Joseph Prado

29 March 2020

## IT FDN 100 B

Assignment 08

**Modifying a script to use objects**

# Introduction

In this assignment, I will do several things.

1. Create a script from instructor provided pseudocode to create a CD Inventory
2. Recycle code from previous CD Inventory assignments to assist in creating script
3. Modify existing code to use objects to represent stored CDs in runtime.

# **Modifying Assignment 06’s and Assignment 07’s CD Inventory Scripts to store CD information in runtime as objects.**

## **Creating CD Class.**

In order to instantiate an CD Object, I need to create a Class, or in other words blueprint for what my objects should look like.[[1]](#footnote-1)

1. **class** CD(object):
2. **def** \_\_init\_\_(self, cdID, title, artist, lst\_Inventory):
3. **if** **not** str(cdID).isnumeric():
4. **print** ("\nYou must enter the CD ID as an integer. Adding CD failed.")
5. input("Press 'Enter' to Continue")
6. **else**:
7. self.\_\_cd\_id = cdID
8. self.\_\_cd\_title = title
9. self.\_\_cd\_artist = artist
10. row = self
11. lst\_Inventory = lst\_Inventory.append(row)

**First, I start off the class with a constructor function (\_\_init\_\_). This code is executed any time the object of this class in instantiated.**[[2]](#footnote-2)

**Later on the main body of the code, this class is called upon when the code instantiates an object after requesting input from the use for CD values. One of those values is cdID which must be in integer form. Before setting up initial attributes, I use an if/else pair to catch any instance where the user did not cdID as a string of numbers.**

**Then, I set up the attributes for the object using the self parameter and dot notation.**[[3]](#footnote-3) **I preface each attribute with a double underscore to make them private to encourage encapsulation and avoid any situation where the attributes are accessed directly without the use of the methods’s parameters.**[[4]](#footnote-4)

**I then use the append() method to add the created CD Object to a list of objects lst\_Inventory that is also included the constructor’s function parameters.**

**So, in short, this function is taking the CD information either inputed by the user or read from a text file (more on that later), creating a CD object, and appending the CD object to a list of CD objects.**

1. @property
2. **def** cd\_id(self):
3. **return** self.\_\_cd\_id
5. @cd\_id.setter
6. **def** cd\_id(self, value):
7. **if** **not** str(value).isnumeric():
8. **print**("ID must be an integer!")
9. **else**:
10. self.\_\_cd\_id = value
12. @property
13. **def** cd\_title(self):
14. **return** self.\_\_cd\_title
16. @cd\_title.setter
17. **def** cd\_title(self, value):
18. **if** str(value).isnumeric():
19. **print**("Title can't just be numbers!")
20. **else**:
21. self.\_\_cd\_title = value
23. @property
24. **def** cd\_artist(self):
25. **return** self.\_\_cd\_artist
27. @cd\_artist.setter
28. **def** cd\_artist(self, value):
29. **if** str(value).isnumeric():
30. **print**("Artist can't just be numbers!")
31. **else**:
32. self.\_\_cd\_artist = value

**In the constructor function, I made all the attributes private. As you’ll notice later in the script, there are a few areas where attributes are accessed directly. Namely, this is when the script reads from the list of objects to display the inventory or save the inventory to text file. In order to enable accessing the attributes from the CD objects, I need to create properties for each private attribute.**[[5]](#footnote-5) **As shows above, I do this using creating methods with the attribute names preceded by @property decorator.**[[6]](#footnote-6) **Each method then returns the value of the attribute.**[[7]](#footnote-7)

**Using similar logic, I create methods prefaced by the decorator xxxx.setter to provide the script write access to these attributes.**[[8]](#footnote-8) **In each case where I do this, I also include some code to trap errors resulting from improperly formatted data. Strictly, speaking, these setter functions are not necessary in this script, since there are no places in the main script that require direct write access to the attributes, but to be safe, I figure I should demonstrate that I know how to use the “setter” functionality it at least once.**

## **Reading and writing from a text file**

I had to modify the scripts from Assignments 6 and 7 and somewhat to accommodate the use of a list of objects instead of a list of dictionaries to hold the information in run time.

1. **class** FileIO:
2. @staticmethod
3. **def** load\_inventory(file\_name, lst\_Inventory):
4. lst\_Inventory.clear()
5. **try**:
6. objFile = open(file\_name, 'r')
7. **for** line **in** objFile:
8. data = line.strip().split(',')
9. objCD = CD(data[0], data[1], data[2], lst\_Inventory)
10. objFile.close()
11. **except** IOError:
12. **print**("\nNo CD file exists.  Add and save CD info to inventory to create a file.")
13. input("Press 'Enter' to continue.")
14. @staticmethod
15. **def** save\_inventory(file\_name, lst\_Inventory):
16. objFile = open(file\_name, 'w')
17. **for** row **in** lst\_Inventory:
18. cdID = row.cd\_id
19. cdID = str(cdID)
20. title = row.cd\_title
21. artist = row.cd\_artist
22. objFile.write("{},{},{}\n".format(cdID, title, artist))
23. objFile.close()

To read of the text file and save the information as list of objects, I iterate through each line of the text file to create a list. I then use indexing to make each item in the list an argument for the attributes used to create the object **objCD** as demonstrated at location 1:11 [here.](https://washington.zoom.us/rec/play/75cocLv9qj43SIeR4gSDV_AvW9TsJ_2sgyUXq_Bcz020WnYFZ1qhZuAWYLS0bTXStm0btlUk7vB9iCpB?continueMode=true)[[9]](#footnote-9) The constructor function invoked as the CD object is created automatically appends the object to the list of CD objects **lstOfCdObjects** so there is no need to run the append function here like we did in Assignment 6 and 7.

**To save to a text file, I iterate through each object in lstofCDObjects. For each object, I use dot notation to access each attribute in the object and assign it to a variable.**[[10]](#footnote-10) **Even, though the attributes are private, I’m able to access them due to the properties I created in the class preceded with the @property decorator.**[[11]](#footnote-11) **I then used the local variables to generate a string that is written to the text file.**

## **Displaying the inventory**

I also needed to make some modification to the function to display the inventory in runtime to the user, since it now exists as a list of objects. I handle this almost identically to how I saved to a text file above.

1. **class** IO:
3. @staticmethod
4. **def** show\_inventory(lst\_Inventory):
5. **print**('======= The Current Inventory: =======')
6. **print**('ID\tCD Title (by: Artist)\n')
7. **for** row **in** lst\_Inventory:
8. cdID = row.cd\_id
9. title = row.cd\_title
10. artist = row.cd\_artist
11. **print**('{}\t{} (by: {})'.format(cdID, title, artist))
12. **print**('======================================')

**In short, I iterate through each object in lstofCDObjects, use dot notation to access each object attribute, assign each attribute to a variable, and then use the print function to display the variables in a readable format. Again, the properties (@property) I created for each attribute in the class CD allow me directly access the attributes like I did here.**

## **Adding a CD**

The last place I had to significantly modify the script was where you choose to add a CD in the main loop. In Assignments 6 and 7, after gathering the user’s input, the script would call a function that would create a dictionary with the CD information and append it to a list of dictionaries. Previously in the script, we replaced that “**add\_CD**” function with the Class CD and the associated constructor methods **\_\_init\_\_**. So instead of invoking that script, we will invoke Class CD and instantiate a CD object. The constructor method also appends the CD Object to the list of objects **lstOfCDObjects**.

1. **elif** strChoice == 'a': #Process user input
2. strID, strTitle, strArtist = IO.input\_new\_cd()
3. objCD = CD(strID, strTitle, strArtist, lstOfCDObjects)
4. IO.show\_inventory(lstOfCDObjects)

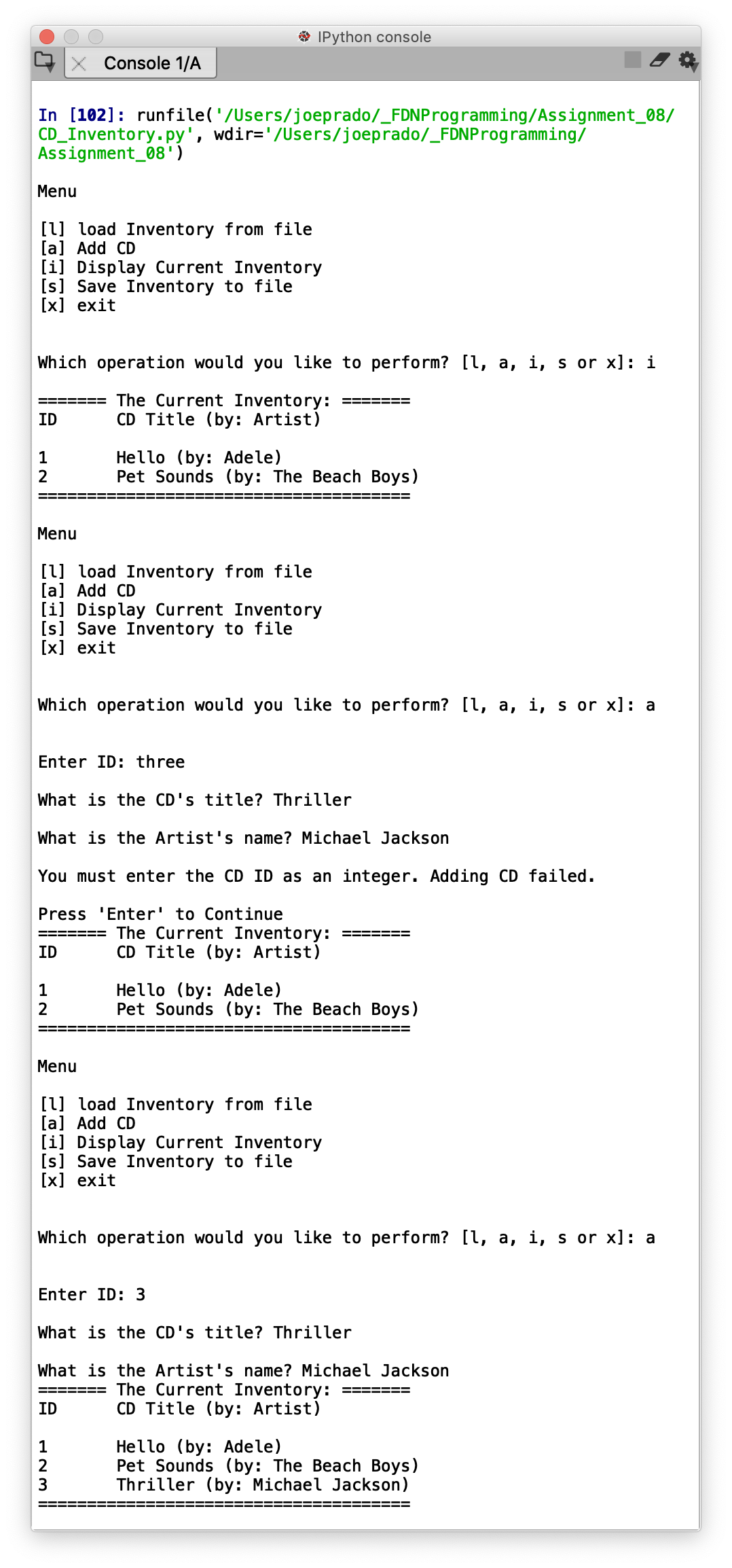
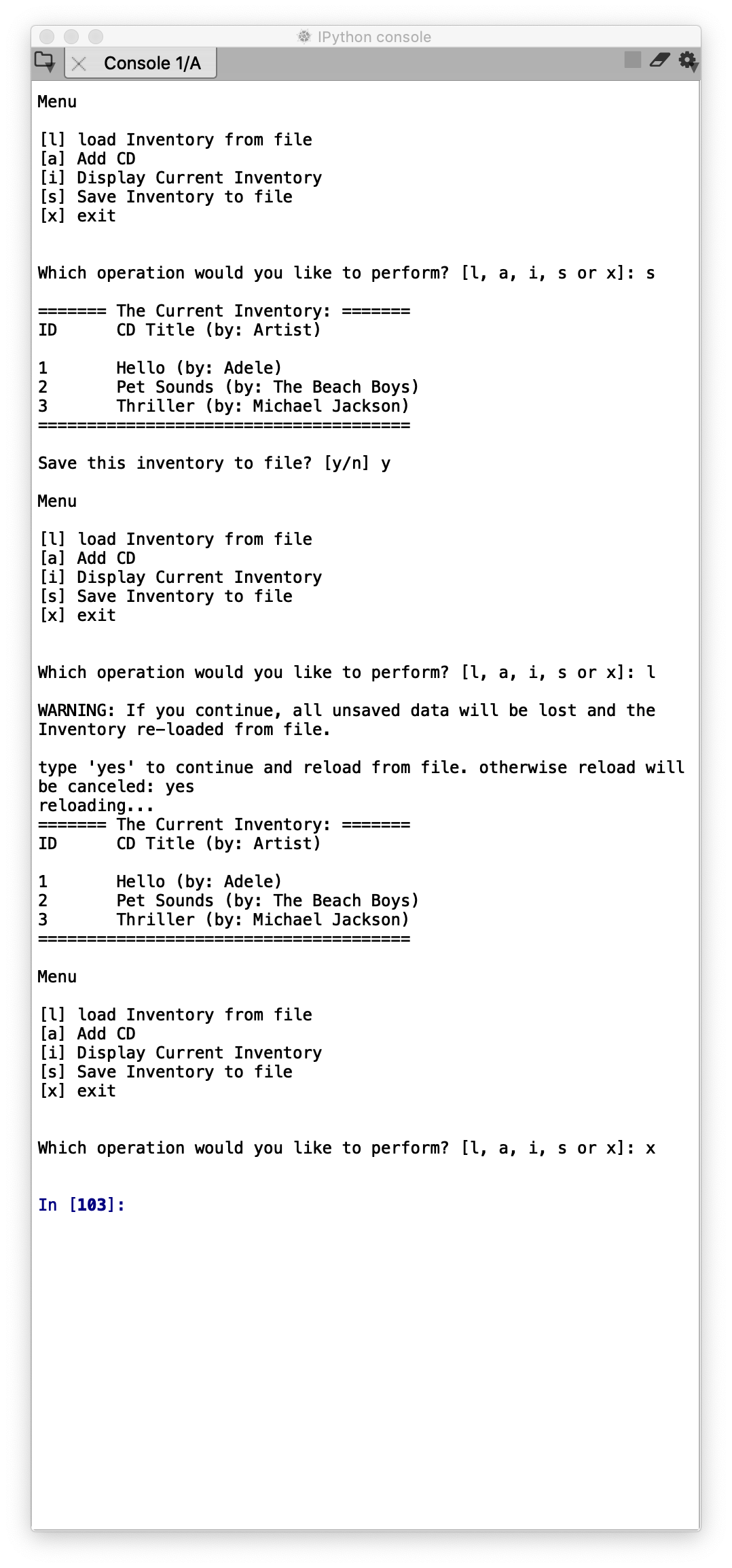
## **Other script housekeeping**

I had to some additional work on the script to get it to work and make it ready for submission. This included

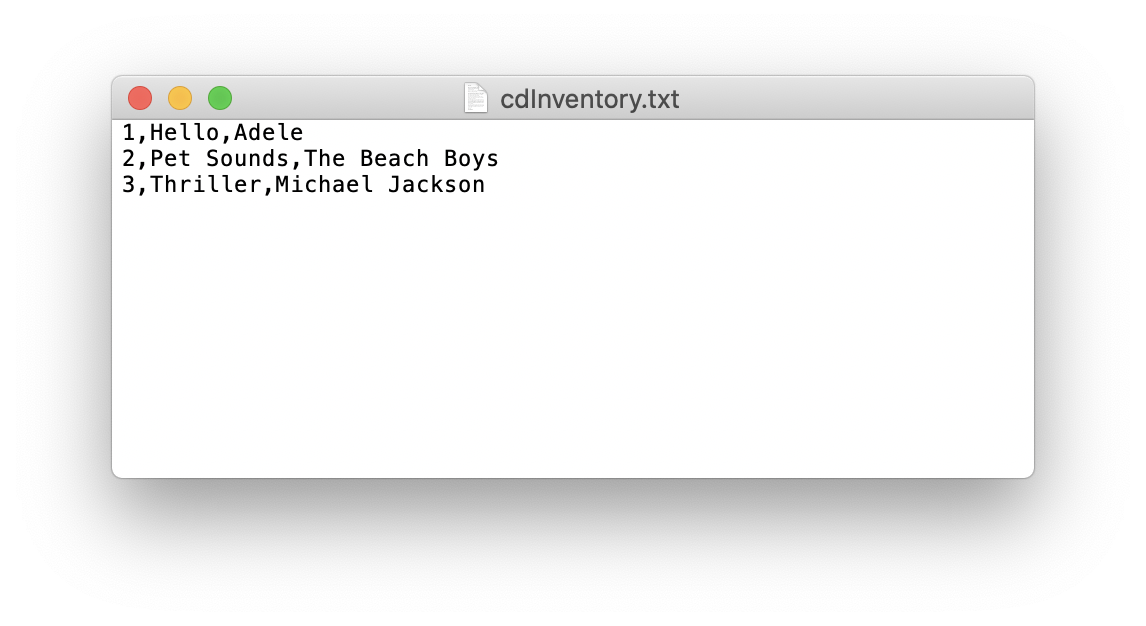
* Deleting the functionality to delete a CD since that is not a capability of the new script.
* Changes to names of the of variables of lists, variables, and parameters used throughout the script
* Updating and adding docstrings.

# Running the script

Running the script in Spyder results in the following:

*Figure 1 – Output from Spyder*

**

*Figure 2 – CDInventory.dat as written to by Spyder*

Running the script in the Terminal results in the following:

# 

# *Figure 3 – Output from Terminal*

**

*Figure 4 – CDInventory.txt as written to by the Terminal.*

Link to GitHub

<https://github.com/joeprado/Assignment_08>

Summary

In this assignment, I:

1. Created a script from instructor provided pseudocode to create a CD Inventory
2. Recycled code from previous CD Inventory assignments to assist in creating script
3. Modified existing code to use objects to represent stored CDs in runtime.

# Appendix

## Listing CD\_Inventory.py

1. #------------------------------------------#
2. # Title: CD\_Inventory.py
3. # Desc: Assignnment 08 - Working with classes
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, created file
6. # DBiesinger, 2030-Jan-01, added pseudocode to complete assignment 08
7. # joeprado, 2020-Mar-28, Added code to adress all TODOs
8. # joeprado, 2020-Mar-20, Changed all object attributes to private
9. # joeprado, 2020-Mar 29, Continued working on code
10. # joeprado, 2020-Mar-29, Changed file name to CD\_Inventory.py
11. # joepradi, 2020-Mar-29, Updated/added docstrings.
12. #------------------------------------------#
14. # -- DATA -- #
15. strFileName = 'cdInventory.txt'
16. lstOfCDObjects = []

19. **class** CD(object):
20. """Stores data about a CD:
22. properties:
23. \_\_cd\_id: (int) with CD ID  (private attribute)
24. \_\_cd\_title: (string) with the title of the CD (private attribute)
25. \_\_cd\_artist: (string) with the artist of the CD (private attribute)
26. methods:
27. \_\_init\_\_(cdID, title, artist, lst\_Inventory): Constructor that conducts error trapping on user CD ID input,
28. automatically creates object attributes, and adds CD information to list of CD objects
29. cd\_id: getter and setter methods to provide read and write to access to private attribute
30. cd\_title: getter and setter methods to provide read and write to access to private attribute
31. cd\_artist: getter and setter methods to provide read and write to access to private attribute
32. """
34. **def** \_\_init\_\_(self, cdID, title, artist, lst\_Inventory):
35. """Constructor funcriton to create atributes for CD Objects
37. Conducts error trapping on user CD ID input, creates object attributes, and adds
38. CD information to list of CD objects
40. Args:
41. cdID (int): CD ID either inputted by user or read from file.
42. title (string): CD Title either inputted by user or read from file.
43. artist (string): CD Artists  either inputted by user or read from file.
44. lst\_Inventory (list of objects): 2D data structure that holds the data during runtime
46. Returns:
47. None
48. """
49. **if** **not** str(cdID).isnumeric():
50. **print** ("\nYou must enter the CD ID as an integer. Adding CD failed.")
51. input("Press 'Enter' to Continue")
52. **else**:
53. self.\_\_cd\_id = cdID
54. self.\_\_cd\_title = title
55. self.\_\_cd\_artist = artist
56. row = self
57. lst\_Inventory = lst\_Inventory.append(row)
59. @property
60. **def** cd\_id(self):
61. """Getter Function to access \_\_cd\_id attribute"""
62. **return** self.\_\_cd\_id
64. @cd\_id.setter
65. **def** cd\_id(self, value):
66. """Setter Function to assign \_\_cd\_id attribute and conduct error trapping"""
67. **if** **not** str(value).isnumeric():
68. **print**("ID must be an integer!")
69. **else**:
70. self.\_\_cd\_id = value
72. @property
73. **def** cd\_title(self):
74. """Getter Function to access \_\_cd\_title attribute"""
75. **return** self.\_\_cd\_title
77. @cd\_title.setter
78. **def** cd\_title(self, value):
79. """Setter Function to assign \_\_cd\_title attribute and conduct error trapping"""
80. **if** str(value).isnumeric():
81. **print**("Title can't just be numbers!")
82. **else**:
83. self.\_\_cd\_title = value
85. @property
86. **def** cd\_artist(self):
87. """Getter Function to access \_\_cd\_artist attribute"""
88. **return** self.\_\_cd\_artist
90. @cd\_artist.setter
91. **def** cd\_artist(self, value):
92. """Setter Function to assign \_\_cd\_artist attribute and conduct error trapping"""
93. **if** str(value).isnumeric():
94. **print**("Artist can't just be numbers!")
95. **else**:
96. self.\_\_cd\_artist = value
98. # -- PROCESSING -- #
99. **class** FileIO:
100. """Processes data to and from file:
102. properties:
103. None
105. methods:
106. load\_inventory(file\_name, lst\_Inventory): loads text file and creates list of CD Objects
107. save\_inventory(file\_name, lst\_Inventory): Saves list of CD Objects into text file.
109. """
110. # TODO Add code to process data from a file
111. @staticmethod
112. **def** load\_inventory(file\_name, lst\_Inventory):
113. """Function to manage data ingestion from file to a list of objects.
115. Reads the data from file identified by file\_name into a 2D table (list of objects)
116. Each line in the file is instantiated into a CD Object, then appended to list of CD objects thru
117. the \_init\_ constructor.
119. Args:
120. file\_name (string): name of file used to read the data from
121. lst\_Inventory (list of objects): 2D data structure that holds the data during runtime
123. Returns:
124. None.
125. """
126. lst\_Inventory.clear()  # this clears existing data and allows to load data from file
127. **try**:
128. objFile = open(file\_name, 'r')
129. **for** line **in** objFile:
130. data = line.strip().split(',')
131. objCD = CD(data[0], data[1], data[2], lst\_Inventory)
132. objFile.close()
133. **except** IOError:
134. **print**("\nNo CD file exists.  Add and save CD info to inventory to create a file.")
135. input("Press 'Enter' to continue.")
137. @staticmethod
138. **def** save\_inventory(file\_name, lst\_Inventory):
139. """Function to save the contents of CD Inventory in volatile memory into a text file.
141. Takes the list of CD Objects, formats it human readable text, and saves it into text file identified
142. by file\_name.

145. Args:
146. file\_name (string): name of file used to save data to.
147. lst\_Inventory (list of objects): 2D data structure that holds the data during runtime
149. Returns:
150. None
151. """
152. objFile = open(file\_name, 'w')
153. **for** row **in** lst\_Inventory:
154. cdID = row.cd\_id
155. cdID = str(cdID)
156. title = row.cd\_title
157. artist = row.cd\_artist
158. objFile.write("{},{},{}\n".format(cdID, title, artist))
159. objFile.close()
161. # -- PRESENTATION (Input/Output) -- #
162. **class** IO:
163. """Handling Input / Output
165. properties:
166. None
168. methods:
169. print\_menu(): Displays a menu of choices to the  user.
170. menu choice(): Gets user input for menu selection
171. show\_inventory(lst\_Inventory): Displays current inventory table
172. intpu\_new\_cd(): Function that collects user input for a new CD to be added to inventory
174. """
176. @staticmethod
177. **def** print\_menu():
178. """Displays a menu of choices to the user
180. Args:
181. None.
183. Returns:
184. None.
185. """
187. **print**('\nMenu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
188. **print**('[s] Save Inventory to file\n[x] exit\n')
190. @staticmethod
191. **def** menu\_choice():
192. """Gets user input for menu selection
194. Args:
195. None.
197. Returns:
198. choice (string): a lower case sting of the users input out of the choices l, a, i, s or x
200. """
201. choice = ' '
202. **while** choice **not** **in** ['l', 'a', 'i', 's', 'x']:
203. choice = input('Which operation would you like to perform? [l, a, i, s or x]: ').lower().strip()
204. **print**()  # Add extra space for layout
205. **return** choice
207. @staticmethod
208. **def** show\_inventory(lst\_Inventory):
209. """Displays current inventory table
211. Iterates through list of CD Objects, accesses object attributes thru dot notation, assigns attributes
212. to seperate variables, then prints in readable format.
214. Args:
215. lst\_Inventory (list of objects): 2D data structure that holds the data during runtime.
217. Returns:
218. None.
219. """
220. **print**('======= The Current Inventory: =======')
221. **print**('ID\tCD Title (by: Artist)\n')
222. **for** row **in** lst\_Inventory:
223. cdID = row.cd\_id
224. title = row.cd\_title
225. artist = row.cd\_artist
226. **print**('{}\t{} (by: {})'.format(cdID, title, artist))
227. **print**('======================================')
229. @staticmethod
230. **def** input\_new\_cd():
231. """Function that collects user input for a new CD to be added to inventory.
233. Args:
234. None.
236. Returns:
237. cdID (string): string representing ID number user entered for CD
238. title (string): string representing CD title entered by user
239. artist (string): string representing artist name entered by user
240. """
241. cdID = input('Enter ID: ').strip()
242. title = input('What is the CD\'s title? ').strip()
243. artist = input('What is the Artist\'s name? ').strip()
244. **return** cdID, title, artist
246. # -- Main Body of Script -- #
247. FileIO.load\_inventory(strFileName, lstOfCDObjects) # When program starts, calls function that reads in the currently saved Inventory
248. **while** True:
249. IO.print\_menu() #Display Menu to user and get choice
250. strChoice = IO.menu\_choice() # Process menu selection
251. **if** strChoice == 'x': #Process exit first
252. **break**
253. **if** strChoice == 'l': #Process load inventory
254. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
255. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
256. **if** strYesNo.lower() == 'yes':
257. **print**('reloading...')
258. FileIO.load\_inventory(strFileName, lstOfCDObjects) #Calls function that loads text file containing CD inventory into runtime.
259. IO.show\_inventory(lstOfCDObjects) #Calls function that displays inventory to user
260. **else**:
261. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
262. IO.show\_inventory(lstOfCDObjects) #Calls function that displays inventory to user
263. **continue**  # start loop back at top.
264. **elif** strChoice == 'a': #Process user input
265. strID, strTitle, strArtist = IO.input\_new\_cd() # Calls function that asks user for new ID, CD Title and Artist
266. objCD = CD(strID, strTitle, strArtist, lstOfCDObjects) # Instatiates CD Object using Class CD. Constructor also adds object to runtime list
267. IO.show\_inventory(lstOfCDObjects) #Calls function that displays inventory with added CD
268. **continue**  # start loop back at top.
269. **elif** strChoice == 'i':  # Process display current inventory
270. IO.show\_inventory(lstOfCDObjects) # Calls function that displays current inventory
271. **continue**  # start loop back at top.
272. **elif** strChoice == 's': #Process save inventory to file
273. IO.show\_inventory(lstOfCDObjects)    # Calls function that displays current inventory.
274. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower() #asks user for confirmation to save
275. **if** strYesNo == 'y': # Process choice
276. FileIO.save\_inventory(strFileName, lstOfCDObjects) # Calls function that saves data
277. **else**:
278. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
279. **continue**  # start loop back at top.
280. **else**:  # catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
281. **print**('General Error')

1. Page 219, Python Programming for Absolute Beginner, Third Edition [↑](#footnote-ref-1)
2. Page 222, Python Programming for Absolute Beginner, Third Edition [↑](#footnote-ref-2)
3. Page 226-227, Python Programming for Absolute Beginner, Third Edition [↑](#footnote-ref-3)
4. Page 232-233, Python Programming for Absolute Beginner, Third Edition [↑](#footnote-ref-4)
5. Page 238, Python Programming for Absolute Beginner, Third Edition [↑](#footnote-ref-5)
6. Page 239, Python Programming for Absolute Beginner, Third Edition [↑](#footnote-ref-6)
7. Page 239, Python Programming for Absolute Beginner, Third Edition [↑](#footnote-ref-7)
8. Page 239-240, Python Programming for Absolute Beginner, Third Edition [↑](#footnote-ref-8)
9. Accessed 29 March 2020. [↑](#footnote-ref-9)
10. Page 227-228, Python Programming for Absolute Beginner, Third Edition [↑](#footnote-ref-10)
11. Page 239, Python Programming for Absolute Beginner, Third Edition [↑](#footnote-ref-11)