Murat Yokus
August 29th, 2021
IT FDN 110 B: Introduction to Programming (Python)
Assignment 08

CDInventory.py Script with Classes and Objects

Introduction

The script shown in this documentation is a continuation of Assignment 07 and includes sections of code to view the current inventory, enter CD data, save data to CDInventory.dat data file, load CD data from CDInventory.dat file, and exit the program. In addition to the previous assignment, Assignment 08 implements classes and objects. The script asks for user inputs of CD info (ID, title, and artist) and stores them in an object. The script was written in the Spyder IDE, and its successful operation was shown in Spyder and anaconda terminal. Finally, the document summarizes my learnings from Module 8.

Steps:

The script starts with defining a class, called *CD*, which is later used for instantiation of a new CD object whenever user wants to add a CD to the inventory (**Listing 1** and **2**). The CD class is used to store the attributes of a new CD (*i.e.*, CD ID, title, and artist) in an object. The *__str__()* method defined within this class is used for displaying the content of the CD inventory in **Listing 6**. Using *__str__()* method within a *for loop* was really useful and a short way of displaying the inventory.

```
#------------------------------------------#
# Title: CDInventory.py
# Desc: Assignnment 08 - Working with classes
# Change Log: (Who, When, What)
# DBiesinger, 2030-Jan-01, created file
# DBiesinger, 2030-Jan-01, added pseudocode to complete assignment 08
# MYokus, 2021-Aug-29, Added Code to work with classes and objects
#------------------------------------------#

import pickle # module for handling binary files
import os.path # module for common pathname manipulation

# -- DATA -- #
strFileName = ''
lstOfCDObjects = []
save_FileName = 'CDInventory.dat' # data storage file to save the data to

class CD:
    """Stores data about a CD:

    properties:
        cd_id: (int) with CD ID
        cd_title: (string) with the title of the CD
        cd_artist: (string) with the artist of the CD
    methods:
        __str__(): Creates a string for CD attributes

    """
    # --Fields --#
    # -- Constructor -- #
    def __init__(self,ID, title, artist):
        # --Attributes -- #
        self.__cd_id = ID
        self.__cd_title = title
        self.__cd_artist = artist
    # -- Properties -- #
    @property
    def cd_id(self):
        return self.__cd_id

    @cd_id.setter
    def cd_id(self, value):
        if type(value) == int:
            self.__cd_id = value
        else:
            raise Exception('Position needs to be integer!')

```

Listing 1- Defining a CD class to store CD data [1/2].

```
    @property
    def cd_title(self):
        return self.__cd_title

    @cd_title.setter
    def cd_title(self, value):
        if type(value) == str:
            self.__cd_title = value
        else:
            raise Exception('Title needs to be string')

    @property
    def cd_artist(self):
        return self.__cd_artist

    @cd_artist.setter
    def cd_artist(self, value):
        if type(value) == str:
            self.__cd_artist = value
        else:
            raise Exception('Length needs to be string')
    # -- Methods -- #
    def __str__(self):
        return (str(self.cd_id) + ',' + self.cd_title + ',' + '(by: ' + self.cd_artist + ')')
```

Listing 2 – Defining a CD class to store CD data [2/2].

The processing section of the script was similar to the Assignment07 (**Listing 3** and **4)**. The *File IO* class includes three functions, *read_Textfile(), read_file()*, and *write_file()*. The *read_Textfile()* function is used to read the CD inventory from a text file when the script is run for the first time. Then, the CD inventory data is saved to the binary file using the *write_file()* function. Lastly, the *read_file()* is used to read the data from the binary file for the subsequent runs.

```python
# --- PROCESSING -- #
class FileIO:
    """Processes data to and from file:

    properties:

    methods:
        read_Textfile(file_name, table): -> A list of CD objects
        read_file(file_name): -> A list of CD objects
        write_file(file_name, table): -> None
    """
    # --Fields --#
    # -- Constructor -- #
            # --Attributes -- #
    # -- Properties -- #
    # -- Methods -- #

    @staticmethod
    def read_Textfile(file_name, table):
        """Function to manage data ingestion from a text file to a list of objects

        Reads the data from a text file identified by file_name into a 2D table
        (list of objects) table one line in the file represents one object row in table.

        Args:
            file_name (string): name of the text file used to read the data from
            table (list of objects): 2D data structure (list of objects) that holds the data during runtime

        Returns:
            None.
        """

        try:
            table.clear()  # this clears existing data and allows to load data from file
            objFile = open(file_name, 'r')
            for line in objFile:
                data = line.strip().split(',') # data type: list
                cdRow = CD(int(data[0]), data[1], data[2]) # data type: object
                table.append(cdRow) # list of objects
            objFile.close()
        except FileNotFoundError as e:
            print('Text file does not exist!')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')

```

Listing 3 – Processing Section: File IO class. read_Textfile() function is used to read the CD inventory from a text file when the script is run for the first time [1/2].

```
118
119     @staticmethod
120     def read_file(file_name):
121         """Function to manage data ingestion from a binary file to a list of objects
122
123         Reads the data from a binary file identified by file_name into a 2D table
124         (list of objects) table one line in the file represents one object row in table.
125
126         Args:
127             file_name (string): name of the binary file used to read the data from
128
129         Returns:
130             data (list of objects): 2D data structure (list of objects)
131         """
132
133         try:
134             data = []
135             with open(file_name, 'rb') as fileObj:
136                 data = pickle.load(fileObj)
137             return data
138         except FileNotFoundError as e:
139             print('Binary file does not exist!')
140             print('Build in error info:')
141             print(type(e), e, e.__doc__, sep='\n')
142
143     @staticmethod
144     def write_file(file_name, table):
145         """Function to save a 2D table (a list of objects) to file via pickle
146
147         Saves the data in a file identified by file_name into a .dat file
148
149         Args:
150             file_name (string): name of binary file used to save the data to
151             table (list of objects): 2D data structure (list of objects) that holds the data during runtime
152
153         Returns:
154             None.
155         """
156         try:
157             with open(file_name, 'wb') as fileObj:
158                 pickle.dump(table, fileObj)
159         except FileNotFoundError as e:
160             print('Binary file does not exist!')
161             print('Build in error info:')
162             print(type(e), e, e.__doc__, sep='\n')
163
164
```

Listing 4 – Processing Section: File I/O class. read_file() and write_file() functions are used for reading and writing a list of CD objects from and to a binary file, respectively [2/2].

The presentation section of the script was also similar to the Assignment07 (**Listing 5** and **6)**. The *IO* class includes four functions, *print_menu(), menu_choice()*, *show_inventory(), and user_input()*. The *print_menu()* and *menu_choice()* functions are used for displaying a menu to user and reading user input for menu selection, respectively. Similarly, *show_inventory()* and *user_input()* functions are used for displaying the current inventory to the user and reading user input for CD data (ID, title, artist), respectively. Lines 225 and 226 in **Listing 6** iterates through the rows of the CD inventory table (a list of CD objects) to display the inventory to user. Lines 240 to 244 in **Listing 6** are used for reading the CD info (ID, title, and artist), creating a new CD object to hold the CD info, and appending the new CD object to the main list of CD objects.

```python

# -- PRESENTATION (Input/Output) -- #
class IO:
    """Handling Input / Output (User Interaction)

    properties:

    methods:
        print_menu(): -> Displays a menu of choices to the user
        menu_choice(): -> Gets user input for menu selection
        show_inventory(table): -> Displays current inventory table
        user_input(table): -> Ask user for new ID, CD Title, and Artist and creates a new object

    """

    @staticmethod
    def print_menu():
        """Displays a menu of choices to the user

        Args:
            None.

        Returns:
            None.
        """

        print('\nMenu\n\n[i] Display Current Inventory\n[a] Add CD\n[s] Save Inventory to file')
        print('[l] load Inventory from file\n[x] exit\n')

    @staticmethod
    def menu_choice():
        """Gets user input for menu selection

        Args:
            None.

        Returns:
            choice (string): a lower case sting of the users input out of the choices i, a, s, l, or x

        """
        choice = ' '
        while choice not in ['i', 'a', 's', 'l', 'x']: # 'While not loop: executes the body of the loop until the cond
            choice = input('Which operation would you like to perform? [i, a, s, l or x]: ').lower().strip()
        print()  # Add extra space for layout
        return choice

```

Listing 5 – Presentation Section: IO class. print_menu() and menu_choice() functions are used for displaying a menu and reading user input for menu selection, respectively [1/2].

```
210     @staticmethod
211     def show_inventory(table):
212         """Displays current inventory table
213
214
215         Args:
216             table (list of objects): 2D data structure (list of objects) that holds the data during runtime.
217
218         Returns:
219             None.
220
221         """
222         print('======= The Current Inventory: =======')
223         print('ID\tCD Title (by: Artist)\n')
224         for row in table:
225             print(row.__str__())
226         print('======================================')
227
228     @staticmethod
229     def user_input(table):
230         """ Ask user for new ID, CD Title, and Artist and creates a new object that contains CD record info
231
232         Args:
233             table (list of objects): 2D data structure (list of objects) that holds the data during runtime.
234
235         Returns:
236             a list of CD objects
237         """
238
239         try:
240             strID = input('Enter ID: ').strip()
241             strTitle = input('What is the CD\'s title? ').strip()
242             stArtist = input('What is the Artist\'s name? ').strip()
243             cdInput = CD(int(strID), strTitle, stArtist) # data type: object
244             table.append(cdInput) # data type: a list of objects
245             return table
246         except ValueError as e:
247             print('That is not an integer!')
248             print('Build in error info:')
249             print(type(e), e, e.__doc__, sep='\n')
250
```

Listing 6 – Presentation Section: IO class. show_inventory() and user_input() functions are used for displaying the current inventory and reading user input for CD data (ID, title, artist), respectively [2/2].

The main body of the script is shown in Listing **7** and **8**. Lines 256 to 261 in **Listing 7** checks for the CDInventory.dat file in the directory using *os.path* module and *os.path.isfile()* function. For the first run of the script, the binary file does not exist. Therefore, the inventory is loaded to memory from the text file. For the subsequent runs, the inventory is loaded from the binary file (CDInventory.dat).

```
#--- Main Body of Script ---#

# 1. When program starts, read in the currently saved Inventory from the .dat file or .txt file
# Load data from file into a list of CD objects on script start
if os.path.isfile('CDInventory.dat'): # if "CDInventory.dat" exits, use function "read_file()"
    strFileName = 'CDInventory.dat' # binary file to read the data from
    lstOfCDObjects = FileIO.read_file(strFileName)
else:                                 # Else, use function "read_Textfile()"
    strFileName = 'CDInventory.txt' # text file to read the data from
    FileIO.read_Textfile(strFileName,lstOfCDObjects)


# 2. start main loop
while True:
    # 2.1 Display Menu to user and get choice
    IO.print_menu()
    strChoice = IO.menu_choice()

    # 3. Process menu selection
    # 3.1 process exit first
    if strChoice == 'x':
        break
    # 3.2 process display current inventory
    if strChoice == 'i':
        IO.show_inventory(lstOfCDObjects) # displays the current CD inventory
        continue  # start loop back at top.
    # 3.3 process add a CD
    elif strChoice == 'a':
        IO.user_input(lstOfCDObjects) # returns a list of CD objects
        print() # add a space
        IO.show_inventory(lstOfCDObjects) # displays the current CD inventory
    # 3.4 process save inventory to file
    elif strChoice == 's':
        # 3.6.1 Display current inventory and ask user for confirmation to save
        IO.show_inventory(lstOfCDObjects) # displays the current CD inventory
        strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
        # 3.6.2 Process choice
        if strYesNo == 'y':
            # 3.6.2.1 save data
            FileIO.write_file(save_FileName, lstOfCDObjects) # saves the current CD inventory to binary file
        else:
            input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
        continue  # start loop back at top.
```

Listing 7 – Main body of the script [1/2].

```
    # 3.5 process load inventory
    elif strChoice == 'l':
        if os.path.isfile('CDInventory.dat'): # if "CDInventory.dat" exits, use function "read_file()"
            strFileName = 'CDInventory.dat'
            lstOfCDObjects = FileIO.read_file(strFileName) # reads from binary file
            IO.show_inventory(lstOfCDObjects) # displays the current CD inventory
        else:                                 # Else, use function "read_Textfile()"
            strFileName = 'CDInventory.txt'
            FileIO.read_Textfile(strFileName,lstOfCDObjects) # reads from text file
    # 3.6 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
    else:
        print('General Error')
```

Listing 8– Main body of the script [2/2].

Successful operation of the script in Spyder IDE was provided in **Figure 1** and **2**.

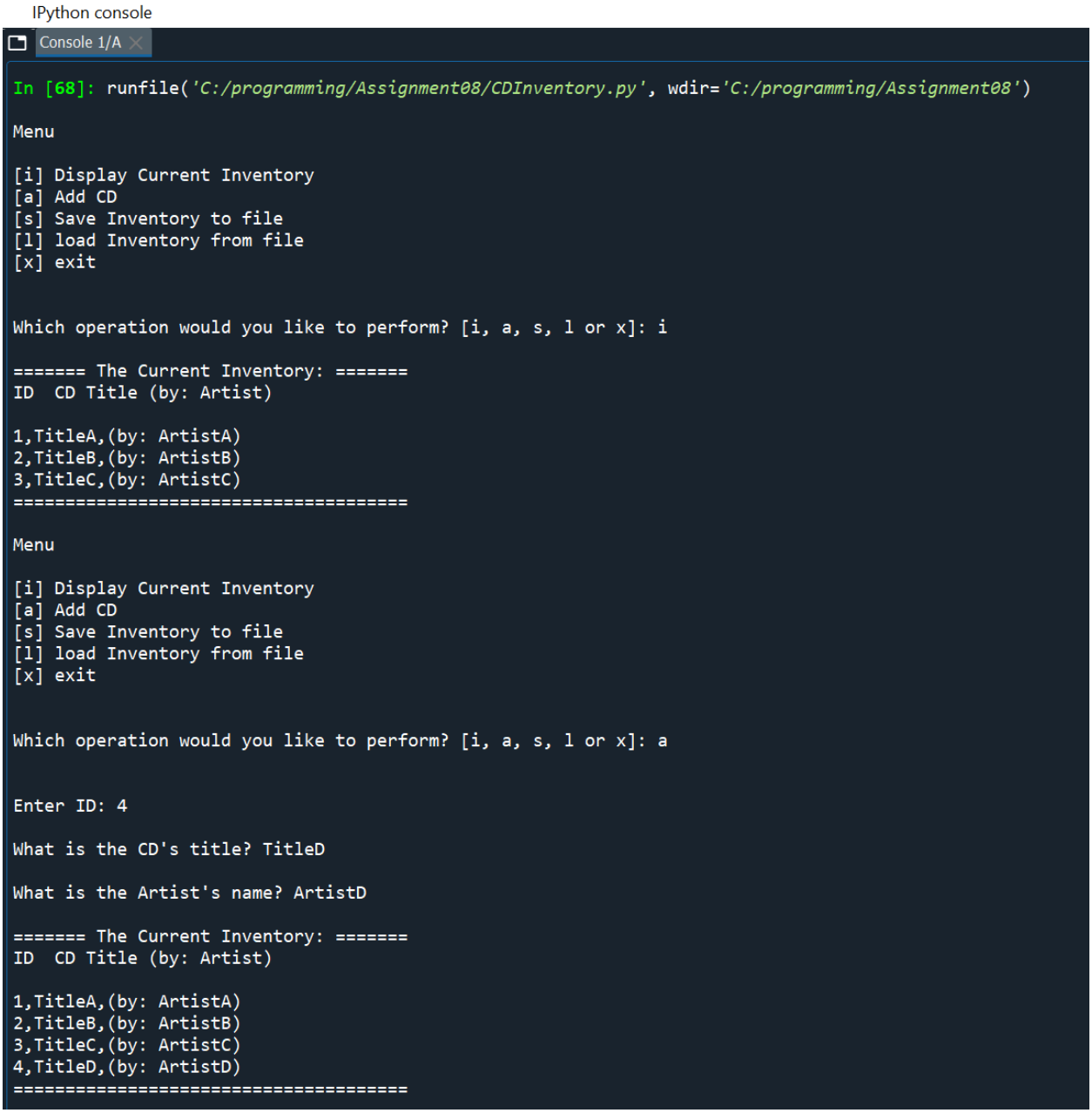

Figure 1– Successful run of the script in Spyder IDE [1/2].

IPython console

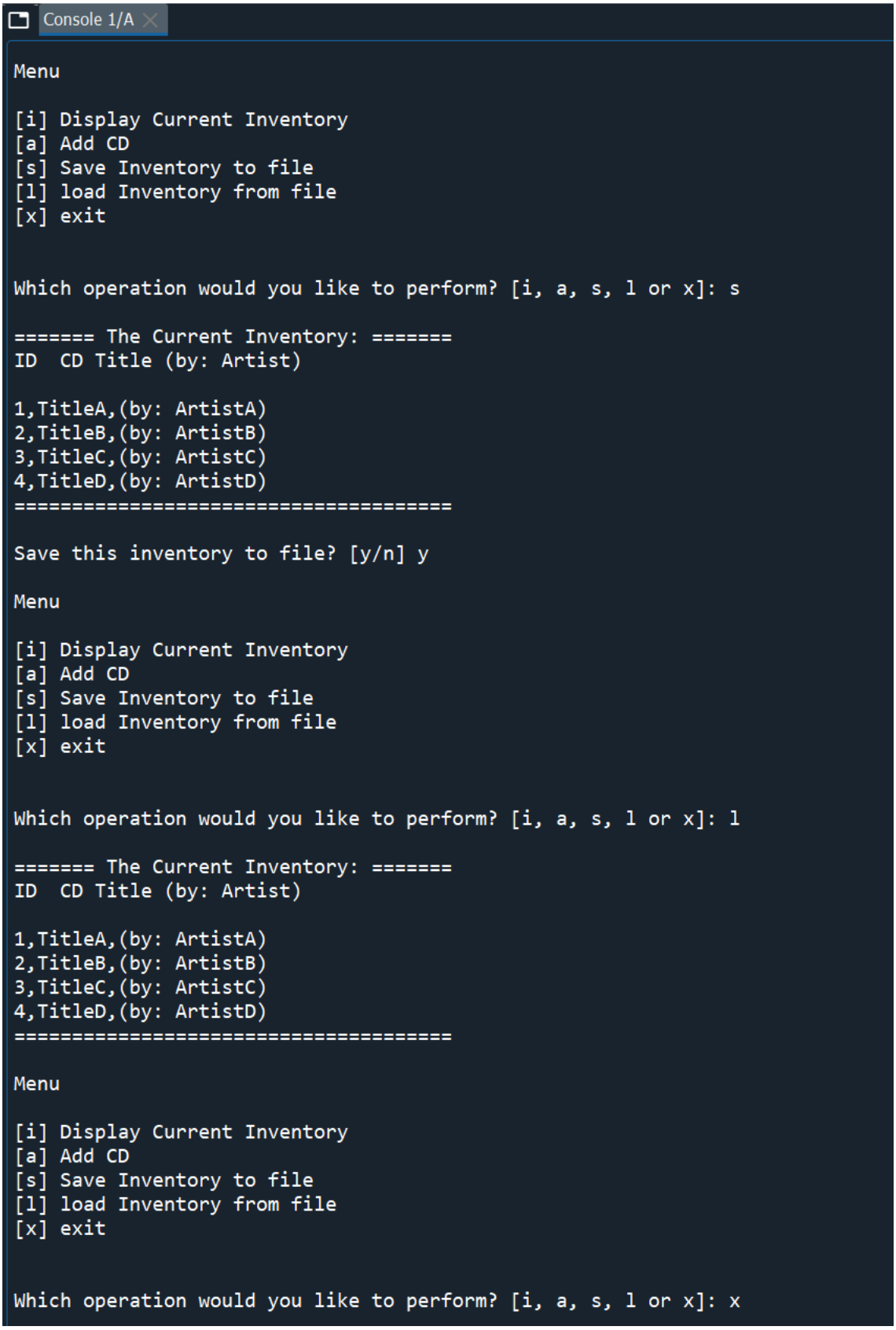

Figure 2– Successful run of the script in Spyder IDE [2/2].

Successful operation of the script in Anaconda Terminal was provided in **Figure 4** and **5**. The screenshot of the CDInventory.txt, from which the CD inventory data is loaded to the memory for the first time running of the script, is given in **Figure 6**.

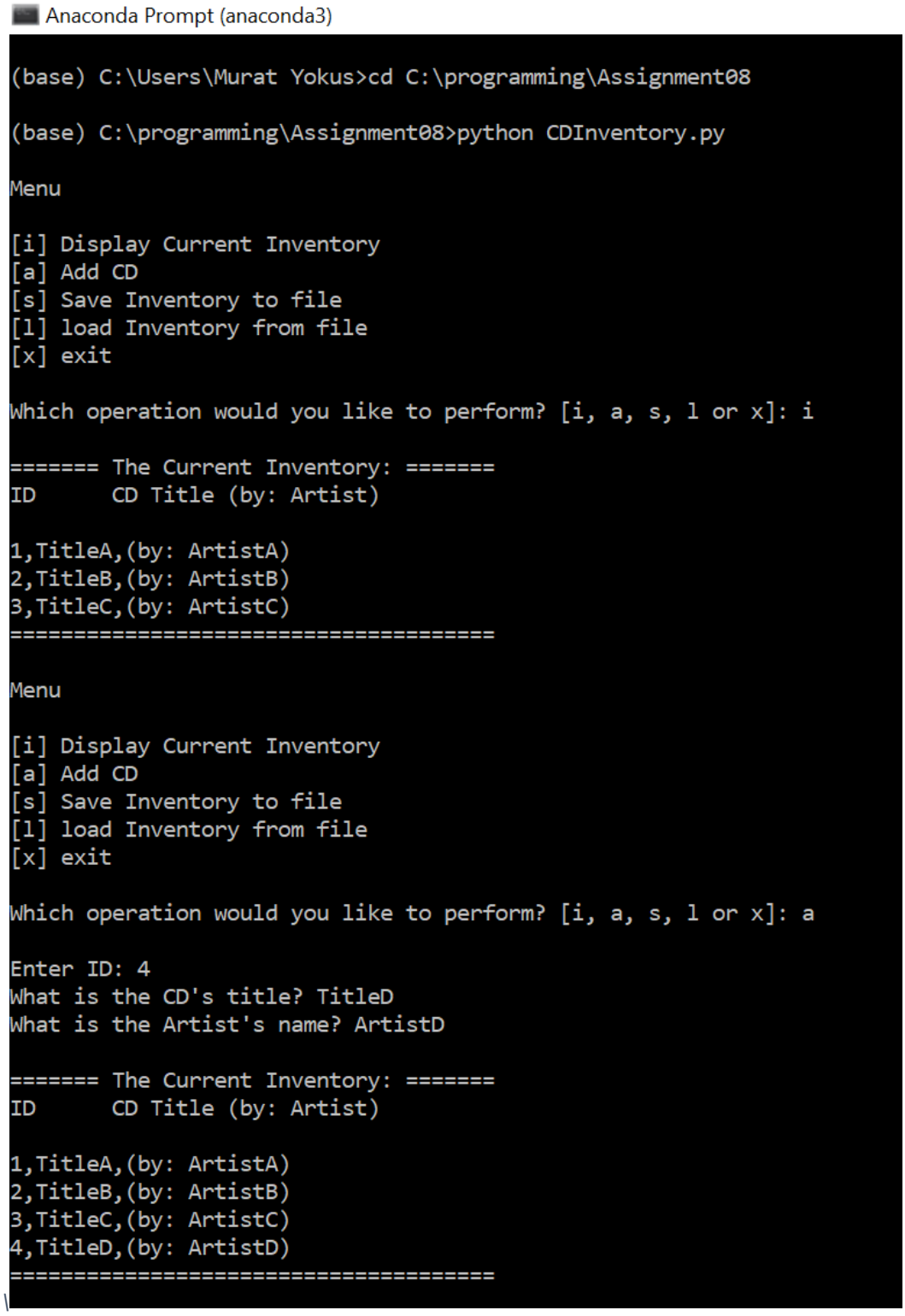

Figure 3 – Successful run of the script in terminal [1/2].

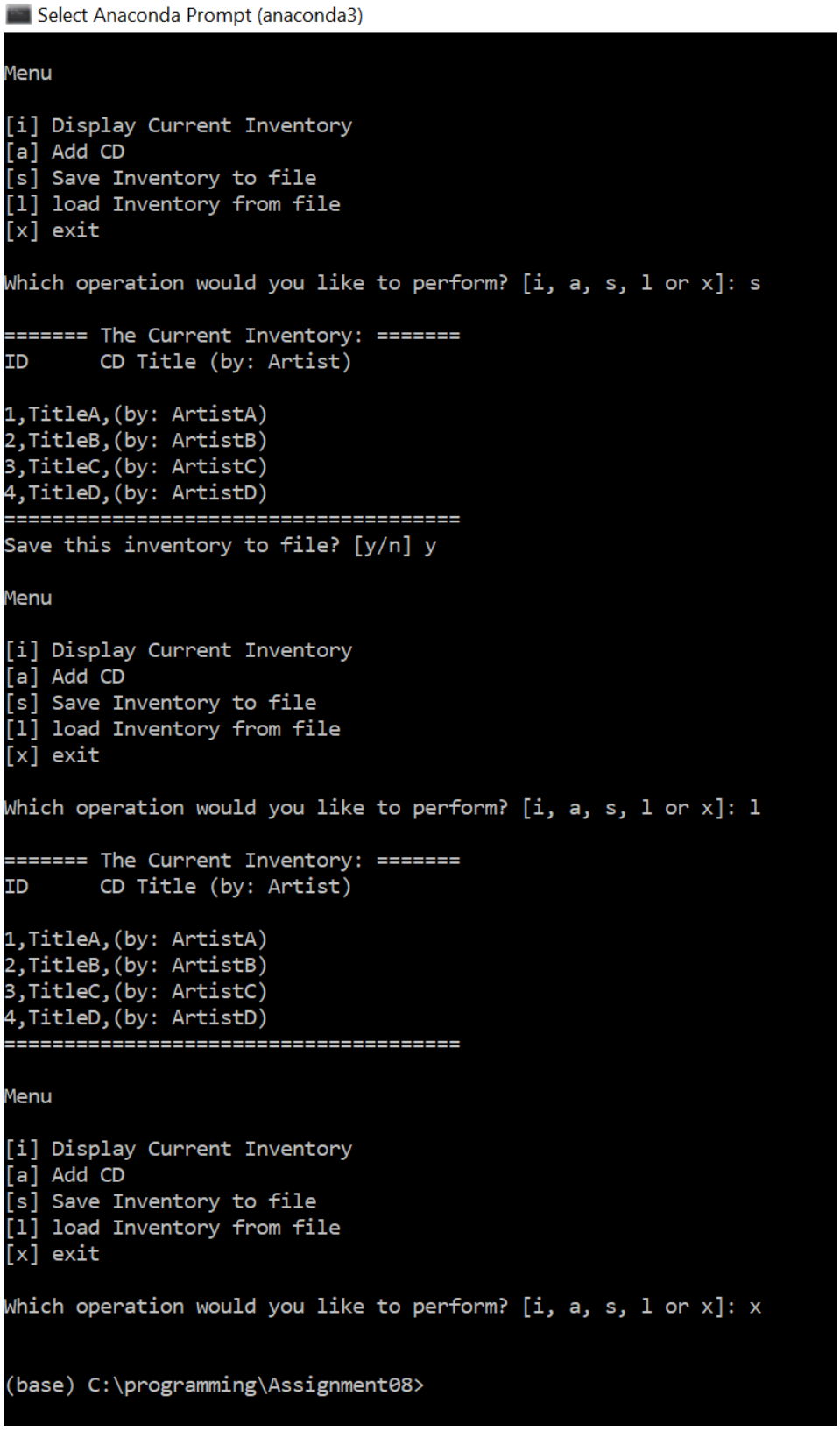

Figure 4 – Successful run of the script in terminal [2/2].

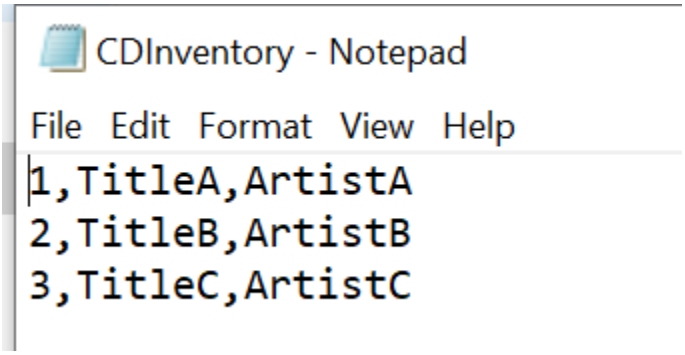

Figure 5– Content of the CDInventory.txt file. This text file was initially used by the script to load the current inventory data to the memory if the CDInventory.dat file does not exist in the file directory.

GitHub Link

The knowledge document, the script, and CDInventory.txt file were uploaded to GitHub/Assignment_08 repository. Link: https://github.com/myokus/Assignment_08

Module 7: Learnings

In the Module 8, I learned the definitions of the following items and practiced them in the course labs.

- Classes, objects, constructors, attributes, properties, methods, static methods, and fields.

Summary

Overall, the objective of this assignment was to implement classes and objects. This document showed step-by-step implementation and operation of Classes and Objects for creation and data manipulation of a CD inventory. Additionally, the script included structured error handing and data storage/reading using binary files. Potential build-in Python errors (user interaction, type casting (*e.g.,* string to integer), or file access operations) were handled using try-except blocks.

Appendix

Listing CDInventory.py

```
#------------------------------------------#
# Title: CDInventory.py
# Desc: Assignnment 08 - Working with classes
# Change Log: (Who, When, What)
# DBiesinger, 2030-Jan-01, created file
# DBiesinger, 2030-Jan-01, added pseudocode to complete assignment 08
# MYokus, 2021-Aug-29, Added Code to work with classes and objects
#------------------------------------------#

import pickle # module for handling binary files
import os.path # module for common pathname manipulation

# -- DATA -- #
strFileName = ''
lstOfCDObjects = []
save_FileName = 'CDInventory.dat'  # data storage file to save the data to
```

```
17.
18. class CD:
19.     """Stores data about a CD:
20.
21.    properties:
22.        cd_id: (int) with CD ID
23.        cd_title: (string) with the title of the CD
24.        cd_artist: (string) with the artist of the CD
25.    methods:
26.        __str__(): Creates a string for CD attributes
27.
28.    """
29.     # --Fields --#
30.     # -- Constructor -- #
31.     def __init__(self,ID, title, artist):
32.         # --Attributes -- #
33.         self.__cd_id = ID
34.         self.__cd_title = title
35.         self.__cd_artist = artist
36.     # -- Properties -- #
37.     @property
38.     def cd_id(self):
39.         return self.__cd_id
40.
41.     @cd_id.setter
42.     def cd_id(self, value):
43.         if type(value) == int:
44.             self.__cd_id = value
45.         else:
46.             raise Exception('Position needs to be integer!')
47.
48.     @property
49.     def cd_title(self):
50.         return self.__cd_title
51.
52.     @cd_title.setter
53.     def cd_title(self, value):
54.         if type(value) == str:
55.             self.__cd_title = value
56.         else:
57.             raise Exception('Title needs to be string')
58.
59.     @property
60.     def cd_artist(self):
61.         return self.__cd_artist
62.
63.     @cd_artist.setter
64.     def cd_artist(self, value):
65.         if type(value) == str:
66.             self.__cd_artist = value
67.         else:
68.             raise Exception('Length needs to be string')
69.     # -- Methods -- #
70.     def __str__(self):
71.         return (str(self.cd_id) + ',' + self.cd_title + ',' + '(by: ' + self.cd_artist + ')')
72.
73.
74. # -- PROCESSING -- #
75. class FileIO:
76.     """Processes data to and from file:
77.
78.    properties:
79.
80.    methods:
81.        read_Textfile(file_name, table): -> A list of CD objects
```

```
        read_file(file_name): -> A list of CD objects
        write_file(file_name, table): -> None
    """
    # --Fields --#
    # -- Constructor -- #
            # --Attributes -- #
    # -- Properties -- #
    # -- Methods -- #

    @staticmethod
    def read_Textfile(file_name, table):
        """Function to manage data ingestion from a text file to a list of objects

        Reads the data from a text file identified by file_name into a 2D table
        (list of objects) table one line in the file represents one object row in table.

        Args:
            file_name (string): name of the text file used to read the data from
            table (list of objects): 2D data structure (list of objects) that holds the data during runtime

        Returns:
            None.
        """

        try:
            table.clear()  # this clears existing data and allows to load data from file
            objFile = open(file_name, 'r')
            for line in objFile:
                data = line.strip().split(',') # data type: list
                cdRow = CD(int(data[0]), data[1], data[2]) # data type: object
                table.append(cdRow) # list of objects
            objFile.close()
        except FileNotFoundError as e:
            print('Text file does not exist!')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')

    @staticmethod
    def read_file(file_name):
        """Function to manage data ingestion from a binary file to a list of objects

        Reads the data from a binary file identified by file_name into a 2D table
        (list of objects) table one line in the file represents one object row in table.

        Args:
            file_name (string): name of the binary file used to read the data from

        Returns:
            data (list of objects): 2D data structure (list of objects)
        """

        try:
            data = []
            with open(file_name, 'rb') as fileObj:
                data = pickle.load(fileObj)
            return data
        except FileNotFoundError as e:
            print('Binary file does not exist!')
            print('Build in error info:')
            print(type(e), e, e.__doc__, sep='\n')

    @staticmethod
    def write_file(file_name, table):
        """Function to save a 2D table (a list of objects) to file via pickle
```

```
146.
147.         Saves the data in a file identified by file_name into a .dat file
148.
149.         Args:
150.             file_name (string): name of binary file used to save the data to
151.             table (list of objects): 2D data structure (list of objects) that holds the data during
    runtime
152.
153.         Returns:
154.             None.
155.         """
156.          try:
157.              with open(file_name, 'wb') as fileObj:
158.                  pickle.dump(table, fileObj)
159.          except FileNotFoundError as e:
160.              print('Binary file does not exist!')
161.              print('Build in error info:')
162.              print(type(e), e, e.__doc__, sep='\n')
163.
164.
165.  # -- PRESENTATION (Input/Output) -- #
166.  class IO:
167.      """Handling Input / Output (User Interaction)
168.
169.     properties:
170.
171.     methods:
172.         print_menu(): -> Displays a menu of choices to the user
173.         menu_choice(): -> Gets user input for menu selection
174.         show_inventory(table): -> Displays current inventory table
175.         user_input(table): -> Ask user for new ID, CD Title, and Artist and creates a new object
176.
177.     """
178.
179.      @staticmethod
180.      def print_menu():
181.          """Displays a menu of choices to the user
182.
183.         Args:
184.             None.
185.
186.         Returns:
187.             None.
188.         """
189.
190.          print('\nMenu\n\n[i] Display Current Inventory\n[a] Add CD\n[s] Save Inventory to file')
191.          print('[l] load Inventory from file\n[x] exit\n')
192.
193.      @staticmethod
194.      def menu_choice():
195.          """Gets user input for menu selection
196.
197.         Args:
198.             None.
199.
200.         Returns:
201.             choice (string): a lower case sting of the users input out of the choices i, a, s, l, or x
202.
203.         """
204.          choice = ' '
205.          while choice not in ['i', 'a', 's', 'l', 'x']: # 'While not loop: executes the body of the
    loop until the condition for loop termination is met'
206.              choice = input('Which operation would you like to perform? [i, a, s, l or x]:
    ').lower().strip()
207.          print()  # Add extra space for layout
```

```
208.          return choice
209.
210.      @staticmethod
211.      def show_inventory(table):
212.          """Displays current inventory table
213.
214.
215.         Args:
216.             table (list of objects): 2D data structure (list of objects) that holds the data during
    runtime.
217.
218.         Returns:
219.             None.
220.
221.         """
222.          print('======= The Current Inventory: =======')
223.          print('ID\tCD Title (by: Artist)\n')
224.          for row in table:
225.              print(row.__str__())
226.          print('======================================')
227.
228.      @staticmethod
229.      def user_input(table):
230.          """ Ask user for new ID, CD Title, and Artist and creates a new object that contains CD
    record info
231.
232.         Args:
233.             table (list of objects): 2D data structure (list of objects) that holds the data during
    runtime.
234.
235.         Returns:
236.             a list of CD objects
237.         """
238.
239.          try:
240.              strID = input('Enter ID: ').strip()
241.              strTitle = input('What is the CD\'s title? ').strip()
242.              stArtist = input('What is the Artist\'s name? ').strip()
243.              cdInput = CD(int(strID), strTitle, stArtist) # data type: object
244.              table.append(cdInput) # data type: a list of objects
245.              return table
246.          except ValueError as e:
247.              print('That is not an integer!')
248.              print('Build in error info:')
249.              print(type(e), e, e.__doc__, sep='\n')
250.
251.
252.  # -- Main Body of Script -- #
253.
254.  # 1. When program starts, read in the currently saved Inventory from the .dat file or .txt file
255.  # Load data from file into a list of CD objects on script start
256.  if os.path.isfile('CDInventory.dat'): # if "CDInventory.dat" exits, use function "read_file()"
257.      strFileName = 'CDInventory.dat' # binary file to read the data from
258.      lstOfCDObjects = FileIO.read_file(strFileName)
259.  else:                                 # Else, use function "read_Textfile()"
260.      strFileName = 'CDInventory.txt' # text file to read the data from
261.      FileIO.read_Textfile(strFileName,lstOfCDObjects)
262.
263.
264.  # 2. start main loop
265.  while True:
266.      # 2.1 Display Menu to user and get choice
267.      IO.print_menu()
268.      strChoice = IO.menu_choice()
269.
```

```
    # 3. Process menu selection
    # 3.1 process exit first
    if strChoice == 'x':
        break
    # 3.2 process display current inventory
    if strChoice == 'i':
        IO.show_inventory(lstOfCDObjects) # displays the current CD inventory
        continue  # start loop back at top.
    # 3.3 process add a CD
    elif strChoice == 'a':
        IO.user_input(lstOfCDObjects) # returns a list of CD objects
        print() # add a space
        IO.show_inventory(lstOfCDObjects) # displays the current CD inventory
    # 3.4 process save inventory to file
    elif strChoice == 's':
        # 3.6.1 Display current inventory and ask user for confirmation to save
        IO.show_inventory(lstOfCDObjects) # displays the current CD inventory
        strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
        # 3.6.2 Process choice
        if strYesNo == 'y':
            # 3.6.2.1 save data
            FileIO.write_file(save_FileName, lstOfCDObjects) # saves the current CD inventory to
binary file
        else:
            input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
        continue  # start loop back at top.
    # 3.5 process load inventory
    elif strChoice == 'l':
        if os.path.isfile('CDInventory.dat'): # if "CDInventory.dat" exits, use function
"read_file()"
            strFileName = 'CDInventory.dat'
            lstOfCDObjects = FileIO.read_file(strFileName) # reads from binary file
            IO.show_inventory(lstOfCDObjects) # displays the current CD inventory
        else:                                 # Else, use function "read_Textfile()"
            strFileName = 'CDInventory.txt'
            FileIO.read_Textfile(strFileName,lstOfCDObjects) # reads from text file
    # 3.6 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
    else:
        print('General Error')
```