Brandon Wayward

2020-Sep-01

IT FDN 110 B SU 20

Assignment 08

Objects and Classes: An introduction to OOP

Introduction

In Module 8 we were thrown headfirst into the depths of OOP, laughed at, and told to swim. This introduction is just a dabble into the beginnings of Object-Oriented Programming with regards to Objects and Classes.

# Classes

A Class is a blueprint that is used to create objects based on the classes definitions. The class defines attributes, methods, and properties that are inherited by the objects instantiated by the class.

The components that make up classes are fields, constructors, attributes, properties, and methods. We’ll take a closer look at each of them individually

A Class Constructor is a specific method that is automatically called upon the instantiation of an object. This starts the foundational building of the object

Fields are like class specific variables an object can call to. They exist outside of objects and are independent of said objects. Value changes to a field can happen at the class level, but not the object level.

Attributes are variables that exist on an object level as opposed to a class level. This means that attributes are directly linked to the object itself.

Properties help the programmer gatekeep access to attributes. This naturally builds out two types of property functions. The Setter, and The Getter. Kinda sounds like a mafia cleanup crew! Setter property functions manage the ability to set (or write to) attributes and Getter property functions manage the ability to get (or read from) attributes.

@staticmethod’ is a decorator that is used before the definition of a class-wide method. Methods can be either defined within an object where the method references to its own object (this is where the keyword “self” is being used) or on a the class level where the function is declared with the decorator “@staticmethod” and can be used by any object of the class.

# Steps Taken:

The steps taken to complete this assignment were as follows:

1. Add more pseudocode to organize my thoughts regarding what to build and where to place it
2. Investigate what current code from the prior assignment could be reutilized. Most of the IO class could be used again, the rest was a wash
3. Storing data about the CD was a little overwhelming. I have decorators added that I am uncertain about in addition to “\_\_”
4. Review code from the labs to reinforce the information, and create the Display Inventory function
5. Add the ability to load inventory and display inventory as they were the most similar. It was at this point I didn’t exactly notice a difference between private and public attributes… so I made the decision to make them all private.
6. Writing to the text file as outlined in the sample code starter was fairly cut and dry string manipulation
7. Then the menu / exit / user input were fairly self-explanatory.

# Summary

In summary, we learned about classes and what makes up the internal workings of a class. This was a fairly difficult assignment due to how I needed to rework through everything in pseudocode and by the nature of the assignment, there was a lot less code recycling done overall. Learning when and how to interact with objects, creating private vs public references, and overall interaction with objects is something I only vaguely grasp. With enough exposure and practice this will be mastered.

# Appendix

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

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. # BWayward, 2020-Aug-27, cannibalized code from assignment 7 for meny, load, add, and write skeletons
8. # BWayward, 2020-Aug-27, attempted to define base functions for user input and CD data manipulation
9. # BWayward, 2020-uug-28, revisited functions, corrected, tested functionality
10. # BWayward, 2020 Aug-29, Updated docstrings (?), commented code, validated processes again
11. # BWayward, 2020-Aug-30, doublechecked.
12. #------------------------------------------#
14. # -- DATA -- #
15. strFileName = 'CDInventory.txt'
16. lstOfCDObjects = []
18. **class** CDData():
19. """Stores data about a CD:
21. properties:
22. position: (int) with CD ID
23. album: (string) with the title of the CD
24. artist: (string) with the artist of the CD
25. methods:
26. append\_cd\_inventory\_memory\_list(objCD, table):  None
27. """
28. **def** \_\_init\_\_(self):
29. self.\_\_intPosition = 1
30. self.\_\_strAlbum = ""
31. self.\_\_strArtist = ""
33. @property
34. **def** position(self):
35. **return** self.\_\_intPosition
36. @position.setter
37. **def** position (self, ind):
38. self.\_\_intPosition = ind
40. @property
41. **def** album (self):
42. **return** self.\_\_strAlbum
43. @album.setter
44. **def** album (self, alb):
45. self.\_\_strAlbum = alb
47. @property
48. **def** artist (self):
49. **return** self.\_\_strArtist
50. @artist.setter
51. **def** artist (self, ar):
52. self.\_\_strArtist = ar
54. **def** \_\_str\_\_(self):
55. **return** str(self.\_\_intPosition) + "\t" + str(self.\_\_strAlbum) + "\tby: " + str(self.\_\_strArtist)
57. @staticmethod
58. **def** append\_cd\_inventory\_memory\_list(objCD, table):
59. """Function to append a newly created CD object (user input or file reading) to the global list of CD objects
61. Uses passed arguements of CD information to create a new entry as a dictionary, then add dictionary to global table
63. Args:
64. objCD (object): CD Object that needs to be added to the list
65. table (list of objects): 2D data structure (list of objects) that holds the data during runtime
67. Returns:
68. None.
69. """
70. table.append(objCD)
71. # -- PROCESSING -- #
73. **class** FileIO:
74. """Processes data to and from file:
76. properties:
78. methods:
79. write\_file(file\_name, table):  None
80. load\_inventory(file\_name):  cdObjLst (a list of CD objects)
82. """
84. @staticmethod
85. **def** write\_file(file\_name, table):
86. """Function to write added data to file
88. appends data fromlist to table.
90. Args:
91. file\_name (string): name of file used to read the data to
92. table (list of objects): 2D data structure (list of objects) that holds data during runtime
94. Returns:
95. None.
96. """
97. objFile = open(file\_name, 'w')
98. **for** obj **in** table:
99. cd\_string = str(obj.position)+ ',' + obj.album + ',' + obj.artist
100. objFile.write(cd\_string + '\n')
101. objFile.close()
103. @staticmethod
104. **def** load\_inventory(file\_name):
106. """Function to manage data ingestion from file to a list of dictionaries
108. Reads the data from file identified by file\_name into a global 2D table
109. (list of dicts) table one line in the file represents one dictionary row in table.
111. Args:
112. file\_name (string): name of file used to read the data from
114. Returns:
115. cdObjLst (list): list of objects
116. """
117. cdObjLst = []
118. **try**:
119. with open(file\_name, 'r') as objFile:
120. **for** line **in** objFile:
121. cdObjName = CDData()
122. data = line.strip().split(',')
123. cdObjName.position = data[0]
124. cdObjName.album = data[1]
125. cdObjName.artist = data[2]
126. cdObjLst.append(cdObjName)
127. **except** IOError:
128. **print**('\nThere is currently no existing inventory file\n')
129. **return** cdObjLst
131. # -- PRESENTATION (Input/Output) -- #
133. **class** IO:
134. """Presents menu, data and requests data from the user:
136. properties:
138. methods:
139. cd\_user\_input(): data to CD (list)
140. print\_menu():  None
141. menu\_choice(): takes user input choice(string)
142. show\_inventory(table):  None
143. """
145. @staticmethod
146. **def** cd\_user\_input():
147. """Gets user input to add new CD
149. Catches ValueError if an integer is not entered.
150. Continues to ask for an integer until one is entered
152. Args:
153. None.
155. Returns:
156. CD (list): Returns list with the 3 values outlined ID, Title and Artist
158. """
159. **while** True:
160. **try**:
161. strID = input('Enter a numerical ID: ').strip()
162. intID = int(strID)
163. **break**
164. **except** ValueError:
165. **print** ('The entered ID is not an integer. Please enter a number')
166. strTitle = input('What is the CD\'s title? ').strip()
167. strArtist = input('What is the Artist\'s name? ').strip()
168. CD = [intID, strTitle, strArtist]
169. **return** CD
171. @staticmethod
172. **def** print\_menu():
173. """Displays a menu of choices to the user
175. Args:
176. None.
178. Returns:
179. None.
180. """
182. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
183. **print**('[s] Save Inventory to file\n[x] exit\n')

186. @staticmethod
187. **def** menu\_choice():
188. """Gets user input for menu selection
190. Args:
191. None.
193. Returns:
194. choice (string): a lower case sting of the users input out of the choices l, a, i, s or x
196. """
197. choice = ' '
198. **while** choice **not** **in** ['l', 'a', 'i', 's', 'x']:
199. choice = input('Which operation would you like to perform? [l, a, i, ,s or x]: ').lower().strip()
200. **print**()
201. **return** choice
203. @staticmethod
204. **def** show\_inventory(table):
205. """Displays current inventory table

208. Args:
209. table (list of CD objects): 2D data structure (list of CD objects) that holds the data
211. Returns:
212. None.
214. """
215. **print**('======= The Current Inventory: =======')
216. **print**('ID\tCD Title (by: Artist)\n')
217. **for** obj **in** table:
218. **print**(obj)
219. **print**('======================================')
221. # -- Main Body of Script -- #
223. # Load data from file
224. FileIO.load\_inventory(strFileName)

227. **while** True:
228. # Display menu to user
229. IO.print\_menu()
230. strChoice = IO.menu\_choice()
231. # let user exit program
232. **if** strChoice == 'x':
233. **break**
234. # let user load inventory from file
235. **if** strChoice == 'l':
236. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
237. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
238. **if** strYesNo.lower() == 'yes':
239. **print**('reloading...')
240. lst = FileIO.load\_inventory(strFileName)
241. lstOfCDObjects = lst
242. IO.show\_inventory (lstOfCDObjects)
243. **else**:
244. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
245. IO.show\_inventory (lstOfCDObjects)
246. **continue**  # start loop back at top.
247. # let user add data to the inventory
248. **elif** strChoice == 'a':
249. UserInputLst = IO.cd\_user\_input() #requests the user input of the new CD object and temp stores it into a list object
250. cdObjName = CDData() # initiate an empty CD object
251. #Writing the user input into the initialized CD object
252. cdObjName.position = UserInputLst[0]
253. cdObjName.album = UserInputLst[1]
254. cdObjName.artist = UserInputLst[2]
255. #Appending the newly created CD to the CD object list
256. CDData.append\_cd\_inventory\_memory\_list(cdObjName, lstOfCDObjects)
257. IO.show\_inventory (lstOfCDObjects)
258. # show user current inventory
259. **elif** strChoice == 'i':
260. IO.show\_inventory (lstOfCDObjects)
261. **continue**
262. # let user save inventory to file
263. **elif** strChoice == 's':
264. # Display current inventory and ask user for confirmation to save
265. IO.show\_inventory(lstOfCDObjects)
266. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
267. # Process choice
268. **if** strYesNo == 'y':
269. # save data
270. FileIO.write\_file(strFileName, lstOfCDObjects)
271. **else**:
272. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
273. **continue**  # start loop back at top.
275. # catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
276. **else**:
277. **print**('General Error')

SPYDER

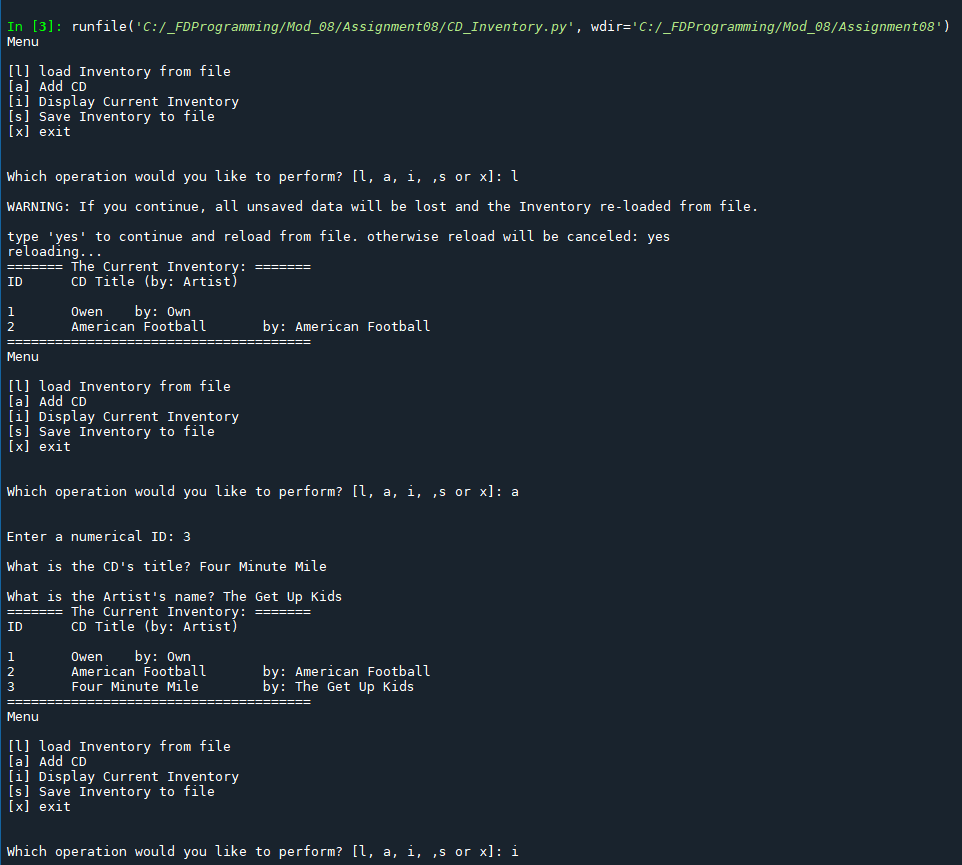


Figure - Script in Spyder

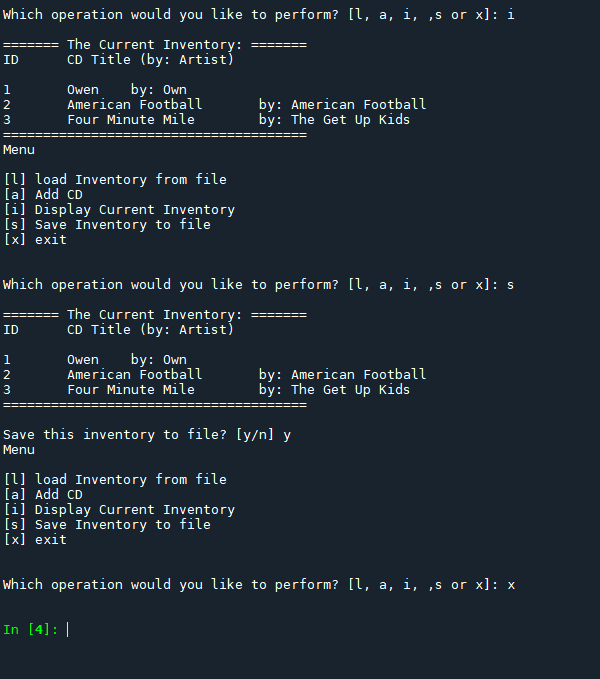


Figure - Script in spyder

TERMINAL

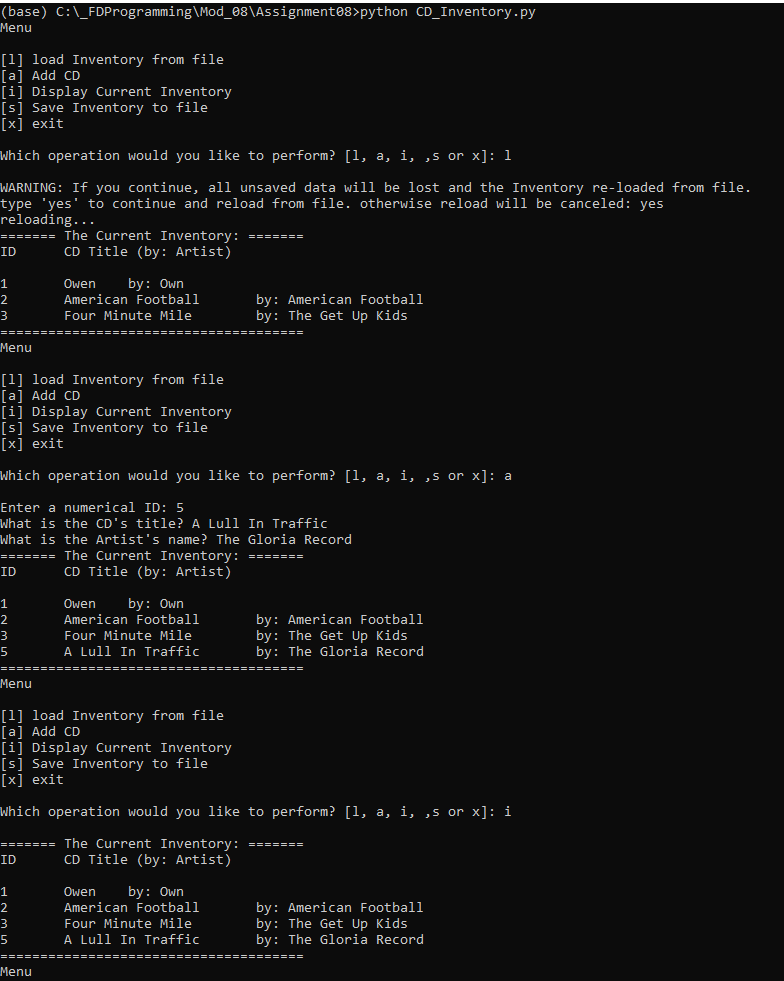


Figure - Script in Terminal 1

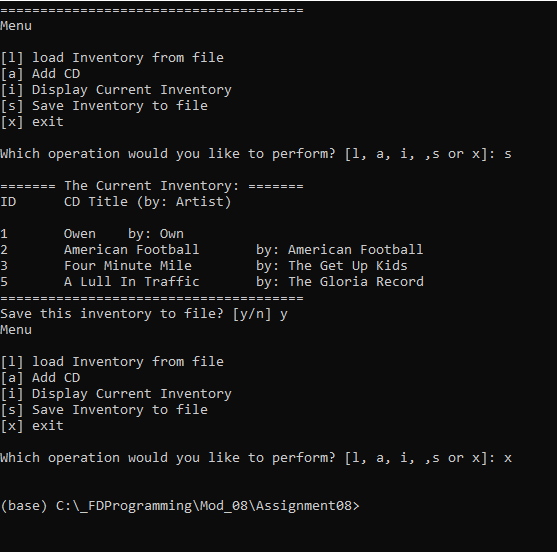


Figure - Script in Terminal 2