Laurel Warner

12/5/2021

Foundations of Programming

Assignment08

Intro to Software Objects

# Introduction

This module introduced the subject of software objects. Some of this I had seen before, but I just didn’t know that it WAS part of software objects. Examples are: classes and decorators like “@staticmethod”. This module also gave more information about git and GitHub as well. This module introduced classes, objects as well as constructors, fields, attributes and methods.

# Classes

A class is like a “blueprint” for an object. The class packages the data and functionality of an object. The object itself is what’s known as an “instantiation” of the class. It’s 1 “instance” of the class with unique attributes. The object as the same functionality as the class that it is a part of. The components that make up a class are fields, constructors, attributes, properties, and methods. Fields are the data stores of a class. They get created the same way as variables, but you don’t need to call them out specifically. I know this is true because even though they are mentioned in this module, they are NOT mentioned at all in the book or in any other web sources I could find. And indeed, once a constructor and attributes are created you can just go ahead and delete your fields if you had previously created them (which is what happened in Lab C). Constructors are invoked when creating an object. They are a way to ensure proper datatypes in the fields. They also allow for population default values for the fields as well so that they all start out the same way before they get changed either by the program or by user input. They are created by calling the \_\_init\_\_() method. (Called “dunder init method”.) Attributes are internal fields or variables that hold data. (So having fields and attributes both is redundant I believe). Attributes are variables and unless you write code to validate values before they are assigned you won’t have any control over what goes into them or how they change during runtime. Properties are methods that have control mechanisms built in that control the validity of values assigned to the attributes and if you make your attributes private then you would use private properties and public properties both in tandem to enforce the interaction with the attributes in such a way that they can’t be changed in ways you don’t want. There are 2 properties for each attribute if you choose to go that route. They are the “getter” and “setter” properties. The “setter” writes or sets the attribute and the “getter” reads or “gets” it. Getter methods let you add code to format the fields or attributes data and the setter methods let you add validation or error handling to the code. Methods are like functions in a script. They let you organize your statements into blocks that can be invoked by calling the method’s name. Calling a method also submits a reference to the object it’s invoked on so the first attribute for a method is the “self” reference.

# Decorators, Docstrings, and the Self Keyword

The keyword “self” is the first parameter in every method. Every method automatically receives a reference to that object, so you have to put “self” so the class knows on which object to use the method. Self is used in all your constructors, as well as in all the attributes, properties, and methods. It is ALWAYS used except for static methods. Decorators are what gets typed before your properties and say things like “@property” or “@property.setter” as well as our old friend “@staticmethod”. They basically change the behavior or scope of functions. They wrap functionality around a method. Using @staticmethod is done when you want a method to be called on the class level and not on an instance level. This allows you to keep track of information across a whole class. This is the only time when you don’t need to use the “self” keyword. Docstrings are included in classes, properties, and methods so that you can type in the pertinent information about it. When you do that, then it allows hints and helps to populate in your IDE environment.

# Labwork

The labwork this week was not too bad. Here is a screenshot of my Lab E which has most everything else in it from Labs A-D (except for the Fields from Lab A).

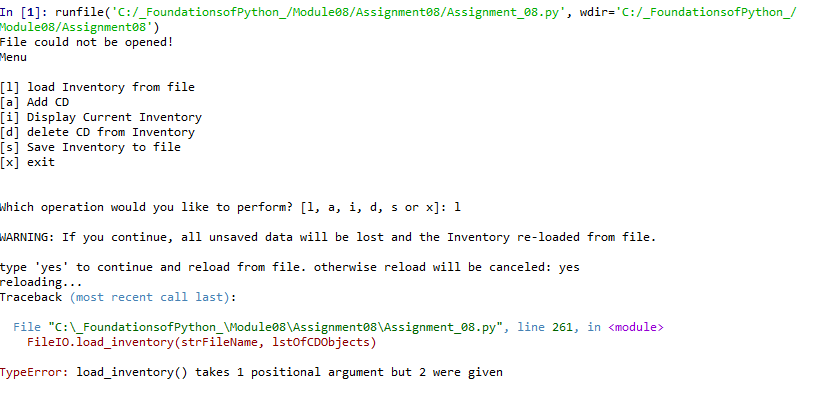


Figure

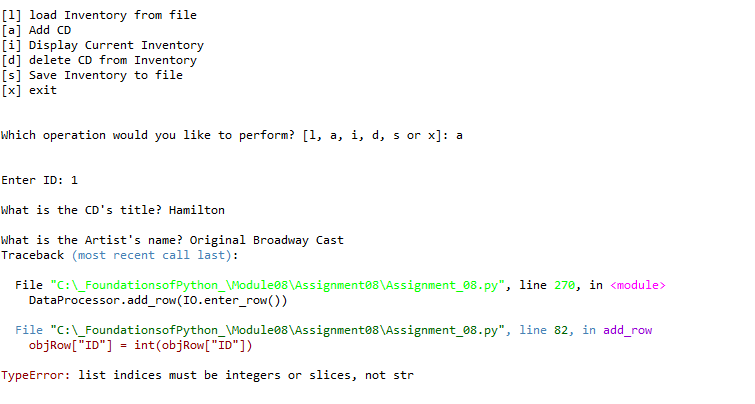
I got to see how to properly construct a class with its’ attributes, properties and methods. This led directly into the assignment for the week.

# Assignment Code

For this assignment I wound up copying and pasting most of my data from lab E (but not the method) and just changing the attributes and properties to match the CD Title, Artist and CD ID that were given. I don’t know how this bears on the rest of the code though, because it doesn’t look like it ever gets called anywhere else in the code. But maybe it’s supposed to and that is 1 area that I messed up on. I added a new class of DataProcessor because it wasn’t there in the starter code. It was in the assignment for prior assignments so I just went ahead and added it here because I knew it would most likely be needed. I updated and changed all the instances of “lstTbl” and “dicRow” to be “lstOfCDObjects” and “objRow” to match this weeks tasks of working with CD objects and not tables of lists or dictionaries. In the FileIO class I copied and pasted my code from prior weeks for the load\_inventory and save\_inventory methods that were given in the starter code. Also with some error handling. Here, also, I updated the instances of “lstTbl” and “tblRow” and “dicRow” (whichever one as the case may be) to match the fact that we are working with objects this week. For the IO class, I copied and pasted my code from Assignment 6 and 7 and it didn’t appear to need much done to it, except in the “enter\_row()” method. I just had to update the Args to be cd\_id, cd\_title, and cd\_artist instead of “strID”, “strTitle”, or “strArtist”. Then in the main body of the code I copied and pasted my code from Assignment 6 and updated all the instances that call different functions to make sure the names of the classes and methods match up. I did get it to the point where Spyder didn’t have any error messages in the margins saying that there was a syntax error. When I ran the code though, I did get runtime errors. My helpful tutor from prior weeks was not available to help me this week due to holiday commitments, so I will not be able to provide screenshots of the code working properly. I can provide a couple screenshots of the errors it gave me however. I hope that helps. Because of these 2 errors, I wasn’t able to test out all the portions of my code.



Figure



Figure

I did upload my files to GitHub at the following address: https://github.com/lellenn/Assignment\_08.

# Summary

In this module I did learn about classes, attributes, properties, and methods. I learned how to set each of these items up with very simplistic information. I learned about what all of these things are. However, I did have errors with my code as I laid out in the prior section and I could not solve them on my own.

# Appendix

1. *#------------------------------------------#*
2. *# Title: Assignment08.py*
3. *# Desc: Assignment 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. *# LWarner, 2021-Dec-05, added code and docstrings for functionality*
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:
23. """
24. *#--Constructor--#*
25. **def \_\_init\_\_(self, cd\_id, cd\_title, cd\_artist):**
26. *#--Attributes--#*
27. self.\_\_cd\_id = cd\_id
28. self.\_\_cd\_title = cd\_title
29. self.\_\_cd\_artist = cd\_artist
31. *#--Properties--#*
33. @property
34. **def** cd\_id(self):
35. **return self.\_\_cd\_id**
37. @cd\_id.setter
38. **def** cd\_id(self, cd\_id):
39. **if** str(cd\_id).isnumeric():
40. **return cd\_id**
41. **else**:
42. **raise** Exception ('ID must be a number!')
44. @property
45. **def cd\_title(self):**
46. **return** self.\_\_cd\_title
48. @cd\_title.setter
49. **def** cd\_title(self, cd\_title):
50. **if str(cd\_title).isnumeric():**
51. **raise** Exception('Title must be a string.')
52. **else**:
53. **return** cd\_title
55. **@property**
56. **def** cd\_artist(self):
57. **return** self.\_\_cd\_artist
59. @cd\_artist.setter
60. **def cd\_artist(self, cd\_artist):**
61. **if** str(cd\_artist).isnumeric():
62. **raise** Exception('Artist must be a string.')
63. **else**:
64. **return** cd\_artist
66. *# -- PROCESSING -- #*
67. **class** DataProcessor:
68. """ Functions to add, delete, and save data in the table"""
70. **@staticmethod**
71. **def** add\_row(objRow):
72. """Function to append a row to the table
73. Gathers the user input for ID, Title and Artist and appends it to the table in memory.
75. **Args:**
76. row: the row of ID, Title, and Artist
77. table: the 2D data structure that holds the data during runtime
79. Returns:
80. **Shows inventory**
81. """
82. objRow["ID"] = int(objRow["ID"])
83. lstOfCDObjects.append(objRow)
85. **@staticmethod**
86. **def** delete\_row(intRowNr, lstTbl):
87. """ Function to delete a row from the table if the user wants.
89. Args:
90. **intRowNr: the ID number for the row to be deleted**
91. lstTbl: the table with the data
93. Returns:
94. The information showing whether or not the row was successfully deleted.
95. **"""**
96. intRowNr = -1
97. blnCDRemoved = False
98. **for** row **in** lstTbl:
99. intRowNr += 1
100. **if row['ID'] == intIDDel:**
101. **del** lstTbl[intRowNr]
102. blnCDRemoved = True
103. **break**
104. **if** blnCDRemoved:
105. **print('The CD was removed')**
106. **else**:
107. **print**('Could not find this CD!')
109. **class** FileIO:
110. **"""Processes data to and from file:**
112. properties:
114. methods:
115. **save\_inventory(file\_name, lst\_Inventory): -> None**
116. load\_inventory(file\_name): -> (a list of CD objects)
118. """
119. @staticmethod
120. **def load\_inventory(file\_name):**
121. """Function to manage data ingestion from file to a list of objects
123. Reads the data from file identified by file\_name into a 2D table
124. (list of objects) table one line in the file represents one object row in table.
126. Args:
127. file\_name (string): name of file used to read the data from
129. Returns:
130. **lstOfCDObjects (list of objects): 2D data structure (list of objects) that holds the data during runtime**
131. """
132. lstOfCDObjects.clear() *# this clears existing data and allows to load data from file*
133. **try**:
134. objFile = open(file\_name, 'r')
135. **for line in objFile:**
136. data = line.strip().split(',')
137. objRow = [int(data[0]), data[1], data[2]]
138. lstOfCDObjects.append(objRow)
139. objFile.close()
140. **except OSError:**
141. **print**('File could not be opened!')
143. @staticmethod
144. **def** save\_inventory(file\_name, lst\_Inventory):
145. **""" Function to allow data to be written to the .txt file that has previously been created**
147. Args:
148. strFileName (string): name of file used to write the data to
149. lstOfCDObjects (list of objects): 2D data structure (list of objects) that holds the data during runtime
151. Returns:
152. None.
153. """
154. **try**:
155. **objFile = open(strFileName, 'w')**
156. **for** row **in** lstOfCDObjects:
157. lstValues = list(row.values())
158. lstValues[0] = str(lstValues[0])
159. objFile.write(','.join(lstValues) + '**\n**')
160. **objFile.close()**
161. **except** OSError:
162. **print**('File could not be opened!')
164. *# -- PRESENTATION (Input/Output) -- #*
165. **class IO:**
166. """Handling Input / Output"""
168. @staticmethod
169. **def** print\_menu():
170. **"""Displays a menu of choices to the user**
172. Args:
173. None.
175. **Returns:**
176. None.
177. """
179. **print**('Menu**\n\n**[l] load Inventory from file**\n**[a] Add CD**\n**[i] Display Current Inventory')
180. **print('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')**
182. @staticmethod
183. **def** menu\_choice():
184. """Gets user input for menu selection
186. Args:
187. None.
189. Returns:
190. **choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x**
192. """
193. choice = ' '
194. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
195. **choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()**
196. **print**() *# Add extra space for layout*
197. **return** choice
199. @staticmethod
200. **def show\_inventory(table):**
201. """Displays current inventory table

204. Args:
205. **lstOfCDObjects (list of objects): 2D data structure (list of objects) that holds the data during runtime.**
207. Returns:
208. None.
210. **"""**
211. **print**('======= The Current Inventory: =======')
212. **print**('ID**\t**CD Title (by: Artist)**\n**')
213. **for** row **in** lstOfCDObjects:
214. **print**('{}**\t**{} (by:{})'.format(\*row.values()))
215. **print('======================================')**
217. @staticmethod
218. **def** enter\_row():
219. """ Gets user input for adding a new CD to the inventory
221. Args:
222. cd\_id: The ID of the CD to be added
223. cd\_title: the Title of the CD to be added
224. cd\_artist: the Artist of the CD to be added
226. Returns:
227. The ID, Title, and Artist info for each CD rendered as a string.
228. """
229. **while** True:
230. **try:**
231. cd\_id = int(input('Enter ID: ').strip())
232. **break**
233. **except** ValueError:
234. **print**('You have not entered a number. Please enter a number')
236. cd\_title = input('What is the CD**\'**s title? ').strip()
237. cd\_artist = input('What is the Artist**\'**s name? ').strip()
238. **return** [ cd\_id, cd\_title, cd\_artist]

241. *# -- Main Body of Script -- #*
242. *# 1. When program starts, read in the currently saved Inventory*
243. FileIO.load\_inventory(strFileName)
245. ***# 2. start main loop***
246. **while** True:
247. *# 2.1 Display Menu to user and get choice*
248. IO.print\_menu()
249. strChoice = IO.menu\_choice()
251. *# 3. Process menu selection*
252. *# 3.1 process exit first*
253. **if** strChoice == 'x':
254. **break**
255. ***# 3.2 process load inventory***
256. **if** strChoice == 'l':
257. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
258. strYesNo = input('type **\'**yes**\'** to continue and reload from file. otherwise reload will be canceled: ')
259. **if** strYesNo.lower() == 'yes':
260. **print('reloading...')**
261. FileIO.load\_inventory(strFileName, lstOfCDObjects)
262. IO.show\_inventory(lstOfCDObjects)
263. **else**:
264. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
265. **IO.show\_inventory(lstOfCDObjects)**
266. **continue** *# start loop back at top.*
267. *# 3.3 process add a CD*
268. **elif** strChoice == 'a':
269. *# 3.3.1 Ask user for new ID, CD Title and Artist*
270. **DataProcessor.add\_row(IO.enter\_row())**
271. IO.show\_inventory(lstOfCDObjects)
272. **continue** *# start loop back at top.*
273. *# 3.4 process display current inventory*
274. **elif** strChoice == 'i':
275. **IO.show\_inventory(lstOfCDObjects)**
276. **continue** *# start loop back at top.*
277. *# 3.5 process delete a CD*
278. **elif** strChoice == 'd':
279. *# 3.5.1 get Userinput for which CD to delete*
280. ***# 3.5.1.1 display Inventory to user***
281. IO.show\_inventory(lstOfCDObjects)
282. *# 3.5.1.2 ask user which ID to remove*
283. intIDDel = int(input('Which ID would you like to delete? ').strip())
284. *# 3.5.2 search thru table and delete CD*
285. **DataProcessor.delete\_row(intIDDel, lstOfCDObjects)**
286. IO.show\_inventory(lstOfCDObjects)
287. **continue** *# start loop back at top.*
288. *# 3.6 process save inventory to file*
289. **elif** strChoice == 's':
290. ***# 3.6.1 Display current inventory and ask user for confirmation to save***
291. IO.show\_inventory(lstOfCDObjects)
292. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
293. *# 3.6.2 Process choice*
294. **if** strYesNo == 'y':
295. ***# 3.6.2.1 save data***
296. FileIO.save\_inventory(strFileName, lstOfCDObjects)
297. **else**:
298. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
299. **continue** *# start loop back at top.*
300. ***# 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:***
301. **else**:
302. **print**('General Error')