Kelly Ros

2020-Mar-16

Foundations of Programming, Python

Assignment 08

Object Oriented Programming

# Introduction

In this module, we began covering the basics of Object-Oriented Programming or “OOP.” Object-Oriented Program is a different way of thinking about programming, the new concept is that everything is an object. Each object has attributes and methods. We began our first steps into understanding OOP and learned about creating classes and defining objects, writing methods and create attributes for objects, instantiating objects, and restricting access to an object’s attributes.

# Intro to Object Oriented Programming

A Class is the blueprint of an object, classes usually contain methods, things that objects do.[[1]](#footnote-1) In our program we create the CD Class. The CD class is initialized with an ID number, title, and artist with the listing shown in listing 1.

1. **class** CD():
2. """Stores data about a CD:
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. formatted: (string) returns a formatted string of the cd infomation
10. saveFormat: (string) returns a formatted string for saving to a file
11. \_\_str\_\_: returns formatted
12. """
14. # -- CONSTRUCTOR -- #
15. **def** \_\_init\_\_(self, cd\_id=1, cd\_title='title', cd\_artist='artist'):
16. self.\_\_cd\_id = cd\_id
17. self.\_\_cd\_title = cd\_title
18. self.\_\_cd\_artist = cd\_artist

Listing 1 – CD class with constructors or initializers

As seen in listing one, this class contains 3 methods, formatted, saveFormat, and \_\_str\_\_. Methods are functions specific to a class, and can only be used on objects that belong to that class.[[2]](#footnote-2) These class methods, just return a formatted string when it is called. We can see the methods in listing 2.

1. # -- METHODS -- #
2. **def** formatted(self):
3. **return** '{}\t{} by: {}'.format(self.\_\_cd\_id, self.\_\_cd\_title, self.\_\_cd\_artist)
4. **def** saveFormat(self):
5. **return** '{},{},{}\n'.format(self.\_\_cd\_id, self.\_\_cd\_title, self.\_\_cd\_artist)
6. **def** \_\_str\_\_(self):
7. **return** self.formatted()

Listing 2 – Methods from CD Class

I used these methods when calling the object that belongs to CD, if I were to just print(cdObj1), it would call the \_\_str\_\_() function and return the formatted function 4 lines above it. The double underscores before each attribute mean that they are a private attribute and can only be called or changed by functions defined within the class. Classes also include properties and setters. Since each attribute in the class CD is private, we can validate given through the setters. In figure 1, we can see lines 55 through 59 from listing 3 at work.

1. # -- PROPERTIES -- #
2. @property
3. **def** cd\_id(self):
4. **return** self.\_\_cd\_id
6. @cd\_id.setter
7. **def** cd\_id(self, idno):
8. **if** idno.isnumeric():
9. self.\_\_cd\_id = int(idno)
10. **else**:
11. **raise** Exception("IDs can only be numeric")
13. @property
14. **def** cd\_title(self):
15. **return** self.\_\_cd\_title
17. @cd\_title.setter
18. **def** cd\_title(self, title):
19. **if** title ==  "":
20. **raise** Exception("CD title cannot be empty")
21. **else**:
22. self.\_\_cd\_title = title
24. @property
25. **def** cd\_artist(self):
26. **return** self.\_\_cd\_artist
28. @cd\_artist.setter
29. **def** cd\_artist(self, artist):
30. **if** artist == "":
31. **raise** Exception("CD artist cannot be empty")
32. **else**:
33. self.\_\_cd\_artist = artist

Listing 3 – Properties and Setters inside CD class

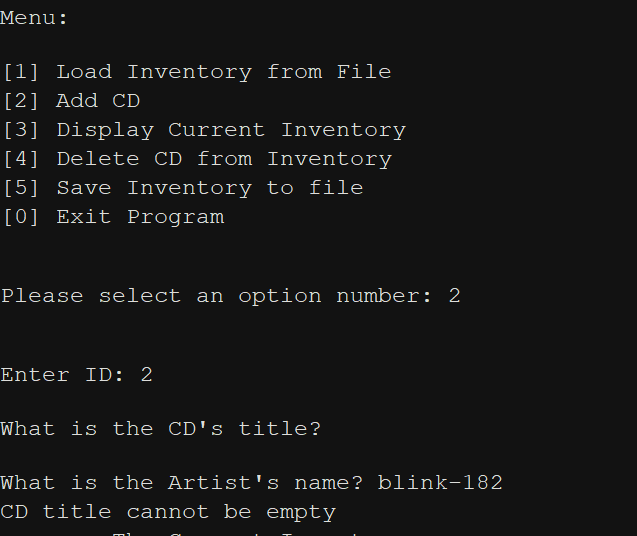


Figure 1 – Testing cd\_title’s validation

We finally understand the use of @staticmethod in our programs. A static method is designed to be called through a class, not an object.[[3]](#footnote-3) This means that we can call the functions from our IO class by just typing IO.print\_menu().

# CD Inventory Program using Object Oriented Programming

In this week’s module, we use our new knowledge of classes and objects to improve our CD Inventory Program. We saw parts of our CD class in listings 1, 2, and 3, which prove the CD Object with the following attributes, ID number, Title, and Artist. The getters and setters in this class do not allow the ID number to be a string and do not let the Title or Artist be empty strings.

Upon running the program, the program attempts to load cdInventory.txt, and if it isn’t found, a FileNotFoundError will raise and state that cdInventory.txt was not loaded and will continue with the program. Successful and unsuccessful loads can be seen in figure 2 and 3, respectively.

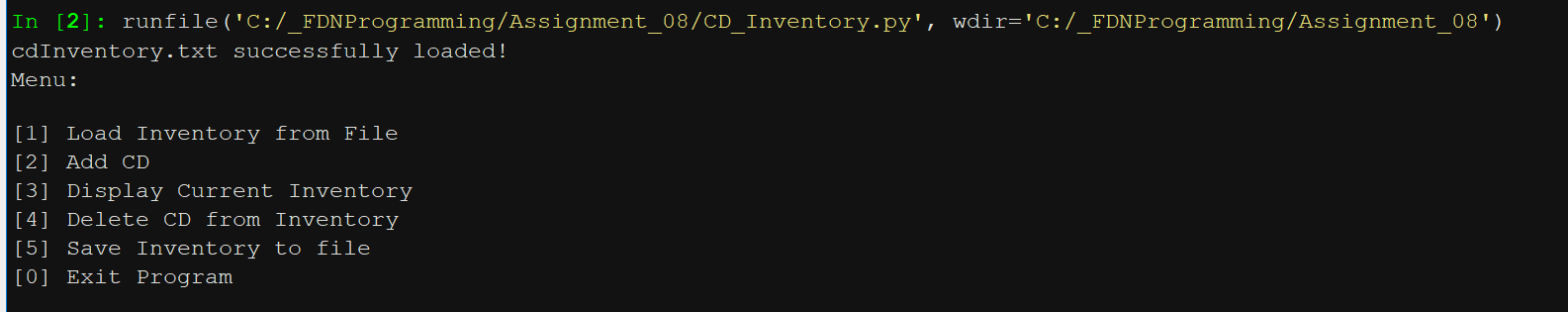


Figure 2 – Successful load upon running CD\_Invntory.py

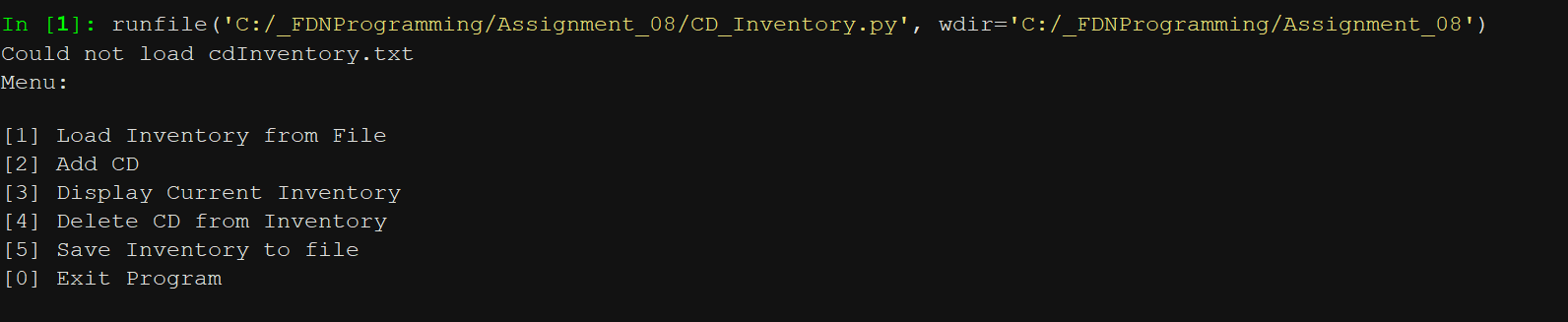


Figure 3 – Unsuccessful load through error handling of FileNotFoundError

Now the user is greeted with a menu, and the user has to select a numbered option. Option 1, is the same load function upon boot, but if selected, the user will be able to clear the current inventory and load or re-load cdInventory.txt. Option 2 allows a user to add a new CD into the inventory. This asks the user for an ID number, CD Title, and the Artist of the CD. This portion uses the getters and setters of the CD class. In order to how they work, the CD’s defaults are set automatically and will change upon correctly assigning attributes. Figure 1 shows an example of incorrectly assigning attributes. The add CD portion can be seen in listing 4.

1. # let user add cd to inventory
2. **elif** strChoice == '2':
3. new\_id, new\_title, new\_artist = IO.input\_CD()
4. newCD = CD()
5. **try**:
6. newCD.cd\_id = new\_id
7. newCD.cd\_title = new\_title
8. newCD.cd\_artist = new\_artist
9. lstOfCDObjects.append(newCD)
10. **except** Exception as e:
11. **print**(e)

Listing 4 – Add CD option of the program

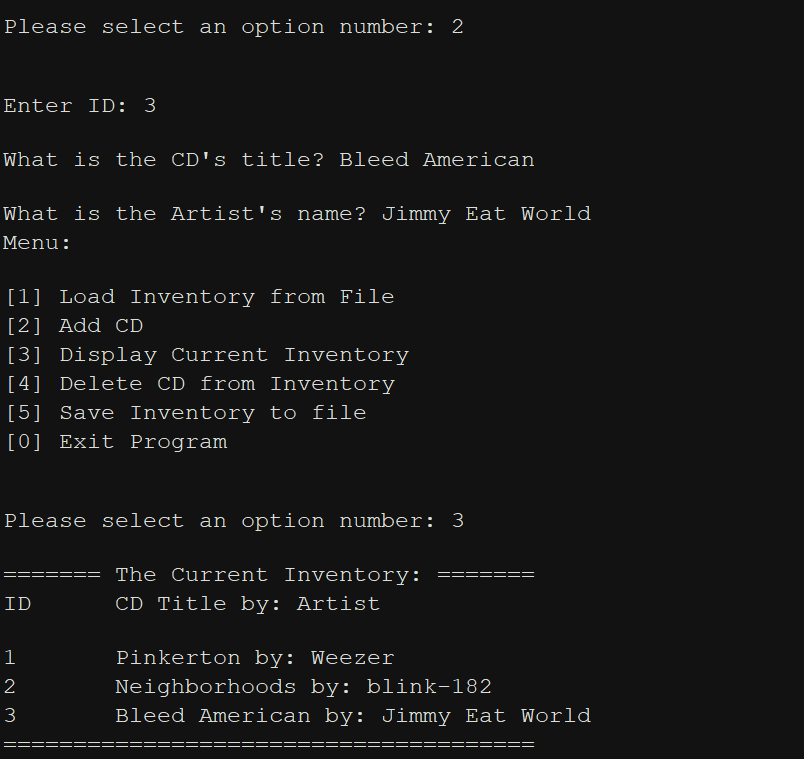


Figure 4 – Using the Add CD option in Spyder

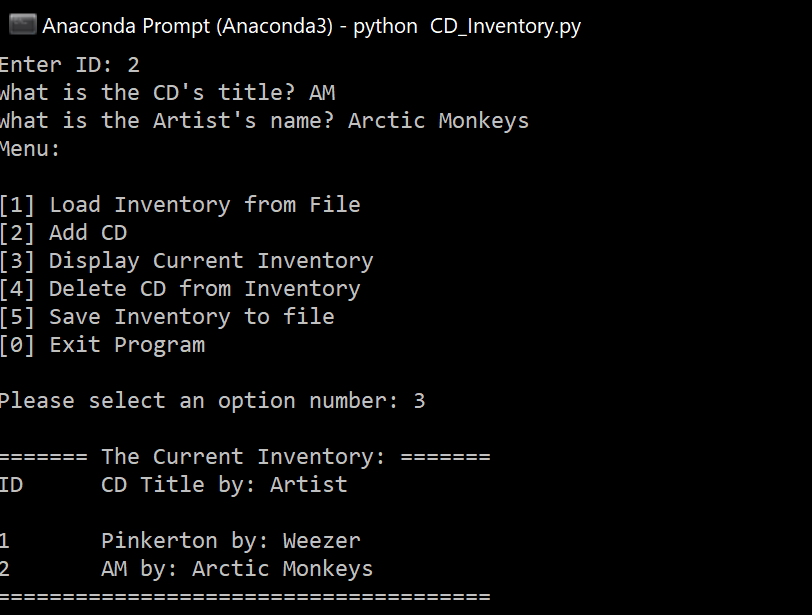


Figure 5 – Using the Add CD option in Anaconda Prompt

The next option displays the current inventory that we have stored in our list of objects, by invoking the show\_Inventory function from the IO class. This potion uses the method cdObj.\_\_str\_\_(), which returns the the formatted() method from the CD Class. This can be shown in listing 5.

1. @staticmethod
2. **def** show\_inventory(table):
3. """Displays current inventory table
5. Args:
6. table (list of obj): 2D data structure that holds the data during runtime
8. Returns:
9. None
11. """
12. **print**('======= The Current Inventory: =======')
13. **print**('ID\tCD Title by: Artist\n')
14. **for** cd **in** table:
15. **print**(cd)
17. **print**('======================================')

Listing 5 – show\_inventory(table) function from IO Class, uses \_\_str\_\_() method from CD class

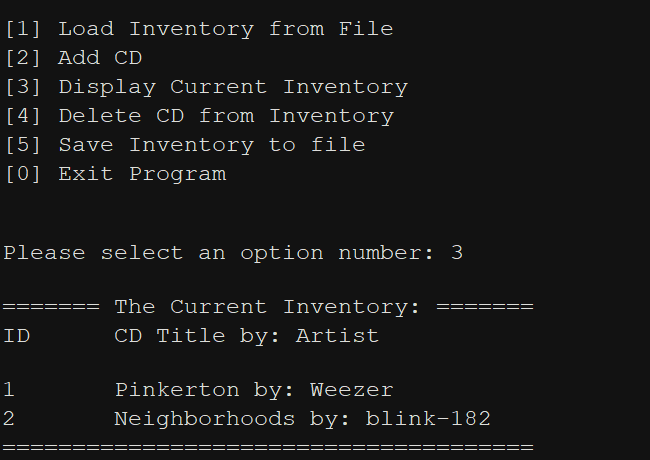


Figure 6 – show\_inventory function on Spyder

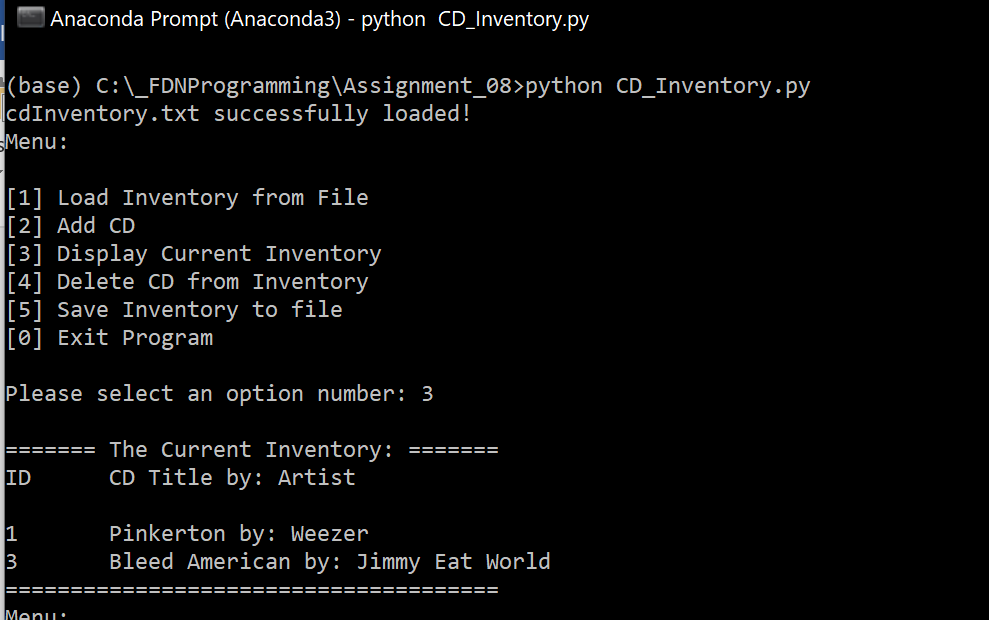


Figure 7 – show\_inventory function on Anaconda Prompt

The next portion wasn’t included in the starter code, but I added it since I thought it was useful from our previous assignments, the delete CD function. This portion works the same as previous assignments, however, instead of looking at dictionary values, we’re looking at the CD class’s cd\_id attribute. The program loops through the list and compares the entered integer to the object’s cd\_id value, and if they are the same, that index of the list is deleted. An example of this can be seen in figure 8.

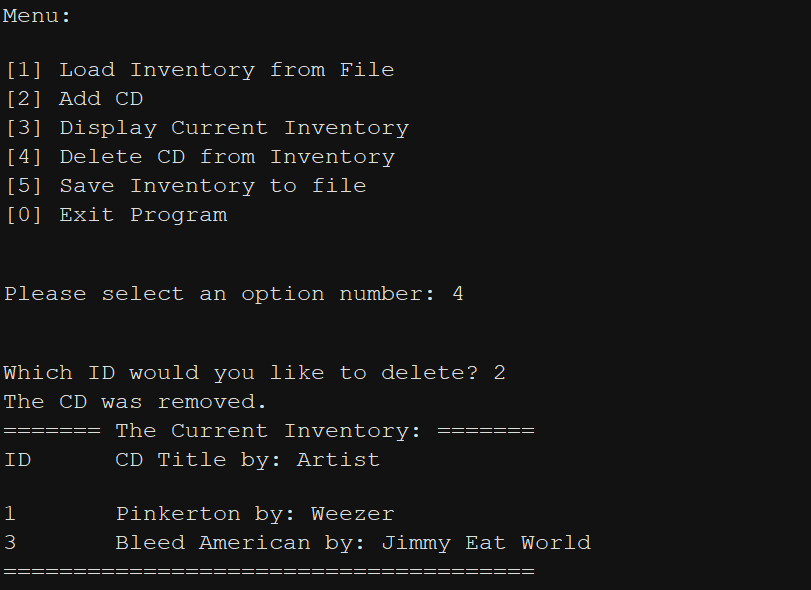


Figure 8 – Delete CD function

The save function is the last option that required the use of a function from the fileIO class. This option saves the current inventory and overwrites, or creates a file cdInventory.txt. This function uses the save\_method() formatting from the CD class. When calling the save\_inventory function, the program goes through each object within the list, and uses the object and calls the save\_method() function to format the string into the text file.

1. @staticmethod
2. **def** save\_inventory(file\_name, table):
3. """Function that saves inventory to text file
5. Args:
6. file\_name (string): name of file used to save data to
7. table (list of obj): 2D data structre that holds data during runtime
8. Returns:
10. """
11. with open(file\_name, 'w') as objFile:
12. **for** cd **in** table:
13. objFile.write(cd.saveFormat())
14. **return** table

Listing 6 – save\_inventory function from fileIO class

The asked if they want to save the file prior to saving, and if ‘y’ is entered, the file will be saved.

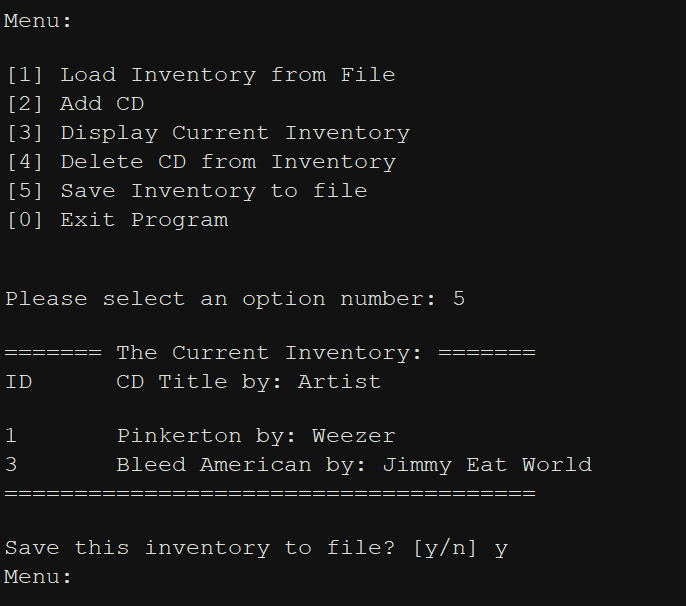


Figure 9 – Saving inventory to file

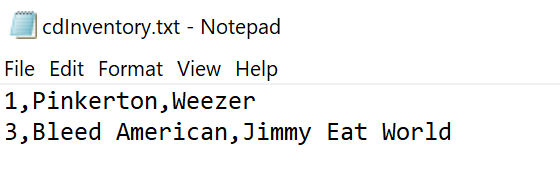


Figure 10 – File after saving

# Summary

In this week’s module, we started with the basics of object-oriented programming. At first it was a little difficult to understand, but after reading a few posts on stackoverflow, seeing more examples, and attempting to correctly explain classes and objects with other members of this class (offline), I’m beginning to understand it a little more. The program has been uploaded to GitHub here: <https://github.com/kellyros12/Assignment_08>

# 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. # KRos, 2020-Mar-14, added constructors, properties and methods to CD Class
8. # KRos, 2020-Mar-15, added in save\_inventory and load\_inventory
9. # KRos, 2020-Mar-16, added in menu options, updated save and load functions
10. # KRos, 2020-Mar-16, updated docstrings
11. # KRos, 2020-Mar-16, added a show\_inventory call into the add cd option
12. #------------------------------------------#
14. # -- DATA -- #
15. strFileName = 'cdInventory.txt'
16. lstOfCDObjects = []

19. **class** CD():
20. """Stores data about a CD:
22. properties:
23. cd\_id: (int) with CD ID
24. cd\_title: (string) with the title of the CD
25. cd\_artist: (string) with the artist of the CD
26. methods:
27. formatted: (string) returns a formatted string of the cd infomation
28. saveFormat: (string) returns a formatted string for saving to a file
29. \_\_str\_\_: returns formatted
30. """
32. # -- CONSTRUCTOR -- #
33. **def** \_\_init\_\_(self, cd\_id=1, cd\_title='title', cd\_artist='artist'):
34. self.\_\_cd\_id = cd\_id
35. self.\_\_cd\_title = cd\_title
36. self.\_\_cd\_artist = cd\_artist
37. # -- ATTRIBUTES -- #
38. # -- PROPERTIES -- #
39. @property
40. **def** cd\_id(self):
41. **return** self.\_\_cd\_id
43. @cd\_id.setter
44. **def** cd\_id(self, idno):
45. **if** idno.isnumeric():
46. self.\_\_cd\_id = int(idno)
47. **else**:
48. **raise** Exception("IDs can only be numeric")
50. @property
51. **def** cd\_title(self):
52. **return** self.\_\_cd\_title
54. @cd\_title.setter
55. **def** cd\_title(self, title):
56. **if** title ==  "":
57. **raise** Exception("CD title cannot be empty")
58. **else**:
59. self.\_\_cd\_title = title
61. @property
62. **def** cd\_artist(self):
63. **return** self.\_\_cd\_artist
65. @cd\_artist.setter
66. **def** cd\_artist(self, artist):
67. **if** artist == "":
68. **raise** Exception("CD artist cannot be empty")
69. **else**:
70. self.\_\_cd\_artist = artist
71. # -- METHODS -- #
72. **def** formatted(self):
73. **return** '{}\t{} by: {}'.format(self.\_\_cd\_id, self.\_\_cd\_title, self.\_\_cd\_artist)
74. **def** saveFormat(self):
75. **return** '{},{},{}\n'.format(self.\_\_cd\_id, self.\_\_cd\_title, self.\_\_cd\_artist)
76. **def** \_\_str\_\_(self):
77. **return** self.formatted()
79. # -- PROCESSING -- #
80. **class** FileIO:
81. """Processes data to and from file:
83. properties:
85. methods:
86. save\_inventory(file\_name, lst\_Inventory): -> None
87. load\_inventory(file\_name): -> (a list of CD objects)
89. """
91. @staticmethod
92. **def** load\_inventory(file\_name, table):
93. """Function that reads data from a file and saves to 2D table
95. Args:
96. file\_name (string): name of file used to read the data from
97. table (list of obj): 2D data structure that holds the data during runtime
98. Returns:
99. table (list of obj): 2D data structure after loading data from file
101. """
102. **try**:
103. with open(file\_name, 'r') as objFile:
104. **for** line **in** objFile:
105. data = line.strip().split(',')
106. cd = CD(int(data[0]),data[1],data[2])
107. table.append(cd)
108. **print**("{} successfully loaded!".format(file\_name))
109. **except** FileNotFoundError:
110. **print**("Could not load {}".format(file\_name))
111. **return** table

114. @staticmethod
115. **def** save\_inventory(file\_name, table):
116. """Function that saves inventory to text file
118. Args:
119. file\_name (string): name of file used to save data to
120. table (list of obj): 2D data structre that holds data during runtime
121. Returns:
123. """
124. with open(file\_name, 'w') as objFile:
125. **for** cd **in** table:
126. objFile.write(cd.saveFormat())
127. **return** table

130. # -- PRESENTATION (Input/Output) -- #
131. **class** IO:
132. """Handles User inputs and outputs"""
134. @staticmethod
135. **def** print\_menu():
136. """Displays a menu of choices to the user
138. Args:
139. None.
141. Returns:
142. None.
143. """
145. **print**('Menu: \n\n[1] Load Inventory from File\n[2] Add CD\n[3] Display Current Inventory')
146. **print**('[4] Delete CD from Inventory\n[5] Save Inventory to file\n[0] Exit Program\n')
148. @staticmethod
149. **def** menu\_choice():
150. """Gets user input for menu selection
152. Args:
153. None
155. Returns:
156. choice (string): a string of the users input out of the choices 0 through 5
158. """
159. choice = ''
160. **while** choice **not** **in** ['0', '1', '2',  '3', '4', '5']:
161. choice = input('Please select an option number: ').strip()
162. **print**() #formatting
163. **return** choice
165. @staticmethod
166. **def** show\_inventory(table):
167. """Displays current inventory table
169. Args:
170. table (list of obj): 2D data structure that holds the data during runtime
172. Returns:
173. None
175. """
176. **print**('======= The Current Inventory: =======')
177. **print**('ID\tCD Title by: Artist\n')
178. **for** cd **in** table:
179. **print**(cd)
181. **print**('======================================')
183. @staticmethod
184. **def** input\_CD():
185. """Asks user for CD inputs
187. Args:
188. None
190. Returns:
191. ID, Title, Artist
193. """
194. **try**:
195. ID = input('Enter ID: ').strip()
196. Title = input('What is the CD\'s title? ').strip()
197. Artist = input('What is the Artist\'s name? ').strip()
198. **return** ID, Title, Artist
199. **except** ValueError:
200. **print**('ID should be an integer.')
202. @staticmethod
203. **def** delete\_CD(idno, table):
204. """Function that removes a CD from table
206. Args:
207. idno (int): CD ID number to be deleted
208. table (list of Obj): 2D list of objects that CD should be removed from
210. Returns:
211. table (list of Obj): edted 2D list
212. """
213. intRowNr = -1
214. blnCDRemoved = False
215. **for** cd **in** table:
216. intRowNr += 1
217. **if** cd.cd\_id == idno:
218. **del** table[intRowNr]
219. blnCDRemoved= True
220. **break**
221. **if** blnCDRemoved:
222. **print**('The CD was removed.')
223. **else**:
224. **print**('Coud not find CD!')
225. **return** table

228. # -- Main Body of Script -- #
229. # Load data from file into a list of CD objects on script start
230. lstOfCDObjects = FileIO.load\_inventory(strFileName, lstOfCDObjects)
231. # Display menu to user
232. **while** True:
234. IO.print\_menu()
235. strChoice = IO.menu\_choice()

238. # let user load inventory from file
239. **if** strChoice == '1':
240. **print**("WARNING: This will clear current inventory.")
241. strYesNo = input("Type 'yes' to continue. ")
242. **if** strYesNo.lower() == 'yes':
243. lstOfCDObjects = [] # clears table
244. lstOfCDObjects = FileIO.load\_inventory(strFileName, lstOfCDObjects)
245. **else**:
246. **print**("Inventory not reloaded")
247. **continue**
248. # let user add cd to inventory
249. **elif** strChoice == '2':
250. new\_id, new\_title, new\_artist = IO.input\_CD()
251. newCD = CD()
252. **try**:
253. newCD.cd\_id = new\_id
254. newCD.cd\_title = new\_title
255. newCD.cd\_artist = new\_artist
256. lstOfCDObjects.append(newCD)
257. **except** Exception as e:
258. **print**(e)
260. IO.show\_inventory(lstOfCDObjects)
261. **continue**
262. #display inventory
263. **elif** strChoice == '3':
264. IO.show\_inventory(lstOfCDObjects)
265. **continue**
266. #added delete CD option
267. **elif** strChoice == '4':
268. **try**:
269. del\_ID = int(input('Which ID would you like to delete? ').strip())
270. IO.delete\_CD(del\_ID, lstOfCDObjects)
271. IO.show\_inventory(lstOfCDObjects)
272. **except** ValueError:
273. **print**("Expected an ID number.")
274. **continue**
275. # let user save inventory to file
276. **elif** strChoice == '5':
277. IO.show\_inventory(lstOfCDObjects)
278. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
279. **if** strYesNo == 'y':
280. FileIO.save\_inventory(strFileName, lstOfCDObjects)
281. **print**('File saved!')
282. **else**:
283. **print**('File was not saved!')
284. **continue**
285. **else**:
286. **print**('Exiting program.')
287. **break**

1. From Python Programming for the Absolute Beginner, Third Edition. Pg. 220 [↑](#footnote-ref-1)
2. From <https://stackoverflow.com/questions/6480676/why-use-classes-instead-of-functions> retrieved on 14-March-2020. [↑](#footnote-ref-2)
3. From Python Programming for the Absolute Beginner, Third Edition. Pg. 231 [↑](#footnote-ref-3)