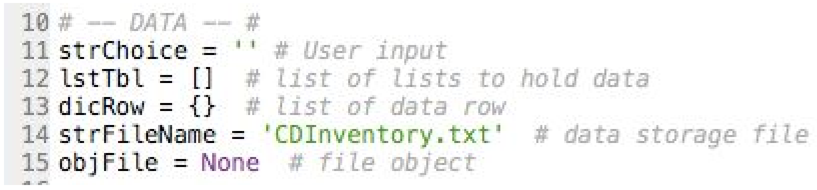
Modifying the script

Reviewing the file

The script for this assignment was based on a pre-existing file called Assignment06_Starter.py. To begin this assignment, I opened the file and reviewed the TODO items.

Data

The first block in the SoC was the data block. I did not change anything in this block. This block established the main data variables for the script.



```
# -- DATA -- #
strChoice = '' # User input
lstTbl = []  # list of lists to hold data
dicRow = {}  # list of data row
strFileName = 'CDInventory.txt'  # data storage file
objFile = None  # file object
```

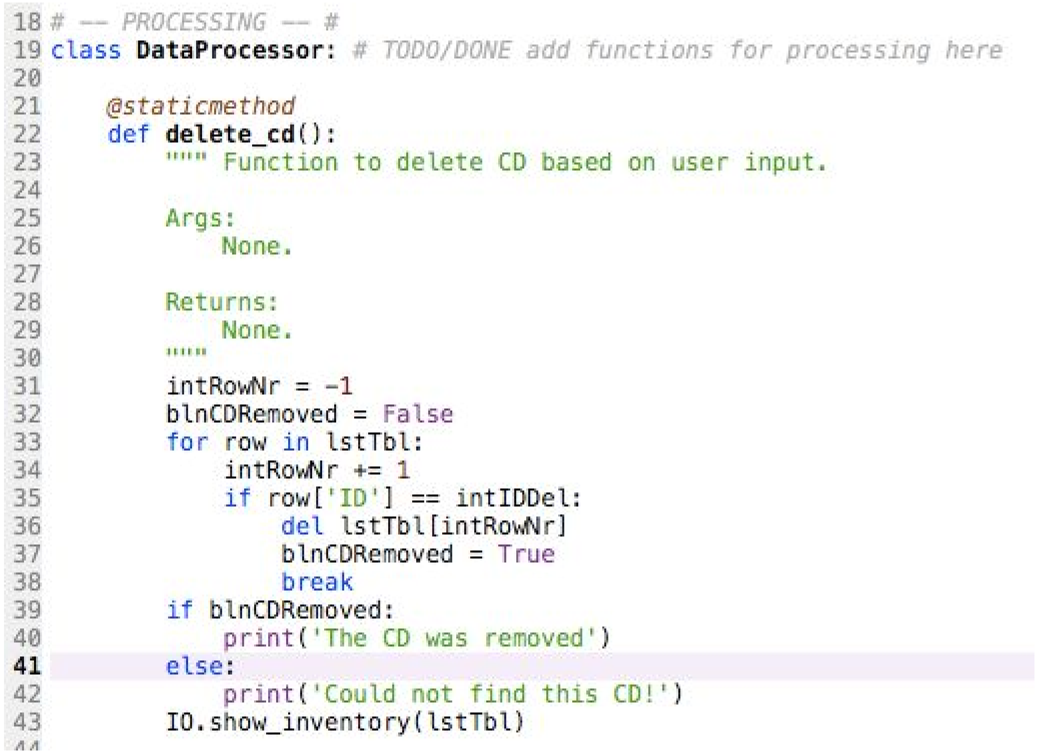
Figure 1: Data block.

Processing: DataProcessor class

The next block in the SoC is the Processing block. This block organizes all processing functions under two classes. The first class is the DataProcessor class. This class contains the delete_cd() function, which was a function that I created based on code that was originally in the while loop (in the Presentation block). I also added a docstring to this function (and all functions hereafter) to describe what this function does.

The del_cd() function allows the user to delete a CD from the inventory by comparing the user input against the row['ID'], which will match the user's input integer to an ID in the table. The row corresponding to the input ID will then be deleted. The print statements help guide the user by showing the user whether he/she entered a valid ID. The delete_cd function ends with another function from the Presentation block, the show_inventory(lstTbl) function, which I will describe later.

[1] I have tried to avoid repeating information from the previous two assignments, and have thus not explained elif statements or Github in detail, or other components of the script that have remained the same from previous assignments.

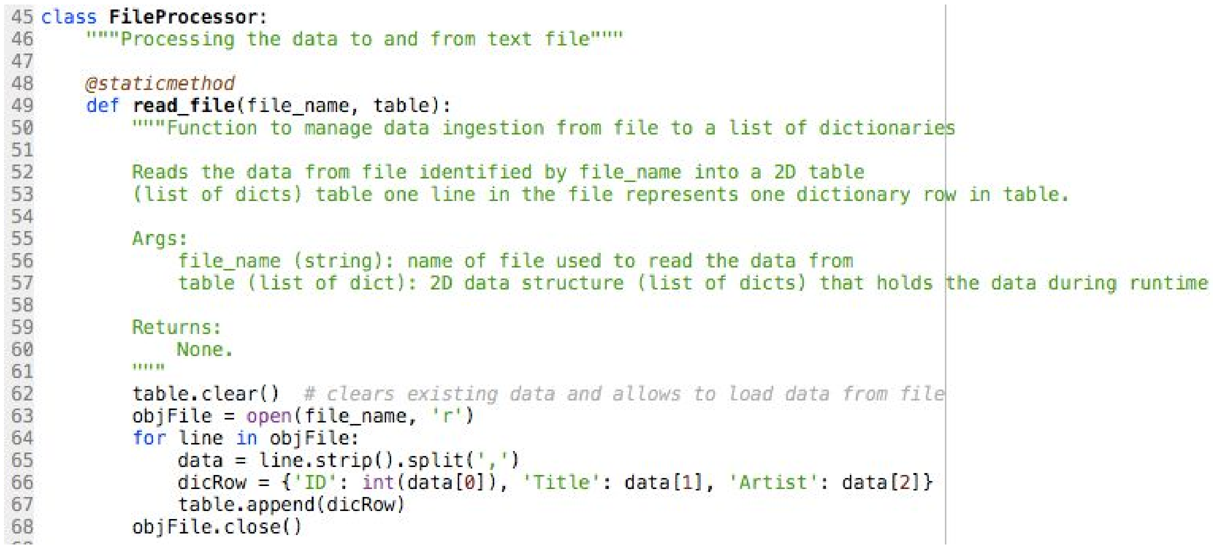


```
# -- PROCESSING -- #
class DataProcessor: # TODO/DONE add functions for processing here

    @staticmethod
    def delete_cd():
        """ Function to delete CD based on user input.

        Args:
            None.

        Returns:
            None.
        """
        intRowNr = -1
        blnCDRemoved = False
        for row in lstTbl:
            intRowNr += 1
            if row['ID'] == intIDDel:
                del lstTbl[intRowNr]
                blnCDRemoved = True
                break
        if blnCDRemoved:
            print('The CD was removed')
        else:
            print('Could not find this CD!')
        IO.show_inventory(lstTbl)
```

Figure 2: Processing block with delete_cd() function.

Processing: FileProcessor class

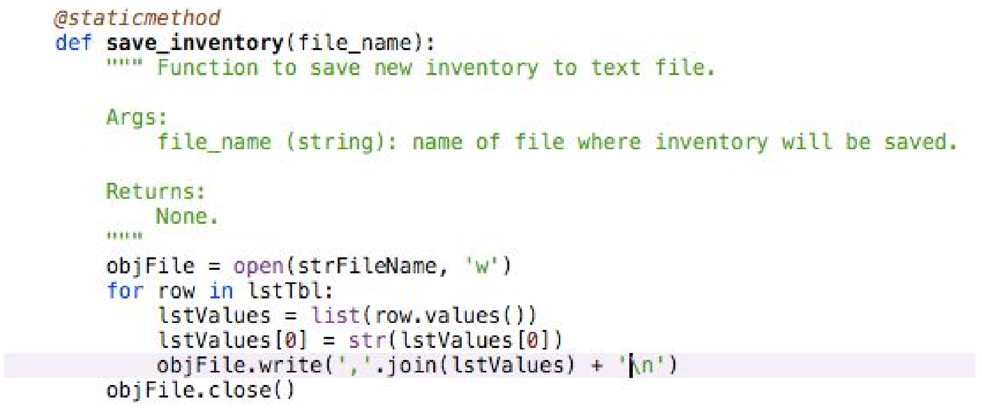
The next class in the Processing block is the FileProcessor. This class organizes all the functions that deal with processing data to and from a text file. The first function in this class is read_file(file_name, table), which came preloaded into the script. This function contains two arguments (or parameters) which are passed through the function: 'file_name', which, in this case, is CDInventory.txt, and 'table', which refers to the 2D data structure that holds the CD inventory data. This function will be called several times in the code, and I will explain its utility when it is called.



```
class FileProcessor:
    """Processing the data to and from text file"""

    @staticmethod
    def read_file(file_name, table):
        """Function to manage data ingestion from file to a list of dictionaries

        Reads the data from file identified by file_name into a 2D table
        (list of dicts) table one line in the file represents one dictionary row in table.

        Args:
            file_name (string): name of file used to read the data from
            table (list of dict): 2D data structure (list of dicts) that holds the data during runtime

        Returns:
            None.
        """
        table.clear()  # clears existing data and allows to load data from file
        objFile = open(file_name, 'r')
        for line in objFile:
            data = line.strip().split(',')
            dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
            table.append(dicRow)
        objFile.close()
```

Figure 3: read_file() function.

The next function in the FileProcessor class is save_inventory(file_name). Using the 'w' command, this function **writes** the row values of the lstTbl to the file CDInventory.txt. Later, I will explain this function's utility in context when it is called.



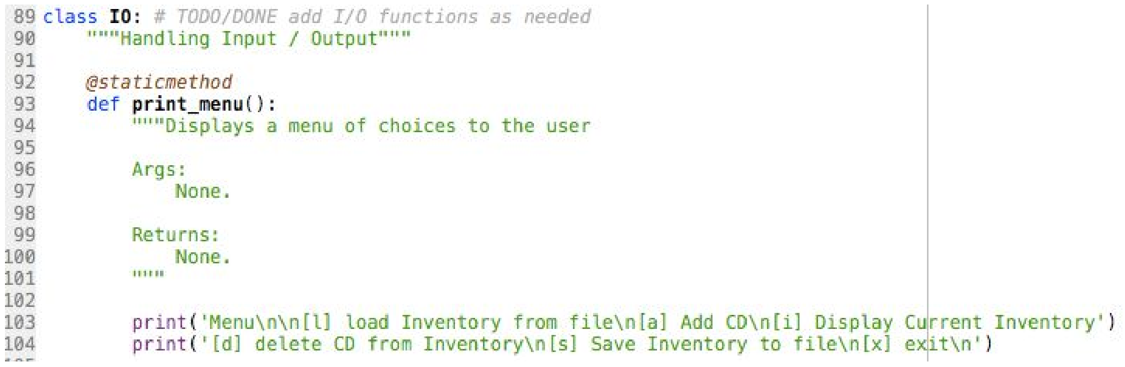
```
    @staticmethod
    def save_inventory(file_name):
        """ Function to save new inventory to text file.

        Args:
            file_name (string): name of file where inventory will be saved.

        Returns:
            None.
        """
        objFile = open(strFileName, 'w')
        for row in lstTbl:
            lstValues = list(row.values())
            lstValues[0] = str(lstValues[0])
            objFile.write(','.join(lstValues) + '\n')
        objFile.close()
```

Figure 4: save_inventory(file_name) function.

There used to be another function in the FileProcessor class called write_file. I removed this function because the script worked without it.

Presentation: IO class

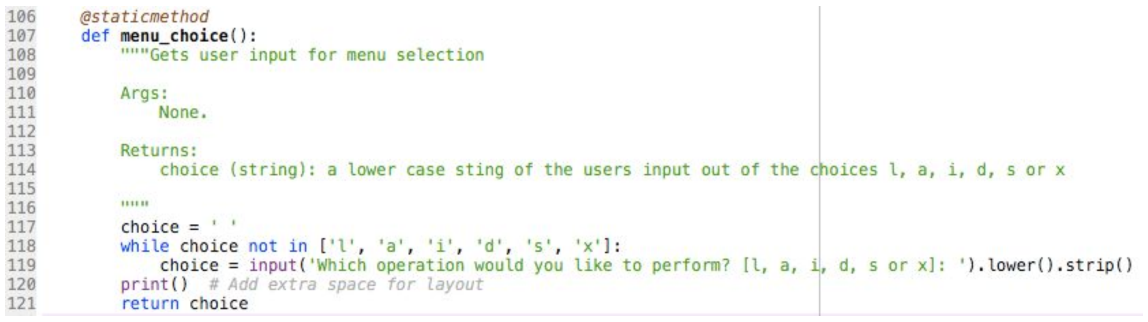
The next block in the SoC is the Presentation block and contains the class IO, which organizes functions that handle input and output. The first function in this class is print_menu(), which came preloaded into the script. This function displays the menu of choices to the user. This function is useful later because instead of having to write print statements in the while loop to display the menu, we only need to call the function, which already contains the descriptive print statements. This makes the code cleaner.



```
class IO: # TODO/DONE add I/O functions as needed
    """Handling Input / Output"""

    @staticmethod
    def print_menu():
        """Displays a menu of choices to the user

        Args:
            None.

        Returns:
            None.
        """

        print('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
        print('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
```

Figure 5: Presentation block with the print_menu() function.

The next function here is menu_choice(), which also came preloaded. This function collects user input based on the options in the menu. This function uses a local variable 'choice' to collect the user input, and a while loop to make sure that the user selects a valid option from the menu. If the user inputs an option that is not based on the menu (as defined in this while loop here), the script will remind the user to pick a valid option. Once the user does choose a valid option, the function returns the user's input. Having this function offers similar benefits to print_menu() in that it keeps the script cleaner later on.

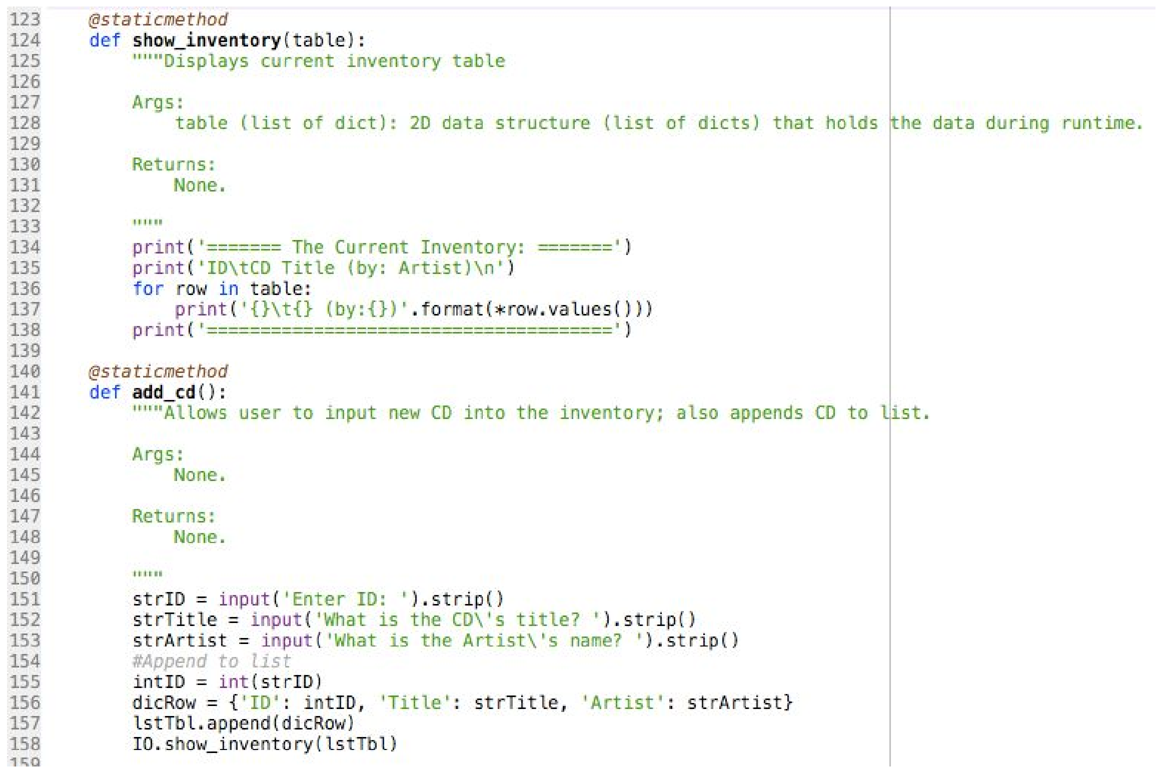


```
    @staticmethod
    def menu_choice():
        """Gets user input for menu selection

        Args:
            None.

        Returns:
            choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x

        """
        choice = ' '
        while choice not in ['l', 'a', 'i', 'd', 's', 'x']:
            choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
        print()  # Add extra space for layout
        return choice
```

Figure 6: menu_choice() function.

The next function under the IO class is show_inventory(table). This function contains a series of print statements that show the user what CDs are currently in memory. Once again, this function keeps the script clean later.

The final function in this class is add_cd(), which I added by taking code from the main while loop and adding it to a function here. This function does two things: first, it collects input data from the user to add a new CD to the inventory, and then it appends that data into the list by consolidating all the inputs into a row. This function also ends with another function, show_inventory(), which I just described.

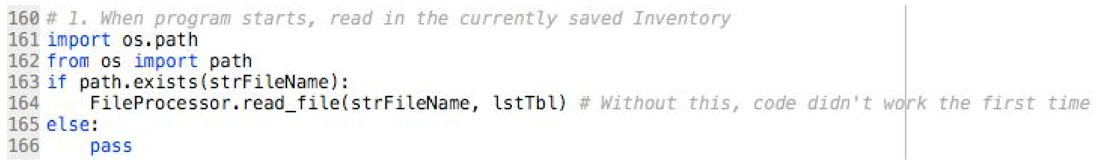


```python
    @staticmethod
    def show_inventory(table):
        """Displays current inventory table

        Args:
            table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.

        Returns:
            None.

        """
        print('======= The Current Inventory: =======')
        print('ID\tCD Title (by: Artist)\n')
        for row in table:
            print('{}\t{} (by:{})'.format(*row.values()))
        print('======================================')

    @staticmethod
    def add_cd():
        """Allows user to input new CD into the inventory; also appends CD to list.

        Args:
            None.

        Returns:
            None.

        """
        strID = input('Enter ID: ').strip()
        strTitle = input('What is the CD\'s title? ').strip()
        strArtist = input('What is the Artist\'s name? ').strip()
        #Append to list
        intID = int(strID)
        dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
        lstTbl.append(dicRow)
        IO.show_inventory(lstTbl)
```

Figure 7: Final two functions in the Presentation block.

Presentation: Calling the read_file() function

We now move to calling all the functions to execute the program. The script began by calling the read_file() function to read in the currently saved inventory. The problem here was that when I ran the code for the first time without any text file create, the script wouldn't run if this was the first statement. To get around this, I used an if statement to check if the text file existed, and, if it did not exist, this block would be passed.[2]



```python
# 1. When program starts, read in the currently saved Inventory
import os.path
from os import path
if path.exists(strFileName):
    FileProcessor.read_file(strFileName, lstTbl) # Without this, code didn't work the first time
else:
    pass
```

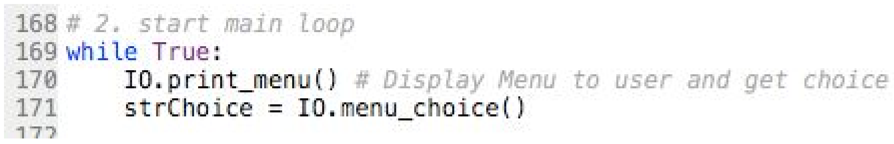
Figure 8: If statement to check if CDInventory.txt exists, and if read_file() can be called.

While loop and display menu

The rest of the script is a series of elif statements nested under a while loop. The first block in this while loop is not an elif statement, though, and calls the print_menu() function to display the menu to the user. Once the user sees the menu, the menu_choice() function is called to allow the

[2] Solution adapted from: https://www.guru99.com/python-check-if-file-exists.html Retrieved 2020-Mar-01.

user to select an option from the menu. These options, and the functions that make them work, will now be explained.



```
# 2. start main loop
while True:
    IO.print_menu() # Display Menu to user and get choice
    strChoice = IO.menu_choice()
```

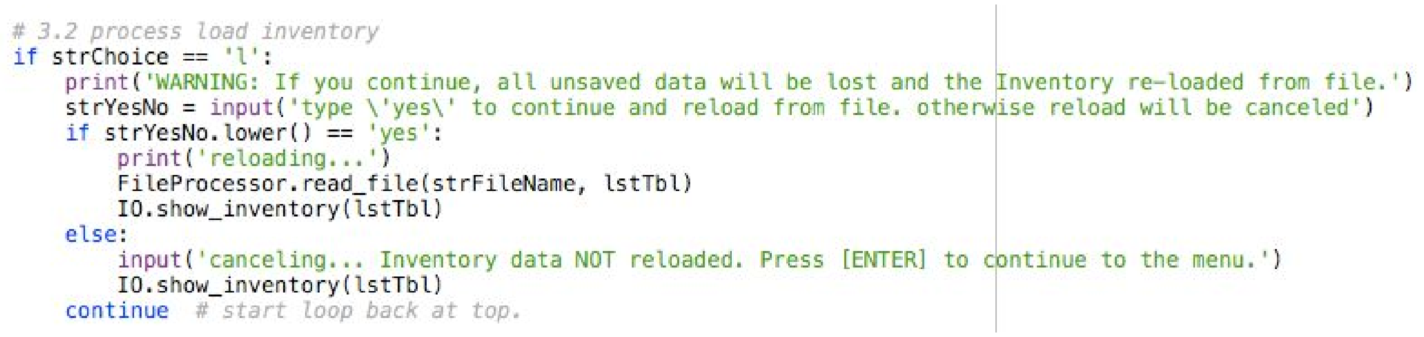
Figure 9: While loop and two functions that display the menu to the user and then get the user's choice from the menu.

Quit program

The first if statement in the while loop allows the user to quit the program. If this choice is selected, it will break the loop.

Load inventory from file

The next elif statement allows the user to load inventory from a file. If the user chooses this option, only the inventory that he/she has saved to the file will be loaded, and any other data will be lost. A print statement warns of this outcome. The strYesNo variable is then defined to collect the user's choice regarding whether he/she wants to continue to load the file or not. A nested if statement cycles through these choices. If the user selects yes, then the read_file() function is called. The code in this function does not write any new data to the file, but it does format the data in the file, as long as it was previously saved. Once the file is read, another function is called to show the current inventory again. If the user does not want to go through with this choice and does not want to read the file, the else statement simply calls the show_inventory() again. This block, and all subsequent blocks, ends with a continue statement. This is important so that all elif statements can be cycled through with each new menu input.



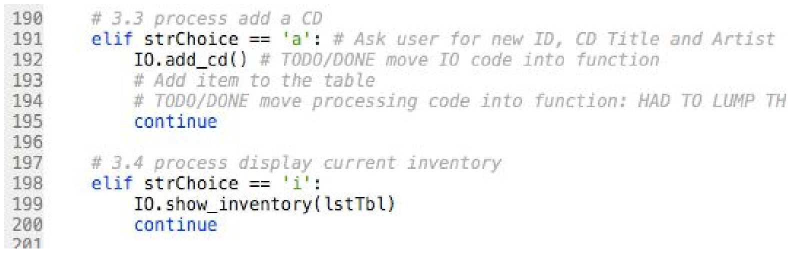
```
# 3.2 process load inventory
if strChoice == 'l':
    print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
    strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
    if strYesNo.lower() == 'yes':
        print('reloading...')
        FileProcessor.read_file(strFileName, lstTbl)
        IO.show_inventory(lstTbl)
    else:
        input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
        IO.show_inventory(lstTbl)
    continue  # start loop back at top.
```

Figure 10: if statement that loads inventory from file with a nested if statement.

Add a CD and display current inventory

The next elif statement allows the user to add a CD to the inventory. To collect the user input, the add_cd() function is called. This function was already described above.

The following elif statement allows the user to display the current inventory by calling the show_inventory() function, which was already described above.

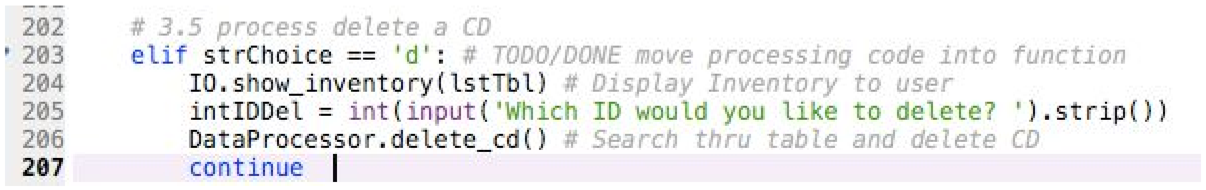


```
    # 3.3 process add a CD
    elif strChoice == 'a': # Ask user for new ID, CD Title and Artist
        IO.add_cd() # TODO/DONE move IO code into function
        # Add item to the table
        # TODO/DONE move processing code into function: HAD TO LUMP TH
        continue

    # 3.4 process display current inventory
    elif strChoice == 'i':
        IO.show_inventory(lstTbl)
        continue
```

Figure 11: Elif statements to add CD and display current inventory.

Delete a CD

The next elif statement required me to create a new function, the process of which I already described above. This elif statement begins with the show_inventory function to show the user the current inventory and make it easier for him/her to select which CD he/she wants to delete. The user is then given the opportunity to enter the ID of the CD he/she wishes to delete, and this input is cast as an integer (because the IDs in the table are integers). Then, the newly created function delete_cd() is called, which finally deletes the CD from the table. The details of the delete_cd() function have already been described.



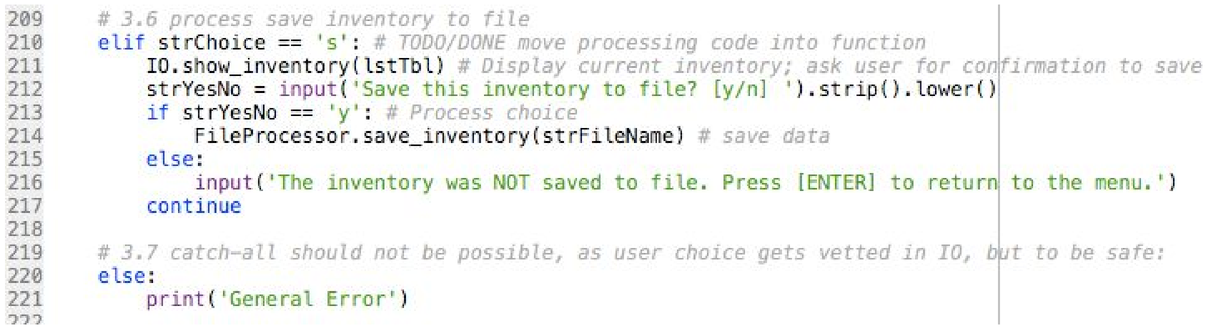
```
    # 3.5 process delete a CD
    elif strChoice == 'd': # TODO/DONE move processing code into function
        IO.show_inventory(lstTbl) # Display Inventory to user
        intIDDel = int(input('Which ID would you like to delete? ').strip())
        DataProcessor.delete_cd() # Search thru table and delete CD
        continue
```

Figure 12: Elif statement allows the user to delete a CD; includes the newly created delete_cd function.

Save inventory to file, final else statement

The final elif statement allows the user to save the inventory to a file. First, the show_inventory() function is called to display the current inventory to the user. The user then must confirm that he/she wants to save the inventory that was just displayed to the text file. Then, a nested if statement goes through the options: if the user selects 'y' for yes, the save_inventory() function is called, and the data is saved to CDInventory.txt. If the user does not select yes, a print statement is returned that tells the user the data was not saved.

The script ends with an else statement: if the user doesn't enter one of the menu options specified at the beginning of the while loop, the program will run through all options until it reaches the else statement, which will print 'General Error.'



```
    # 3.6 process save inventory to file
    elif strChoice == 's': # TODO/DONE move processing code into function
        IO.show_inventory(lstTbl) # Display current inventory; ask user for confirmation to save
        strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
        if strYesNo == 'y': # Process choice
            FileProcessor.save_inventory(strFileName) # save data
        else:
            input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
        continue

    # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
    else:
        print('General Error')
```

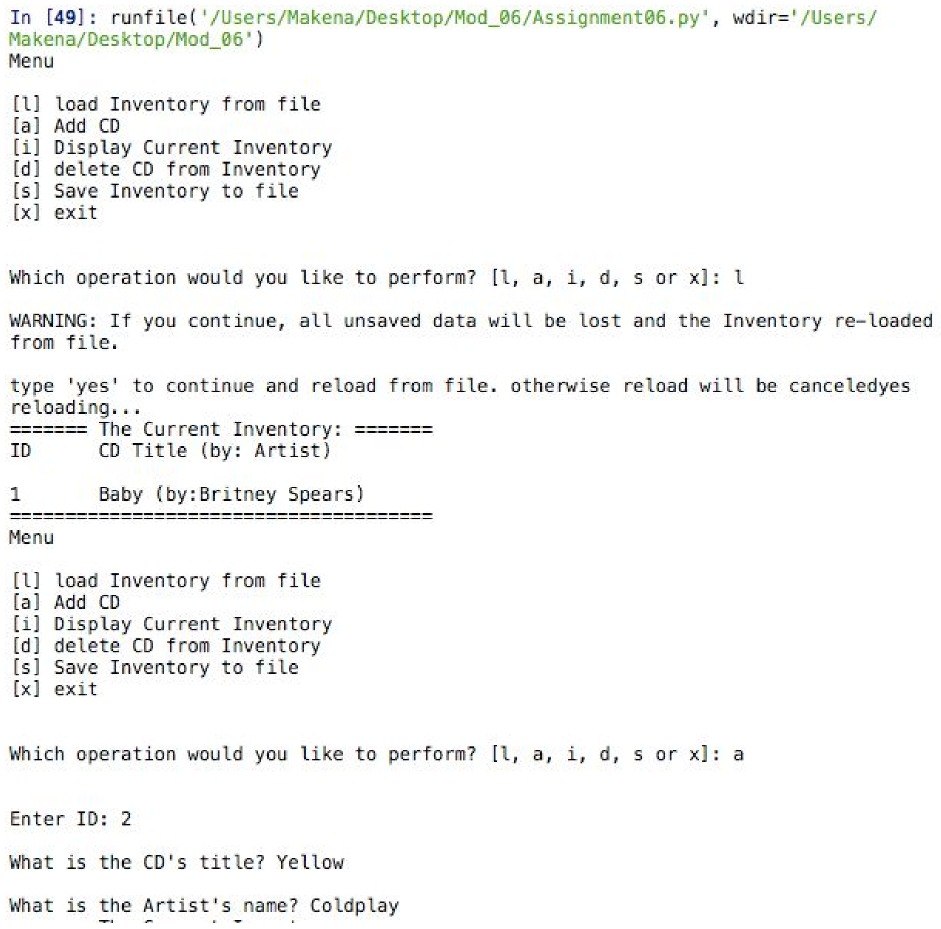
Figure 13: Elif statement to save data to file; final else statement.

Saving the file

To finish the file, I added marked the TODOs as Done, removed some of the pseudocode to clean the script, and added my contributions to the header. I saved this file to a folder created for this course and renamed the file CDInventory.py.

Running the script, verifying functioning

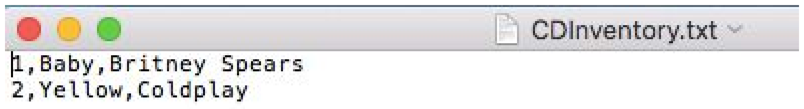
To test the script, I first ran and tested all functions of the script in Spyder. Everything worked.



```
In [49]: runfile('/Users/Makena/Desktop/Mod_06/Assignment06.py', wdir='/Users/
Makena/Desktop/Mod_06')
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit


Which operation would you like to perform? [l, a, i, d, s or x]: l

WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded
from file.

type 'yes' to continue and reload from file. otherwise reload will be canceledyes
reloading...
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Baby (by:Britney Spears)
======================================
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit


Which operation would you like to perform? [l, a, i, d, s or x]: a


Enter ID: 2

What is the CD's title? Yellow

What is the Artist's name? Coldplay
```

Figure 14: A portion of the working script in spyder.

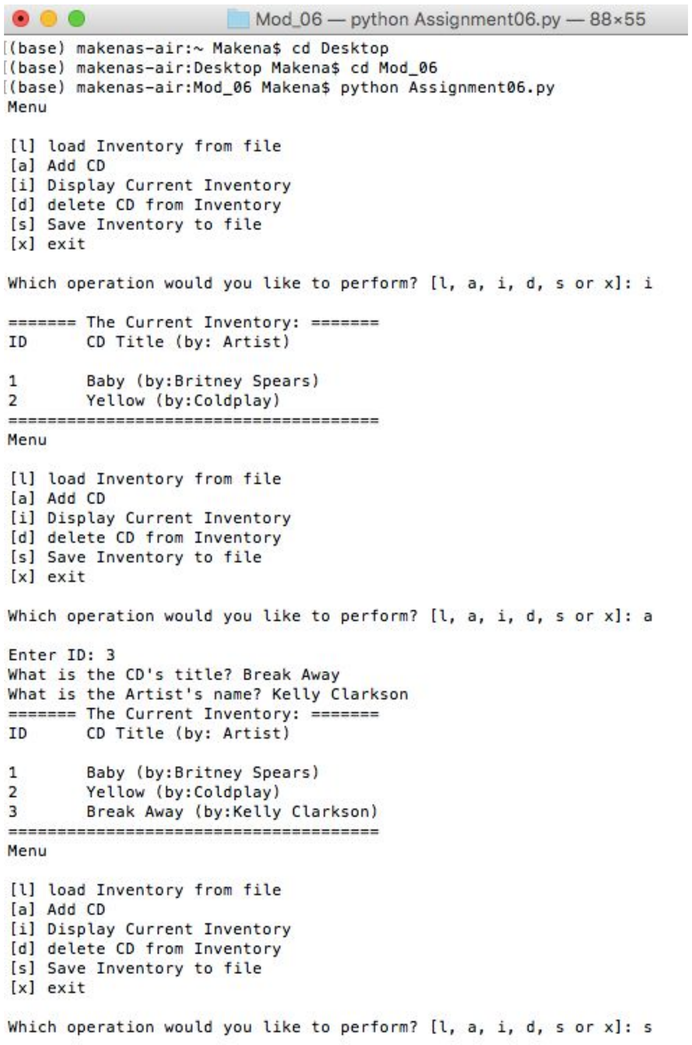
I checked the text file to make sure the data had been written in properly; it had.



```
1,Baby,Britney Spears
2,Yellow,Coldplay
```

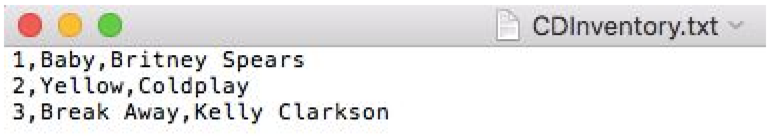
Figure 15: CDInventory.txt with the written data.

I then tested the script from the terminal window, with different data inputs, from my desktop. It also worked properly. The new data was also in the text file.



```
Mod_06 — python Assignment06.py — 88×55
(base) makenas-air:~ Makena$ cd Desktop
(base) makenas-air:Desktop Makena$ cd Mod_06
(base) makenas-air:Mod_06 Makena$ python Assignment06.py
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, d, s or x]: i

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Baby (by:Britney Spears)
2       Yellow (by:Coldplay)
======================================
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, d, s or x]: a

Enter ID: 3
What is the CD's title? Break Away
What is the Artist's name? Kelly Clarkson
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Baby (by:Britney Spears)
2       Yellow (by:Coldplay)
3       Break Away (by:Kelly Clarkson)
======================================
Menu

[l] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, d, s or x]: s
```

Figure 16: The working script in IDLE.



```
CDInventory.txt
1,Baby,Britney Spears
2,Yellow,Coldplay
3,Break Away,Kelly Clarkson
```

Figure 11: Data displayed in CDInventory.txt.

Uploading to Github

Part of this assignment also involved saving the script to Github. To save this script (and this knowledge document) to Github, I created a repository called Assignment_06 and placed the files there. I then shared the link to this repository on the discussion board for this part of the assignment on canvas. Here is the link to my repository: https://github.com/makenaflory/Assignment06

Summary

In this assignment I modified the CD inventory to include functions. I described these functions using docstrings. All functions were organized according to classes, and the script was organized according to the SoC. The program allowed the user to view current inventory from in memory, load data from a text file, add new data to memory, save data to a text file, and quit the program. This script was saved to Github.

Questions and Comments

In the add_cd function, I combined an IO function with a processing function. When I tried to split them up, the function didn't work; I kept on getting an error that the variables in the IO function (strID, strTitle, etc) were defined but not used. I was able to fix this by merging the IO and processing into one IO processing function, but I don't know if that's the most organized thing to do.