Evan Malina

12/5/2021

Foundations of Programming Python

Assignment 8

# Introduction

For this assignment, I took code from Assignment 6 to fill in most of the functionality of the script. I used properties and methods in a new ‘cd’ class. The code went back to assignment 6 and wrote to .txt rather than pickling. I included exemptions in the CD class, but I’m not sure that they are actually doing anything, or rather just hiding in the background. I did use the class to populate a dictionary in the main body of the script.

# Code

## Github

Github link to my code and knowledge document here;

## <https://github.com/ewmalina/Assignment_08>

## Class

The primary new concepts in this script were related to the CD class. I added a dunder \_\_init\_\_ constructor which assigned new private variable names to the CD ID, Artist and Title.

### Properties

For each variable; ID, Title, Artist, I created a getter and setter property. I believe, as written, all that these would possibly be doing is checking that the variables were the correct type. I have checked the functionality in running the script and the ID will accept a number so it doesn’t seem to be working. I know this was discussed during the office hours, and it seems to be that the properties are not really being called so nothing actually happens.

### Docstring

I also learned about class docstrings. I added the new property and methods too the docstring. This is a nice summary compared to how my previous codes were.

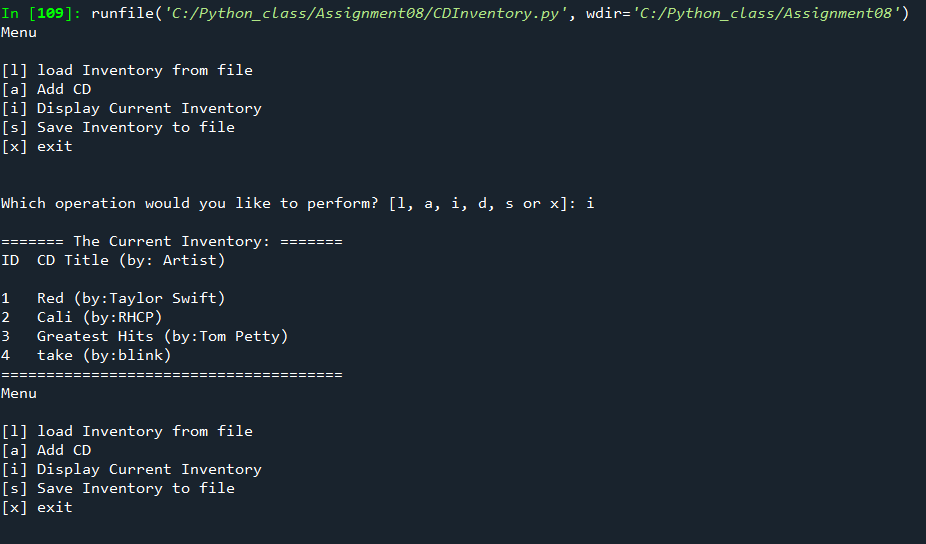


Figure 1 - Integer D add ValueError

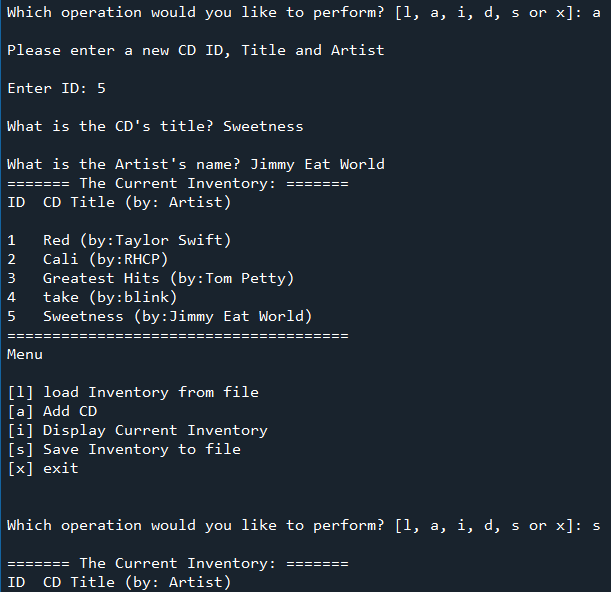


Figure 2 - Spyder Script 1

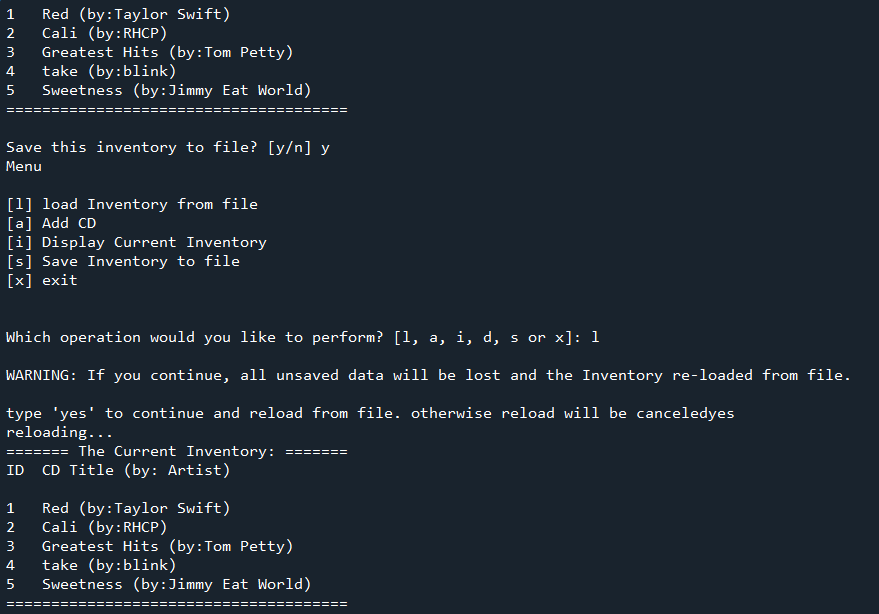


Figure 3 - Spyder Script 2

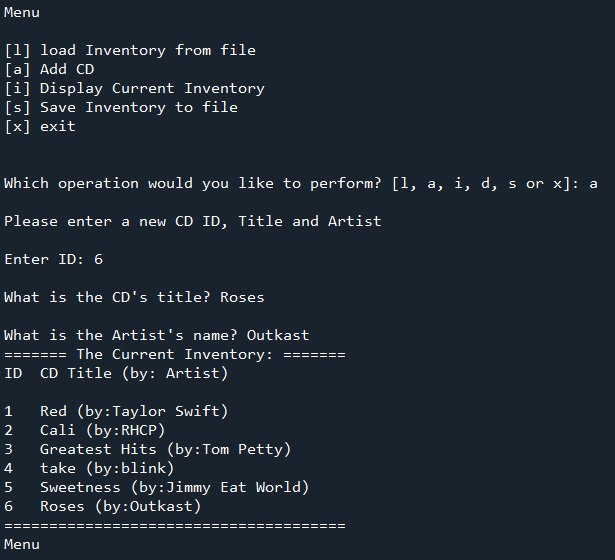


Figure 4 - Spyder script run part 3

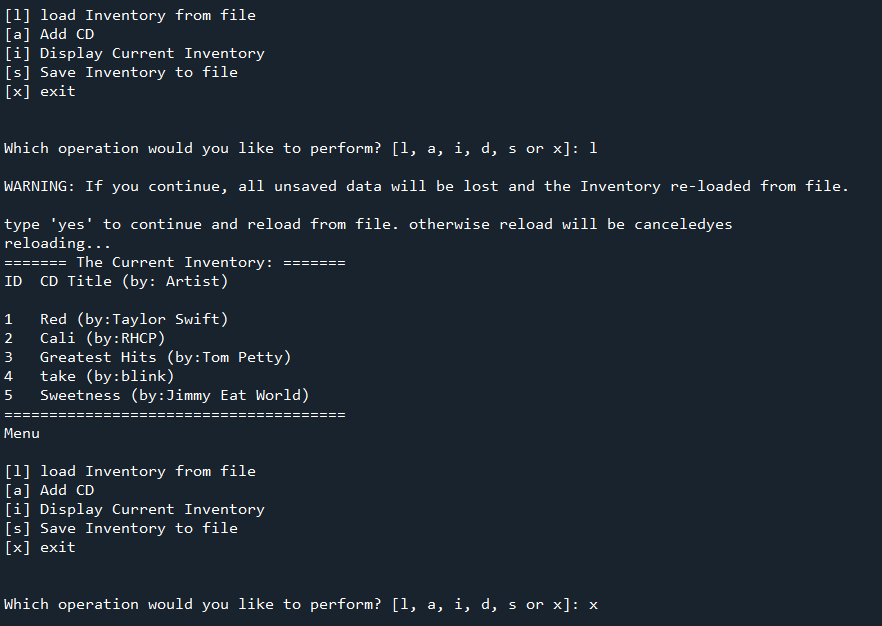


Figure 5 - Spyder script run part 4

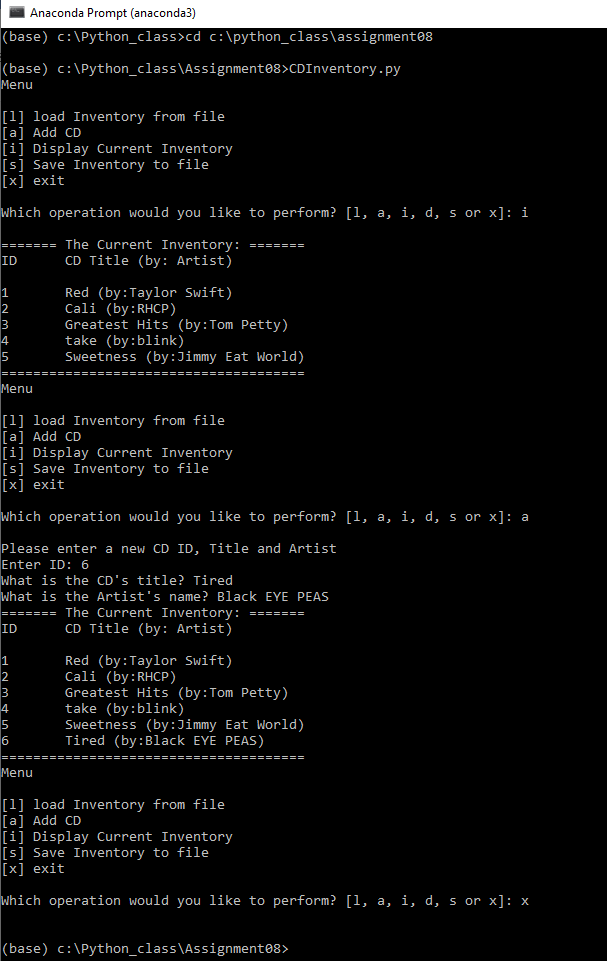


Figure 6 - Spyder script run part 5

1. *#------------------------------------------#*
2. *# Title: CDInventory.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. *# EMalina, 2021-Dec-05 added code to complete assignment 8*
8. *#------------------------------------------#*
10. ***# -- DATA -- #***
11. strFileName = 'cdInventory.txt'
12. lstOfCDObjects = []
14. **class** CD():
15. **"""Stores data about a CD:**
17. properties:
18. cd\_id: (int) with CD ID
19. cd\_title: (string) with the title of the CD
20. **cd\_artist: (string) with the artist of the CD**
21. methods: none
22. """
23. *# TODOne Add Code to the CD class*
24. *# -- Constructor -- #*
26. **def** \_\_init\_\_(self, idid, title, artist):
27. *# -- Attributes -- #*
28. self.\_\_cd\_id = idid
29. self.\_\_cd\_title = title
30. **self.\_\_cd\_artist = artist**
32. *# -- Properties -- #*
33. @property
34. **def** cd\_id(self):
35. **return self.\_\_cd\_id**
37. @cd\_id.setter
38. **def** cd\_id(self, idid):
39. **if** str(idid).isnumeric():
40. **self.\_\_cd\_id = idid**
41. **else**:
42. **raise** Exception('Non-numeric ID')
44. @property
45. **def cd\_title(self):**
46. **return** self.\_\_cd\_title
48. @cd\_title.setter
49. **def** cd\_title(self, title):
50. **if str(title).isnumeric():**
51. **raise** Exception('Title cannot be number')
52. **else**:
53. self.\_\_cd\_title = title
55. **@property**
56. **def** cd\_artist(self):
57. **return** self.\_\_cd\_artist
59. @cd\_artist.setter
60. **def cd\_artist(self, artist):**
61. **if** str(artist).isnumeric():
62. **raise** Exception('artist cannot be number')
63. **else**:
64. self.\_\_cd\_artist = artist

67. *# -- PROCESSING -- #*
68. **class** FileIO:
69. """Processes data to and from file:
71. properties: none
73. methods:
74. save\_inventory(file\_name, lst\_Inventory): -> None
75. **load\_inventory(file\_name): -> (a list of CD objects)**
77. """
78. *# TODOne Add code to process data from a file*
79. @staticmethod
80. **def load\_inventory(file\_name, table):**
81. """Function to manage data ingestion from file to a list of dictionaries
83. Reads the data from file identified by file\_name into a 2D table
84. (list of dicts) table one line in the file represents one dictionary row in table.
86. Args:
87. file\_name (string): name of file used to read the data from
88. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
90. **Returns:**
91. None.
92. """
93. table.clear() *# this clears existing data and allows to load data from file*
94. **try**:
95. **objFile = open(file\_name, 'r')**
96. **for** line **in** objFile:
97. data = line.strip().split(',')
98. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
99. table.append(dicRow)
100. **objFile.close()**
101. **return** table
102. **except** FileNotFoundError **as** e:
103. **print**('File not found')
104. **print**('Built in error info: ')
105. **print(type(e), e, e.\_\_doc\_\_, sep='\n')**
106. table = []
107. **return** table
109. *# TODOne Add code to process data to a file*
110. **@staticmethod**
111. **def** save\_inventory(file\_name, table):
112. """Function to save the CD to file
113. Args:
114. file\_name (str): file name for saving
115. **table (list): current inventory of CDs. A list of dictionaries**
117. Returns: none
118. """
119. **try**:
120. **objFile = open(file\_name, 'w')**
121. **for** row **in** table:
122. lstValues = list(row.values())
123. lstValues[0] = str(lstValues[0])
124. objFile.write(','.join(lstValues) + '**\n**')
125. **objFile.close()**
126. **except** FileNotFoundError **as** e:
127. **print**('File not found')
128. **print**('Built in error info: ')
129. **print**(type(e), e, e.\_\_doc\_\_, sep='**\n**')
131. *# -- PRESENTATION (Input/Output) -- #*
132. **class** IO:
133. *# TODOne add docstring*
134. """Processes data to and from file:
136. properties: none
138. methods:
139. menu(): -> None (prints menu)
140. **selection(): -> (menu choice)**
141. inventory(table): -> None (prints inventory)
142. add\_data(): -> var1, var2, var3 (cd\_ID, cd\_Title, cd\_Artist)
144. """
145. ***# TODOne add code to show menu to user***
146. **def** menu():
147. """Displays a menu of choices to the user
149. Args:
150. **None.**
152. Returns:
153. None.
154. """
155. **print('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')**
156. **print**('[d] delete CD from Inventory**\n**[s] Save Inventory to file**\n**[x] exit**\n**')
158. *# TODOne add code to captures user's choice*
159. **def** selection():
160. **"""Gets user input for menu selection**
162. Args:
163. None.
165. **Returns:**
166. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
168. """
169. choice = ' '
170. **while choice not in ['l', 'a', 'i', 'd', 's', 'x']:**
171. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
172. **print**() *# Add extra space for layout*
173. **return** choice
175. ***# TODOne add code to display the current data on screen***
176. **def** inventory(table):
177. """Displays current inventory table
179. Args:
180. **table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.**
182. Returns:
183. None.
185. **"""**
187. **print**('======= The Current Inventory: =======')
188. **print**('ID**\t**CD Title (by: Artist)**\n**')
189. **for** row **in** table:
190. **print('{}\t{} (by:{})'.format(\*row.values()))**
191. **print**('======================================')
193. *# TODOne add code to get CD data from user*
194. **def** add\_data():
195. **"""requests user to input new CD information**
197. Args: none
199. Returns: tuple of 3;
200. **var1 (int): ID number**
201. var2 (str): CD title
202. var3 (str): CD artist
203. """
204. **print**('Please enter a new CD ID, Title and Artist')
205. **var1 = input('Enter ID: ').strip()**
206. var2 = input('What is the CD**\'**s title? ').strip()
207. var3 = input('What is the Artist**\'**s name? ').strip()
208. **return**(var1, var2, var3)
209. **pass**

212. *# -- Main Body of Script -- #*
213. *# TODOne Add Code to the main body*
214. *# Load data from file into a list of CD objects on script start*
215. **FileIO.load\_inventory(strFileName, lstOfCDObjects)**
216. **while** True:
217. *# Display menu to user*
218. IO.menu()
219. strChoice = IO.selection()
220. ***# let user exit program***
221. **if** strChoice == 'x':
222. **break**
223. *# show user current inventory*
224. **if** strChoice == 'i':
225. **IO.inventory(lstOfCDObjects)**
226. **continue** *# start loop back at top.*
227. *# let user add data to the inventory*
228. **elif** strChoice == 'a':
229. strID, strTitle, strArtist = IO.add\_data()
230. **newCD = CD(strID, strTitle, strArtist, lstOfCDObjects)**
231. dicRow = {'ID': '{}'.format(newCD.cd\_id), 'Title': '{}'.format(newCD.cd\_title), 'Artist': '{}'.format(newCD.cd\_artist)}
232. lstOfCDObjects.append(dicRow)
233. IO.inventory(lstOfCDObjects)
234. **continue** *# start loop back at top.*
235. ***# let user save inventory to file***
236. **elif** strChoice == 's':
237. IO.inventory(lstOfCDObjects)
238. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
239. **if** strYesNo == 'y':
240. **FileIO.save\_inventory(strFileName, lstOfCDObjects)**
241. **else**:
242. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
243. **continue** *# start loop back at top.*
244. *# let user load inventory from file*
245. **if strChoice == 'l':**
246. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
247. strYesNo = input('type **\'**yes**\'** to continue and reload from file. otherwise reload will be canceled')
248. **if** strYesNo.lower() == 'yes':
249. **print**('reloading...')
250. **FileIO.load\_inventory(strFileName, lstOfCDObjects)**
251. IO.inventory(lstOfCDObjects)
252. **else**:
253. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
254. IO.inventory(lstOfCDObjects)
255. **continue *# start loop back at top.***
256. **else**:
257. **print**('General Error')

Listing 1 - CDInventory.py script

# Summary

I really struggled with some of these concepts. I’m confused about what is going on in the background. The idea of calling a method or property via the class (i.e. xyz = class.method() makes sense to me. I also am very clear about how a method could call on another method (for example that *tricky* Kung Fu lesson). I’m more confused about the \_\_str\_\_ concept, and also in the getter/setter what is actually being passed back and forth. Like, in Listing 8 it states “return self.\_\_message.title(). I understand that comment, but in the setter property, what is value? (def message(self, value):…). I am also confused a bit about what is actually being passed back and forth with self. I definitely have more learning to do on this topic.