Kenny Lu

2020-Mar-16

IT FDN 100 B Winter 2020

Assignment08

GitHub <https://github.com/lukenny/Assignment_08>

# Introduction

In this eighth module, we cover object-oriented programming. We’re introduced to Classes, Methods and Constructors. It’s by far the hardest topic because we are leaving scripting and entering the complexity and powerfulness of programming. There are so many concepts and technical terms I’m definitely feeling a bit overwhelmed. I honestly think just getter and setter can be a module by itself.

Topic 1 – Dunder is short for “double-underscore”

One of the most common things we see in the python code base is \_\_init\_\_. It’s called the “dunder init”, it’s short for “double-underscore”. It’s best to illustrate with an example below. In this example, we are making a new object by calling the class directly, almost as if it was a function. When the object is created, it’s initialized by \_\_init\_\_ method on the object. The new object is invoked as self and the arguments to the class are passed as arguments after self. In this case, name is “Paul Smith”. And when the method is called, the object greeter is passed as self and other argument used is also being passed in after self.

1. In [6]: **class** Greeter(object):
2. ...:     """Object-oriented greetings"""
3. ...:
4. ...:     **def** \_\_init\_\_(self, name):
5. ...:         self.name = name
6. ...:
7. ...:     **def** greet(self, salutation="Hello"):
8. ...:         **print**("{}, {}".format(salutation, self.name))
9. ...:
11. In [7]: greeter = Greeter("Paul Smith")
13. In [8]: greeter.greet()
14. Hello, Paul Smith
16. In [9]: greeter.greet("Bonjour")
17. Bonjour, Paul Smith

Listing 1 – sample output to demonstrate dunder

Using [pyhurry](https://pyhurry.readthedocs.io/en/latest/classes.html) (external reference) webpage

# Topic 2 – Attributes, Getters and Setters

In order to explain getters and setters, we have to introduce the concept of data encapsulation which is to hide the attributes of the object class from other classes so accidental alteration of the data doesn’t happen inadvertently. Essentially, they are methods getters get value and setters set value of private attributes. Let’s use the snippet below to demonstrate this concept. Below is a class where it keeps track of favorite subjects. As of now, if you run Listing 2 it doesn’t do anything because these objects don’t have property set which is “FavoriteSubject”. We can set it directly but that’s definitely not the best approach. With Listing 3, Five objects are created, all of them have unique values for the property FavoriteSubject:

1. **class** Kid:
2. **def** \_\_init\_\_(self):
3. self.FavoriteSubject = "None"
5. Agnes = Kid()
6. Mia = Kid()
7. Sophia = Kid()
8. Mika = Kid()
9. Miro = Kid ()

Listing 2 – sample output to demonstrate getters and setters part one

1. **class** Kid:
2. **def** \_\_init\_\_(self):
3. self.FavoriteSubject = "None"
5. **def** getFavoriteSubject(self):
6. **return** self.FavoriteSubject
8. **def** setFavoriteSubject(self, FavoriteSubject):
9. self.FavoriteSubject = FavoriteSubject
11. Agnes = Kid()
12. Mia = Kid()
13. Sophia = Kid()
14. Mika = Kid()
15. Miro = Kid()
17. Agnes.setFavoriteSubject("Math")
18. Mia.setFavoriteSubject("Computer Science")
19. Sophia.setFavoriteSubject("Science")
20. Mika.setFavoriteSubject("English")
21. Miro.setFavoriteSubject("French")
23. **print**(Agnes.FavoriteSubject)
24. **print**(Mia.FavoriteSubject)
25. **print**(Sophia.FavoriteSubject)
26. **print**(Mika.FavoriteSubject)
27. **print**(Miro.FavoriteSubject)

Listing 3 – sample output to demonstrate getters and setters part two

1. python3 example.py
2. Math
3. Computer Science
4. Science
5. English
6. French

Listing 4 –sample output to demonstrate getters and setters part three

Using [tutorialpoint](https://www.tutorialspoint.com/getter-and-setter-in-python) (external reference) webpage

Using [pythonbasic](https://pythonbasics.org/getter-and-setter/) (external reference) webpage

# Summary

All in all, this assignment was more challenging than the rest for me because firstly the TODO deliverables were pseudocode and it could be interpreted in so many different ways. But I think the structure of this assignment resembled the closest of what code should look like in real world where classes and methods were widely adopted. For me the hardest part was to let go of what I learned with how we did the previous assignments and refactored it with concepts and techniques that I was not all that familiar with.

# Appendix

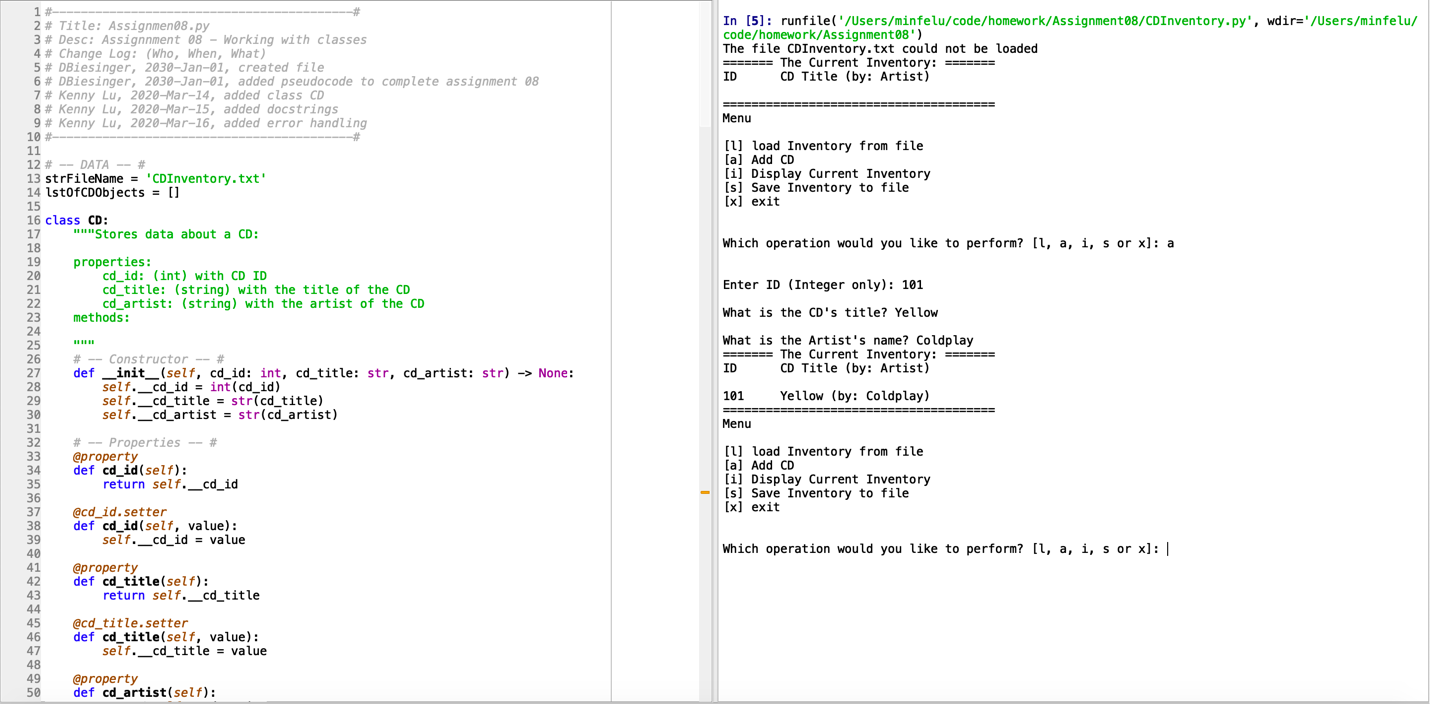


Figure 1 – Example output from script initial loaded and add data in Spyder

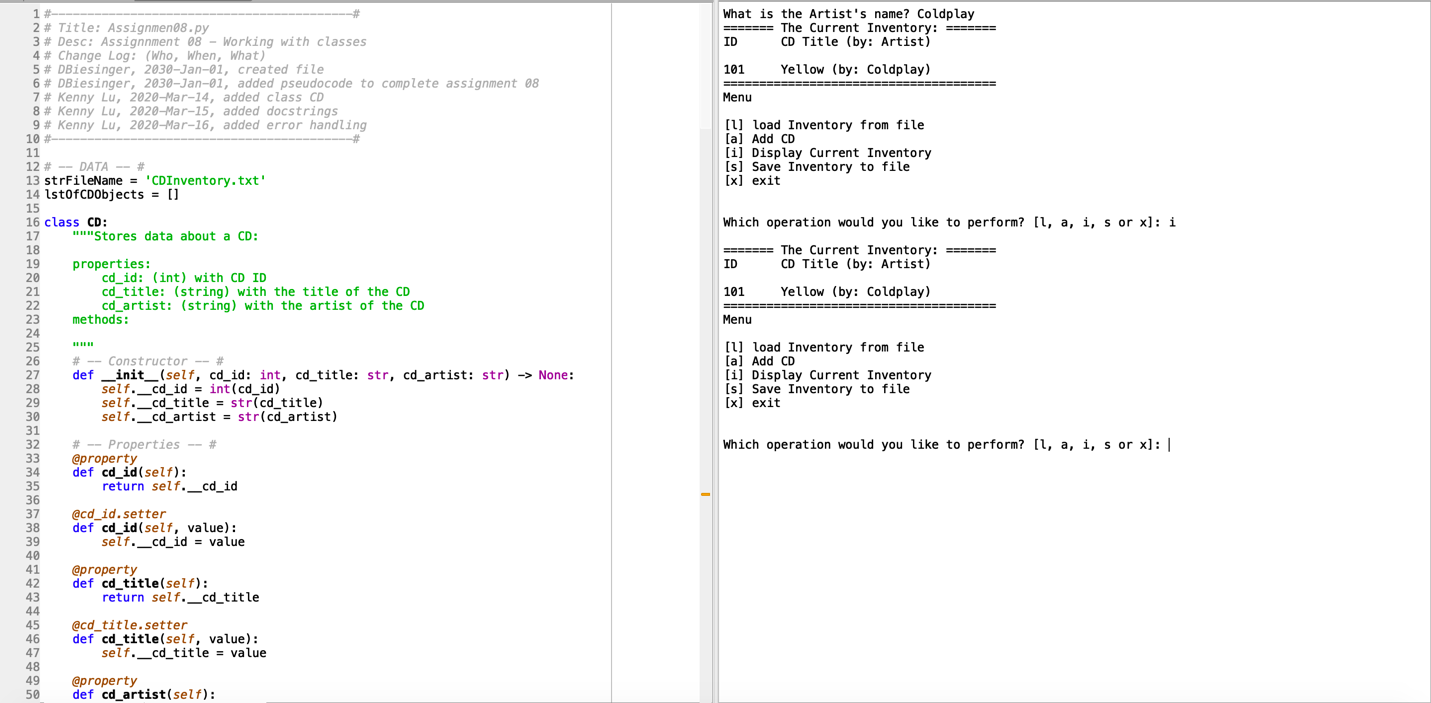


Figure 2 – Example output from script display inventory

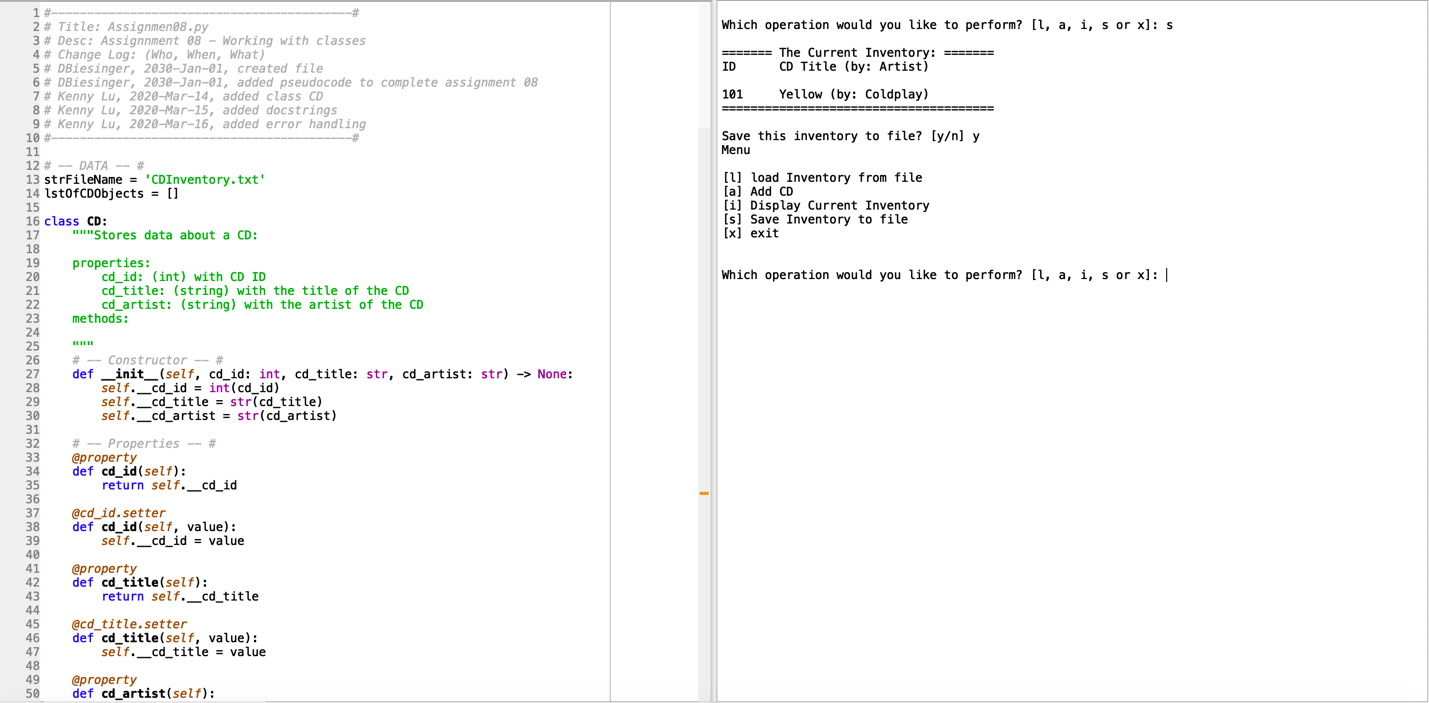


Figure 3 – Example output from script save inventory

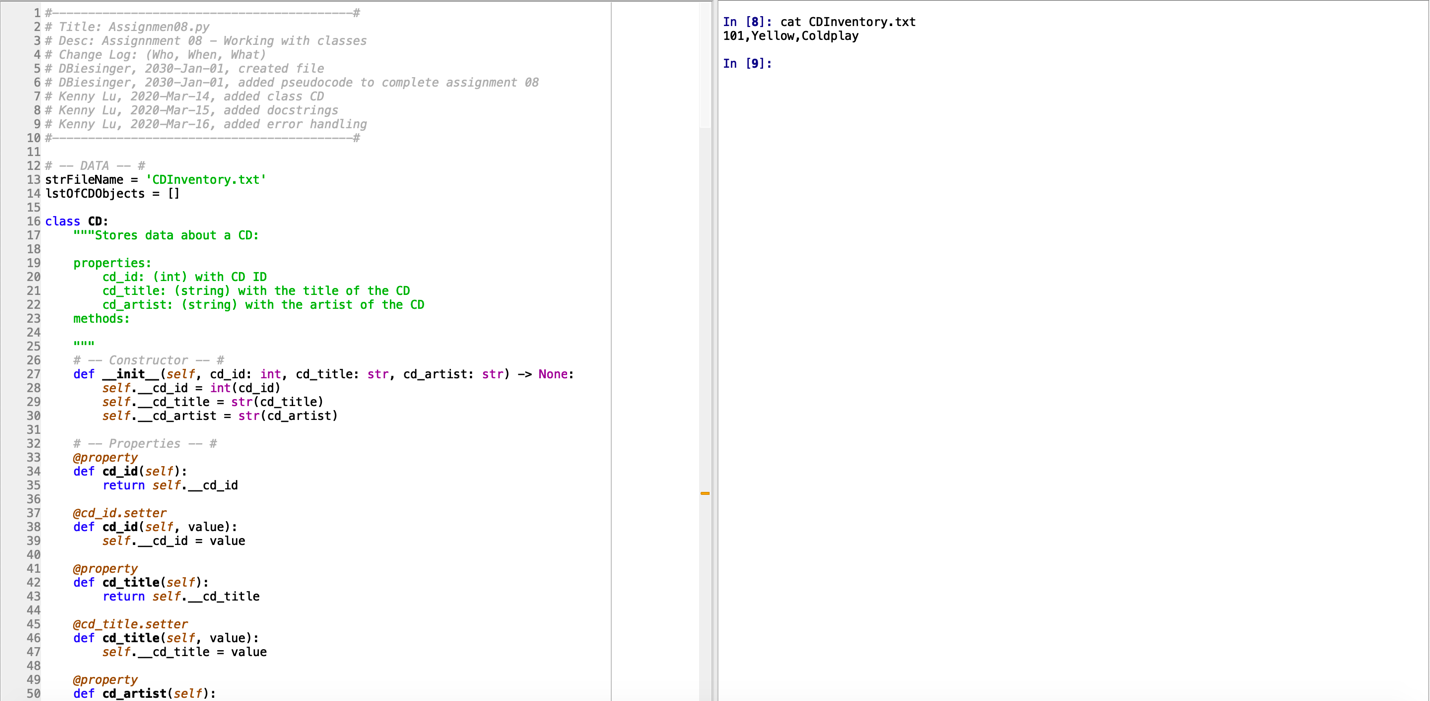
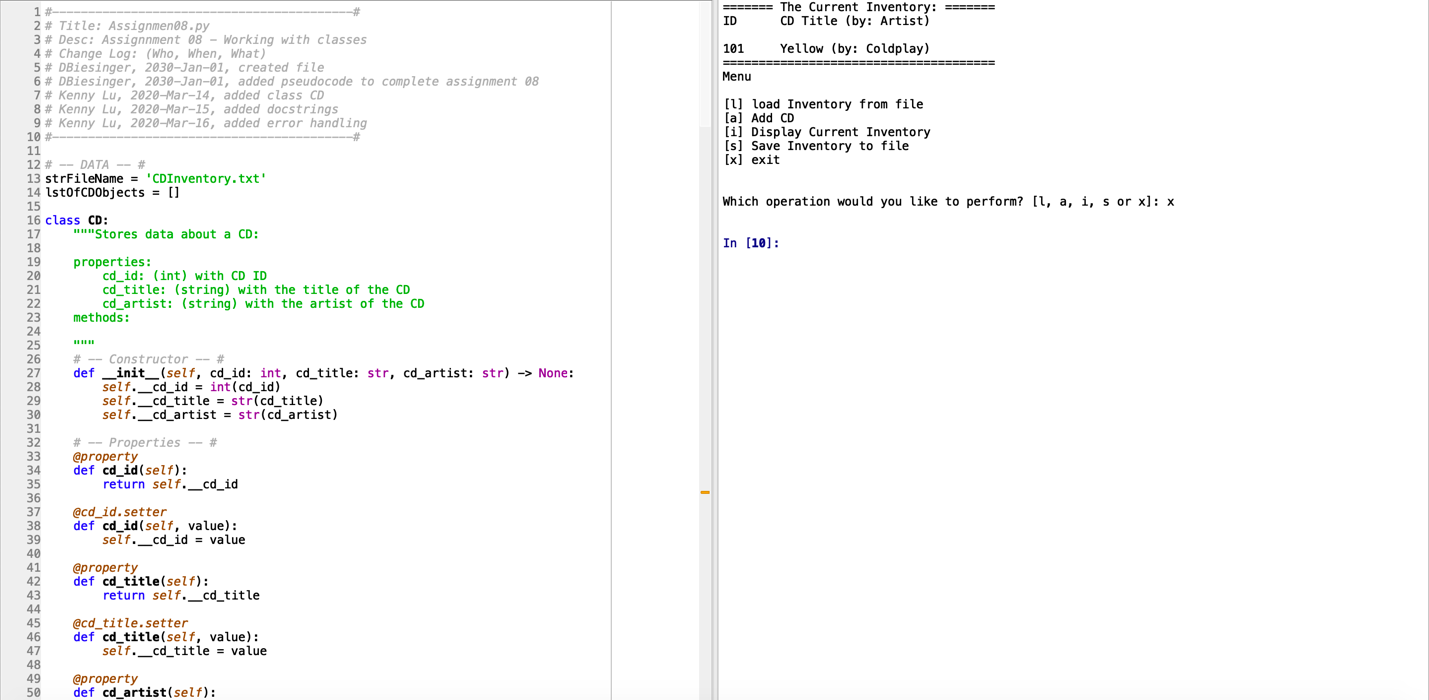


Figure 4 – Verify CDInventory.txt has saved inventory

  
Figure 5 – Example output from script exit program

1. #------------------------------------------#
2. # Title: Assignmen08.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. # Kenny Lu, 2020-Mar-13, added initial commit class CD
8. # Kenny Lu, 2020-Mar-14, added docstrings
9. # Kenny Lu, 2020-Mar-15, added error handling
10. # Kenny Lu, 2020-Mar-16, updated comments
11. #------------------------------------------#
13. # -- DATA -- #
14. strFileName = 'CDInventory.txt'
15. lstOfCDObjects = []
17. **class** CD:
18. """Stores data about a CD:
20. properties:
21. cd\_id: (int) with CD ID
22. cd\_title: (string) with the title of the CD
23. cd\_artist: (string) with the artist of the CD
24. methods:
26. """
27. # -- Constructor -- #
28. **def** \_\_init\_\_(self, cd\_id: int, cd\_title: str, cd\_artist: str) -> None:
29. self.\_\_cd\_id = int(cd\_id)
30. self.\_\_cd\_title = str(cd\_title)
31. self.\_\_cd\_artist = str(cd\_artist)
33. # -- Properties -- #
34. @property
35. **def** cd\_id(self):
36. **return** self.\_\_cd\_id
38. @cd\_id.setter
39. **def** cd\_id(self, value):
40. self.\_\_cd\_id = value
42. @property
43. **def** cd\_title(self):
44. **return** self.\_\_cd\_title
46. @cd\_title.setter
47. **def** cd\_title(self, value):
48. self.\_\_cd\_title = value
50. @property
51. **def** cd\_artist(self):
52. **return** self.\_\_cd\_artist
54. @cd\_artist.setter
55. **def** cd\_artist(self, value):
56. self.\_\_cd\_artist = value
58. # -- Methods -- #
59. **def** \_\_str\_\_(self):
60. # Cd data as formatted string
61. **return** '{}\t{} (by: {})'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
63. **def** get\_textdata(self):
64. # Cd data as text data csv format
65. **return** '{},{},{}\n'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
67. # -- PROCESSING -- #
68. **class** DataProcessor:
69. """Handling data processing adding data"""
71. @staticmethod
72. **def** add\_item(CdData, table):
73. """function to add CD data to the inventory table.
75. Args:
76. CdData (tuple): (ID, Title, Artist) to be added to inventory.
77. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
79. Returns:
80. None.
82. """
83. Id, Title, Artist = CdData
84. row = CD(Id, Title, Artist)
85. table.append(row)
87. **class** FileIO:
88. """Processes data to and from file:
90. properties:
92. methods:
93. save\_inventory(file\_name, lst\_Inventory): -> None
94. load\_inventory(file\_name): -> (a list of CD objects)
96. """
97. @staticmethod
98. **def** load\_inventory(file\_name: strFileName) -> list:
99. """Function to manage data ingestion from file
101. Load the data from text file identified by file\_name
103. Args:
104. file\_name (string): name of file used to read the data from
105. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
107. Returns:
108. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
109. """
110. lst\_Inventory = []
111. **try**:
112. lst\_Inventory.clear()
113. with open(strFileName, 'r') as file:
114. **for** line **in** file:
115. data = line.strip().split(',')
116. row = CD(data[0], data[1], data[2])
117. lst\_Inventory.append(row)
118. **except** FileNotFoundError:
119. **print**("The file {} could not be loaded".format(file\_name))
121. **return** lst\_Inventory
123. @staticmethod
124. **def** save\_inventory(file\_name: strFileName, lst\_Inventory: list) -> None:
125. """Function to write data in memory to file
127. Load the data from memory and write to a text based human readable storage file
129. Args:
130. file\_name (string): name of file used to read the data from
131. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
133. Returns:
134. None.
135. """
136. **try**:
137. with open(file\_name, 'w') as file:
138. **for** i **in** lst\_Inventory:
139. file.write(i.get\_textdata())
140. **except** IOError:
141. **print**("ERROR: The file {} could not be written or saved. Returning to the menu.".format(file\_name) + '\n')
143. # -- PRESENTATION (Input/Output) -- #
144. **class** IO:
145. """Handling Input / Output"""
147. @staticmethod
148. **def** print\_menu():
149. """Displays a menu of choices to the user
151. Args:
152. None.
154. Returns:
155. None.
156. """
157. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
158. **print**('[s] Save Inventory to file\n[x] exit\n')
160. @staticmethod
161. **def** menu\_choice():
162. """Gets user input for menu selection
164. Args:
165. None.
167. Returns:
168. choice (string): a lower case string of the users input out of the choices l, a, i, d, s or x
170. """
171. choice = ' '
172. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
173. choice = input('Which operation would you like to perform? [l, a, i, s or x]: ').lower().strip()
174. **print**()  # Add extra space for layout
175. **return** choice
177. @staticmethod
178. **def** add\_data():
179. """Get data from user to be added to data structure.
181. Returns:
182. Id (integer): ID of the new CD
183. Title (string): Title of the new CD
184. Artist (string): Artist of the new CD
186. """
187. **while** True:
188. str\_Id = input('Enter ID (Integer only): ').strip()
189. **if** str\_Id.lower() == 'exit':
190. **break**
191. # Try-except to ensure input is integer
192. **try**:
193. Id = int(str\_Id)
194. **except** ValueError as e:
195. **print**("Oops! Please enter integer only. Try again or type \'exit\' to return to the menu.")
196. **continue**
197. Title = input('What is the CD\'s title? ').strip()
198. Artist = input('What is the Artist\'s name? ').strip()
199. **return** Id, Title, Artist
201. @staticmethod
202. **def** show\_inventory(table):
203. """Displays current inventory table
205. Args:
206. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
208. Returns:
209. None.
211. """
212. **print**('======= The Current Inventory: =======')
213. **print**('ID\tCD Title (by: Artist)\n')
214. **for** row **in** table:
215. **print**(row)
216. **print**('======================================')
218. # -- Main Body of Script -- #
220. # 1. When program starts, read in the currently saved Inventory
221. lstOfCDObjects = FileIO.load\_inventory(strFileName)
222. IO.show\_inventory(lstOfCDObjects)
224. # 2. start main loop
225. **while** True:
226. # 2.1 Display Menu to user and get choice
227. IO.print\_menu()
228. strChoice = IO.menu\_choice()
230. # 3. Process menu selection
231. # 3.1 process exit first
232. **if** strChoice == 'x':
233. **break**
234. # 3.2 process load inventory
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. lstOfCDObjects = FileIO.load\_inventory(strFileName)
241. IO.show\_inventory(lstOfCDObjects)
242. **else**:
243. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
244. IO.show\_inventory(lstOfCDObjects)
245. **continue**  # start loop back at top.
246. # 3.3 process add a CD
247. **elif** strChoice == 'a':
248. # 3.3.1 Ask user for new ID, CD Title and Artist
249. # 3.3.2 Add item to the table
250. # Try-except in case user wants to exit out of the loop
251. CdInfo = IO.add\_data()
252. # try:
253. DataProcessor.add\_item(CdInfo, lstOfCDObjects)
254. # except Exception:
255. #     print("Returning to the menu...")
256. IO.show\_inventory(lstOfCDObjects)
257. # 3.4 process display current inventory
258. **elif** strChoice == 'i':
259. IO.show\_inventory(lstOfCDObjects)
260. **continue** # start loop back at top.
261. # 3.5 process save inventory to file
262. **elif** strChoice == 's':
263. # 3.5.1 Display current inventory and ask user for confirmation to save
264. IO.show\_inventory(lstOfCDObjects)
265. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
266. # 3.5.2 Process choice
267. **if** strYesNo == 'y':
268. # 3.5.2.1 save data
269. FileIO.save\_inventory(strFileName, lstOfCDObjects)
270. **else**:
271. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
272. **continue**  # start loop back at top.
273. # 3.6 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
274. **else**:
275. **print**('General Error')

Figure 6 – CDInventory.py