Andrew Hanson

2020 Sep 01

IT FDN 110 B Su 20

Assignment08

Classes, OOP, and private objects

# Intro

This Module focused on Object Oriented Programing or OOP. Topics under this heading included classes, fields, constructors, destructors, attributes, properties, methods, decorators, private objects, and type hints. Classes are blueprints for an object that help construct a custom type of data. Fields store data common to all instances of an object. Constructors specify how an object of a class is instantiated. Destructors specify how an object of a class in de-instantiated. Attributes hold data specific to an instance of a class object. Properties specify how class attributes are assigned. Methods are functions that relate to class objects. Decorators modify functions. Private objects are mostly accessible only inside an instance of an objects by functions within that objects class. These include private functions, private methods, private fields, attributes, and properties.

# Data

The main data of the program is stored in CD objects. This data includes the CDs numeric ID, song title, and song artist. These CD objects are appended to a list and stored in a data file with pickle in binary format. New entries are created via user input. The data stored in the file is read in when the program is initialized and when the user chooses the [LOAD] load Inventory from file option. Data is saved manually by the user when choosing the [SAVE] Save Inventory to file. Data can be deleted before saving to remove it from the file. The data is worked with in memory between reading and writing requests.

## Class CD

The CD class is used to store CD related data. The CD class contains the private field \_\_numCDs which keep track of the number of instantiated CD objects and is currently not used in the program. The CD class contains the attributes ID, title, and artist. It also contains properties related to those attributes. Those attributes store the numerical ID number for the CD, the song title for the CD, and the song artist for the CD. The CD class contains the methods CDs, \_\_incrementer, \_\_atributestrings, and \_\_str\_\_. The CDs and \_\_incrementer method both relate to the \_\_numCDs field and are currently not used in the program. The \_\_atributestrings and \_\_str\_\_ methods modify the dunder string return of class objects to provide details about the class objects attributes in a formatted string. This function is also currently unused in the program. The CD class contains a constructor that assigns positional arguments to the attributes of the CD object when it is instantiated.

1. **class** CD:
2. """ Stores data about a CD.
4. fields: \_\_numCDs
5. Attributes: (ID,title,artist)
6. properties: (\_\_ID,\_\_title,\_\_artist)
7. Methods: (CDs,\_\_incrementer,\_\_atributestrings,\_\_str\_\_)"""
9. #fields#
10. \_\_numCDs = 0 #this is the private variable for the number of CD objects instatiated
12. #constructor#
13. **def** \_\_init\_\_(self, i, t, a): # (i,t,a) are positional arguments that stand for ID, title, and artist
14. """ Initializes the CD object.
15. This method is implicitly called when the CD class is called with no other method.
17. Attributes: (ID,title,artist)
18. ID (integer): This is the numeric lable for the CD.
19. title (string): This is the song title for the CD.
20. artist (string): This is the artist of the song for the CD.
22. Returns: instantiated CD class object"""
24. #atributes#
25. self.ID = i
26. self.title = t
27. self.artist = a
28. CD.\_\_incrementer() #this function increments the tracker for instatiated CD objects
30. #properties#
31. @property # cd\_id: (int) with CD ID
32. **def** ID(self):
33. """ This is the getter property for the ID atribute. """
34. **return** self.\_\_ID
35. @ID.setter
36. **def** ID(self, value):
37. """ This is the setter property for the ID atribute. """
38. **if** type(value) != int:
39. **raise** Exception('The CD ID must be an integer!')
40. **else**:
41. self.\_\_ID = value
43. @property #cd\_title: (string) with the title of the CD
44. **def** title(self):
45. """ This is the getter property for the title atribute. """
46. **return** self.\_\_title
47. @title.setter
48. **def** title(self, value):
49. """ This is the setter property for the title atribute. """
50. **if** type(value) != str:
51. **raise** Exception('The title must be a string!')
52. #TODO: add duplicate entry check
53. **else**:
54. self.\_\_title = value
56. @property #cd\_artist: (string) with the artist of the CD
57. **def** artist(self):
58. """ This is the getter property for the artist atribute. """
59. **return** self.\_\_artist
60. @artist.setter
61. **def** artist(self, value):
62. """ This is the setter property for the artist atribute. """
63. **if** type(value) != str:
64. **raise** Exception('The artist must be a string!')
65. **else**:
66. self.\_\_artist = value
68. #methods#
69. @staticmethod
70. **def** CDs():
71. """ This function returns the number of initialized CDs
73. Args: None
75. Returns: f'there are {CD.\_\_numCDs} CD(s)'      """
76. **return** f'there are {CD.\_\_numCDs} CD(s)'
78. @staticmethod
79. **def** \_\_incrementer():
80. """ This function increments the number of initialized CDs """
81. CD.\_\_numCDs += 1
83. **def** \_\_atributestrings(self):
84. """ this function returns a formated string containing the CDs atributes """
85. **return** f"{self.\_\_ID} - {self.\_\_title} - {self.\_\_artist}"
87. **def** \_\_str\_\_(self):
88. """ This customizes the dunder string function for this class
89. This function calls the \_\_atributestrings function """
90. **return** self.\_\_atributestrings()

Listing 1 - Code for the CD class.

### [ADD] Add CD

When the user chooses [ADD] Add CD from the menu, they will be prompted to input data to create CD entries into the Inventory. The default assumption for this program is that users would want to input multiple CD entries at once, so this option will loop through data prompts until the user inputs the keyword “BACK”. This keyword is non case sensitive and also not sensitive to extraneous white space characters before or after the keyword. If the user inputs a non-integer input for the ID they will be prompted to input an integer.

After the user exits the loop with the “BACK” keyword, the inputs are used to create CD class objects and the objects are added to a list for use throughout the program. These entries are not saved automatically and must be manually saved by the user with the [SAVE] Save Inventory to file option.

1. @staticmethod
2. **def** add():
3. """
4. takes initial user input
5. checks that the ID entry is an Integer before return.
6. loops until input is 'back'
7. returns the user's inputs as a list of tuples.
9. Args:
10. None.
12. Returns:
13. lstTplUserinput (list of tuples): 2D data structure (list of tuples) that holds multiple user input tuples.
15. """
16. **print**('input "BACK" to exit')
17. Userinputs = []
18. **while** True:
19. **while** True:
20. strID = input('Enter ID: ').strip()
21. **if** strID.upper().strip() == 'BACK':
22. **break**
23. **try**:
24. int(strID)
25. **break**
26. **except**(ValueError):
27. **print**('Invalid entry, please enter a number.')
28. **continue**
29. **except** Exception as e:
30. **print**(e)
31. **print**('Please, try again.')
32. **continue**
33. **if** strID.lower().strip() == 'BACK':
34. **break**
35. strTitle = input('What is the CD\'s title? ').strip()
36. **if** strTitle.lower().strip() == 'BACK':
37. **break**
38. strArtist = input('What is the Artist\'s name? ').strip()
39. **if** strArtist.lower().strip() == 'BACK':
40. **break**
41. tplUserinputs = (strID,strTitle,strArtist)
42. Userinputs.append(tplUserinputs)
43. **print**('input "BACK" to exit')
44. **return** Userinputs

Listing 2 - code for add function

A screen shot of a computer

Description automatically generated

Figure 1 - ADD running in spyder

# Processing

Data is processed to and from a data file with the pickle module in binary format. Data is processed and formatted into strings for presentation purposes. Data is also processed for deletion when the user chooses [DEL] delete CD from Inventory.

## Class FileIO

The FileIO class is responsible for operations with the CDInventory.dat data file. The class contains two important methods, one related to reading from the file, and the other related to writing to the file. The functions in this class use pickle to read and write python objects in binary format to and from the data file.

1. **class** FileIO:
2. """ Processes data to and from file. """
4. # methods:
6. @staticmethod # save\_inventory(file\_name, lst\_Inventory): -> None
7. **def** save\_inventory(file\_name : str, lst\_Inventory : list) -> None:
9. @staticmethod
10. **def** load\_inventory(file\_name: str) -> list: # load\_inventory(file\_name): -> (a list of CD objects)

Listing 3 - abbreviated code for FileIO class

### [LOAD] load Inventory from file

When the user chooses [LOAD] load Inventory from file from the menu, they will be prompted with a warning message and asked if they would like to continue. Then if they proceed the load\_inventory function will be called. This will use the pickle module to deserialized the data from the CDInventory.dat data file into a python object of a list containing CD class objects. This list will then replace the list currently being worked with in memory. The load\_inventory function is also responsible for data ingestion at the beginning of the runtime. The main body code contains a code block that creates a CDinventory.dat file if none exists. There is error handling in this function to account for an empty file or for general errors. In both cases the user is prompted with a message about the error.

1. @staticmethod
2. **def** load\_inventory(file\_name: str) -> list: # load\_inventory(file\_name): -> (a list of CD objects)
3. """Function to manage data ingestion from file
5. Reads the data from file identified by file\_name into a list of objects
6. (list of CD objects) list one item in the list represents one CD object.
7. Checks for error with empty file.
9. Args:
10. file\_name (string): name of file used to read the data from
11. table (list of CD objetcs): 2D data structure (list of objects) that holds the data during runtime
13. Returns:
14. table.
15. """
16. table = []  # this clears existing data and allows to load data from file
17. with open(file\_name, 'rb') as objFile:
18. **try**:
19. table = pickle.load(objFile) #note: load() loads one line of data
20. **except** (EOFError):
21. **print**('Data file is empty!'.upper())
22. **except** Exception as e:
23. **print** (e)
24. **return** table

Listing 4 - code for load\_inventory function

### [SAVE] Save Inventory to file

When the user chooses the [SAVE] Save Inventory to file option from the menu, they will be prompted with a warning and asked if they would like to continue. Then if they proceed the save\_inventory function will be called. This will use the pickle module to serialize the data from the list so it can be stored in the CDInventory.dat data file in binary format. This allows the program to save and retrieve data between use. The function contains general error handling that prompts the user with a message.

1. @staticmethod # save\_inventory(file\_name, lst\_Inventory): -> None
2. **def** save\_inventory(file\_name : str, lst\_Inventory : list) -> None:
3. """Function writes inventory to file
5. opens the file
6. takes in the inventory list
7. writes a python object using pickle to a data file in binary format.
8. closes file
9. checks for genneral error and prints to user
11. Args:
12. file\_name (string): name of file used to read the data from
13. lst\_Inventory (list of objects): 2D data structure (list of objects) that holds the data during runtime
15. Returns:
16. None.
17. """
18. with open(file\_name, 'wb') as objFile:
19. **try**:
20. pickle.dump(lst\_Inventory, objFile)
21. **except** Exception as e:
22. **print**(e)

Listing 5 - code for the save\_inventory function

## Class DataProcessor

The DataProcessor class is responsible for operations involving the data in memory. The class contains one important method responsible for removing data entries.

### [DEL] delete CD from Inventory

When the user chooses the [DEL] delete CD from Inventory option from the menu, they are shown the current inventory entries and prompted to input the numerical ID number for the entry they would like removed from the list. If the user inputs a non-integer entry they will be prompted with an error message and returned to the main menu. If they input an integer that does not belong to a CD they will be prompted with an error message and returned to the main menu.

1. @staticmethod
2. **def** delete\_entry(ID, table):
3. """Deletes an entry from memory.
4. Modifies table to remove selected entry.
6. Args:
7. table (list of CD objects): 2D data structure (list of objects) that holds the data during runtime
8. ID (integer): numerical ID for CD entry
10. Returns:
11. None.
12. """
13. intRowNr = -1
14. blnCDRemoved = False
15. **for** row **in** table:
16. intRowNr += 1
17. **if** row.ID == ID:
18. **del** table[intRowNr]
19. blnCDRemoved = True
20. **break**
21. **if** blnCDRemoved:
22. **print**('The CD was removed')
23. **else**:
24. **print**('Could not find this CD!')

Listing 6 - code for delete\_entry function

A screenshot of a cell phone

Description automatically generated

Figure 2 - DEL running in the console

# Presentation

Presentation is handled by the IO class. There is an interactive user menu. There is a function that displays the current inventory, and there is a function to handle data entry by the user.

## Class IO

The IO class is responsible for the user menu, presenting data to the user, and taking input from the user. The class contains four important methods, one for data input, one for data output, and two for the user menu. The functions are add, show\_inventory, print\_menu, and menu\_choice.

### [ADD] Add CD

When the user chooses the [ADD] Add CD option from the menu, they are prompted for three pieces of data. They are asked for a numerical ID to identify the CD. They are asked for a song title for the CD. Then, they are asked for a song artist for the CD. The ID input is rejected if it is not an integer and the user is asked for another input.

1. **elif** strChoice == 'ADD':
3. # 3.3.1 Ask user for new ID, CD Title and Artist
4. lstTplUserInputs = IO.add()
6. # 3.3.2 Add item to the table
7. **for** item **in** lstTplUserInputs:
8. intID = int(item[0])
9. strTitle = item[1]
10. strArtist = item[2]
11. CDobj = CD(intID, strTitle, strArtist)
12. lstOfCDObjects.append(CDobj)
13. IO.show\_inventory(lstOfCDObjects)

Listing 7 - main body code for “[ADD] Add CD”

### [SHOW] display Current Inventory

When the user chooses the [SHOW] display Current Inventory option from the menu, they are prompted with many formatted strings displaying the contents of the CD inventory. Behind the scenes, the data in converted into a list of dictionaries to work with code that was previously used to display the inventory from a list of dictionaries. The entries are sorted in ascending order by numerical ID before display to the user.

1. **def** show\_inventory(lis):
2. #TODO update docstring
3. """displays current inventory to user in console

6. Args:
7. lis (list of CDobjects): 2D data structure (list of CDobjects) that holds the data during runtime.
9. Returns:
10. None.
12. """
13. table = []
14. **for** CDobj **in** lis:
15. intID = CDobj.ID
16. strTitle = CDobj.title
17. strArtist = CDobj.artist
18. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
19. table.append(dicRow)
21. table = sorted(table, key = **lambda** i: i['ID'])
22. # https://www.geeksforgeeks.org/ways-sort-list-dictionaries-values-python-using-lambda-function/
23. # https://www.youtube.com/watch?v=3dt4OGnU5sM
25. **print**('======= The Current Inventory: =======')
26. **print**('ID\tCD Title (by: Artist)')
27. **for** row **in** table:
28. **print**('{}\t{} (by:{})'.format(\*row.values()))
29. **print**('======================================')

Listing 8 - code for show\_inventory function

### Menu

The main menu is the primary point of user interactivity and control of the program. This makes it a very important part of the program. When the user inputs a choice other than a menu option they are prompted with an error message and asked to input a new choice.

#### print\_menu()

The function print\_menu in the class IO is responsible for making the user aware of their options and how to interact with the program. Placing the menu in a function allows additional functionality to be more easily added and helps with the separation of concerns. The options are: LOAD, ADD, SHOW, DEL, SAVE or EXIT.

1. @staticmethod
2. **def** print\_menu():
3. """displays a menu of choices to the user
5. Args:
6. None.
8. Returns:
9. None.
10. """
12. **print**('Menu\n\n[LOAD] load Inventory from file\n[ADD] Add CD\n[SHOW] display Current Inventory')
13. **print**('[DEL] delete CD from Inventory\n[SAVE] Save Inventory to file\n[EXIT] exit\n')

Listing 9 - code for print\_menu function

#### menu\_choice()

the function menu\_choice in the class IO is responsible for asking for the user’s choice input and returning it to the while loop so that the program can fulfil their desires. This is the main interface by which the user interacts with the program.

1. @staticmethod
2. **def** menu\_choice():
3. """Gets user input for menu selection
5. Args:
6. None.
8. Returns:
9. choice (string): an upper case sting of the users input out of the choices LOAD, ADD, SHOW, DEL, SAVE or EXIT
11. """
12. choice = ' '
13. **while** choice **not** **in** ['LOAD', 'ADD', 'SHOW', 'DEL', 'SAVE', 'EXIT']:
14. choice = input('Which operation would you like to perform? [LOAD, ADD, SHOW, DEL, SAVE or EXIT]: ').upper().strip()
15. **if** choice **not** **in** ['LOAD', 'ADD', 'SHOW', 'DEL', 'SAVE', 'EXIT']:
16. **print**("Invalid input, try again!")
17. **print**()  # Add extra space for layout
18. **return** choice
20. # -- Main Body of Script -- #
21. # Load data from file into a list of CD objects on script start
22. # 1.0 Create inventory file if none exists
23. **try**:
24. with open(strFileName, 'a') as objFile:
25. **pass**
26. **except** Exception as e:
27. **print**(e)
28. **print**(type(e), e, e.\_\_doc\_\_, sep = '\n')
29. **print**('CDInventory.dat file could not be acccessed or created, please contact support!')
30. **print**('Program will not function properly without file access!')
31. **print**("Please take notes so the problem can be recreated!")

Listing 10 - code for menu\_choice function

#### [EXIT] exit

When the user chooses the [EXIT] exit option from the menu, the program ends.

1. **while** True:
3. # 2.1 display Menu to user and get choice
4. IO.print\_menu()
5. strChoice = IO.menu\_choice()
7. # let user exit program
8. # 3. Process menu selection
9. # 3.1 process exit first
10. **if** strChoice == 'EXIT':
11. **break**

Listing 11 - abbreviated code for main menu including EXIT

# Challenges

The code from module08’s labs was used to construct the CD class. It required little modification, however the labs themselves took some effort to get through. Lots of documentation was required for the new class and functions. Some updating for previous functions was necessary to work with the new variable and data type. More error catching was added to the code from the previous assignment when copied to the current program. Type hints were implemented with the fileIO methods. A method for presenting the list in ascending order was found on the internet. The show inventory function was given code to fill in the expected dictionary format with CD attributes.

# Summary

This module was an introduction to Object Oriented Programing or OOP. A data class was used with a constructor, attributes, and properties. Classes are useful in giving additional control over your data. Class objects can be stored in binary with pickle the same way lists or dictionaries are stored. Private objects were gone over in significant detail.

# Appendix

## Sources

<https://www.geeksforgeeks.org/ways-sort-list-dictionaries-values-python-using-lambda-function/>

<https://www.youtube.com/watch?v=3dt4OGnU5sM>

## Repository

<https://github.com/clichekhfan/Assignment08>

## Questions

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

The difference is the same as the difference between the list data type and an instantiation of a list. Lists contain common methods and attributes, but they have individual data within those attributes.

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

Fields, constructor, destructor, attributes, properties, and methods

### • What is the purpose of a class constructor?

The class constructor specifies how the class object populates it’s attributes and instantiates the class object.

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

When referring to an instance of the class object.

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

When the method is common across all instances of the class objects.

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

Field contain data common to all objects of a class. Attributes contain data individual to the instance of the class object. Properties specify how the attributes are assigned.

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

Methods are functions common to objects of a class. Properties specify how the attributes of a class object are assigned.

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

So humans can understand your bizzaro code. Docstring makes your code more accessible and understandable. Classes in particular can be opaque to novice programmers and benefit greatly from documentation about their usage.

## CDInventory.py

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. # AHanson, 2020-Aug-31, add more psudocode to outline
8. # AHanson, 2020-Aug-31, created classes methods and functions
9. # AHanson, 2020-Aug-31, created main body code
10. # AHanson, 2020-Aug-31, added load option
11. # AHanson, 2020-Aug-31, added delete option
12. # AHanson, 2020-Aug-31, fixed bugs with properties
13. # AHanson, 2020-Aug-31, labeled load\_inventory and save\_inventory with type hints
14. # AHanson, 2020-Aug-31, labeled all functions and classes with placeholder docstring
15. # AHanson, 2020-Aug-31, updated docstring
16. #------------------------------------------#
18. **import** pickle
20. # -- DATA -- #
21. strChoice = '' # User input
22. strFileName = 'CDInventory.dat' # data storage file
23. lstOfCDObjects = [] #list of CD class objects
24. objFile = None  # file object
26. **class** CD:
27. """ Stores data about a CD.
29. fields: \_\_numCDs
30. Attributes: (ID,title,artist)
31. properties: (\_\_ID,\_\_title,\_\_artist)
32. Methods: (CDs,\_\_incrementer,\_\_atributestrings,\_\_str\_\_)"""
34. #fields#
35. \_\_numCDs = 0 #this is the private variable for the number of CD objects instatiated
37. #constructor#
38. **def** \_\_init\_\_(self, i, t, a): # (i,t,a) are positional arguments that stand for ID, title, and artist
39. """ Initializes the CD object.
40. This method is implicitly called when the CD class is called with no other method.
42. Attributes: (ID,title,artist)
43. ID (integer): This is the numeric lable for the CD.
44. title (string): This is the song title for the CD.
45. artist (string): This is the artist of the song for the CD.
47. Returns: instantiated CD class object"""
49. #atributes#
50. self.ID = i
51. self.title = t
52. self.artist = a
53. CD.\_\_incrementer() #this function increments the tracker for instatiated CD objects
55. #properties#
56. @property # cd\_id: (int) with CD ID
57. **def** ID(self):
58. """ This is the getter property for the ID atribute. """
59. **return** self.\_\_ID
60. @ID.setter
61. **def** ID(self, value):
62. """ This is the setter property for the ID atribute. """
63. **if** type(value) != int:
64. **raise** Exception('The CD ID must be an integer!')
65. **else**:
66. self.\_\_ID = value
68. @property #cd\_title: (string) with the title of the CD
69. **def** title(self):
70. """ This is the getter property for the title atribute. """
71. **return** self.\_\_title
72. @title.setter
73. **def** title(self, value):
74. """ This is the setter property for the title atribute. """
75. **if** type(value) != str:
76. **raise** Exception('The title must be a string!')
77. #TODO: add duplicate entry check
78. **else**:
79. self.\_\_title = value
81. @property #cd\_artist: (string) with the artist of the CD
82. **def** artist(self):
83. """ This is the getter property for the artist atribute. """
84. **return** self.\_\_artist
85. @artist.setter
86. **def** artist(self, value):
87. """ This is the setter property for the artist atribute. """
88. **if** type(value) != str:
89. **raise** Exception('The artist must be a string!')
90. **else**:
91. self.\_\_artist = value
93. #methods#
94. @staticmethod
95. **def** CDs():
96. """ This function returns the number of initialized CDs
98. Args: None
100. Returns: f'there are {CD.\_\_numCDs} CD(s)'      """
101. **return** f'there are {CD.\_\_numCDs} CD(s)'
103. @staticmethod
104. **def** \_\_incrementer():
105. """ This function increments the number of initialized CDs """
106. CD.\_\_numCDs += 1
108. **def** \_\_atributestrings(self):
109. """ this function returns a formated string containing the CDs atributes """
110. **return** f"{self.\_\_ID} - {self.\_\_title} - {self.\_\_artist}"
112. **def** \_\_str\_\_(self):
113. """ This customizes the dunder string function for this class
114. This function calls the \_\_atributestrings function """
115. **return** self.\_\_atributestrings()
117. # -- PROCESSING -- #
119. **class** FileIO:
120. """ Processes data to and from file. """
121. #fields#
123. #constructor#
125. #attributes#
127. # properties:
129. # methods:
131. @staticmethod # save\_inventory(file\_name, lst\_Inventory): -> None
132. **def** save\_inventory(file\_name : str, lst\_Inventory : list) -> None:
133. """Function writes inventory to file
135. opens the file
136. takes in the inventory list
137. writes a python object using pickle to a data file in binary format.
138. closes file
139. checks for genneral error and prints to user
141. Args:
142. file\_name (string): name of file used to read the data from
143. lst\_Inventory (list of objects): 2D data structure (list of objects) that holds the data during runtime
145. Returns:
146. None.
147. """
148. with open(file\_name, 'wb') as objFile:
149. **try**:
150. pickle.dump(lst\_Inventory, objFile)
151. **except** Exception as e:
152. **print**(e)
154. @staticmethod
155. **def** load\_inventory(file\_name: str) -> list: # load\_inventory(file\_name): -> (a list of CD objects)
156. """Function to manage data ingestion from file
158. Reads the data from file identified by file\_name into a list of objects
159. (list of CD objects) list one item in the list represents one CD object.
160. Checks for error with empty file.
162. Args:
163. file\_name (string): name of file used to read the data from
164. table (list of CD objetcs): 2D data structure (list of objects) that holds the data during runtime
166. Returns:
167. table.
168. """
169. table = []  # this clears existing data and allows to load data from file
170. with open(file\_name, 'rb') as objFile:
171. **try**:
172. table = pickle.load(objFile) #note: load() loads one line of data
173. **except** (EOFError):
174. **print**('Data file is empty!'.upper())
175. **except** Exception as e:
176. **print** (e)
177. **return** table
179. **class** DataProcessor:
180. ''''' processses data within memory durring runtime '''
182. #fields#
184. #constructor#
186. #attributes#
188. # properties:
190. # methods:
192. @staticmethod
193. **def** delete\_entry(ID, table):
194. """Deletes an entry from memory.
195. Modifies table to remove selected entry.
197. Args:
198. table (list of CD objects): 2D data structure (list of objects) that holds the data during runtime
199. ID (integer): numerical ID for CD entry
201. Returns:
202. None.
203. """
204. intRowNr = -1
205. blnCDRemoved = False
206. **for** row **in** table:
207. intRowNr += 1
208. **if** row.ID == ID:
209. **del** table[intRowNr]
210. blnCDRemoved = True
211. **break**
212. **if** blnCDRemoved:
213. **print**('The CD was removed')
214. **else**:
215. **print**('Could not find this CD!')
217. # -- PRESENTATION (Input/Output) -- #
218. **class** IO:
219. """ manages IO operations """
221. #fields#
223. #constructor#
225. #attributes#
227. # properties:
229. # methods:
231. @staticmethod
232. **def** add():
233. """
234. takes initial user input
235. checks that the ID entry is an Integer before return.
236. loops until input is 'back'
237. returns the user's inputs as a list of tuples.
239. Args:
240. None.
242. Returns:
243. lstTplUserinput (list of tuples): 2D data structure (list of tuples) that holds multiple user input tuples.
245. """
246. **print**('input "BACK" to exit')
247. Userinputs = []
248. **while** True:
249. **while** True:
250. strID = input('Enter ID: ').strip()
251. **if** strID.upper().strip() == 'BACK':
252. **break**
253. **try**:
254. int(strID)
255. **break**
256. **except**(ValueError):
257. **print**('Invalid entry, please enter a number.')
258. **continue**
259. **except** Exception as e:
260. **print**(e)
261. **print**('Please, try again.')
262. **continue**
263. **if** strID.lower().strip() == 'BACK':
264. **break**
265. strTitle = input('What is the CD\'s title? ').strip()
266. **if** strTitle.lower().strip() == 'BACK':
267. **break**
268. strArtist = input('What is the Artist\'s name? ').strip()
269. **if** strArtist.lower().strip() == 'BACK':
270. **break**
271. tplUserinputs = (strID,strTitle,strArtist)
272. Userinputs.append(tplUserinputs)
273. **print**('input "BACK" to exit')
274. **return** Userinputs
276. @staticmethod
277. **def** show\_inventory(lis):
278. #TODO update docstring
279. """displays current inventory to user in console

282. Args:
283. lis (list of CDobjects): 2D data structure (list of CDobjects) that holds the data during runtime.
285. Returns:
286. None.
288. """
289. table = []
290. **for** CDobj **in** lis:
291. intID = CDobj.ID
292. strTitle = CDobj.title
293. strArtist = CDobj.artist
294. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
295. table.append(dicRow)
297. table = sorted(table, key = **lambda** i: i['ID'])
298. # https://www.geeksforgeeks.org/ways-sort-list-dictionaries-values-python-using-lambda-function/
299. # https://www.youtube.com/watch?v=3dt4OGnU5sM
301. **print**('======= The Current Inventory: =======')
302. **print**('ID\tCD Title (by: Artist)')
303. **for** row **in** table:
304. **print**('{}\t{} (by:{})'.format(\*row.values()))
305. **print**('======================================')
307. @staticmethod
308. **def** print\_menu():
309. """displays a menu of choices to the user
311. Args:
312. None.
314. Returns:
315. None.
316. """
318. **print**('Menu\n\n[LOAD] load Inventory from file\n[ADD] Add CD\n[SHOW] display Current Inventory')
319. **print**('[DEL] delete CD from Inventory\n[SAVE] Save Inventory to file\n[EXIT] exit\n')
321. @staticmethod
322. **def** menu\_choice():
323. """Gets user input for menu selection
325. Args:
326. None.
328. Returns:
329. choice (string): an upper case sting of the users input out of the choices LOAD, ADD, SHOW, DEL, SAVE or EXIT
331. """
332. choice = ' '
333. **while** choice **not** **in** ['LOAD', 'ADD', 'SHOW', 'DEL', 'SAVE', 'EXIT']:
334. choice = input('Which operation would you like to perform? [LOAD, ADD, SHOW, DEL, SAVE or EXIT]: ').upper().strip()
335. **if** choice **not** **in** ['LOAD', 'ADD', 'SHOW', 'DEL', 'SAVE', 'EXIT']:
336. **print**("Invalid input, try again!")
337. **print**()  # Add extra space for layout
338. **return** choice
340. # -- Main Body of Script -- #
341. # Load data from file into a list of CD objects on script start
342. # 1.0 Create inventory file if none exists
343. **try**:
344. with open(strFileName, 'a') as objFile:
345. **pass**
346. **except** Exception as e:
347. **print**(e)
348. **print**(type(e), e, e.\_\_doc\_\_, sep = '\n')
349. **print**('CDInventory.dat file could not be acccessed or created, please contact support!')
350. **print**('Program will not function properly without file access!')
351. **print**("Please take notes so the problem can be recreated!")
353. # 1.1 When program starts, read in the currently saved Inventory
354. lstOfCDObjects = FileIO.load\_inventory(strFileName)
356. # SHOWlay menu to user
357. **while** True:
359. # 2.1 display Menu to user and get choice
360. IO.print\_menu()
361. strChoice = IO.menu\_choice()
363. # let user exit program
364. # 3. Process menu selection
365. # 3.1 process exit first
366. **if** strChoice == 'EXIT':
367. **break**
369. # show user current inventory
370. # 3.2 process display current inventory
371. **elif** strChoice == 'SHOW':
372. IO.show\_inventory(lstOfCDObjects)
373. **continue**  # start loop back at top.
375. # let user add data to the inventory
376. # 3.3 process add a CD
377. **elif** strChoice == 'ADD':
379. # 3.3.1 Ask user for new ID, CD Title and Artist
380. lstTplUserInputs = IO.add()
382. # 3.3.2 Add item to the table
383. #TODO move code into function
384. **for** item **in** lstTplUserInputs:
385. intID = int(item[0])
386. strTitle = item[1]
387. strArtist = item[2]
388. CDobj = CD(intID, strTitle, strArtist)
389. lstOfCDObjects.append(CDobj)
390. IO.show\_inventory(lstOfCDObjects)
391. #TODO flag end
393. # 3.4 process save inventory to file
394. # let user save inventory to file
395. **elif** strChoice == 'SAVE':
397. # 3.4.1 display current inventory and ask user for confirmation to save
398. IO.show\_inventory(lstOfCDObjects)
399. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
401. # 3.4.2 Process choice
402. **if** strYesNo == 'y':
404. # 3.4.2.1 save data
405. FileIO.save\_inventory(strFileName, lstOfCDObjects)
407. **else**:
408. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
409. **continue**  # start loop back at top.
411. # let user load inventory from file
412. # 3.5 process load inventory
413. **elif** strChoice == 'LOAD':
414. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
415. strYesNo = input('type \'yes\' (case sensitive) to continue and reload from file. otherwise reload will be canceled: ')
417. **if** strYesNo.lower() == 'yes':
418. **print**('reloading...')
419. lstOfCDObjects = FileIO.load\_inventory(strFileName)
420. IO.show\_inventory(lstOfCDObjects)
422. **else**:
423. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
424. IO.show\_inventory(lstOfCDObjects)
426. # 3.6 process delete a CD
427. **elif** strChoice == 'DEL':
428. # 3.6.1 get Userinput for which CD to delete
429. # 3.6.1.1 display Inventory to user
430. IO.show\_inventory(lstOfCDObjects)
431. # 3.6.1.2 ask user which ID to remove
433. **try**:
434. intIDDel = int(input('Which ID would you like to delete? ').strip())
436. **except**(ValueError):
437. **print**('Delete ID input must be integer!')
438. **continue**
440. **except** Exception as e:
441. **print**(e)
442. **continue**
444. # 3.6.2 search thru table and delete CD
445. DataProcessor.delete\_entry(intIDDel, lstOfCDObjects)
446. IO.show\_inventory(lstOfCDObjects)
447. **continue**  # start loop back at top.
449. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
450. **else**:
451. **print**('General Error')

Listing 12 - CDInventory.py