Beemnet Workeneh

09/03/2020

FDN 100

Assignment 08

Software Objects

# Introduction

In this assignment we will compare classes and objects that are made from a class. We will look into the components that make up the standard pattern of a class, what the purpose of a class constructor is, when to use the keyword “self” vs when to use “@staticmethod”. We will look into the relationship between fields, attributes, and property functions. We will see how to differentiate property and method. We will learn the purpose of including a docstring in a class. We will then use what we learn to modify the CDInventory program from the last assignment and we will upload that to github.

# Exploration Questions

*What is the difference between a class and the objects made from a class?*

Classes define attributes and methods. Classes are like blueprints, they are designed for creating objects. A programmer can create many objects from the same class. Each object created the same class will have a similar structure. But each object can be modified to have different attributes while maintaining the same structure.

The object is an instantiation of the class. Each instance or individual object, has the same functionality as the class but is “customized” by the attributes used during initialization.

Real-life objects have characteristics and behaviors. Attributes are characteristics of software objects and Methods are behaviors of software objects or what the object can do. Lets say the object was an alien spacecraft, its attributes would be location and energy level while its methods could be ability to make space jumps or fire lasers. Another way to think about methods is as functions associated with an object.

*What are the components that make up the standard pattern of a class?*

The following are components that make up the standard pattern of a class:

*Class definition*

*Fields*

*Constructor*

*Attributes*

*Properties*

*Methods*

*What is the purpose of a class constructor?*

Constructors are special methods that we can write that are automatically invoked right after a new object is created. The constructor method is usually used to set up the initial attribute values of an object. A constructor method is defined using “def \_\_init\_\_(self)”. Python already recognizes \_\_init\_\_() and this constructor method is automatically called by any newly created object right after the object is live. It can take more parameters other than *self*.

*When do you use the keyword "self"?*

Every instancemethod-a method that every object of a class has- have to have the “self” as the first parameter. This parameter provides a way for a method to refer to the object itself. As the first parameter in every method, self automatically receives a reference to the object invoking the method. Through self, a method can get at the object invoking it and access the object’s attributes and methods or even create new attributes for the object.

*When do you use the keyword "@staticmethod"?*

@staticmethod is a decorator used right before defining a static method. Static methods are used to work with class attributes. A class attribute is a value that is associated with a class itself. Let’s say we want to know how many alien spaceship objects that we have. We can create a static attribute for the alien spaceship class called total and we can have the value of total increased when a new spaceship is created (or decreased when the spaceship attribute turns to “shot down”). We can then use a static method function that we can use to display the total number of spaceships. The static method is designed to be invoked through a class and not an object.

*How are fields and attributes and property functions related?*

A property is an object with methods that allow indirect access to attributes and can impose restriction on that access. Properties allow you to manage exactly how an attribute (could be a private attribute) is accessed or changed Fields are the data stores of a class and get treated the same way as variables. Attribute values can be set equal to field values.

To differentiate when they are used in a class, we call functions methods and variables and constants are fields.

*What is the difference between a property and a method?*

Properties are specialized methods that are used to make class attributes private and enforce the interaction with them. Methods are like functions in a script: They allow you to organize your statements into blocks that can be invoked by calling the method. Methods do not take on the decorator *@property.*

*Why do you include a docstring in a class?*

We include a docstring in a class to explain how to use it to create an object.

# CDInventory Program

In this assignment I modified the CD Inventory program from the past assignments.

*The CD Class*

This class is used to create the CD objects:

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:
10. """
11. **def** \_\_init\_\_(self, cdId, cdTitle, cdArtist):
12. #   -- Attributes  -- #
13. self.\_\_cd\_id = cdId
14. self.\_\_cd\_title = cdTitle
15. self.\_\_cd\_artist = cdArtist
17. # -- Properties -- #
18. @property
19. **def** cd\_id(self):
20. **return** self.\_\_cd\_id
21. @cd\_id.setter
22. **def** cd\_id(self, value):
23. **if** type(value) **is** int:
24. self.\_\_cd\_id = int(value)
25. **else**:
26. **raise** Exception('The ID has to be an integer')
28. @property
29. **def** cd\_title(self):
30. **return** self.\_\_cd\_title
31. @cd\_title.setter
32. **def** cd\_title(self, value):
33. **if** str(value) == '':
34. **raise** Exception('The title cannot be empty')
35. **else**:
36. self.\_\_title = str(value)
38. @property
39. **def** cd\_artist(self):
40. **return** self.\_\_cd\_artist
41. @cd\_artist.setter
42. **def** cd\_artist(self, value):
43. **if** str(value) == '':
44. **raise** Exception('The artist cannot be empty')
45. **else**:
46. self.\_\_cd\_artist= str(value)

Listing - The CD class

Calling CD() and passing in cd\_id, cd\_artist, and cd\_title will create a CD object. I have property methods for the cd\_id, cd\_artist, and cd\_title that have exception handling. The cd\_id only accepts integers, the cd\_artist and cd\_title do not accept empty strings.

*The FileIO class*

This class organizes the functions to save CD Inventory data to a binary file and to load CD Inventory data from a binary file. The functions are very similar to the read and write functions from the previous assignment except for a few name changes and docstring updates.

1. **class** FileIO:
2. """Processes data to and from file:
4. properties:
6. methods:
7. save\_inventory(file\_name, lst\_CDObjects): -> None
8. load\_inventory(file\_name): -> (a list of CD objects)
10. """
11. @staticmethod
12. **def** save\_inventory(file\_name, lst\_CDObjects):
13. """ save inventory to a binary file
14. Args:
15. file\_name (str): file to open to save CD inventory to
16. lst\_CDObjects(list of obj): list of CD objects to write into file
17. Return: None
18. """
19. with open(file\_name, 'wb') as objFile:
20. pickle.dump(lst\_CDObjects, objFile)
22. @staticmethod
23. **def** load\_inventory(file\_name, lst\_CDObjects):
24. """ load inventory from a binary file
25. Args:
26. file\_name(str): file to open to load CD data from
27. Return:
28. a list of CD objects
29. """
30. **try**:
31. with open(file\_name, 'rb') as objFile:
32. lst\_CDObjects = pickle.load(objFile)
33. **return** lst\_CDObjects
34. **except** FileNotFoundError:
35. **print**("The file {} could not be loaded".format(file\_name))
36. **return** lst\_CDObjects

Listing The FileIO class

*The IO Class*

In the IO class I reuse the print\_menu(), menu\_choice function from the previous versions of the CDInventory program. The show\_inventory() function is also from previous assignments except it’s been updated to display from the CD object attributes instead of a dictionary row’s values. See line 151 in the CDInventory.py (Appendix)

The get\_int\_value() and get\_new\_cd() were written and demonstrated by Douglas Klos in response to the previous assignment. The get\_int\_value() handles exception handling for entering an integer value and is reusable in the adding CD and deleting CD functionalities.

I rewrote get\_new\_cd() to take in a list of CD IDs in the current inventory and to check if the CD ID being added is unique. In order to do this I added my own custom Exception class called IDAlreadyExistsError (see CDInventory.py line 263 in the Appendix) . It’s possible to also have the integer input happen in a while loop while the condition of the loop checks to see if the integer ID entered is in the list of existing CD IDs. To get the list of current CD IDs I added a function called get\_current\_IDs() that takes the list of CD Objects and iterates through each to obtain the ID value and append it to a list that it returns (see CDInventory.py line 203 and 291 in the Appendix)

The add\_cd() function receives the CD ID, artist, and title entries that were obtained from the get\_new\_CD function and the current list of CD objects. It creates a new CD Object with that have the attributes equal to the CD entry values that were passed in. It will then append the CD Object to the list of CD Objects and pass back the list (see CDInventory.py line 216 in the Appendix).

The delete\_cd() functions is similar to what was used in the previous assignment. It has been modified to iterate through the list of CD Objects and compare the cd\_id attribute of each CD object against the ID to be deleted (see CDInventory.py line 251-253 in the Appendix).

The main body of script is virtually the same as what was used with the previous assignment, it just has some name changes for the new functions.

### Testing the script

*File not found*

For this part I deleted the binary file from the folder and then started the file. The main body of the script starts by trying to load from the file before continuing to the while loop.

Table - Displaying FileNotFound Exception handling

|  |  |
| --- | --- |
| Executed in Spyder | Executed in Anaconda Prompt |
|  |  |

*Adding a CD: Catching non int ID, empty artist and empty title errors*

Table Adding CD and Exceptioin Handling

|  |  |
| --- | --- |
| Executed in Spyder | Executed in Anaconda Prompt |
|  |  |

*Display Current Inventory*

Table Displaying Inventory

|  |  |
| --- | --- |
| Executed in Spyder | Executed in Anaconda Prompt |
|  |  |

*Save Inventory to file*

While I have the script open in both Spyder and Anaconda Prompt, I interact with it first on Spyder then on Prompt. By doing so I overwrite the file that was written using Spyder.

Table Saving Inventory to file

|  |  |
| --- | --- |
| Executed in Spyder | Executed in Anaconda Prompt |
|  |  |

*Load Inventory from file*

I restart the Kernel in Spyder and load the file and I see that it is the same as what was saved by Anaconda Prompt

Table Loading Inventory from file

|  |  |
| --- | --- |
| Executed in Spyder | Executed in Anaconda Prompt |
|  |  |

# Summary

In this assignment we get introduced to object oriented programing. We saw how objects are created from classes. We saw the standard components of a class. We saw the use of the class constructor, the use of the keyword “self” and “@staticmethod”. We saw the relationship between fields, attributes, and property functions. We differentiated properties and methods. We then modified the CDInventory script to create CD objects, display their contents, and save them.

# Appendix

## GitHub Repository

The CDInventory code can be found under the following link

https://github.com/beemnet20/Assignment\_08

## CDInventory.py

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. # BWorkeneh, 2020-Mar-09, modified code
8. #------------------------------------------#
9. **import** pickle
10. # -- DATA -- #
11. strFileName = 'cdInventory.dat'
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:
22. None
24. """
25. **def** \_\_init\_\_(self, cdId, cdTitle, cdArtist):
26. #   -- Attributes  -- #
27. self.\_\_cd\_id = cdId
28. self.\_\_cd\_title = cdTitle
29. self.\_\_cd\_artist = cdArtist
31. # -- Properties -- #
32. @property
33. **def** cd\_id(self):
34. **return** self.\_\_cd\_id
35. @cd\_id.setter
36. **def** cd\_id(self, value):
37. **if** type(value) **is** int:
38. self.\_\_cd\_id = int(value)
39. **else**:
40. **raise** Exception('The ID has to be an integer')
42. @property
43. **def** cd\_title(self):
44. **return** self.\_\_cd\_title
45. @cd\_title.setter
46. **def** cd\_title(self, value):
47. **if** str(value) == '':
48. **raise** Exception('The title cannot be empty')
49. **else**:
50. self.\_\_title = str(value)
52. @property
53. **def** cd\_artist(self):
54. **return** self.\_\_cd\_artist
55. @cd\_artist.setter
56. **def** cd\_artist(self, value):
57. **if** str(value) == '':
58. **raise** Exception('The artist cannot be empty')
59. **else**:
60. self.\_\_cd\_artist= str(value)
62. # -- PROCESSING -- #
63. **class** FileIO:
64. """Processes data to and from file:
66. properties:
68. methods:
69. save\_inventory(file\_name, lst\_CDObjects): -> None
70. load\_inventory(file\_name): -> (a list of CD objects)
72. """
73. @staticmethod
74. **def** save\_inventory(file\_name, lst\_CDObjects):
75. """ save inventory to a binary file
76. Args:
77. file\_name (str): file to open to save CD inventory to
78. lst\_CDObjects(list of obj): list of CD objects to write into file
79. Return: None
80. """
81. with open(file\_name, 'wb') as objFile:
82. pickle.dump(lst\_CDObjects, objFile)
84. @staticmethod
85. **def** load\_inventory(file\_name, lst\_CDObjects):
86. """ load inventory from a binary file
87. Args:
88. file\_name(str): file to open to load CD data from
89. Return:
90. a list of CD objects
91. """
92. **try**:
93. with open(file\_name, 'rb') as objFile:
94. lst\_CDObjects = pickle.load(objFile)
95. **return** lst\_CDObjects
96. **except** FileNotFoundError:
97. **print**("The file {} could not be loaded".format(file\_name))
98. **return** lst\_CDObjects

101. # -- PRESENTATION (Input/Output) -- #
102. **class** IO:
104. """Handling Input / Output"""
106. @staticmethod
107. **def** print\_menu():
108. """Displays a menu of choices to the user
110. Args:
111. None.
113. Returns:
114. None.
115. """
117. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
118. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
120. **def** menu\_choice():
121. """Gets user input for menu selection
123. Args:
124. None.
126. Returns:
127. choice (string): a lower case string of the users input out of the choices l, a, i, d, s or x
129. """
130. choice = ''
131. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
132. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
133. **print**()
134. **return** choice
136. @staticmethod
137. **def** show\_inventory(lst\_CDObjects):
138. """Displays current inventory table
140. Args:
141. lst\_CDObjects (list of objects): 2D data structure (list of CD objects) that holds the data during runtime.
143. Returns:
144. None.
146. """
147. **print**('======= The Current Inventory: =======')
148. **print**('ID\tCD Title (by: Artist)\n')
150. **for** row **in** lst\_CDObjects:
151. **print**('{}\t{} (by:{})'.format(row.cd\_id, row.cd\_title, row.cd\_artist))
152. **print**('======================================')
154. @staticmethod
155. **def** get\_int\_value(input\_message, error\_message):
156. """ Prompts the user to enter an integer value
158. Args:
159. input\_message (string): String presented to the user when prompting for data
160. error\_message (string): String presented to the user if bad data is entered
162. Returns:
163. int\_id (int): Valid integer supplied by the user
164. """
166. **while** True:
167. **try**:
168. int\_id = int(input(input\_message).strip())
169. **return** int\_id
170. **except** ValueError:
171. **print**(error\_message)
173. @staticmethod
174. **def** get\_new\_cd(listofIDs):
175. """ Gets new CD information from the user
177. Args:
178. listofIDs(list): list containing the IDs in the current inventory
180. Returns:
181. intID (int): ID value for the new CD
182. strTitle (string): Title of the new CD album
183. strArtist (string): Artist of the new CD album
184. """

187. **while** True:
188. **try**:
189. intID = IO.get\_int\_value("Enter ID: ", "This is not an integer value!")
191. **if** intID **in** listofIDs:
192. **raise** IDAlreadyExistsError
193. strTitle = ''
194. **while** strTitle == '':
195. strTitle = input('What is the CD\'s title? ').strip()
196. strArtist = ''
197. **while** strArtist == '':
198. strArtist = input('What is the Artist\'s name? ').strip()
199. **return** intID, strTitle, strArtist
200. **except** IDAlreadyExistsError:
201. **print**('That ID already exists \nEnter new ID')
203. **def** get\_current\_IDs(lst\_CDObjects):
204. """ returns a list of IDs in the current inventory
205. Args:
206. list\_Inventory(list): list of CD objects containing CD information
207. Returns:
208. listofIds(list): list of IDs in the current inventory
209. """
210. listofIds = []
211. **for** rows **in** lst\_CDObjects:
212. listofIds.append(rows.cd\_id)
213. **return** listofIds
215. @staticmethod
216. **def** add\_cd(cd\_id, cd\_title, cd\_artist, lst\_CDObjects):
217. """ function to add a CD to the inventory
218. Tells user to enter an integer if a value that cannot be type cast to
219. integer is entered
221. Args:
222. cd\_id (int): ID value for the new CD
223. cd\_title (string): Title of the new CD album
224. cd\_artist (string): Artist of the new CD album
225. lst\_CDObjects: the list of CDObjects containing the CD entries
227. Returns:
228. the modified list of CD Objects with new etries of CDs
229. """
231. # Add item to the table
232. newCDObj = CD(cd\_id, cd\_title, cd\_artist)
233. lst\_CDObjects.append(newCDObj)
234. **return** lst\_CDObjects
236. @staticmethod
237. **def** delete\_cd(lst\_CDObjects, ID):
238. """ function to delete an entry from the inventory
240. Args:
241. lst\_CDObjects: the list of CD objects containing the CD entries
242. ID: the integer ID of the CD to be deleted
244. Returns:
245. the modified list of CD Objects with the object containing the ID removed
246. """
248. # search thru lst of CD Objects and delete CD
249. intRowNr = -1
250. blnCDRemoved = False
251. **for** row **in** lst\_CDObjects:
252. intRowNr += 1
253. **if** row.cd\_id == ID:
254. **del** lst\_CDObjects[intRowNr]
255. blnCDRemoved = True
256. **break**
257. **if** blnCDRemoved:
258. **print**('The CD was removed')
259. **else**:
260. **print**('Could not find this CD!')
261. **return** lst\_CDObjects
263. **class** IDAlreadyExistsError(Exception):
264. """ the ID already exists"""
265. **def** \_\_str\_\_(self):
266. **return** 'the ID already exists'

269. # -- Main Body of Script -- #
270. lstOfCDObjects = FileIO.load\_inventory(strFileName, lstOfCDObjects)
271. **while** True:
272. IO.print\_menu()
273. strChoice = IO.menu\_choice()
274. **if** strChoice == 'x':
275. **break**
276. **if** strChoice == 'l':
277. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
278. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
279. **if** strYesNo.lower() == 'yes':
280. **print**('reloading...')
281. lstTbl = FileIO.load\_inventory(strFileName, lstOfCDObjects)
282. IO.show\_inventory(lstOfCDObjects)
283. **else**:
284. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
285. IO.show\_inventory(lstOfCDObjects)
286. **continue**
287. **elif** strChoice == 'i':
288. IO.show\_inventory(lstOfCDObjects)
289. **continue**
290. **elif** strChoice == 'a':
291. idList = IO.get\_current\_IDs(lstOfCDObjects)
292. intID, strTitle, strArtist = IO.get\_new\_cd(idList)
293. lstTbl = IO.add\_cd(intID, strTitle, strArtist, lstOfCDObjects)
294. IO.show\_inventory(lstOfCDObjects)
295. **continue**
296. **elif** strChoice == 'd':
297. IO.show\_inventory(lstOfCDObjects)
298. intIDDel = IO.get\_int\_value("Please enter an ID to delete: ", "Please enter only an integer value.")
299. lstOfCDObjects = IO.delete\_cd(lstOfCDObjects, intIDDel)
300. IO.show\_inventory(lstOfCDObjects)
301. **continue**
302. **elif** strChoice == 's':
303. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
304. **if** strYesNo == 'y':
305. FileIO.save\_inventory(strFileName, lstOfCDObjects)
306. **else**:
307. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
308. **continue**
309. **else**:
310. **print**('General Error')