Arielle Waller

August 19, 2020

Foundations of Programming (Python)

Assignment06

Assignment06

Introduction

The task for this assignment was to modify a CD inventory script that allowed the user to load inventory from a file, display the current inventory, delete a CD from the inventory, save the inventory to a file, and exit. In modifying the script, we were required to edit and add to existing functions and classes to organize repetitive tasks.

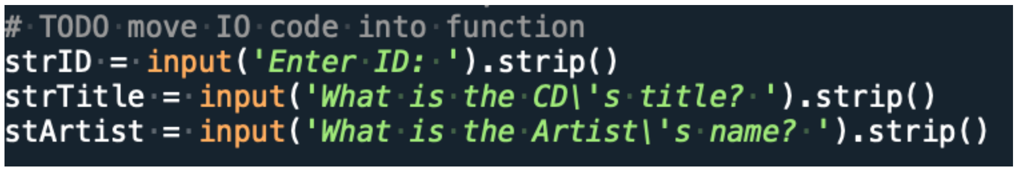
The Approach

When I first began editing the script, I made the mistake of not reviewing it completely. I began adding functions to classes without realizing that the code many of them would contain was already created further down in the script.

I started with a fresh Assignment06_Starter.py script, skipping to the "TODO"s in the main *while loop*.

Collecting User Input to Add a New CD

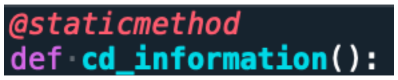
The first *TODO* in the main *while* loop involved the code block associated with the user choosing to add a CD to the inventory table. It instructed me to move the IO code asking the user for the CD's ID, title, and artist into a function. I copied this code block and navigated to *class IO* since this is where my new method would be stored.



```
# TODO move IO code into function
strID = input('Enter ID: ').strip()
strTitle = input('What is the CD\'s title? ').strip()
stArtist = input('What is the Artist\'s name? ').strip()
```

Figure 1 – Copying the original code to be placed in a function.

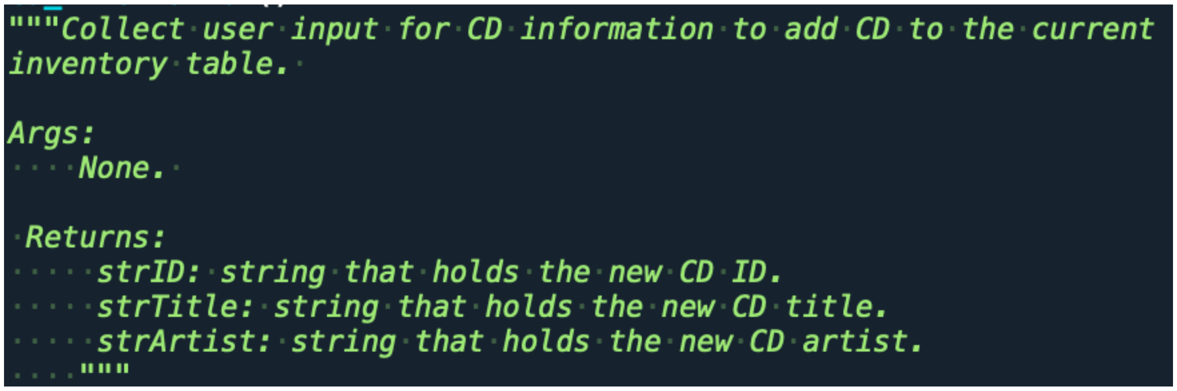
I created a new static method called *cd_information* that required no arguments.



```
@staticmethod
def cd_information():
```

Figure 2 - Creating a new static method with no arguments.

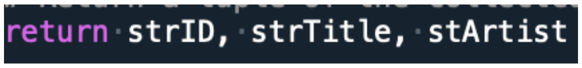
In the docstring, I included a brief description of the method, arguments, and the returns.



```
"""Collect user input for CD information to add CD to the current
inventory table.

Args:
    None.

 Returns:
     strID: string that holds the new CD ID.
     strTitle: string that holds the new CD title.
     strArtist: string that holds the new CD artist.
    """
```

Figure 3 - Describing the function with a docstring.

I pasted the code block prompting the user for the CD's ID, title, and artist and assigned the inputs to the variables *strID, strTitle*, and *stArtist* respectively. Then, I added a *return* statement so that a tuple of these variables would be returned upon calling the function.

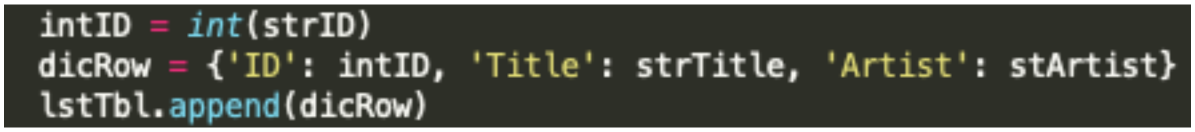


```
return strID, strTitle, stArtist
```

Figure 4 - Returning a tuple of the user's CD information input.

Adding the User Input to the Table

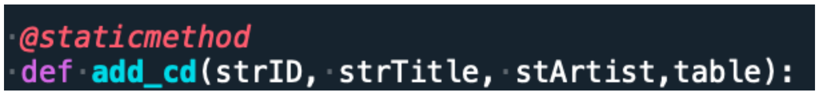
The next *TODO* was part of the same conditional statement, the use case of adding a new CD to the table. I was instructed to move the code block that processed the user's input into a function. Again, I copied the code block, this time navigating to *class DataProcessor.*



```
intID = int(strID)
dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
lstTbl.append(dicRow)
```

Figure 5 - Copying the original code block to be pasted into a function.

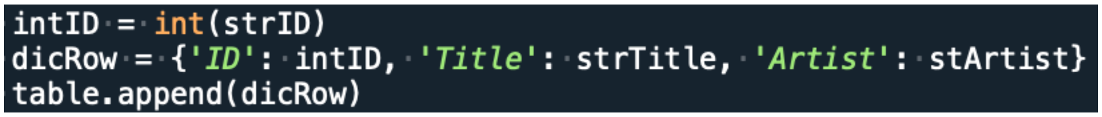
For this new method, *add_cd*, I required four arguments: *strID, strTitle, stArtist, and table*.



```
@staticmethod
def add_cd(strID, strTitle, stArtist,table):
```

Figure 6 - Creating a static method requiring four arguments.

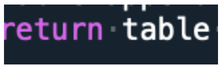
As previously, I used a docstring to provide additional information about my method. Next, I pasted the code block. I changed any references in this function to *lstTbl* to *table* to minimize confusion if other tables were to be worked with.



```
intID = int(strID)
dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
table.append(dicRow)
```

Figure 7 – Pasting and modifying the copied code block to be more inclusive of all tables.

I added a statement to return *table* if the function were to be called.

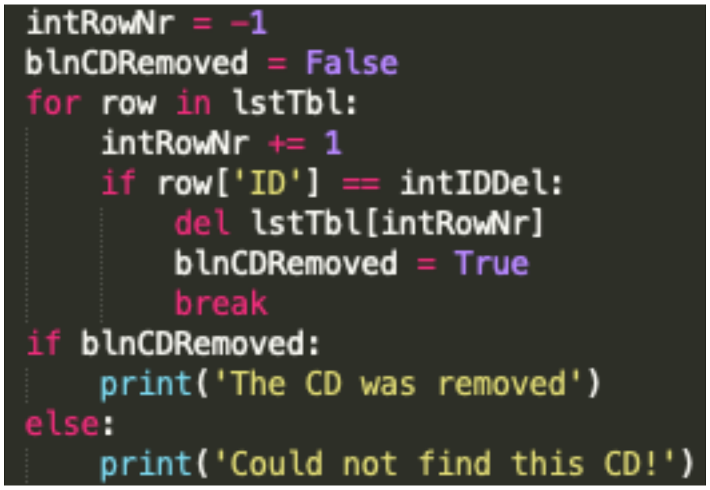


```
return table
```

Figure 8 - Returning the modified table.

Deleting Inventory

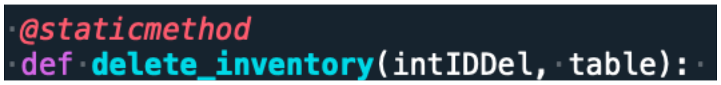
I found the next *TODO* in the conditional statement for deleting a CD from the inventory table.



```
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!')
```

Figure 9 - Copying the original code block to be added to a function.

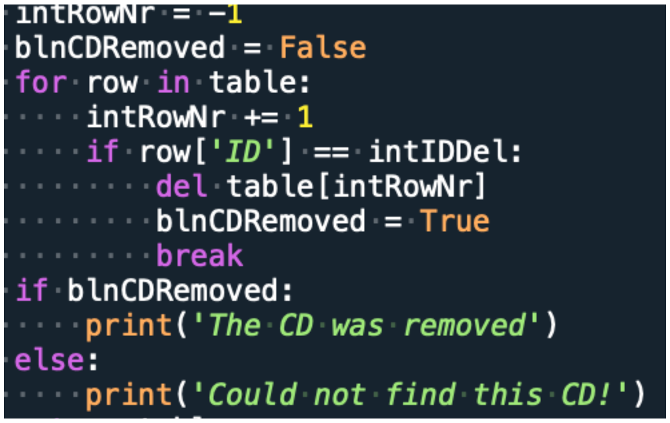
It instructed me to move the processing code into a function. Copying the code, I navigated to *class DataProcessor* and created the method *delete_inventory*. For this method, I required two arguments, the ID of the CD to be deleted (*intIDDel*) and the table from which the CD would be deleted (*table).*



```
@staticmethod
def delete_inventory(intIDDel, table):
```

Figure 10 - Creating a method requiring two arguments.

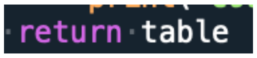
I described the function in a docstring, pasted the code, and changed references of *lstTbl* to *table*.



```
intRowNr = -1
blnCDRemoved = False
for row in table:
    intRowNr += 1
    if row['ID'] == intIDDel:
        del table[intRowNr]
        blnCDRemoved = True
        break
if blnCDRemoved:
    print('The CD was removed')
else:
    print('Could not find this CD!')
```

Figure 11 - Pasting and modifying the original code to be more inclusive of all tables.

I added a *return* statement to return the modified table upon calling the function.

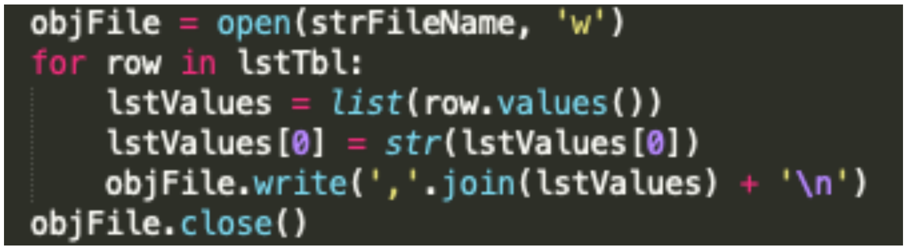


```
return table
```

Figure 12 - Returning the modified table.

Saving Inventory to a File

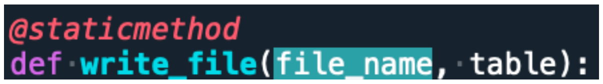
The following *TODO* instructed me to move the processing code for saving data to the file into a function.



```
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 13 - Copying the original code to be placed into a function.

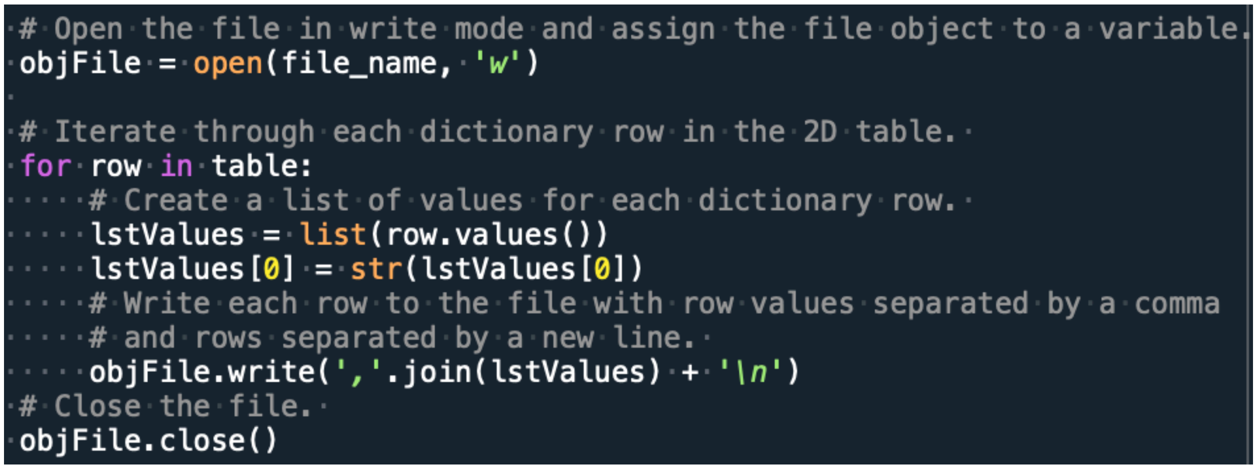
I copied the code, went to class *FileProcessor*, and created a method called *write_file*. The method took two arguments, the filename (*file_name*) and a table (*table*).



```
@staticmethod
def write_file(file_name, table):
```

Figure 14 - Creating a static method with two arguments.

Once more, I described the function with a docstring, pasted the code, and changed all references of *lstTbl* to *table*.



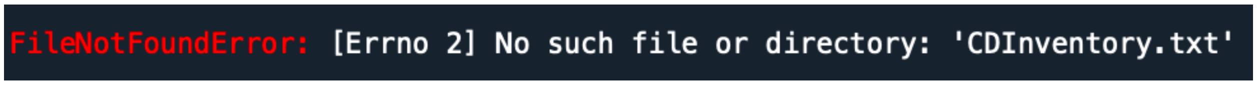
```
# Open the file in write mode and assign the file object to a variable.
objFile = open(file_name, 'w')

# Iterate through each dictionary row in the 2D table.
for row in table:
    # Create a list of values for each dictionary row.
    lstValues = list(row.values())
    lstValues[0] = str(lstValues[0])
    # Write each row to the file with row values separated by a comma
    # and rows separated by a new line.
    objFile.write(','.join(lstValues) + '\n')
# Close the file.
objFile.close()
```

Figure 15 - Pasting and modifying the original code to be more inclusive of all tables.

Finally, I removed the *pass* statements to check the functionality was as intended.

Testing the Script

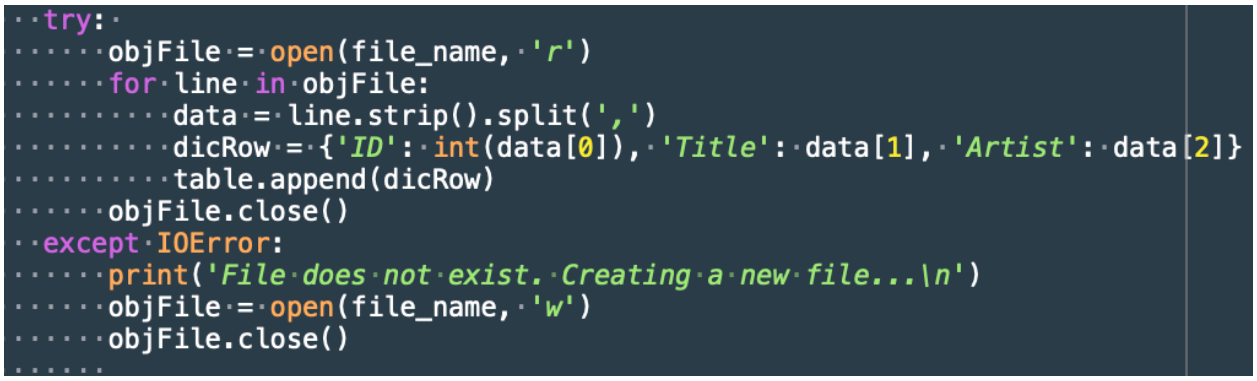
While running the script in Spyder, I got an error that no file or directory existed.



```
FileNotFoundError: [Errno 2] No such file or directory: 'CDInventory.txt'
```

Figure 16 - Error resulting from no CDInventory.txt file.

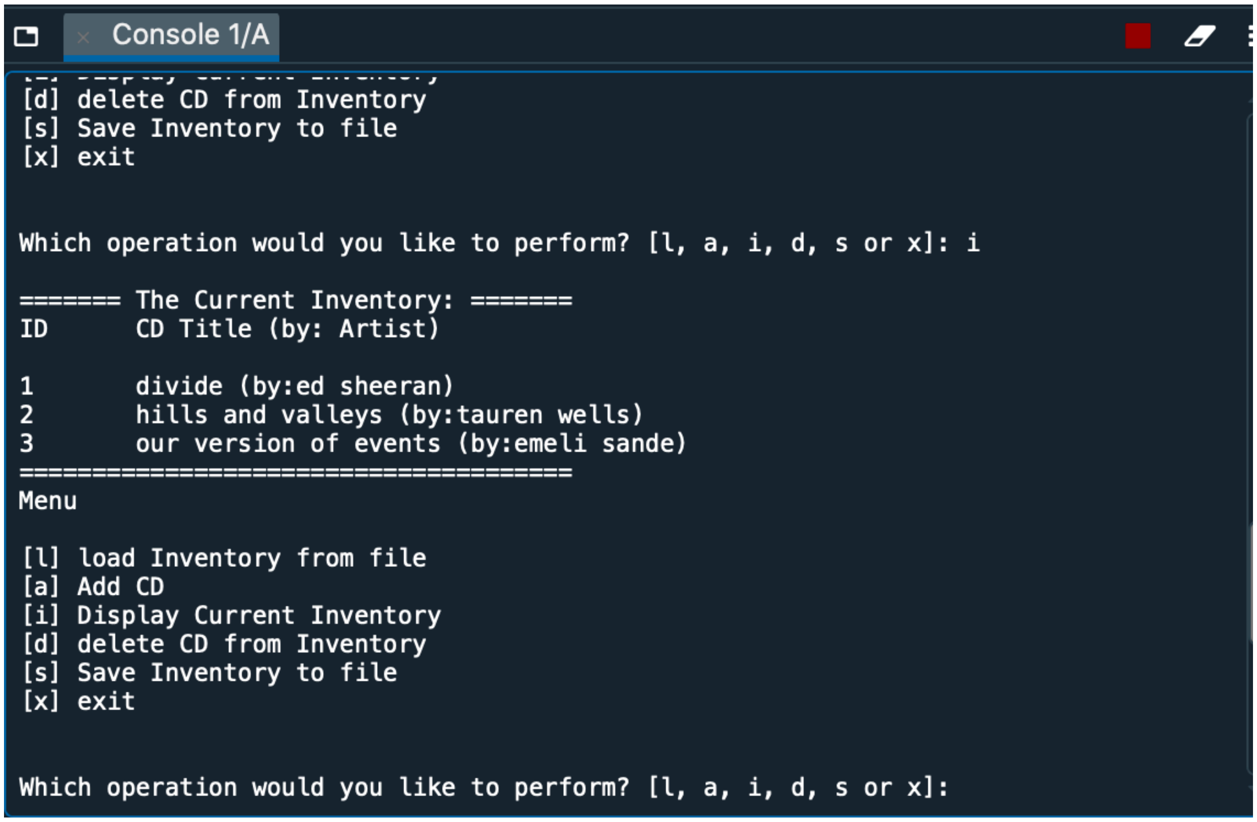
Using last week's knowledge[i], I added a *try-except* statement to the *read_file* method in *class FileProcessor*. If the file did not exist, I added a statement informing the user of this and that a new file would be created. I then created a new file with *'write'* mode before closing it.



```
  try:
      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()
  except IOError:
      print('File does not exist. Creating a new file...\n')
      objFile = open(file_name, 'w')
      objFile.close()
```

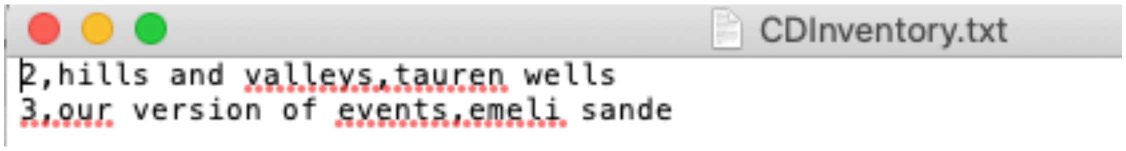
Figure 17 - Using try-except to handle the 'FileNotFoundError.'

Running the Script in Spyder



```
[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       divide (by:ed sheeran)
2       hills and valleys (by:tauren wells)
3       our version of events (by:emeli sande)
======================================
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]:
```

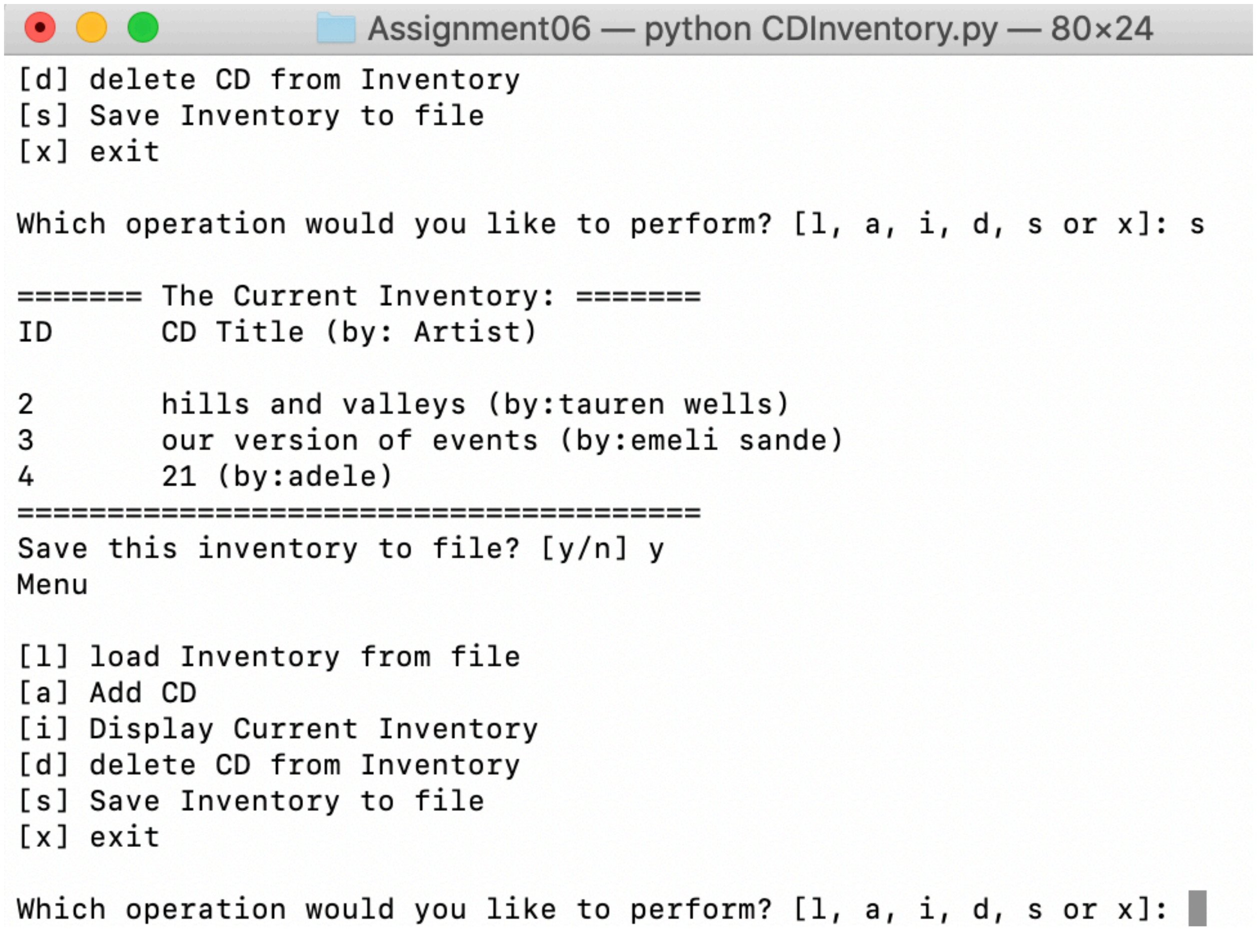
Figure 18 - Running CDInventory.py in Spyder.



```
2,hills and valleys,tauren wells
3,our version of events,emeli sande
```

Figure 19 - CDInventory.txt in TextEdit after deleting a CD in Spyder and saving to the file.

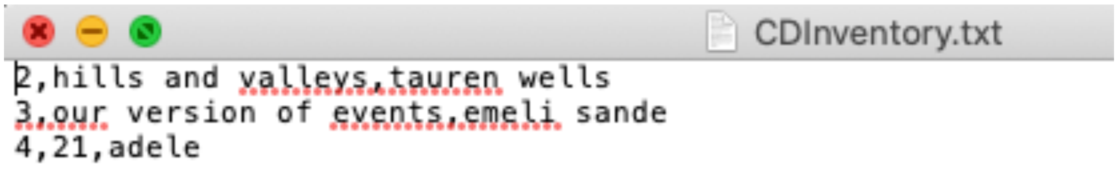
Running the Script in the Terminal



Assignment06 — python CDInventory.py — 80×24

```
[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

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

2       hills and valleys (by:tauren wells)
3       our version of events (by:emeli sande)
4       21 (by:adele)
======================================
Save this inventory to file? [y/n] y
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]: 
```

Figure 20 - Running CDInventory.py in the terminal.



CDInventory.txt

```
2,hills and valleys,tauren wells
3,our version of events,emeli sande
4,21,adele
```

Figure 21 - CDInventory.txt in TextEdit window after adding a CD in the terminal and saving to the file.

Summary

In summary, this week’s assignment involved making a CD inventory script more concise and efficient through classes and functions. It was slightly easier this week than last week as far as working with an existing script. The most frequent error I made was with indentation. Several times my script told me that it could not find an attribute for a class because I had not properly indented. Other errors involved calls to functions without using parenthesis, not specifying a return value, and having an incorrect number of arguments stated.

Appendix

Using planetb's webpage[ii] (external site):

```
#------------------------------------------#
# Title: CDInventory.py
# Desc: Working with classes and functions.
# Change Log: (Who, When, What)
# AWaller, 2020-August-19, Created File
# AWaller, 2020-August-19, Added cd_information function to class IO.
# AWaller, 2020-August-19, Added add_cd function to class DataProcessor.
# AWaller, 2020-August-19, Added delete_inventory method to class DataProcessor.
# AWaller, 2020-August-19, Added write_file method to class FileProcessor.
# AWaller, 2020-August-19, Added try-except error handling to read_file method in FileProcessor.


#------------------------------------------#

# -- 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

dicRow1 = {'ID': 1, 'Title': 'follklore', 'Artist': 'taylor swift'}
# dicRow2= {'ID': 2, 'Title': 'solitude', 'Artist': 'tori kelly'}
lstTbl.append(dicRow1)
# lstTbl.append(dicRow2)

# -- PROCESSING -- #
class DataProcessor:
    # TODONE: Added functions for processing here.

    @staticmethod
    def add_cd(strID, strTitle, stArtist,table):
        """Add a CD to the current inventory table.


        Args:
            strID: string that holds the CD ID.
            strTitle: string that holds the CD title.
            strArtist: string that holds the CD artist.
            table (list of dict): 2D data structure (list of dicts) that holds
            the data during runtime.

        Returns:
            table: 2D data structure (list of dicts) that holds the data
            during runtime. Modified to include the new CD.

```

```python
        """

        # 3.3.2 Add item to the table
        intID = int(strID)
        dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
        table.append(dicRow)
        return table


    @staticmethod
    def delete_inventory(intIDDel, table):
        """Deletes a CD from the current inventory table


        Args:
            intIDDel: the ID integer of the CD to be deleted
            table (list of dict): 2D data structure (list of dicts) that holds
            the data during runtime.

        Returns:
            table (list of dict): 2D data structure (list of dicts) that holds
            the data during runtime. If intIDDel is found, this list is edited
            to remove the applicable dictionary.

        """
        # 3.5.2 search thru table and delete CD
        intRowNr = -1
        blnCDRemoved = False
        for row in table:
            intRowNr += 1
            if row['ID'] == intIDDel:
                del table[intRowNr]
                blnCDRemoved = True
                break
        if blnCDRemoved:
            print('The CD was removed')
        else:
            print('Could not find this CD!')
        return table

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.
              """
        # Clear the existing data and allow to load data from file.
        table.clear()
        try:
            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()
        except IOError:
            print('File does not exist. Creating a new file...\n')
            objFile = open(file_name, 'w')
            objFile.close()


    @staticmethod
    def write_file(file_name, table):
        """Function to write data from the list of dictionaries to a file.

        Takes the data from the 2D table and represeents one dictionary row in
        the table as one line in the file. (The values of each dictionary row
        are reperesented as one line in the table)

        Args:
            file_name (string): name of file used to write the data to
            table (list of dict): 2D data structure (list of dicts) that holds
            the data during runtime
            table (list of dict): 2D data structure (list of dicts) that holds
            the data during runtime.
        Returns:
            None.

        """
        # Open the file in write mode and assign the file object to a variable.

        objFile = open(file_name, 'w')

        # Iterate through each dictionary row in the 2D table.
        for row in table:
            # Create a list of values for each dictionary row.
            lstValues = list(row.values())
            lstValues[0] = str(lstValues[0])
            # Write each row to the file with row values separated by a comma
            # and rows separated by a new line.
            objFile.write(','.join(lstValues) + '\n')
        # Close the file.
        objFile.close()



# -- PRESENTATION (Input/Output) -- #

class IO:
    """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')


    @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


    @staticmethod
    def show_inventory(table):
        """Displays a CD from the 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 cd_information():
        """Collect user input for CD information to add CD to the current
        inventory table.

        Args:
            None.

```

```
                 Returns:
                     strID: string that holds the new CD ID.
                     strTitle: string that holds the new CD title.
                     strArtist: string that holds the new CD artist.
                    """
                # Prompt user for CD information.
                strID = input('Enter ID: ').strip()
                strTitle = input('What is the CD\'s title? ').strip()
                stArtist = input('What is the Artist\'s name? ').strip()
                # Return a tuple of the collected values.
                return strID, strTitle, stArtist


        # # 1. When program starts, read in the currently saved Inventory
        FileProcessor.read_file(strFileName, lstTbl)

        # 2. start main loop
        while True:
            # 2.1 Display Menu to user and get choice
            IO.print_menu()
            strChoice = IO.menu_choice()
            # 3. Process menu selection
            # 3.1 process exit first
            if strChoice == 'x':
                break
            # 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.
            # 3.3 process add a CD
            elif strChoice == 'a':
                # 3.3.1 Ask user for new ID, CD Title and Artist. TO-DONE: Moved IO code
                # into function.
                # # Unpack the tuple returned from the function.
                cd_id, title, artist = IO.cd_information()
                # 3.3.2 Add CD to the table.
                # TO-DONE: Moved processing code into function.
                DataProcessor.add_cd(cd_id, title, artist, lstTbl)
                # Display the new inventory table.
                IO.show_inventory(lstTbl)
                continue  # start loop back at top.
            # 3.4 process display current inventory
            elif strChoice == 'i':
                IO.show_inventory(lstTbl)
                continue  # start loop back at top.
            # 3.5 process delete a CD
            elif strChoice == 'd':
                # 3.5.1 get Userinput for which CD to delete
            # 3.5.1.1 display Inventory to user
            IO.show_inventory(lstTbl)
            # 3.5.1.2 ask user which ID to remove.
            intIDDel = int(input('Which ID would you like to delete? ').strip())
            # 3.5.2 search thru table and delete CD.TO-DONE - Moved processing code
            # into function.
            DataProcessor.delete_inventory(intIDDel, lstTbl)
            # Display the new CD inventory table.
            IO.show_inventory(lstTbl)
            continue  # start loop back at top.
        # 3.6 process save inventory to file
        elif strChoice == 's':
            # 3.6.1 Display current inventory and ask user for confirmation to save
            IO.show_inventory(lstTbl)
            strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
            # 3.6.2 Process choice
            if strYesNo == 'y':
                # 3.6.2.1 save data
                # TO-DONE: Moved processing code into write_function.
                FileProcessor.write_file(strFileName, lstTbl)
            else:
                input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
            continue  # start loop back at top.
        # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
        else:
            print('General Error')
```

[i] Retrieved 2020-August-19
[ii] Retrieved 2020-August-19