Tiffany Hou  
IT FDN 100B  
2020 March 16  
Assignment 08

Module 8

# Overview

In this assignment, we recreate the CD Inventory script from scratch using Object Oriented Programming. The primary difference is that instead of using a 2D list to manage our CD data, we use a single list that holds instances of a CD object.

We also continue to explore and solidify our knowledge of classes and programming principles through the use of docstrings and decorators.

# Key Concepts

## Object Oriented Programming

*Object oriented programming* is a methodology where every single thing in a script is an object, each with its own characteristics and capabilities (attributes and methods). *Objects*are an *instance* (concrete occurrence[[1]](#footnote-1)) of a class.

## Classes

A *class* is the structure with all information that an object should contain. It can contain various fields and methods used for managing the data, including (but are not required) the aspects below.

### Attributes

*Attributes* can refer to *class attributes* or *object attributes*, or fields (variables) that contain information about the class as a wholeor about a specific instance*[[2]](#footnote-2).* Every instance within a class shares the same class attributes, but each instance has its own values for its object attributes.

### Methods

*Methods* are like functions within a class. They can refer to a specific instance by having the self keyword (by convention) as the first parameter when being called or be on the class-level.

We can also modify the behavior of a method by using decorators, which are special wrappers that Python uses to modify the behavior of functions[[3]](#footnote-3).

To call a method on a class-level, we use the @staticmethod decorator before defining the method.

### Constructors & Destructors

*Constructors* are special methods that are run during the *instantiation* or creation of an object instance. This is where the object attributes are defined and populated. The constructor can parameterized and have values passed in for the object attributes or have default values[[4]](#footnote-4).

We can make an attribute private by pre-pending a double underscore (\_\_) to its name. This will make it inaccessible except via a *property*, which is a method used to access or modify the attribute (see below).

A screenshot of a cell phone

Description automatically generated

Listing 1 – A parametrized constructor for a CD object with private object attributes

As their name implies, *destructors* are methods that are run during the deletion of an object. There is a default destructor that Python runs, but a developer can add additional code if needed.

### Properties

Since we had previously made our CD object attributes private, we must use *setter (mutator)* and *getter* *(accessor*) properties to modify or fetch data.

As can be seen below, a getter property is defined by adding the @property decorator as a wrapper to the method. Since we’re simply fetching the data, we return the private id attribute of the object.

To modify the private data, we add a decorator comprised of the function name, followed by .setter (@id.setter)[[5]](#footnote-5). For our example, I’ve added a try statement to enforce that any attempt to modify the CD ID must be an integer.

A screenshot of a social media post

Description automatically generated

Listing 2 - getter and setter properties

# The Code

Since the code is getting quite long with docstrings and we’re using concepts/features from previous iterations of the CD inventory script, I will not be adding snippets for everything. Instead, I’ll summarize the main highlights.

As usual, code snippets are formatted using PlanetB’s syntax highlighter[[6]](#footnote-6).

## CD Class

The CD class is our object class that will hold the ID, title, and artist for each CD instance.

In listing 3, we can see the fields are initialized in the parameterized constructor method, filling in the object attributes with the values passed in.

A screenshot of a cell phone

Description automatically generated

Listing 3 - constructor for CD Class

Beyond our constructor method, the only methods in the class are our property method. Each of the 3 object attributes above have their own getter and setter methods, which are needed since our object attributes are private.

As seen in listing 4 below, only @id.setter has any code in it, which enforces the ID needing to be an integer. Although we do not use the setter method for ID at this point (we do not try to modify the ID beyond the instantiation), adding the try except block now makes sure the data type is enforced in case someone adds code to modify it later on.

Both @title.setter and @artist.setter exist, but only contain the pass keyword at this time. However, I’ve chosen to include them so that it is less likely for later developers to forget add enforcements later on.

A screenshot of a social media post

Description automatically generated

Listing 4 - @id.setter

## Class FileIo

Our FileIO class is responsible for functions that process data to and from the CDInventory.txt file. It does not have any class properties, but has two functions: load\_inventory() and save\_inventory().

Bot of the functions directly access the CD class. I spent a lot of time trying to avoid referencing one class from another, but decided that since these are two different kinds of classes (processing and object) and the whole point is to manage CD objects, it would be okay to do so.

### Load\_inventory()

As seen in listing 5, load\_inventory() gets data from our save file and overwrites the current inventory. Since our data is saved in a text file, we first open the file using a with statement. Then using a for loop, we separate and strip the fields in each line, then directly access the CD object class to initialize each CD instance and append the instance to the table, repeating until all lines are read, and return the table.

An if statement is used to provide a little more context to the user. The whole thing is encapsulated in a try except block to handle a FileNotFoundError.

A screenshot of a cell phone

Description automatically generated

Listing 5 - load\_inventory()

### Save\_inventory()

In listing 6, the save\_inventory() function uses a list to format each CD instance’s attributes. Like load\_inventory(), we use a with statement interact with the file and for loop to get each CD instance.

A screenshot of a cell phone

Description automatically generated

Listing 6 - save\_inventory()

## DataProcessor Class

While not outlined in the pseudocode, I decided to add a DataProcessor class to help manage some of the data interactions, using features from previous iterations of the CD Inventory script. As seen in Listing 7, I decided to add a delete\_entry() function (although not asked for) just to get more familiar with manipulating objects.

A screenshot of a cell phone

Description automatically generated

Listing 7 - Class DataProcessor DocString

The only function here that varies in its ideas is the delete\_entry() function, which uses a for loop and if statement to identify the CD entry to be deleted. Both the CD instance and entry in the inventory are deleted. I have found that the script can work properly without deleting the CD instance, but I decided to add that statement anyways to help conserve memory and keep things clean.

I also added error handling for a TypeError since I found that the script will break if someone tries to delete an entry before adding one.

A screenshot of a cell phone

Description automatically generated

Listing 8 - Delete\_entry()

## IO Class

As in previous iterations, the IO class deals with collecting inputs and outputs to and from the user.

A screenshot of a cell phone

Description automatically generated

Listing 9 - IO class docstring

The only function with significant changes is the add\_cd() function, which has been updated to use CD instances to hold data rather than a dictionary or tuple inside the CD inventory 2D-list.

First, we pass in arguments from the main body of the script (the auto-generated CD ID and current inventory list). We then collect the CD title and artist, and create a new CD instance by passing in the ID, title, and artist. The new CD instance is then appended to the table. As noted in the comment of my except statement, I had to use a try except block when appending because an empty table list was being treated as a NoneType object, which I was not able to append to.

I chose to add these in this IO class function since the pseudocode asked us to get the data from the user and add it to the inventory table.

A screenshot of a social media post

Description automatically generated

Listing 10 - add\_cd()

## Main Body

The main body of our script behaves very similarly to previous iterations, built primarily of a while loop with an if-elif-else block nested inside to manage the different menu options.

## GItHub Repository

Repository: <https://github.com/tiffhou/Assignment_08>

## REsults

A screenshot of a social media post

Description automatically generated

Spyder Output 1 - Script Starting up and the iNitial Load

A screenshot of a cell phone

Description automatically generated

Terminal Output 1 - Script starting up in Terminal and the CDInventory.txt file

# Summary

In this module, we embrace the power of OOP by using a CD object class to manage our individual CDs as instances of the class. We learn about how to create instances and how instances can be managed and interacted with through properties and methods.

We also learn about the importance of using decorators to modify Python’s interaction with a method or function and about the importance of docstrings as scripts grow in complexity.

## OUtstanding Question

You’ll notice in my various functions that I’ve added quite a few error handling statements for if my list is empty and has not been populated before.

Despite declaring the list as a global variable and passing it into my functions, I’ve been having issues with certain errors if it hasn’t been populated before at all (I’ve noticed that these errors don’t happen if the list has been populated then emptied).

Thoughts on this would be appreciated.

# Appendix

## Listing – CD\_Inventory.py

1. #------------------------------------------#
2. # Title: CD\_Inventory.py
3. # Desc: Assignnment 08 - Working with classes to create CD Inventory script
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. # THou, 2020-Mar-13: updated header
8. # THou, 2020-Mar-13: added to CD class; added save and load functions;
9. #   added DataProcessor class and functions; added IO functions;
10. #   updated main body of script
11. # THou, 2020-Mar-14: added error handling & formatting
12. # THou, 2020-Mar-15: added formatting, doc strings, tweaked error handling
13. # THou, 2020-Mar-16: added additional error handling; updated docstrings
14. #------------------------------------------#
16. # -- DATA -- #
17. strFileName = 'CDInventory.txt'
18. lstInv = [] #list of CD objects comprising the inventory
19. strChoice = '' #user menu choice
20. strInput = '' #user input
21. intID = ''
22. strTitle = ''
23. strArtist = ''

26. **class** CD:
27. """Stores data about a CD:
29. properties:
30. cd\_id (int): the ID of the CD; private
31. cd\_title (str): the title of the CD; private
32. cd\_artist (str): the artist of the CD; private
33. """
35. #--- Constructor ---#
36. **def** \_\_init\_\_(self, cdid, cdtitle, cdartist):
37. """ Constructor for a CD instance
39. Args:
40. self
41. cdid (int): CD ID
42. cdtitle (str): CD title
43. cdartist (str): CD artist
45. Return:
46. None
47. """
48. self.\_\_id = cdid
49. self.\_\_title = cdtitle
50. self.\_\_artist = cdartist
52. #--- Properties ---#
53. @property
54. **def** id(self):
55. """ Getter for CD ID
57. Args:
58. self
60. Return:
61. \_\_id (str): the private CD ID attribute
62. """
63. **return** self.\_\_id
65. @id.setter
66. **def** id(self, value):
67. """ Setter for the CD ID; enforces the integer type
69. Args:
70. self
72. Raises:
73. ValueError: if ID is not an integer
75. Return:
76. None
77. """
78. **try**:
79. value = int(value)
80. self.\_\_id = value
81. **except** ValueError:
82. **raise** ValueError('ID must be an integer') # raise ValueError to provide more context
84. @property
85. **def** title(self):
86. """ Getter for CD title
88. Args:
89. self
91. Return:
92. \_\_title (str): the private CD title attribute
93. """
94. **return** self.\_\_title
96. @title.setter
97. **def** title(self):
98. """ Setter for CD title; currently empty"""
99. # add enforcement and validation here in future
100. **pass**
102. @property
103. **def** artist(self):
104. """ Getter for CD artist
106. Args:
107. Self
109. Return:
110. \_\_artist (str): the private CD artist attribute
111. """
112. **return** self.\_\_artist
114. @artist.setter
115. **def** artist(self):
116. """ Setter for CD artist; currently empty"""
117. # add enforcement and validation here in future
118. **pass**


122. # -- PROCESSING -- #
123. **class** FileIO:
124. """Processes data to and from file:
126. properties:
127. None
129. Functions:
130. load\_inventory(file\_name): -> (a list of CD objects)
131. save\_inventory(file\_name, lst\_Inventory): -> None
132. """
134. # Code to process data from a file
135. @staticmethod
136. **def** load\_inventory(file\_name):
137. """overwrite the current inventory
139. Args:
140. file\_name (str): name of the file to be loaded from
142. Exceptions:
143. FileNotFoundError: if the file cannot be found
145. Return:
146. table (list): the current inventory table loaded with data from the file
147. """
148. **print**('Loading data from {}\n.\n.\n.\n'.format(strFileName))
149. **try**:
150. with open(file\_name, 'r') as fileObj:
151. table = []
152. **for** line **in** fileObj:
153. data = line.strip().split(',')
154. newCD = CD(int(data[0]), data[1], data[2])
155. table.append(newCD)
156. **if** len(table) > 0:
157. **print**('Inventory loaded from {}'.format(file\_name))
158. **else**:
159. **print**('No data in {} to be loaded.'.format(file\_name))
160. **return** table
161. **except** FileNotFoundError:
162. **print**('{} not found. No data loaded.'.format(file\_name))
164. # Code to process data to a file
165. @staticmethod
166. **def** save\_inventory(file\_name, inventory):
167. """write the current inventory to the save file
169. Args:
170. file\_name (str): the name of the file to save to
171. inventory (list): the current inventory to be written to file
173. Return:
174. """
175. lstValues = []
176. with open(file\_name, 'w') as fileObj:
177. **for** cd **in** inventory:
178. lstValues = [cd.id, cd.title, cd.artist]
179. lstValues[0] = str(lstValues[0])
180. fileObj.write(','.join(lstValues) + '\n')
181. **print**('Inventory saved to {}'.format(file\_name))


185. **class** DataProcessor:
186. """Processes data in memory
188. Properties:
189. None
191. Methods:
192. get\_int(strVal): -> intVal
193. delete\_entry(strID, lstInv): -> table
194. generate\_cd\_id(lstInv): -> cd\_id
195. """
197. @staticmethod
198. **def** get\_int(strVal):
199. """Casts the user input as an integer
201. Args:
202. strVal (str): the value inputted by the user
204. Exceptions:
205. ValueError: if the user input cannot be cast as an integer
207. Returns:
208. intVal (int): the value cast as an integer
209. """
210. **try**:
211. intVal = int(strVal)
212. **return** intVal
213. **except** ValueError:
214. **print**('Input must be an integer')
215. **return** None
217. @staticmethod
218. **def** delete\_entry(delID, table):
219. """Deletes a CD entry based on inputted ID (delID).
221. Args:
222. delID (int): the ID inputted by the user for the entry to be deleted
223. table (list): the current inventory list
225. Raises:
226. TypeError: if the list has not been populated before and is a NoneType
228. Return:
229. table (list): the updated inventory list
230. """
231. intRowNr = -1
232. blnCDRemoved = False
233. **try**:
234. **for** cd **in** table:
235. intRowNr += 1
236. **if** cd.id == delID:
237. **del** cd
238. **del** table[intRowNr]
239. blnCDRemoved = True
240. **break**
241. **if** blnCDRemoved:
242. **print**('The CD was removed.\n')
243. **else**:
244. **print**('Could not find this CD!\n')
245. **except** TypeError:
246. **print**('No entries in table to delete.')
247. **return** table
249. @staticmethod
250. **def** generate\_cd\_id(table):
251. """generates a CD ID, for uniqueness in current inventory
253. Args:
254. table (list): the current inventory table
256. Return:
257. cd\_id (int): the auto-generated CD ID
258. """
259. cd\_id = 1
260. **if** table **is** **not** None:
261. **for** cd **in** table:
262. **if** cd.id == cd\_id:
263. cd\_id += 1
264. **return** cd\_id


268. # -- PRESENTATION (Input/Output) -- #
269. **class** IO:
270. """Presenting outputs and collecting inputs to and from the user
272. Properties:
274. Methods:
275. show\_menu()
276. menu\_choice()
277. show\_inventory(lstInv)
278. add\_cd(intID, lstInv) --> table
279. """
281. # Code to show menu to user
282. @staticmethod
283. **def** show\_menu():
284. """show the menu choices to the user
286. Args:
287. None
289. Return:
290. None
291. """
292. **print**('\n\n[[  CD Inventory Menu  ]]\n')
293. **print**('[i]\tShow current inventory\n[a]\tAdd CD to inventory\n[d]\tDelete CD from inventory\n[s]\tSave inventory to file\n[l]\tLoad inventory from file\n[x]\tExit CD Inventory')
295. # Code to captures user's choice
296. @staticmethod
297. **def** menu\_choice():
298. """get the menu choice from user & checks for valid choi
300. Args:
301. None
303. Return:
304. choice (str): the validated choice
305. """
306. choice = None
307. **while** choice **not** **in** ['i', 'a', 'd', 's', 'l', 'x']:
308. choice = input('Select your option: ').strip().lower()
309. **print**()
310. **return** choice
312. # Code to display the current data on screen
313. @staticmethod
314. **def** show\_inventory(table):
315. """Displays current inventory table.
317. Args:
318. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
320. Raises:
321. TypeError: if the list has not been populated before and is a NoneType
323. Returns:
324. None.
325. """
326. **try**:
327. **if** len(table) > 0:
328. **print**('======= The Current Inventory: =======')
329. **print**('ID\tCD Title (by: Artist)\n')
330. **for** cd **in** table:
331. **print**('{}\t{} (by: {})'.format(cd.id, cd.title, cd.artist))
332. **print**('======================================')
333. **else**:
334. **print**('No entries in inventory table to show.\n')
335. **except** TypeError:
336. **print**('No entries in inventory table to show.\n')

339. # Code to get CD data from user and add to inventory table
340. @staticmethod
341. **def** add\_cd(cd\_id, table):
342. """get the CD fields from the user
344. Args:
345. cd\_id (int): the auto-generated CD ID
347. Exceptions:
348. AttributeError: if the list is a NoneType and cannot be appended to
350. Returns:
351. table (list): the updated inventory list
352. """
353. newCD = ''
354. **print**('CD ID: ', cd\_id)
355. cd\_title = input('CD\'s title: ').strip()
356. cd\_artist = input('CD Artist\'s name: ').strip()
357. newCD = CD(int(cd\_id), cd\_title, cd\_artist)
358. **print**(newCD.id)
359. **try**:
360. table.append(newCD)
361. **except** AttributeError: #can't append if list is None
362. table = [newCD]
363. **print**('\nCD added to inventory.\n')
364. **return** table


368. # -- Main Body of Script -- #
370. # Load data from file into a list of CD objects on script start
371. lstInv = FileIO.load\_inventory(strFileName)
372. **if** lstInv **is** **not** None:
373. IO.show\_inventory(lstInv)

376. **while** True:
377. IO.show\_menu()
378. strChoice = IO.menu\_choice()

381. # Display menu to user
382. # show user current inventory
383. **if** strChoice == 'i':
384. IO.show\_inventory(lstInv)
385. **continue**

388. # let user add data to the inventory
389. **elif** strChoice == 'a':
390. #get unique CD ID
391. intID = DataProcessor.generate\_cd\_id(lstInv)
393. #pass CD ID into add\_cd()
394. lstInv = IO.add\_cd(intID, lstInv)
395. IO.show\_inventory(lstInv)
396. **continue**

399. # let user delete a CD
400. **elif** strChoice == 'd':
401. IO.show\_inventory(lstInv)
402. #get int ID of entry to delete
403. strInput = input('Which ID would you like to delete? ')
404. intID = DataProcessor.get\_int(strInput)
406. #pass ID of entry into delete\_entry()
407. **if** intID **is** **not** None:
408. lstInv = DataProcessor.delete\_entry(intID, lstInv)
409. IO.show\_inventory(lstInv)
410. **continue**  # start loop back at top.

413. # let user save inventory to file
414. **elif** strChoice == 's':
415. strInput = input('Overwrite the data in {}? (\'yes\' to continue)\n'.format(strFileName)).strip().lower()
416. **if** strInput == 'yes':
417. FileIO.save\_inventory(strFileName, lstInv)
418. **continue**

421. # let user load inventory from file
422. **elif** strChoice == 'l':
423. #check if there is data to be overwritten, then load inventory
424. **try**:
425. **if** len(lstInv) > 0:
426. IO.show\_inventory(lstInv)
427. strInput = input('Are you sure you want to overwrite the existing inventory? (enter \'yes\' to continue)\n').strip().lower()
428. **if** strInput == 'yes':
429. lstInv = FileIO.load\_inventory(strFileName)
430. IO.show\_inventory(lstInv)
431. **else**:
432. lstInv = FileIO.load\_inventory(strFileName)
433. IO.show\_inventory(lstInv)
434. **except** TypeError: #if lstInv is NoneType
435. lstInv = FileIO.load\_inventory(strFileName)
436. IO.show\_inventory(lstInv)
437. **continue**

440. # let user exit program
441. **elif** strChoice == 'x':
442. **print**('Thanks for using the CD inventory.')
443. **break**
445. # restarts menu in case of input error
446. **else**:
447. **print**('Invalid selection. Please try again.')
448. **continue**

1. <https://en.wikipedia.org/wiki/Instance_(computer_science)>, accessed 2020-Mar-13 [↑](#footnote-ref-1)
2. <https://medium.com/swlh/class-and-object-attributes-python-8191dcd1f4cf>, <https://www.tutorialsteacher.com/python/python-class>, accessed 2020-Mar-14 [↑](#footnote-ref-2)
3. <https://realpython.com/primer-on-python-decorators/#simple-decorators>, accessed 2020-Mar-14 [↑](#footnote-ref-3)
4. <https://www.geeksforgeeks.org/constructors-in-python/>, accessed 2020-Mar-14 [↑](#footnote-ref-4)
5. <https://www.python-course.eu/python3_properties.php>, accessed 2020-Mar-14 [↑](#footnote-ref-5)
6. <http://www.planetb.ca/projects/syntaxHighlighter/>, accessed 2020-Mar-15 [↑](#footnote-ref-6)