Britta Anson

September 7, 2020

Foundations of Programming: Python

Assignment 08

Modified CD Inventory Script: Working with Classes

# Introduction

This week we learned about object-oriented programming (OOP). We were introduced to creating classes, including constructors, attributes, properties, and methods.

# General background

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

Classes are blueprints for an object, whereas objects made from a class share the features of the class but are distinct and can be changed without making changes to the class or other objects made from that class.

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

A standard class contains fields, constructors, attributes, properties, and methods.[[1]](#footnote-1)

## What is the purpose of a class constructor?

The class constructor allows for pre-population and ensures proper datatypes of fields. Python uses the dunder init method \_\_init\_\_().[[2]](#footnote-2)

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

The keyword “self” is used within a class constructor and within any method within a class in order to represent the reference to the object location being referred to.[[3]](#footnote-3)

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

This decorator indicates a method to be called on a class level (as opposed to an instance level).[[4]](#footnote-4)

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

Fields are the data stores of a class.[[5]](#footnote-5) Attributes are internal variables that hold data.[[6]](#footnote-6) Properties are special methods that allow for controlling validity of values assigned to attributes.[[7]](#footnote-7)

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

Methods are like functions in a script in the way they are called, except a method call also includes a reference to the object it is working on. Properties are special methods that work on attributes.

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

Docstrings are included in a class in order to provide information about the class in a way that is easily referenced by humans and allows other programmers to easily understand the program. IDEs like Spyder can also pull information included in a standardly formatted docstring in order to create built-in documentation of the code which can be accessed using CTRL + I in Spyder.[[8]](#footnote-8)

# Modifying the CD Inventory script

The goal this week was to apply the different class methods we learned this week to rewrite the CD Inventory program.

I started by creating the properties of the ID, title, and artist, modeled on the example of Lab08\_D as Dirk explained it in the September 1 online class. I used setters to check for the correct data type.

A screenshot of text

Description automatically generated

Listing - New property & setters for ID, title, and artist data

I pulled in code for the other functions and script from previous versions of the script from assignments 6 and 7.

Next, I updated the menu item to add data in a way that referenced the new “CD” class using newly created values from the “get\_data” function as arguments.

A screen shot of a computer

Description automatically generated

Listing - Updated "add CD" menu choice

Besides this, most of the code remains similar to previous versions. I’m not entirely sure if it is meant to be dramatically different in other places.

# RUNNING the FINAL code

## Running in Spyder

A screenshot of a cell phone

Description automatically generated

Figure – CD\_Inventory.py running in Spyder console (1 of 3)

A close up of text on a black background

Description automatically generated

Figure – CD\_Inventory.py running in Spyder console (2 of 3)

A screenshot of a cell phone

Description automatically generated

Figure – CD\_Inventory.py running in Spyder console (3 of 3)

## Running in the Terminal

A screenshot of a cell phone

Description automatically generated

Figure - CD\_Inventory.py running in Terminal (1 of 2)

A picture containing bird

Description automatically generated

Figure - CD\_Inventory.py running in Terminal (2 of 2)

# Posted to github

The assignment documents are posted to my GitHub repository “Assignment\_08” at: <https://github.com/bricolagerie/Assignment_08>.

# Summary

This week was very challenging, and I confess I was a bit stumped by this assignment. I feel like I partially solved the assignment, but I have a feeling that it is not entirely what was expected. That said, I was able to successfully create properties of a class and pass in argument values and reference them again in the program.

# Appendix

## Listing CDInventory.py

1. #------------------------------------------#
2. # Title: Assignmen08.py
3. # Desc: Assignnment 08 - Working with classes
4. # Change Log: (Who, When, What)
5. # BAnson, 2020-Sep-07, created file based on pseudocode provided by
6. #   DBiesinger in starter file
7. # BAnson, 2020-Sep-07, added code for class CD, including docstrings
8. #   Borrowed pieces of code from Assignment06 and Assignment07 for FileIO and IO functions
9. #   Rewrote "add data" option to reference CD class properties
10. #------------------------------------------#
12. # -- DATA -- #
13. strFileName = 'cdInventory.txt'
14. lstOfCDObjects = []
16. **class** CD():
17. """Stores data about a CD:
19. properties:
20. cd\_id: (int) with CD ID
21. cd\_title: (string) with the title of the CD
22. cd\_artist: (string) with the artist of the CD
23. methods:
24. \_\_str\_\_: returns string of CD data
26. """
27. # TODid Add Code to the CD class
28. # -- Constructor -- #
29. **def** \_\_init\_\_(self, ID, ttl, art):
30. # -- Attributes -- #
31. self.\_\_ID = ID
32. self.\_\_title = ttl
33. self.\_\_artist = art
35. # -- Properties -- #
36. @property
37. **def** ID(self):
38. **return** self.\_\_ID  # Creates private attribute
40. @ID.setter
41. **def** ID (self, value):
42. """Sets value of ID based on user input and checks value type is integer.
44. Args:
45. value (int): user input ID
47. Returns:
48. ID value as integer; or error message
50. """
51. **if** type(value) == int:
52. self.\_\_ID == value
53. **else**:
54. **raise** Exception('ID needs to be integer')

57. @property
58. **def** title(self):
59. **return** self.\_\_title  # Creates private attribute
61. @title.setter
62. **def** title(self, value):
63. """Sets value of title based on user input and checks value type is string.
65. Args:
66. value: (string) user input title
68. Returns:
69. Title value as string; or error message
71. """
72. **if** type(value) == str:
73. self.\_\_title = value
74. **else**:
75. **raise** Exception('Title needs to be string')

78. @property
79. **def** artist(self):
80. **return** self.\_\_artist  # Creates private attribute
82. @artist.setter
83. **def** artist(self, value):
84. """Sets value of artist based on user input and checks value type is string.
86. Args:
87. value: (string) user input artist
89. Returns:
90. Artist value as string; or error message
92. """
93. **if** type(value) == str:
94. self.\_\_artist = value
95. **else**:
96. **raise** Exception('Artist needs to be string')
98. # -- Methods -- #
99. **def** \_\_str\_\_(self):
100. **return** '{:>2}, {}, {}'.format(self.\_\_ID, self.\_\_title, self.\_\_artist)

103. # -- PROCESSING -- #
104. **class** FileIO:
105. """Processes data to and from file:
107. properties:
109. methods:
110. save\_inventory(file\_name, lst\_Inventory): -> None
111. load\_inventory(file\_name): -> (a list of CD objects)
113. """
114. # TODid Add code to process data from a file
115. @staticmethod
116. **def** load\_inventory(file\_name, lst\_Inventory):
117. """Function to manage data ingestion from file to a list of dictionaries
119. Reads the data from file identified by file\_name into a 2D table
120. (list of dicts) table one line in the file represents one dictionary row in table.
122. Args:
123. file\_name (string) = name of file used to read the data from
124. lst\_Inventory (list of dict) = 2D data structure (list of dicts) that holds the data during runtime
126. Returns:
127. None.
128. """
129. lst\_Inventory.clear()  # this clears existing data and allows to load data from file
130. objFile = open(file\_name, 'r')
131. **for** line **in** objFile:
132. data = line.strip().split(',')
133. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
134. lst\_Inventory.append(dicRow)
135. objFile.close()
137. # TODid Add code to process data to a file
138. @staticmethod
139. **def** save\_inventory(file\_name, lst\_Inventory):
140. """Writes data from inventory table stored in memory to a text file (or creates empty text file if none exists
142. Args:
143. file\_name: name of the file to open and write data to (or create)
144. lst\_Inventory: (list of dicts) 2D data structure that holds the data during runtime.
146. Returns:
147. Text file containing data of current table in memory
148. """
150. objFile = open(file\_name, 'w')
151. **for** row **in** lst\_Inventory:
152. lstValues = list(row.values())
153. lstValues[0] = str(lstValues[0])
154. objFile.write(','.join(lstValues) + '\n')
155. objFile.close()

158. # -- PRESENTATION (Input/Output) -- #
159. **class** IO:
160. # TODid add docstring
161. """Handling Input / Output:
163. properties:
165. methods:
166. print\_menu: -> Displays menu to user
167. menu\_choice: -> None
168. show\_inventory: -> Displays current CD Inventory in memory
169. get\_data: -> (new CD data)
171. """
172. # TODid add code to show menu to user
173. @staticmethod
174. **def** print\_menu():
175. """Displays a menu of choices to the user
177. Args:
178. None.
180. Returns:
181. None.
182. """
184. **print**('\nMenu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
185. **print**('[s] Save Inventory to file\n[x] exit\n')
187. # TODid add code to captures user's choice
188. @staticmethod
189. **def** menu\_choice():
190. """Gets user input for menu selection
192. Args:
193. None.
195. Returns:
196. choice: (string) a lower case sting of the users input out of the choices l, a, i, d, s or x
198. """
199. choice = ' '
200. **while** choice **not** **in** ['l', 'a', 'i', 's', 'x']:
201. choice = input('Which operation would you like to perform? [l, a, i, s or x]: ').lower().strip()
202. **print**()  # Add extra space for layout
203. **return** choice
205. # TODid add code to display the current data on screen
206. @staticmethod
207. **def** show\_inventory(lst\_Inventory):
208. """Displays current inventory table
210. Args:
211. lst\_Inventory: (list of dicts) 2D data structure that holds the data during runtime.
213. Returns:
214. None.
216. """
217. **print**('\n======= The Current Inventory: =======')
218. **print**('ID\tCD Title (by: Artist)\n')
219. **for** row **in** lst\_Inventory:
220. **print**('{}\t{} (by: {})'.format(\*row.values()))
221. **print**('======================================')
223. # TODid add code to get CD data from user
224. @staticmethod
225. **def** get\_data():
226. """Collects three pieces of data from user: ID, title, and artist
228. Args:
229. None
231. Returns:
232. Tuple containing values corresponding to ID (strID), title (strTitle), and artist (strArtist)
234. """
236. strID = input('Enter ID: ').strip()
237. intID = int(strID)
238. strTitle = input('What is the CD\'s title? ').strip()
239. strArtist = input('What is the Artist\'s name? ').strip()
240. tplUserData = (intID, strTitle, strArtist)
241. **return** tplUserData

244. # -- Main Body of Script -- #
245. # TODid Add Code to the main body
246. # Load data from file into a list of CD objects on script start
247. # Display menu to user
248. # show user current inventory
249. # let user add data to the inventory
250. # let user save inventory to file
251. # let user load inventory from file
252. # let user exit program
254. # 1. When program starts, read in the currently saved Inventory, or create empty file
256. **try**:
257. FileIO.load\_inventory(strFileName, lstOfCDObjects)
258. **except**:
259. FileIO.save\_inventory(strFileName, lstOfCDObjects)

262. # 2. start main loop
263. **while** True:
264. # 2.1 Display Menu to user and get choice
265. IO.print\_menu()
266. strChoice = IO.menu\_choice()
268. # 3. Process menu selection
269. # 3.1 process exit first
270. **if** strChoice == 'x':
271. choice = input('Are you sure you want to exit? [y/n]: ') # prevent exiting accidentally
272. **if** choice.lower() == 'y':
273. **break**
274. **else**:
275. **continue**

278. # 3.2 process load inventory
279. **if** strChoice == 'l':
280. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
281. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')
282. **if** strYesNo.lower() == 'yes':
283. **print**('reloading...')
284. FileIO.load\_inventory(strFileName, lstOfCDObjects)
285. IO.show\_inventory(lstOfCDObjects)
286. **else**:
287. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
288. IO.show\_inventory(lstOfCDObjects)
289. **continue**  # start loop back at top.

292. # 3.3 process add a CD
293. **elif** strChoice == 'a':
294. # 3.3.1 Ask user for new ID, CD Title and Artist
295. tplNewCd = IO.get\_data()
296. intID, strTitle, strArtist = tplNewCd
297. newCD = CD(intID, strTitle, strArtist)
299. # # Review entry
300. **print**('\nYou entered: ', newCD)
301. choice = input('Add this data to table? [y/n]: ')
302. **if** choice.lower() == 'y':
303. # 3.3.2 Add item to the table
304. dicRow = {'ID': newCD.ID, 'Title': newCD.title, 'Artist': newCD.artist}
305. **print**(dicRow)
306. lstOfCDObjects.append(dicRow)
307. IO.show\_inventory(lstOfCDObjects)
308. **else**:
309. **print**('Data not added to table.')
310. **continue**  # start loop back at top.

313. # 3.4 process display current inventory
314. **elif** strChoice == 'i':
315. IO.show\_inventory(lstOfCDObjects)
316. **continue**  # start loop back at top.

319. # 3.5 process save inventory to file
320. **elif** strChoice == 's':
321. # 3.6.1 Display current inventory and ask user for confirmation to save
322. IO.show\_inventory(lstOfCDObjects)
323. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
324. # 3.6.2 Process choice
325. **if** strYesNo == 'y':
326. # 3.6.2.1 save data
327. FileIO.save\_inventory(strFileName, lstOfCDObjects) # function call replaces previous code
328. **else**:
329. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
330. **continue**  # start loop back at top.

333. # 3.6 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
334. **else**:
335. **print**('General Error')

1. Dirk Biesinger, “Figure 1 – structure of a class (pseudocode), Module 08 Document, p. 2 [↑](#footnote-ref-1)
2. Ibid, pp. 3-4 [↑](#footnote-ref-2)
3. Dirk Biesinger, “Foundations of Programming (Python) Module 08 Part 2”, <https://www.youtube.com/watch?v=yXldCdMm9vI> (retrieved Sep-07-20). [↑](#footnote-ref-3)
4. Biesinger, Module 08 Doc, p. 13 [↑](#footnote-ref-4)
5. Ibid, p. 3 [↑](#footnote-ref-5)
6. Ibid, p. 5 [↑](#footnote-ref-6)
7. Ibid, p. 6 [↑](#footnote-ref-7)
8. Ibid, p. 16 [↑](#footnote-ref-8)