Brent Kieszling

9/03/2020

IT FDN 110: Introduction to Programming (Python)

Module 8 Learning Document

# Introduction

In this paper I will cover the standout points of Module 8. Some of the topics to expect are:object oriented programming, constructors, attributes, properties, methods, and dunder string. Additionally, I will discuss notes on the development of the program I am submitting for Assignment 8.

# Lessons

Object oriented programming (OOP) is the concept of building a generic blue print for an object and then working with that object in the code. Consider a car manufacturing facility: the facility is able to make many copies of a car based on the car’s blueprint. Some things are common to all of the cars; they all need a frame, a VIN, an engine, and wheels. However, we would run into problems if everyone’s car had the same VIN. So, we while they all generically need a VIN each car needs its’ own number. Figure 1 below shows an example of this in code.



Figure 1 - OOP Building Cars

Class ‘TrackInfo()’ is the blueprint for our cars. Each car has the same generic information seen under attributes (VIN, color, and transmission). A new car is created each time ‘TrackInfo()’ is called. Each of these cars, ‘objCar1’ and ‘objCar2’, have their own information for each of the attributes. An upside to this style of programming is that scaling the number of cars to “infinity and beyond” becomes simple.

‘TrackInfo()’ has a field called Constructor. A constructor is a function that executes when the object is initially created. This constructor has four arguments it expects. Self, which represents the instance of the object (objCar1). ‘int1’, a number for the VIN. ‘msg2’ the color of the car. ‘msg3’ the type of transmission. The constructor then assigns the passed in arguments to their respective attributes.

Attributes can be made private to the object as well. This is done by adding two underscores before the attribute. Figure 2 below shows an example of this.

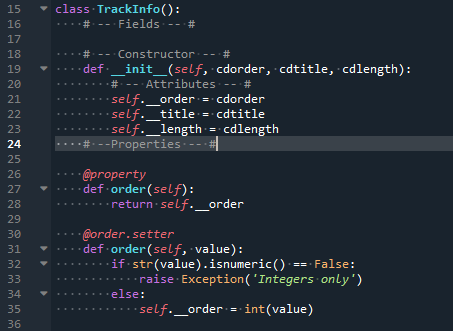
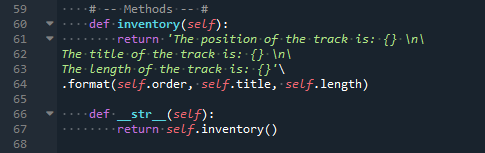


Figure 2 - Excerpt from Lab08\_D

When attributes are established as private they are not accessible outside of the function. The next part of the blueprint is properties. Properties are methods used to either set or get an attribute. In order to work correctly the getter must be defined before the setter. In Figure 2 the getter for the private attribute ‘order’ is defined in lines 26-28 and the setter is lines 30-35. The setter contains exception handling to ensure the correct type of information is being used.

Methods are like actions the object can perform. Figure 3 shows a class with a method that returns the stored attributes of the object, see Appendix A.5 for full code.



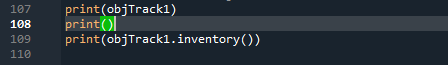


Figure 3 - Excerpt from Lab08\_E

There are two methods being shown above. Line 109 shows an example of the first method ‘inventory(self)’ being called for ‘objTrack1’. The second method ‘\_\_str\_\_(self)’ overwrites Python’s default dunder string for an object. The default dunder string would refer to the object’s location in memory. Line 107 shows the new dunder string in action. Figure 4 shows the result of both of these calls.

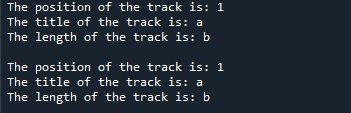


Figure 4 - Example Output from Lab08\_E

# Program: CD Inventory Using Objects

We were tasked with updating a program that would enable someone to save a digital record of their CDs, load the record from a file, and save new data. The code for this can be seen in Appendix B.1. The required updates were: make a class to create individual CD objects, store the CD objects in a list, save the list to a binary file, add functionality to load saved data, and create a menu for the user to interact with.

To build the CD class I established setters and getters for each attribute (collection ID, CD title, and artist). Then I added methods to display the attributes and overwrite the default dunder string. I utilized ‘pickling’ for file input and output processing. I was able to use the same code from the previous assignment with minimal changes to accomplish this. For the menu, I created a static method in the IO class to display the menu. I used a second static method to get and return the user’s selection. The user’s selection is used to navigate an ‘if/elif’ statement until the correct category is found. If ‘x’ is selected the program terminates. If ‘l’ is selected, the user is warned that loading data will overwrite the current file and they are given the option to stop. This is controlled by static method ‘load\_check()’. If the user continues then the active list is updated with the saved list. If a is selected ‘IO.new\_cd()’ gets the user input for the CD information and then this information is used to create a new CD object. The new CD object is then added to the active list. I displays the attributes of each CD object in the active list. S, saves the active list to a binary file. Finally error handling was added.

# Summary

I was successful in updating the CD Inventory program. I enjoyed learning how create objects from blue prints. It feels like object oriented programming is the bases for individual character creation in video games. I’m excited to make objects that can do more complex things.

# Appendix A.1

1. #-------------------------------------------#
2. #Tittle: Lab08\_A, IT FDN 110
3. #Desc: Creating a CD with classes - fields
4. #Change Log: (Who, When, What)
5. #Brent Kieszling, 2020, Aug, 28, created file
6. #-------------------------------------------#
8. #DATA---------------------------------------



13. #PROCESS------------------------------------
15. **class** TrackInfo():
16. # -- Fields -- #
17. order = None
18. title = ''
19. length = ''


23. #PRESENTATION INPUT/OUTPUT------------------

26. objTrack1 = TrackInfo()
27. #get order
28. **while** True:
29. a = input('What order is this track on the CD? ')
30. **try**:
31. # This alows me to store the information as an interger vice string
32. trackOrder = int(a)
33. **break**
34. **except**:
35. **print**('Please provide a numerical value.')
36. objTrack1.order = trackOrder
38. #get track name
39. objTrack1.title = input('What is the name of this track? ')
41. #get length of the track
42. objTrack1.length = input('What is the legnth of the track (MM:SS) ')
44. **print**('The position of the track **is**: {} \n\
45. The title of the track **is**: {} \n\
46. The length of the track **is**: {}'\
47. .format(objTrack1.order, objTrack1.title, objTrack1.length))

# Appendix A.2

1. #-------------------------------------------#
2. #Tittle: Lab08\_B, IT FDN 110
3. #Desc: Creating a CD with classes - constructor
4. #Change Log: (Who, When, What)
5. #Brent Kieszling, 2020, Aug, 28, created file
6. #-------------------------------------------#
8. #DATA---------------------------------------



13. #PROCESS------------------------------------
15. **class** TrackInfo():
16. # -- Fields -- #
17. order = ''
18. title = ''
19. length = ''
21. # -- Constructor -- #
22. **def** \_\_init\_\_(self, msg1, msg2, msg3):
23. # -- Attributes -- #
24. self.order = msg1
25. self.title = msg2
26. self.length = msg3

29. #PRESENTATION INPUT/OUTPUT------------------

32. objTrack1 = TrackInfo(1, 'Over the Rainbow', '03:22')

35. **print**('The position of the track **is**: {} \n\
36. The title of the track **is**: {} \n\
37. The length of the track **is**: {}'\
38. .format(objTrack1.order, objTrack1.title, objTrack1.length))

# Appendix A.3

1. #-------------------------------------------#
2. #Tittle: Lab08\_C, IT FDN 110
3. #Desc: Creating a CD with classes - constructor, just attributes
4. #Change Log: (Who, When, What)
5. #Brent Kieszling, 2020, Aug, 28, created file
6. #-------------------------------------------#
8. #DATA---------------------------------------



13. #PROCESS------------------------------------
15. **class** TrackInfo():
16. # -- Fields -- #
17. #order = ''
18. #title = ''
19. #length = ''
21. # -- Constructor -- #
22. **def** \_\_init\_\_(self, int1, msg2, msg3):
23. # -- Attributes -- #
24. self.order = int1
25. self.title = msg2
26. self.length = msg3
28. **class** IO:
29. """Handling Input / Output"""
30. **def** new\_cd():
31. """Allows the user to add a CD to the active inventory table
33. Args:
34. None.
36. Returns:
37. intID (interger): Serialized CD order
38. strTitle (string): Tittle of track
39. strLength (string): Length of CD (MM:SS)
41. """
42. strID = input('Enter ID: ').strip()
43. **while** True:
44. #This try handles the case where a non interger is entered
45. **try**:
46. intID = int(strID)
47. **break**
48. **except**:
49. strID = input('Please enter an interger for the ID. ')
50. strTitle = input('What is the tracks\'s name? ').strip()
51. strLength = input('How long is the track (MM:SS)? ').strip()
52. **return** intID, strTitle, strLength
54. #PRESENTATION INPUT/OUTPUT------------------
56. newCDOrder, newCDTitle, newCDLength = IO.new\_cd()
57. objTrack1 = TrackInfo(newCDOrder, newCDTitle, newCDLength)


61. **print**('The position of the track **is**: {} \n\
62. The title of the track **is**: {} \n\
63. The length of the track **is**: {}'\
64. .format(objTrack1.order, objTrack1.title, objTrack1.length))

# Appendix A.4

1. #-------------------------------------------#
2. #Tittle: Lab08\_D, IT FDN 110
3. #Desc: Creating a CD with classes - setters and getters
4. #Change Log: (Who, When, What)
5. #Brent Kieszling, 2020, Aug, 28, created file
6. #-------------------------------------------#
8. #DATA---------------------------------------



13. #PROCESS------------------------------------
15. **class** TrackInfo():
16. # -- Fields -- #
18. # -- Constructor -- #
19. **def** \_\_init\_\_(self, cdorder, cdtitle, cdlength):
20. # -- Attributes -- #
21. self.\_\_order = cdorder
22. self.\_\_title = cdtitle
23. self.\_\_length = cdlength
24. # --Properties -- #
26. @property
27. **def** order(self):
28. **return** self.\_\_order
30. @order.setter
31. **def** order(self, value):
32. **if** str(value).isnumeric() == False:
33. **raise** Exception('Integers only')
34. **else**:
35. self.\_\_order = int(value)
37. @property
38. **def** title(self):
39. **return** self.\_\_title
41. @title.setter
42. **def** title(self, value):
43. **if** value.isstring():
44. self.\_\_title = value
45. **else**:
46. **raise** Exception('Not a valid input')
48. @property
49. **def** length(self):
50. **return** self.\_\_length
52. @length.setter
53. **def** length(self, value):
54. **if** value.isstring():
55. self.\_\_length = value
56. **else**:
57. **raise** Exception('Not a valid input')
59. #PRESENTATION INPUT/OUTPUT------------------
61. **class** IO:
62. """Handling Input / Output"""
63. **def** new\_cd():
64. """Allows the user to add a CD to the active inventory table
66. Args:
67. None.
69. Returns:
70. intID (interger): Serialized CD order
71. strTitle (string): Tittle of track
72. strLength (string): Length of CD (MM:SS)
74. """
75. strID = input('Enter ID: ').strip()
76. **while** True:
77. #This try handles the case where a non interger is entered
78. **try**:
79. intID = int(strID)
80. **break**
81. **except**:
82. strID = input('Please enter an interger for the ID. ')
83. strTitle = input('What is the tracks\'s name? ').strip()
84. strLength = input('How long is the track (MM:SS)? ').strip()
85. **return** intID, strTitle, strLength

88. #get new cd input
89. newCDOrder, newCDTitle, newCDLength = IO.new\_cd()
91. #pass the saved input into the expected fields for the initiation of the object
92. objTrack1 = TrackInfo(newCDOrder, newCDTitle, newCDLength)
94. # Diplays fields of objTrack1 ('order', title', and 'length') that were given
95. #values when the object was created.
96. **print**('The position of the track **is**: {} \n\
97. The title of the track **is**: {} \n\
98. The length of the track **is**: {}'\
99. .format(objTrack1.order, objTrack1.title, objTrack1.length))

# Appendix A.5

1. #-------------------------------------------#
2. #Tittle: Lab08\_E, IT FDN 110
3. #Desc: Creating a CD with classes - using \_\_str\_\_(self)
4. #Change Log: (Who, When, What)
5. #Brent Kieszling, 2020, Aug, 28, created file
6. #-------------------------------------------#
8. #DATA---------------------------------------



13. #PROCESS------------------------------------
15. **class** TrackInfo():
16. # -- Fields -- #
18. # -- Constructor -- #
19. **def** \_\_init\_\_(self, cdorder, cdtitle, cdlength):
20. # -- Attributes -- #
21. self.\_\_order = cdorder
22. self.\_\_title = cdtitle
23. self.\_\_length = cdlength
25. # -- Properties -- #
26. @property
27. **def** order(self):
28. **return** self.\_\_order
30. @order.setter
31. **def** order(self, value):
32. **if** str(value).isnumeric() == False:
33. **raise** Exception('Integers only')
34. **else**:
35. self.\_\_order = int(value)
37. @property
38. **def** title(self):
39. **return** self.\_\_title
41. @title.setter
42. **def** title(self, value):
43. **if** value.isstring():
44. self.\_\_title = value
45. **else**:
46. **raise** Exception('Not a valid input')
48. @property
49. **def** length(self):
50. **return** self.\_\_length
52. @length.setter
53. **def** length(self, value):
54. **if** value.isstring():
55. self.\_\_length = value
56. **else**:
57. **raise** Exception('Not a valid input')
59. # -- Methods -- #
60. **def** inventory(self):
61. **return** 'The position of the track **is**: {} \n\
62. The title of the track **is**: {} \n\
63. The length of the track **is**: {}'\
64. .format(self.order, self.title, self.length)
66. **def** \_\_str\_\_(self):
67. **return** self.inventory()

70. #PRESENTATION INPUT/OUTPUT------------------
72. **class** IO:
73. """Handling Input / Output"""
74. **def** new\_cd():
75. """Allows the user to add a CD to the active inventory table
77. Args:
78. None.
80. Returns:
81. intID (interger): Serialized CD order
82. strTitle (string): Tittle of track
83. strLength (string): Length of CD (MM:SS)
85. """
86. strID = input('Enter ID: ').strip()
87. **while** True:
88. #This try handles the case where a non interger is entered
89. **try**:
90. intID = int(strID)
91. **break**
92. **except**:
93. strID = input('Please enter an interger for the ID. ')
94. strTitle = input('What is the tracks\'s name? ').strip()
95. strLength = input('How long is the track (MM:SS)? ').strip()
96. **return** intID, strTitle, strLength

99. #get new cd input
100. newCDOrder, newCDTitle, newCDLength = IO.new\_cd()
102. #pass the saved input into the expected fields for the initiation of the object
103. objTrack1 = TrackInfo(newCDOrder, newCDTitle, newCDLength)
105. # Diplays fields of objTrack1 ('order', title', and 'length') that were given
106. #values when the object was created.
107. **print**(objTrack1)
108. **print**()
109. **print**(objTrack1.inventory())

# Appendix B.1

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. #Brent Kieszling, 2020-Sep-03, Major updates include: Created CD class, added methods to IO: menu, menu choice, and add new cd.
8. #------------------------------------------#
9. **import** os
10. **import** pickle
12. # -- DATA -- #
13. strFileName = 'cdInventory.dat'
14. lstOfCDObjects = []
15. **class** CD:
16. """Stores data about a CD:
18. Args:
19. cdNum: (int) CD collection ID
20. cdTitle: (string) CD title
21. cdArtist: (Artist) CD artist
23. properties:
24. cd\_id: (int) with CD ID
25. cd\_title: (string) with the title of the CD
26. cd\_artist: (string) with the artist of the CD
28. methods:
29. inventory (self): Returns the current attributes of the CD
30. \_\_str\_\_(self): Customized string description of the object
31. """
32. # -- Fields -- #
34. # -- Constructor -- #
35. **def** \_\_init\_\_(self, cdNum, cdTitle, cdArtist):
36. # -- Attributes -- #
37. self.\_\_order = cdNum
38. self.\_\_title = cdTitle
39. self.\_\_artist = cdArtist
41. # -- Properties -- #
42. @property
43. **def** cd\_id(self):
44. **return** self.\_\_order
46. @cd\_id.setter
47. **def** cd\_id(self, value):
48. **if** str(value).isnumeric() == False:
49. **raise** Exception('Integers only')
50. **else**:
51. self.\_\_order = int(value)
53. @property
54. **def** cd\_title(self):
55. **return** self.\_\_title
57. @cd\_title.setter
58. **def** cd\_title(self, value):
59. **if** value.isstring():
60. self.\_\_title = value
61. **else**:
62. **raise** Exception('Not a valid input, string expected')
64. @property
65. **def** cd\_artist(self):
66. **return** self.\_\_artist
68. @cd\_artist.setter
69. **def** cd\_artist(self, value):
70. **if** value.isstring():
71. self.\_\_artist = value
72. **else**:
73. **raise** Exception('Not a valid input')
75. # -- Methods -- #
76. **def** inventory(self):
77. **return** '{}\t{} (by:{})'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
79. **def** \_\_str\_\_(self):
80. **return** self.inventory()

83. # -- PROCESSING -- #
84. **if** os.path.exists(strFileName) != True:
85. objFile = open(strFileName, 'ab')
86. objFile.close()
88. **class** FileIO:
89. """Processes data to and from file:
91. properties:
93. methods:
94. write\_file(file\_name, lstCDs): Saves active list to binary file. No returns
95. read\_file(file\_name, lstCDs): Returns a list of CD objects from the saved file
97. """
98. @staticmethod
99. **def** read\_file(file\_name, lstCDs):
100. """Function to import a list of objects (lstCDs) from a binary file.
102. Args:
103. file\_name (string): name of file used to read the data from
104. lstCDs (list of objects): list of CD objects to hold data during runtime.
106. Returns:
107. lstCDs (list of objects): list of CD objects
108. """
109. lstCDs.clear()
110. #The try statement handles an instance where there is no saved data
111. **try**:
112. with open(file\_name, 'rb') as fileObj:
113. lstCDs = pickle.load(fileObj)
114. **except**:
115. **pass**
116. **return** lstCDs
118. @staticmethod
119. **def** write\_file(file\_name, lstCDs):
120. """Function to save a list of objects (lstCDs) to a binary file
122. Args:
123. file\_name (string): name of file used to read the data from
124. lstCDs (list of objects): list of CD objects to hold data during runtime.
126. Returns:
127. None.
128. """
129. with open(file\_name, 'wb') as fileObj:
130. pickle.dump(lstCDs, fileObj)
131. **pass**
133. # -- PRESENTATION (Input/Output) -- #
134. **class** IO:
135. """User interaction with the program
136. Args:
137. None.
139. properties:
140. None.
142. methods:
143. print\_menu(): Displays menu options to user
144. menu\_choice(): Gets user menu choice. Returns the choice
145. show\_inventory(lstCDs): Displays each CD object in the current list of
146. CDs.
147. new\_cd(): Gets user input for new CD information. Returns CD info.
148. load\_check(): Warns the user about the impact of loading and gives them
149. an option to stop. Returns a flag tracking the decision.
150. """
152. @staticmethod
153. **def** print\_menu():
154. """Displays a menu of choices to the user
156. Args:
157. None.
159. Returns:
160. None.
161. """
163. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
164. **print**('[s] Save Inventory to file\n[x] exit\n')
166. @staticmethod
167. **def** menu\_choice():
168. """Gets user input for menu selection
170. Args:
171. None.
173. Returns:
174. choice (string): a lower case sting of the users input out of the choices l, a, i, s or x
176. """
177. choice = ' '
178. **while** choice **not** **in** ['l', 'a', 'i', 's', 'x']:
179. choice = input('Which operation would you like to perform? [l, a, i, s or x]: ').lower().strip()
180. **print**()  # Add extra space for layout
181. **return** choice
183. @staticmethod
184. **def** show\_inventory(lstCDs):
185. """Displays current inventory table
187. Args:
188. lstCDs (list of objects): list of CD objects to hold data during runtime.
190. Returns:
191. None.
193. """
194. **print**('======= The Current Inventory: =======')
195. **print**('ID\tCD Title (by: Artist)\n')
196. **for** row **in** lstCDs:
197. **print**(row)
198. **print**('======================================')
200. @staticmethod
201. **def** new\_cd():
202. """Allows the user to add a CD to the active inventory table
204. Args:
205. None.
207. Returns:
208. intID (interger): Serialized ID
209. strTitle (string): Tittle of CD
210. stArtist (string): Name of artist
212. """
213. # 3.3.1 Ask user for new ID, CD Title and Artist
214. strID = input('Enter ID: ').strip()
215. **while** True:
216. #This try handles the case where a non interger is entered
217. **try**:
218. intID = int(strID)
219. **break**
220. **except**:
221. strID = input('Please enter an interger for the ID.')
222. strTitle = input('What is the CD\'s title? ').strip()
223. stArtist = input('What is the Artist\'s name? ').strip()
224. **return** intID, strTitle, stArtist
226. @staticmethod
227. **def** load\_check():
228. """Checks to see if the user wants to overwrite current data with saved data
230. Args:
231. None.
233. Returns:
234. flag (boolean): Tracks user response to the load check.
236. """
237. flag = False
238. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
239. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled. ')
240. **if** strYesNo.lower() == 'yes':
241. **print**('reloading...')
242. flag = True
243. **else**:
244. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
245. **return** flag


249. # -- Main Body of Script -- #
251. # Initialize the active list with saved data
252. lstOfCDObjects = FileIO.read\_file(strFileName, lstOfCDObjects)
254. # start main loop
255. **while** True:
256. IO.print\_menu()
257. strSlection = IO.menu\_choice()
258. # exit first
259. **if** strSlection == 'x':
260. **break**
262. # load inventory
263. **if** strSlection == 'l':
264. answer = IO.load\_check()
265. **if** answer == True:
266. lstOfCDObjects = FileIO.read\_file(strFileName, lstOfCDObjects)
267. **else**:
268. **pass**
269. **pass**
271. # add a CD
272. **elif** strSlection == 'a':
273. newCDID, newCDTitle, newCDArtist = IO.new\_cd()
274. objNewCD = CD(newCDID, newCDTitle, newCDArtist)
275. lstOfCDObjects.append(objNewCD)
276. **continue**  # start loop back at top.
278. # display current inventory
279. **elif** strSlection == 'i':
280. IO.show\_inventory(lstOfCDObjects)
281. **continue**  # start loop back at top.
283. # save inventory to file
284. **elif** strSlection == 's':
285. IO.show\_inventory(lstOfCDObjects)
286. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
287. # 3.6.2 Process choice
288. **if** strYesNo == 'y':
289. # 3.6.2.1 save data
290. FileIO.write\_file(strFileName, lstOfCDObjects)
291. **else**:
292. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
293. **continue**  # start loop back at top.
295. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
296. **else**:
297. **print**('General Error')

# Appendix B.2

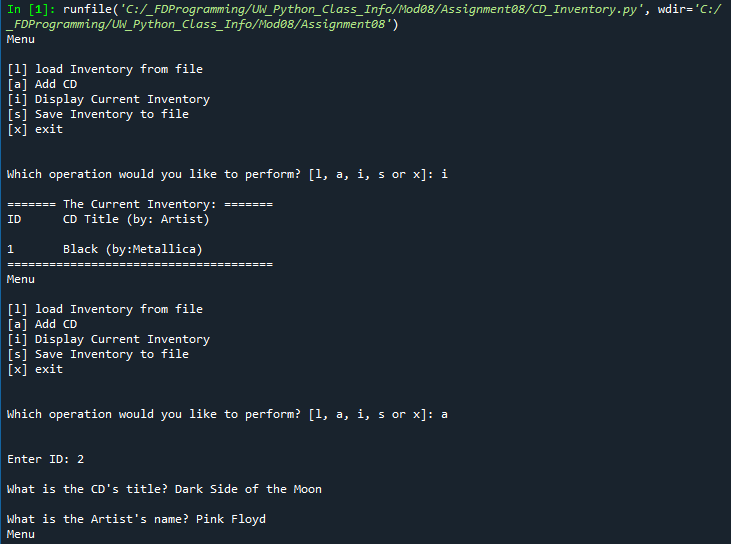


Figure 5 - Program run in Spyder part 1/3

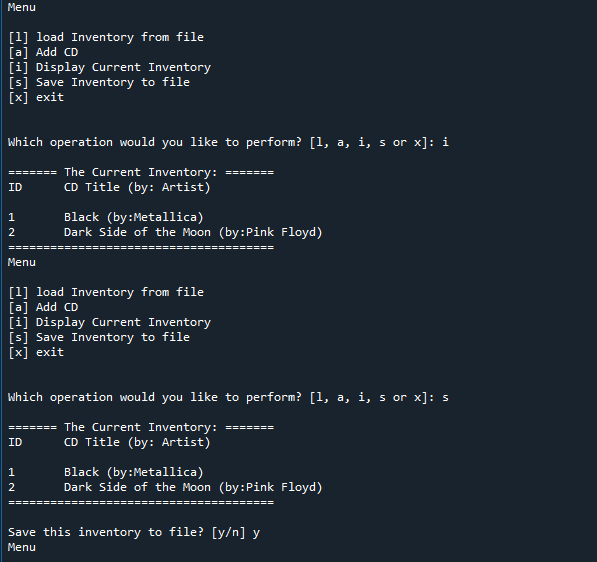


Figure 6 - Program run in Spyder part 2/3

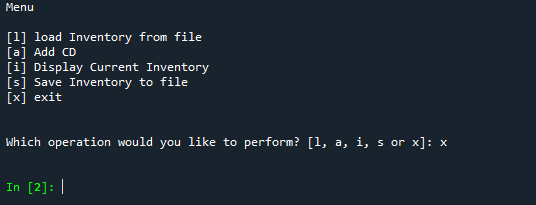


Figure 7 - Program run in Spyder part 3/3

# Appendix B.3

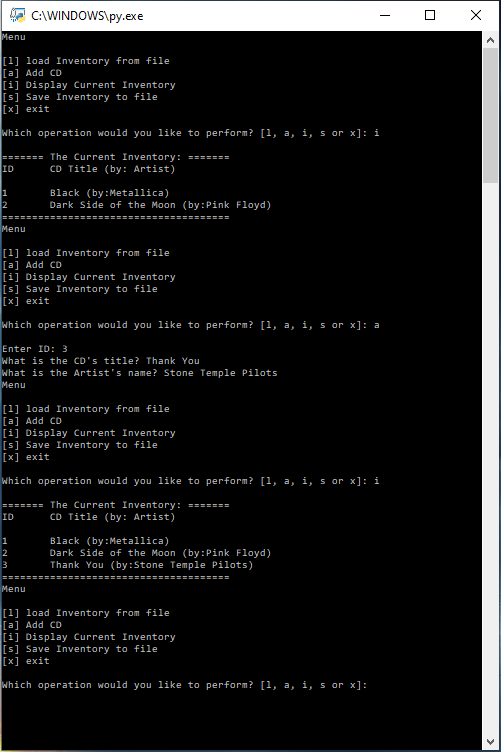


Figure 8 – Program run in Terminal

# GitHub Link

1. https://github.com/Brent-K/Assignment\_08

# References