Rei Bengu
August 28, 2021
IT FDN 110 B
Assignment 08

Object-Oriented Programming

Introduction

In this module we explored object-oriented programming. While we have been using attributes and methods for some time now which are involved in object-oriented programming, we focused on bringing it all together in this module. To put these skills to the test we improved upon the CD Inventory script yet again to take advantage of objects, and work with objects instead of dictionaries to house the data for our CD Inventory and right off the bat the benefits of objects is clear. With objects the code becomes a lot more realistic, if you have a stack of CDs, each CD has its own set of individual attributes, its place in the stack, the title of the CD, the artist of the CD. With objects, each object represents a CD with its own attributes. The script used to practice these methods as well as this knowledge document can also be found at github to see the progression of this assignment.

CD Inventory Script

To put to practice the skills learned in this module, the basic script created and refined over past modules which asked the user to enter basic CD data to create an expanding table of information was revisited and updated with the concepts learned in this module on object-oriented programming. Because this program has gone through several iterations, I will be focusing primarily on the new features of this iteration and its propagated effects on the program and will not re-iterate the contents of the program that can be found in previous modules.

The next iteration of this program shifts from utilizing lists of dictionaries to store data on CDs into objects. The class CD is used to create objects which represent an individual CD. Each object has three attributes: cd_id, cd_title and cd_artist. The code below shows how the CD class is created which is used to define the objects as well as the decorators used to establish the getters and setters for each of the attributes.

```
1. class CD:
2.     """Stores data about a CD:
3.
4.     properties:
5.         cd_id: (int) with CD ID
6.         cd_title: (string) with the title of the CD
7.         cd_artist: (string) with the artist of the CD
8.     methods:
9.
10.         """
11.
12.         def __init__(self, cd_id, cd_title, cd_artist):
13.             self.id = cd_id
14.             self.title = cd_title
15.             self.artist = cd_artist
16.
17.         @property
18.         def cd_id(self):
19.             return self.id
```

```
20.
21.         @property
22.         def cd_title(self):
23.             return self.title
24.
25.         @property
26.         def cd_artist(self):
27.             return self.artist
28.
29.         @cd_id.setter
30.         def cd_id(self, value):
31.             self.id = value
32.
33.         @cd_title.setter
34.         def cd_title(self, value):
35.             self.id = value
36.
37.         @cd_artist.setter
38.         def cd_artist(self, value):
39.             self.artist = value
40.
41.         def __str__(self):
42.             return (str(self.cd_id) + ',' +  self.cd_title + ',' +
    self.cd_artist )
43.
```

The code below shows how objects are created by a user's input and then added to a list of objects saved in memory. The users input is collected as normal, however CD(ID, title, artist) is used to create the object with the users inputs, and is appended to the list of objects. An example of how this list is accessed can be seen in the next sample of code, and highlights the benefits of using objects.

```
1. # -- PRESENTATION (Input/Output) -- #
2. class IO:
3.     """Handles Input/Output with user"""
4.
5.     @staticmethod
6.     def new_entry():
7.         """
8.         Asks the user to enter details for their new entry, and adds it to
    memory
9.
10.             Args:
11.                 None.
12.
13.             Returns:
14.                 ID (string): ID of new entry
15.
16.                 Title (string): Title of new entry
17.
18.                 Artist(string): Artist of new entry
```

```
            """
            ID = int(input('Enter ID: ').strip())
            title = input('What is the CD\'s title? ').strip()
            artist = input('What is the Artist\'s name? ').strip()
            lstOfCDObjects.append( CD(ID, title, artist))
```

The code below displays the contents of the list of objects to the user. Utilitizing a for loop, each element within the list represents an individual CD with its own unique set of attributes (ID, Title and Artist) like how each CD in a stack has its own individual attributes.

```
# -- PRESENTATION (Input/Output) -- #
class IO:
    """Handles Input/Output with user"""

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


            Args:
                list (list of elements): data structure that holds the data during runtime.

            Returns:
                None.

            """
            print('======= The Current Inventory: =======')
            print('ID\tCD Title (by: Artist)\n')
            for obj in list:
                print('{}\t{} (by:{})'.format( obj.cd_id, obj.cd_title, obj.cd_artist))
            print('======================================')

```

Figure 1 shows an example output within the Spyder IDE. Figure 2 shows the contents of CDInventory.dat file after the run, because the file is written in binary it is not easily readable without manipulation. Figure 3 is an example of the program running in a terminal.

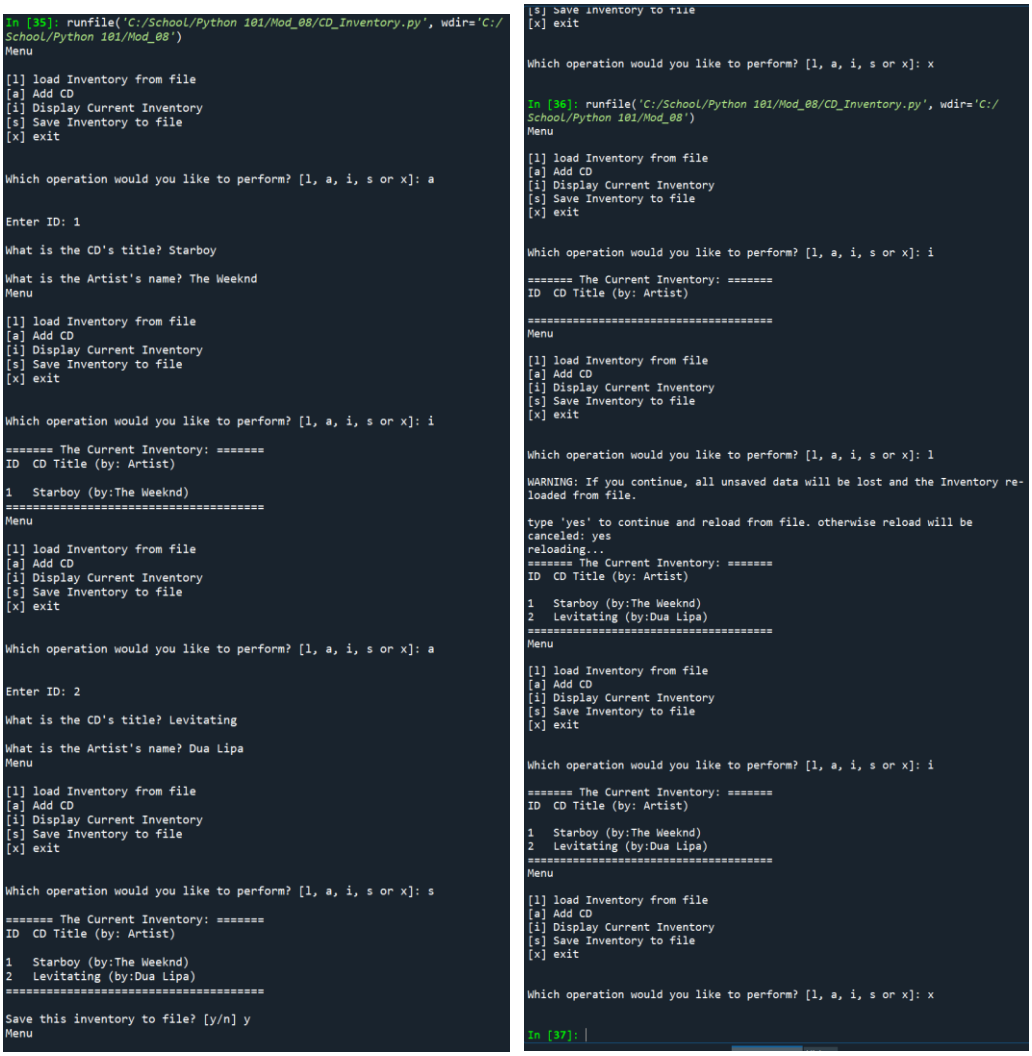

Figure 1 – Example output from working script in Spyder IDE

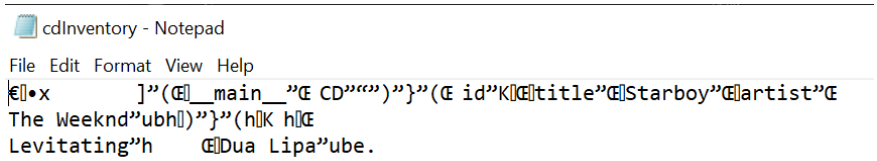

Figure 2 - CDInventory.dat generated by the script after it was run in Spyder IDE

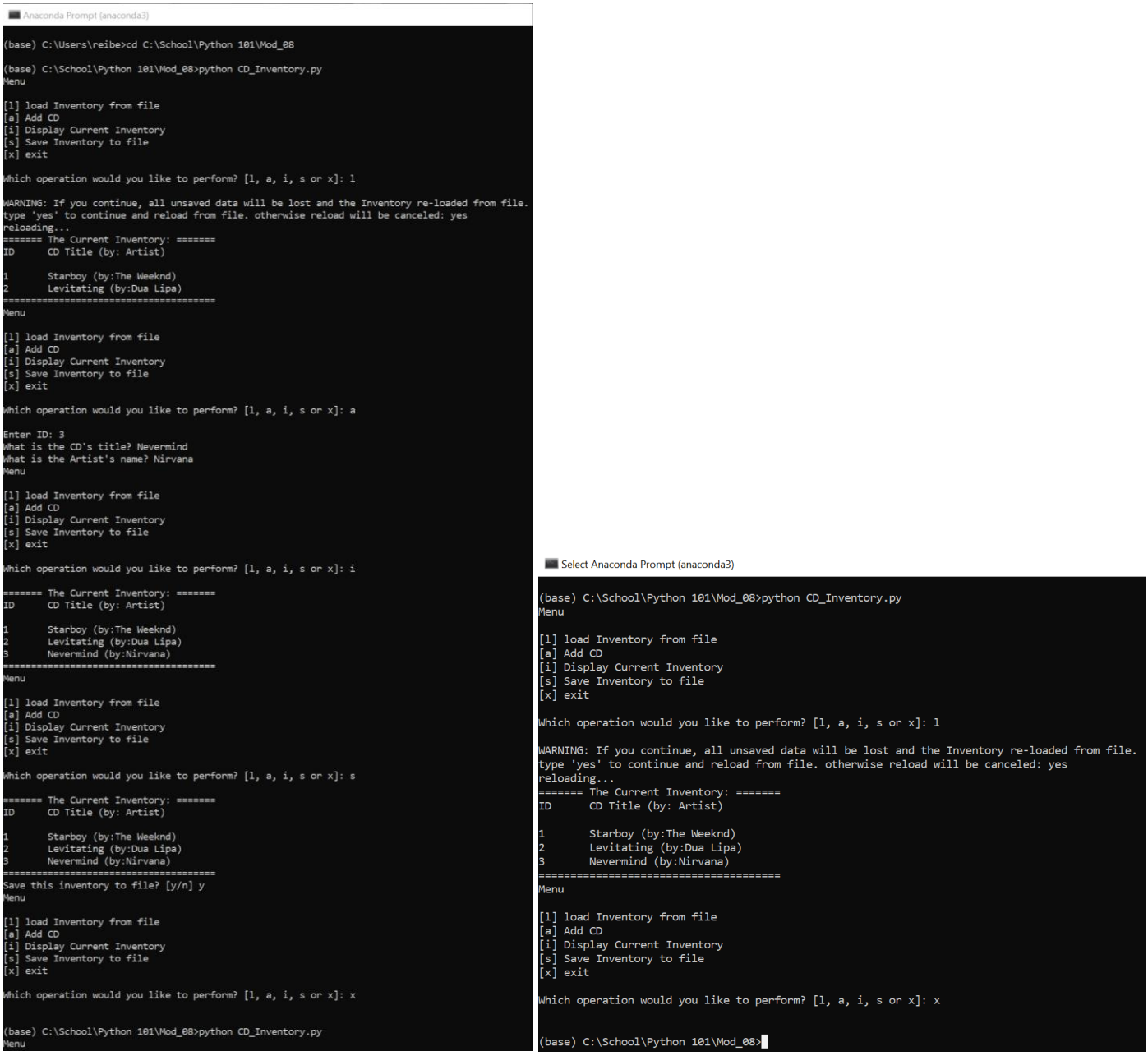

Figure 3 - Example of the program running in a terminal.

Summary

Throughout this module, we focused on adding functionality to improve the program. By using objects, we can continue to improve the functionality of the program with minimal updates to the rest of the program by updating the properties within the class instead. There are still some concepts that are rather confusing with object-oriented programming which will make more sense with different applications of object-oriented programming.