Muthulekshmi Sivathanu

03-07-2021

Foundations of programming python

Assignment-08

Object Oriented Programming

# Introduction

This module deals about classes, objects, Constructor, getters, setters, built-in methods.

# \_\_init\_\_ function

All classes have a function called \_\_init\_\_ . This is always executed when the class is being initiated. In Figure 1 we have not explicitly called the \_\_init\_\_() function , but the function is automatically called , when the object is created . When we create another object for the same class, that object will also use the same blueprint of the class. But it will have its own copy in memory. Changes in one object will not affect other object of the same type. \_\_init\_\_() function is called the constructor in python

Graphical user interface, application

Description automatically generated

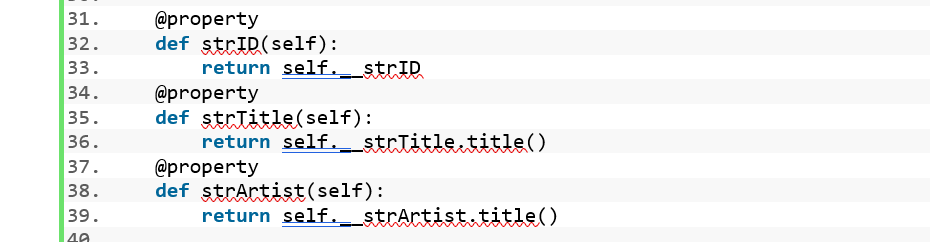
Figure \_\_init\_\_ function initializing variables.

# Properties

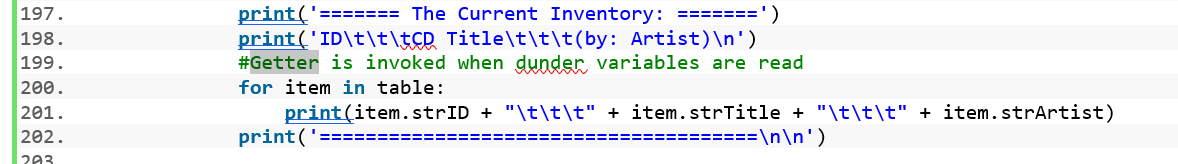
There are some special methods in python called properties. We will create these two properties for each attribute. One property is called getter and the other property is called accessor or setter. Getter property is used for reading a attribute and setter property is for writing an attribute.

## Getter

Getter is called whenever a private variable of a class is accessed. In CDInventory program getter and setter are defined in CD Class which is shown in Listing 1 . Getters are called in FIleIO Class when we store the variables in file as listed in Listing 2



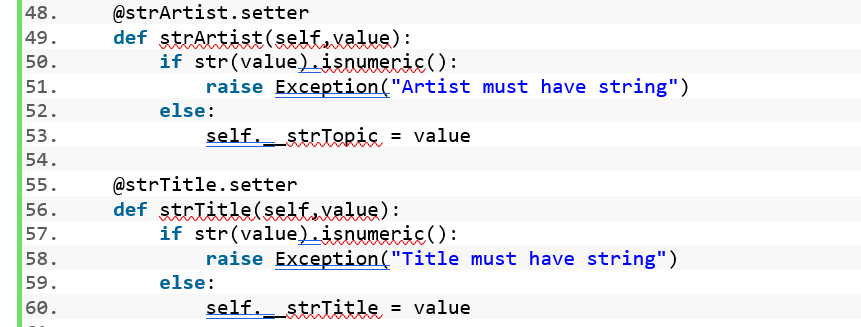
Listing Getter definition in CDInventory program



Listing Getter called in CDInventory program.

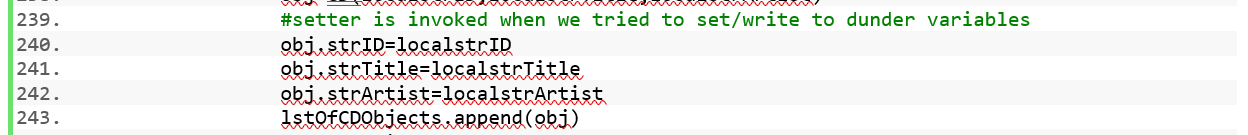
## Setter

Setter is used to define value checks for the private variables inside a class as in Listing 3 below



Listing Setter in CDInventory program

Setter is called when a value is assigned to a private variable. In Listing 4 setter is called when the values are assigned to the class variable



Listing Setter usage in CDInventory.py

## Summary

This module I learnt about classes, objects, Constructor, getters, setters, built-in methods. I used setters and getters in CDInventory program.

## Appendix

CDInventory.py

1. #------------------------------------------#
2. # Title: Assignmen08.py
3. # Desc: Assignnment 08 - Working with classes
4. # Change Log: (Who, When, What)
5. #Muthu , 03-05-2021 , Created CDInventory file with objects
6. #Muthu , 03-06-2021 , updated program to read / write binary file
7. #Muthu , 03-07-2021 , updated docstrings
8. #------------------------------------------#
9. **import** pickle
11. # -- DATA -- #
12. strFileName = 'cdInventory.txt'
13. lstOfCDObjects = []
15. **class** CD:
16. """Stores data about a CD:
18. properties:
19. strId: (int) with CD ID
20. strTitle: (string) with the title of the CD
21. strArtist: (string) with the artist of the CD
22. methods:
24. """
26. **def** \_\_init\_\_(self,localID,localTitle,localArtist):
27. self.\_\_strID = localID
28. self.\_\_strArtist=localArtist
29. self.\_\_strTitle=localTitle
31. @property
32. **def** strID(self):
33. **return** self.\_\_strID
34. @property
35. **def** strTitle(self):
36. **return** self.\_\_strTitle.title()
37. @property
38. **def** strArtist(self):
39. **return** self.\_\_strArtist.title()
41. @strID.setter
42. **def** strID(self,value):
43. **if** **not** str(value).isnumeric():
44. **raise** Exception("ID must be an integer")
45. **else**:
46. self.\_\_strID = value
48. @strArtist.setter
49. **def** strArtist(self,value):
50. **if** str(value).isnumeric():
51. **raise** Exception("Artist must have string")
52. **else**:
53. self.\_\_strTopic = value
55. @strTitle.setter
56. **def** strTitle(self,value):
57. **if** str(value).isnumeric():
58. **raise** Exception("Title must have string")
59. **else**:
60. self.\_\_strTitle = value
62. # -- PROCESSING -- #
63. **class** FileIO:
64. """Processes data to and from file:
66. properties:
68. methods:
69. save\_inventory(file\_name, lst\_Inventory): -> None
70. load\_inventory(file\_name): -> (a list of CD objects)
72. """
74. @staticmethod
75. **def** load\_inventory(strFileName):
76. """Function to manage data ingestion from file to a list of dictionaries
78. Reads the data from file identified by file\_name into a 2D table
79. (list of dicts) table one line in the file represents one dictionary row in table.
81. Args:
82. file\_name (string): name of file used to read the data from
83. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
85. Returns:
86. None.
87. """
88. locallstOfCDObjects=[]
89. **try**:
90. with open(strFileName, 'rb') as file\_handle:
91. locallstOfCDObjects.clear()
92. lstoflst = pickle.load(file\_handle)
93. **for** data **in** lstoflst:
94. obj=CD(data[0],data[1],data[2])
95. locallstOfCDObjects.append(obj)
96. **except**:
97. **print**("Error opening file, creating new database")
98. with open(strFileName, 'wb') as file\_handle:
99. **pass**
101. **return**(locallstOfCDObjects)
103. @staticmethod
104. **def** save\_inventory(strFileName,lstOfCDObjects):
105. """Function to write data from list of dictionaries to a file
107. Reads the data from 2D table row by row
108. Create a list with values from the key value pair of each row
109. Join the contents of this list with a "," seperator
110. Write the list to the file
112. Args:
113. file\_name (string): name of file used to write the data
114. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
116. Returns:
117. None.
118. """
120. lstTable=[]
121. lstValues=[]
122. **for** item **in** lstOfCDObjects:
123. lstValues=[]
124. lstValues.append(str(item.strID))
125. lstValues.append(item.strTitle)
126. lstValues.append(item.strArtist)
127. lstTable.append(lstValues)
128. with open(strFileName,'wb') as objFile:
129. pickle.dump(lstTable,objFile)


133. **pass**
135. # -- PRESENTATION (Input/Output) -- #
136. **class** IO:
137. """Add/Delete new CD Data entered by user to/from the table
138. Print menu . capture users choice , show inventory
139. methods :
140. print\_menu () -> None
141. menu\_choice() -> returns choice
142. show\_inventory(listofCDObjects) -> None
143. ask\_details() -> returns id , title and artist
144. add\_cd(id,title,artist) --> None
145. delete\_cd(id,listofCDObjects) --> status
146. """
147. @staticmethod
148. **def** print\_menu():
149. """Displays a menu of choices to the user
151. Args:
152. None.
154. Returns:
155. None.
156. """
158. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
159. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')

162. @staticmethod
163. **def** menu\_choice():
164. """Gets user input for menu selection
166. Args:
167. None.
169. Returns:
170. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
172. """
173. **try**:
174. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
175. **if** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
176. **raise** Exception('invalid choice')
177. **return** choice
179. **except** Exception as e:
180. **print**('built-in error info')
181. **print**(type(e),e,e.\_\_doc\_\_,sep='\n')


185. @staticmethod
186. **def** show\_inventory(table):
187. """Displays current inventory table

190. Args:
191. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
193. Returns:
194. None.
196. """
197. **print**('======= The Current Inventory: =======')
198. **print**('ID\t\t\tCD Title\t\t\t(by: Artist)\n')
199. #Getter is invoked when dunder variables are read
200. **for** item **in** table:
201. **print**(item.strID + "\t\t\t" + item.strTitle + "\t\t\t" + item.strArtist)
202. **print**('======================================\n\n')

205. @staticmethod
206. **def** ask\_details():
207. """Ask 3 different data from user ( ID , title , artist)
209. Args:
210. None.
212. Returns:
213. CD ID , CD title and Artist name
214. """
216. idfromuser = input('Enter ID: ').strip()
217. titlefromuser = input('What is the CD\'s title? ').strip()
218. artistfromuser = input('What is the Artist\'s name? ').strip()
219. **return**(idfromuser,titlefromuser,artistfromuser)
221. @staticmethod
222. **def** add\_cd(localstrID,localstrTitle,localstrArtist):
223. """Function to add data entered by user to the 2D table
225. Write CD Details as a dictionary entry .
226. Add this dictionary to the 2D list .
228. Args:
229. strID (string ): ID for the CD entered by user
230. strTitle (string ) : Title for the CD entered by user
231. strArtist ( String ) : Artist name for the CD entered by user
233. Returns:
234. None.
235. """
237. **try**:
238. obj=CD(localstrID,localstrTitle,localstrArtist)
239. #setter is invoked when we tried to set/write to dunder variables
240. obj.strID=localstrID
241. obj.strTitle=localstrTitle
242. obj.strArtist=localstrArtist
243. lstOfCDObjects.append(obj)
244. **except** Exception as e:
245. **print**('Error from class CD Setter :')
246. **print**(type(e),e,e.\_\_doc\_\_,sep='\n')
248. @staticmethod
249. **def** delete\_CD(intIDDel,lstOfCDObjects):
250. """Function to delete a row identified by user from 2D table
252. Checks if there is a match between the ID entered by the user and the ID present in the list .
253. If there is a match delete the row that matches the ID .
254. Display the latest content in the table .
256. Args:
257. intIDDel (Integer) : The CD ID entered by user
258. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
260. Returns:
261. Status of CD removal.
262. """
263. intRowNr = -1
264. blnCDRemoved = False
265. **for** item **in** lstOfCDObjects:
266. intRowNr += 1
267. **if** int(item.strID) == intIDDel:
268. **del** lstOfCDObjects[intRowNr]
269. blnCDRemoved = True
270. **break**
271. **return**(blnCDRemoved)

274. # -- Main Body of Script -- #
276. # Load data from file into a list of CD objects on script start
277. # 1. When program starts, read in the currently saved Inventory
279. lstOfCDObjects=FileIO.load\_inventory(strFileName)
281. # 2. start main loop
282. **while** True:
283. # 2.1 Display Menu to user and get choice
285. IO.print\_menu()
286. strChoice = IO.menu\_choice()
288. # 3. Process menu selection
289. # 3.1 process exit first

292. **if** strChoice == 'x':
293. **break**
295. # 3.2 process load inventory
296. **if** strChoice == 'l':
297. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
298. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled : ')
299. **if** strYesNo.lower() == 'yes':
300. **print**('reloading...')
301. lstOfCDObjects=FileIO.load\_inventory(strFileName)
302. **else**:
303. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu: ')
304. IO.show\_inventory(lstOfCDObjects)
305. **continue**

308. # 3.3 process add a CD
309. **elif** strChoice == 'a':
310. # 3.3.1 Ask user for new ID, CD Title and Artist
311. ID,title,artist=IO.ask\_details()
313. # 3.3.2 Add item to the table
314. IO.add\_cd(ID,title,artist)


318. IO.show\_inventory(lstOfCDObjects)
319. **continue**  # start loop back at top.
321. # 3.4 process display current inventory
322. **elif** strChoice == 'i':
323. IO.show\_inventory(lstOfCDObjects)
324. **continue**  # start loop back at top.
326. # 3.5 process delete a CD
327. **elif** strChoice == 'd':
328. # 3.5.1 get Userinput for which CD to delete
329. # 3.5.1.1 display Inventory to user
330. IO.show\_inventory(lstOfCDObjects)
331. # 3.5.1.2 ask user which ID to remove
333. # Need to error trap this as it can still crash the program
334. **try**:
335. intIDDel = int(input('Which ID would you like to delete? ').strip())
336. # 3.5.2 search thru table and delete CD
337. status=IO.delete\_CD(intIDDel,lstOfCDObjects)
338. **if** status:
339. **print**('The CD was removed\n\n')
340. **else**:
341. **print**('Could not find this CD!')
342. **continue**  # start loop back at top.
343. **except** ValueError:
344. **print**("ID to remove must be an integer")
346. # 3.6 process save inventory to file
347. **elif** strChoice == 's':
348. # 3.6.1 Display current inventory and ask user for confirmation to save
349. IO.show\_inventory(lstOfCDObjects)
350. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
351. # 3.6.2 Process choice
352. **if** strYesNo == 'y':
353. # 3.6.2.1 save data
354. FileIO.save\_inventory(strFileName, lstOfCDObjects)
355. **else**:
356. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
357. **continue**  # start loop back at top.
359. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:

Text

Description automatically generated

Figure Spyder output of CDInventory program

Text

Description automatically generated

Figure Terminal output of CDInventory program