Jurg Huerlimann

Sep. 2 2020

IT FDN 110 B SU 20: Foundations of Programming: Python

Assignment Module 08

Introduction to Object Oriented Programming, Classes, Objects, Fields Constructors, Attributes and Methods

# Introduction

In the eight module and assignment of this course, we were introduced to the art of Object Oriented Programming (OOP).

Object-oriented programming is a method of structuring a program by bundling related properties and behaviors into individual objects than can be called individually in the code. These objects act like the blue-prints for specific functions, which then can be called and individualized and configured.   
We also learned about all the components that define a class, including Fields, Constructors, Attributes, Properties and Methods.

Questions we had to investigate and answer:

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

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with *actual values*.

An object consists of :

* **State :** It is represented by attributes of an object. It also reflects the properties of an object.
* **Behavior :** It is represented by methods of an object. It also reflects the response of an object with other objects.
* **Identity :** It gives a unique name to an object and enables one object to interact with other objects.[[1]](#footnote-1)

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

The components that define a class are Fields, Constructors, Attributes, Properties and Methods.

## What is the purpose of a class constructor?

A constructor is a special type of method (function) which is used to initialize the instance members of the class.

Constructors can be of two types.

1. Parameterized Constructor
2. Non-parameterized Constructor

Constructor definition is executed when we create the object of this class. Constructors also verify that there are enough resources for the object to perform any start-up task.

In Python, the method the **\_\_init\_\_()** simulates the constructor of the class. This method is called when the class is instantiated. It accepts the **self**-keyword as a first argument which allows accessing the attributes or method of the class.

We can pass any number of arguments at the time of creating the class object, depending upon the **\_\_init\_\_()** definition. It is mostly used to initialize the class attributes. Every class must have a constructor, even if it simply relies on the default constructor.[[2]](#footnote-2)

When do you use the keyword "self?

Python self variable is used to bind the instance of the class to the instance method. We have to explicitly declare it as the first method argument to access the instance variables and methods. This variable is used only with the instance methods.[[3]](#footnote-3)

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

Static method can be called without creating an object or instance. Simply create the method and call it directly. This is in a sense orthogonal to object orientated programming: we call a method without creating objects.[[4]](#footnote-4)

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

Fields, attributes and property functions are all used the define the behavior of a class.   
Fields are the data stores of a class, whereas attributes are internal fields or variables that hold data.

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

In Python, the main purpose of Property() function is to create a property of a class.[[5]](#footnote-5) Where as a method is associated to an object (dependent). A method is implicitly passed the object on which it is invoked.

A method can operate on the data (instance variables) that is contained by the corresponding class.

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

A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the \_\_doc\_\_ special attribute of that object.[[6]](#footnote-6)

The docstring is included in class to document what the function of the class is, and the arguments that are being passed. Docstrings are being defined by using triple double quotes at the beginning and end of the text. Docstrings are being indented by 4 characters.

# Assignment08: Modify Python Script to Object Oriented functionality

In this eight assignment we were tasked to modify the Assignment\_08\_Starter.py script, adding code to show a menu to user, capture the user's choice, display the current data on screen and code to get CD data from user, as well as writing/reading data from/to data file, using classes and all we learned so far.

## Resulting script:

The listing of the resulting script can be found in the Appendix. It is the result of doing the lab exercises in Module 08, reading up on the commands and methods, experimenting and last but not least, feedback from the class forum, and the usual expert advice from Dirk and Doug.

## Testing script in Spyder:

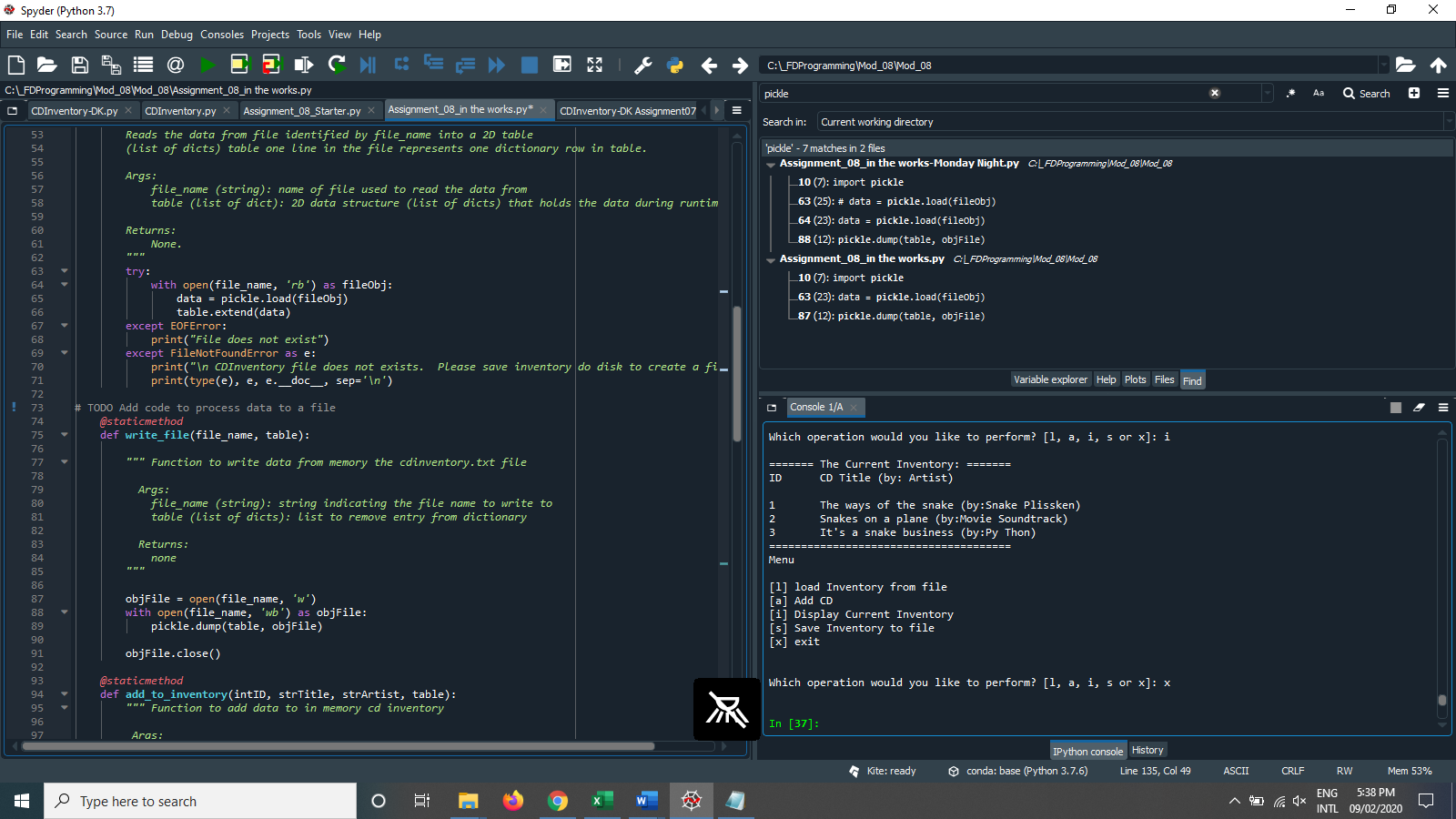


Figure 2 Script running in Spyder

## Testing script in Terminal:

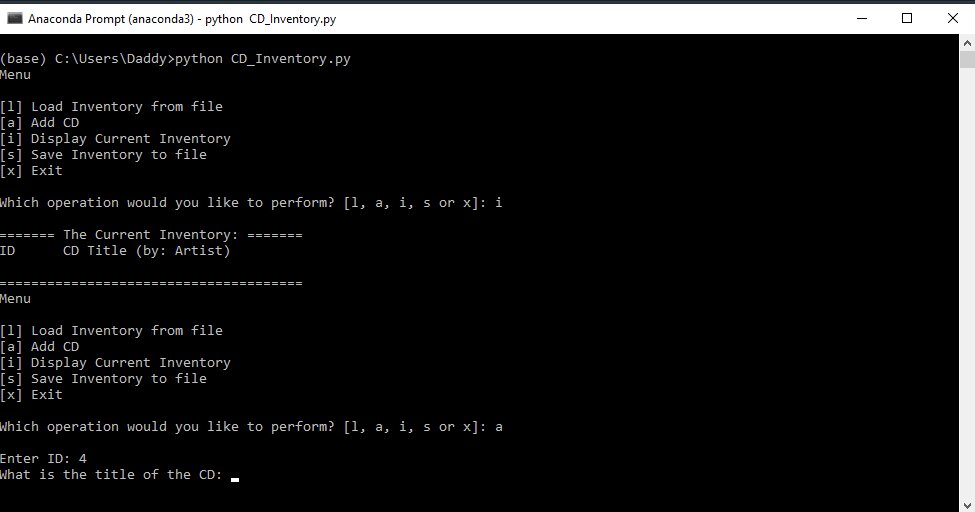


Figure 3 Script running in Terminal

# Summary

In this eighth assignment I ran into some issues with reading from the data file, couldn’t get the code to do what I needed. Something with the variables I tried to pass didn’t work out. I was getting this error:   
  
File "C:\\_FDProgramming\Mod\_08\Mod\_08\Assignment\_08\_in the works.py", line 177, in <module>

load = FileIO.read\_file()

TypeError: read\_file() missing 2 required positional arguments: 'file\_name' and 'table'

All the other functions work, and this time I am writing the data to a binary CD\_Inventory.dat file.

The description of the assignment was not very clear if we had to convert the code to OOP or only implement the code using classes. I opted for the latter, as the OOP part is still way too fuzzy.

These kinds of assignments are way to time intensive to being able to spend the necessary time and effort during one week, to come up with high-quality code with the little experience most of us have.

# Posting to GitHub:

The CDInventory.py script and knowledge document can be found on GitHub under this link:  
  
<https://github.com/jlhuerlimann/Assignment_08>

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. # Jurg Huerlimann 2020-Aug-31 added get\_CD\_info and FileIO functions
8. # Jurg Huerlimann 2020-Sep-02 added menu functions and writing to file
9. # Jurg Huerlimann 2020-Sep-02 added docstrings and pickle function
10. #------------------------------------------#
12. **import** pickle    #importing pickle to create a binary data file
14. # -- DATA -- #
15. strFileName = 'cdInventory.dat'
16. lstOfCDObjects = []
18. #@staticmethod
19. **class** CD:
20. **def** add\_cdinfo():
21. """Stores data about a CD:
23. properties:
24. cdID: (int) with CD ID
25. cdTitle: (string) with the title of the CD
26. cdArtist: (string) with the artist of the CD
27. methods:
29. """
31. cdID = int(input('Enter ID: ').strip())
32. cdTitle = input('What is the title of the CD: ').strip()
33. cdArtist = input('What is the name of the Artist: ').strip()
35. **return** cdID, cdTitle, cdArtist
37. # -- PROCESSING -- #
38. **class** FileIO:
39. """Processes data to and from file:
41. properties:
43. methods:
44. save\_inventory(file\_name, lst\_Inventory): -> None
45. load\_inventory(file\_name): -> (a list of CD objects)
47. """
48. # TODO Add code to process data from a file
49. @staticmethod
50. **def** read\_file(file\_name, table):
51. """Function to manage data ingestion from file to a list of dictionaries
53. Reads the data from file identified by file\_name into a 2D table
54. (list of dicts) table one line in the file represents one dictionary row in table.
56. Args:
57. file\_name (string): name of file used to read the data from
58. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
60. Returns:
61. None.
62. """
63. **try**:
64. with open(file\_name, 'rb') as fileObj:
65. data = pickle.load(fileObj)
66. table.extend(data)
67. **except** EOFError:
68. **print**("File does not exist")
69. **except** FileNotFoundError as e:
70. **print**("\n CDInventory file does not exists.  Please save inventory do disk to create a file.\n")
71. **print**(type(e), e, e.\_\_doc\_\_, sep='\n')
73. # TODO Add code to process data to a file
74. @staticmethod
75. **def** write\_file(file\_name, table):
77. """ Function to write data from memory the cdinventory.txt file
79. Args:
80. file\_name (string): string indicating the file name to write to
81. table (list of dicts): list to remove entry from dictionary
83. Returns:
84. none
85. """
87. objFile = open(file\_name, 'w')
88. with open(file\_name, 'wb') as objFile:
89. pickle.dump(table, objFile)
91. objFile.close()
93. @staticmethod
94. **def** add\_to\_inventory(intID, strTitle, strArtist, table):
95. """ Function to add data to in memory cd inventory
97. Args:
98. intID (string): string indicating ID of the new CD
99. strTitle (string): string indicating Title of the new CD
100. strArtist (string): string indicating Artist of the new CD
101. table (list of dicts): list to append new dictionary entry to
103. Returns:
104. table (list of dicts): list of dictionaries with the new entry appended.
105. """
106. table.append({"ID": intID, "Title": strTitle, "Artist": strArtist})
108. # -- PRESENTATION (Input/Output) -- #
109. **class** IO:
111. @staticmethod
112. **def** menu\_choice():
113. """Gets user input for menu selection
114. Args:
115. None.
116. Returns:
117. choice (string): a lower case sting of the users input out of the ch oices l, a, i, s or x
118. """
119. choice = ' '
120. **while** choice **not** **in** ['l', 'a', 'i', 's', 'x']:
121. choice = input('Which operation would you like to perform? [l, a, i, s or x]: ').lower().strip()
122. **print**()  # Add extra space for layout
123. **return** choice
125. @staticmethod
126. **def** print\_menu():
127. """Displays a menu of choices to the user
128. Args:
129. None.
130. Returns:
131. None.
132. """
134. **print**('Menu\n\n[l] Load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
135. **print**('[s] Save Inventory to file\n[x] Exit\n')
137. @staticmethod
138. **def** show\_inventory(table):
139. """Displays current inventory table
140. Args:
141. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
142. Returns:
143. None.
144. """
145. **print**('======= The Current Inventory: =======')
146. **print**('ID\tCD Title (by: Artist)\n')
147. **for** row **in** table:
148. **print**('{}\t{} (by:{})'.format(\*row.values()))
149. **print**('======================================')
150. # -- Main Body of Script -- #
152. **while** True:
153. IO.print\_menu()
154. strChoice = IO.menu\_choice()
156. **if** strChoice == 'x':
157. **break**
159. **if** strChoice == 'l':
160. load = FileIO.read\_file()
161. **if** load:
162. FileIO.read\_file(strFileName, lstOfCDObjects)
163. **else**:
164. input('\ncanceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
165. **continue**  # start loop back at top.
167. **elif** strChoice == 'a':
168. cd\_id, cd\_title, cd\_artist = CD.add\_cdinfo()
169. FileIO.add\_to\_inventory(cd\_id, cd\_title, cd\_artist, lstOfCDObjects)
170. IO.show\_inventory(lstOfCDObjects)
171. **continue**  # start loop back at top.
173. **elif** strChoice == 'i':
174. IO.show\_inventory(lstOfCDObjects)
175. **continue**  # start loop back at top.

178. **elif** strChoice == 's':
179. IO.show\_inventory(lstOfCDObjects)
180. FileIO.write\_file(strFileName, lstOfCDObjects)
181. **continue**  # start loop back at top.
183. **else**:
184. **print**('General Error')

Listing 3 Code for CDInventory.py

1. Source: <https://www.geeksforgeeks.org/python-classes-and-objects/> 2020-Sep-01 [↑](#footnote-ref-1)
2. Source: <https://www.javatpoint.com/python-constructors> 2020-Sep-01 [↑](#footnote-ref-2)
3. Source: <https://www.askpython.com/python/python-self-variable> 2020-Sep-01 [↑](#footnote-ref-3)
4. Source: <https://pythonbasics.org/static-method/> 2020-Sep-01 [↑](#footnote-ref-4)
5. Source: <https://www.geeksforgeeks.org/python-property-function> 2020-Sep-02 [↑](#footnote-ref-5)
6. Source: <https://www.python.org/dev/peps/pep-0257/#what-is-a-docstring> 2020-Sep-01 [↑](#footnote-ref-6)