Nick Culbert

2021 Mar 10

IT FDN 110 A Wi 21

Assignment 08

Object-Oriented Programming

# Introduction

For this assignment, I modified my “CDInventory.py” script from the previous assignment to incorporate elements of object-oriented programming, including storing and manipulating the inventory of CDs with a list of custom objects rather than a list of objects made from the built-in dictionary class.

Note: planetb returned an internal service error, so I used the site [hilite.me](http://hilite.me/) [[1]](#footnote-1)(external reference) to highlight my code for this document.

# Research

Before beginning to write code, I first did further research on object-oriented programming since the concept felt very difficult to grasp.

This YouTube video, “[OOP in Python | Object Oriented Programming](https://www.youtube.com/watch?v=qiSCMNBIP2g)” [[2]](#footnote-2) (external reference) made the concept of the class as a ‘blueprint’ and object as an instance of something which uses that blueprint very clear by providing several different examples. It also touched on the four pillars of OOP (I did not watch the last 20 minutes or so because that went too in-depth into unfamiliar concepts).

# # -- DATA -- #

Since this script would be relying on a list of objects, I only needed to create global variables for the file name and that list (Figure 1).

strFileName = 'CDInventory.txt'

lst\_Inventory = []

Figure - Global variables

# class CD

This class is in the Data section of the script because it does not actually manipulate the data values which it takes as inputs. Instead, it instructs the way in which objects created from the class store their data.

It begins by initializing the three fields from what its objects will receive as input. I did not need to define these as class variables before initializing since Python creates them implicitly. To practice the convention, I also made the fields private by adding double underscores to their beginnings (Figure 2).

**def** **\_\_init\_\_**(self, cd\_id, cd\_title, cd\_artist):

# -- Attributes -- #

self.\_\_cd\_id = cd\_id

self.\_\_cd\_title = cd\_title

self.\_\_cd\_artist = cd\_artist

Figure – Initialization

Then, I added a pair of getter and setter methods for each field. Doing so allows access to these private fields while separating the input and output functionality (Figure 3). While I could have included error handling in the setter function for the cd\_id field, I decided to leave this error handling in the get\_new\_CD method under the IO class because it would involve printing to the user, which would not adhere to the SoC (Figure 4).

# Getters

**@property**

**def** **cd\_id**(self):

**return** self.\_\_cd\_id

**@property**

**def** **cd\_title**(self):

**return** self.\_\_cd\_title

**@property**

**def** **cd\_artist**(self):

**return** self.\_\_cd\_artist

# Setters

**@cd\_id**.setter

**def** **cd\_id**(self, value):

self.\_\_cd\_id = value

**@cd\_title**.setter

**def** **cd\_title**(self, value):

self.\_\_cd\_title = value

**@cd\_artist**.setter

**def** **cd\_artist**(self, value):

self.\_\_cd\_artist = value

Figure - Getters and setters

cd\_id = ''

**while** type(cd\_id) == str:

**try**:

cd\_id = int(input('Enter ID: ').strip())

**except** **ValueError** **as** e:

print('Fail! You did not enter an integer.**\n**'

f'Error encountered: {e}**\n**'

'Please enter an integer....')

**except** **Exception** **as** e:

print('There was an error...**\n**'

f'Error encountered: {e}**\n**')

cd\_title = input('What is the CD**\'**s title? ').strip()

cd\_artist = input('What is the Artist**\'**s name? ').strip()

**return** cd\_id, cd\_title, cd\_artist

Figure - Error handling

In general, adding error handling within the class allows for ‘abstraction’, which basically means that programmers can create objects and manipulate them or let them interact with each other without worrying about the object’s underlying functionality.

Creating this CD class was the largest change from the previous assignment. The majority of the rest of the script pulls from my previous code, leaves the static methods intact, and makes minor adjustments to account for using the custom CD objects.

# # -- PROCESSING -- #

# class FileIO

The load\_inventory and save\_inventory methods functions exactly the same as before, with variable names changed to reflect Dirk’s conventions in the starter script.

# class IO

Since the ‘delete CD’ functionality was not included in the starter script, I removed that from the print\_menu and menu\_choices methods.

Then, I moved the code from the show\_inventory method into the main body and deleted that method. There, I iterated through the list of CD objects and, each time, referenced the relevant getter functions to return and print the data stored in those objects (Figure 5). If I had kept the show\_inventory method in the IO class, it would have violated the principle of not calling custom classes within other custom classes.

# show user current inventory

**elif** strChoice == 'i':

print()

print('======= The Current Inventory: =======')

print('ID**\t**CD Title (by: Artist)**\n**')

**for** CD **in** lst\_Inventory:

print(f'{CD.cd\_id}**\t**{CD.cd\_title} (by:{CD.cd\_artist})')

print('======================================')

print()

**continue**

Figure - Displaying the current inventory

# Main Body

Under the option for adding a new CD, I first declared and assigned new variables by calling the get\_new\_CD method from the IO class. Then, I created an object called newCD by passing these variables into the CD class. Finally, I appended that object to the list of objects (Figure 6). Creating a new object using the same variable name, newCD, works because each separate instance creates a new reference within the computer’s memory, and what is being appended to the list of objects is that unique reference.

# let user add data to the inventory

**elif** strChoice == 'a':

cd\_id, cd\_title, cd\_artist = IO.get\_new\_CD()

newCD = CD(cd\_id, cd\_title, cd\_artist)

lst\_Inventory.append(newCD)

IO.show\_inventory(lst\_Inventory)

**continue**

Figure - Adding data to the inventory

# Running the script

Figures 7 and 8 are screenshots of my script running in Spyder and Terminal, respectively.

Text

Description automatically generated

Figure 7 - Running the script in Spyder

Text

Description automatically generated

Figure 8 - Running the script in Terminal

# Summary

For this assignment, I modified my previous code to use elements of object-oriented programming. At this point, my two biggest questions are: were there aspects of this program that could have been turned into objects that I missed? Should I have put the error handling in the CD class?

# Appendix

#------------------------------------------#

# Title: Assignmen08.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

# nickculbert, 2021-Mar-10, added main body

#------------------------------------------#

# -- DATA -- #

**import** **pickle**

strFileName = 'CDInventory.txt'

lst\_Inventory = []

**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:

None.

"""

# -- Constructor -- #

**def** **\_\_init\_\_**(self, cd\_id, cd\_title, cd\_artist):

# -- Attributes -- #

self.\_\_cd\_id = cd\_id

self.\_\_cd\_title = cd\_title

self.\_\_cd\_artist = cd\_artist

# -- Properties -- #

# Getters

**@property**

**def** **cd\_id**(self):

**return** self.\_\_cd\_id

**@property**

**def** **cd\_title**(self):

**return** self.\_\_cd\_title

**@property**

**def** **cd\_artist**(self):

**return** self.\_\_cd\_artist

# Setters

**@cd\_id**.setter

**def** **cd\_id**(self, value):

self.\_\_cd\_id = value

**@cd\_title**.setter

**def** **cd\_title**(self, value):

self.\_\_cd\_title = value

**@cd\_artist**.setter

**def** **cd\_artist**(self, value):

self.\_\_cd\_artist = value

# -- PROCESSING -- #

**class** **FileIO**:

"""Processes data to and from file:

properties:

None.

methods:

save\_inventory(file\_name, lst\_Inventory): -> None

load\_inventory(file\_name): -> (a list of CD objects)

"""

# -- Methods -- #

**@staticmethod**

**def** **load\_inventory**(file\_name, lst\_Inventory):

"""Ingests data from file to a list of CD objects

Reads the pickled data from file identified by file\_name and assigns

that data to the field lst\_Inventory.

Args:

file\_name (string): name of file used to read the data from

Returns:

lst\_Inventory (list of CDs): holds the inventory during runtime

"""

lst\_Inventory.clear()

**try**:

**with** open(file\_name, 'ab+') **as** file:

file.seek(**0**)

lst\_Inventory = pickle.load(file)

**except** **EOFError**:

print('**\n**Your CD Inventory is empty!**\n**')

**except** **Exception** **as** e:

print('There was an error...**\n**'

f'Error encountered: {e}**\n**')

**finally**:

**return** lst\_Inventory

**@staticmethod**

**def** **save\_inventory**(file\_name, lst\_Inventory):

"""Pickles current inventory data to text file

Args:

file\_name (string): name of file used to read the data from

lst\_Inventory (list of CDs): holds the inventory during runtime

Returns:

None.

"""

**with** open(file\_name, 'ab') **as** file:

pickle.dump(lst\_Inventory, file)

# -- PRESENTATION (Input/Output) -- #

**class** **IO**:

"""Handling user Input / Output"""

**@staticmethod**

**def** **print\_menu**():

"""Displays a menu of choices to the user

Args:

None.

Returns:

None.

"""

print('==== Menu === **\n\n**'

'[l] load Inventory from file **\n**'

'[a] Add CD**\n**[i] Display Current Inventory **\n**'

'[s] Save Inventory to file **\n**'

'[x] exit **\n**')

**@staticmethod**

**def** **menu\_choice**():

"""Gets user input for menu selection

Args:

None.

Returns:

choice (string): a lower case string of the users input

out of the choices l, a, i, d, s or x

"""

choice = ' '

**while** choice **not** **in** ['l', 'a', 'i', 's', 'x']:

choice = input('Please choose an operation: ').lower().strip()

print() # extra space for layout

**return** choice

**@staticmethod**

**def** **get\_new\_CD**():

"""Gets user input for adding a new CD

Args:

None.

Returns:

cd\_id (integer): CD's ID

cd\_title (string): CD's title

cd\_artist (string): artist's name

"""

cd\_id = ''

**while** type(cd\_id) == str:

**try**:

cd\_id = int(input('Enter ID: ').strip())

**except** **ValueError** **as** e:

print('Fail! You did not enter an integer.**\n**'

f'Error encountered: {e}**\n**'

'Please enter an integer....')

**except** **Exception** **as** e:

print('There was an error...**\n**'

f'Error encountered: {e}**\n**')

cd\_title = input('What is the CD**\'**s title? ').strip()

cd\_artist = input('What is the Artist**\'**s name? ').strip()

**return** cd\_id, cd\_title, cd\_artist

# -- Main Body of Script -- #

# Load data from file into a list of CD objects on script start

lst\_Inventory = FileIO.load\_inventory(strFileName, lst\_Inventory)

**while** **True**:

# Display menu to user

IO.print\_menu()

strChoice = IO.menu\_choice()

# let user exit program

**if** strChoice == 'x':

**break**

# let user load inventory from file

**if** strChoice == 'l':

print('WARNING: If you continue,'

' all unsaved data will be lost'

' and the Inventory re-loaded from file.')

strYesNo = input('Type **\'**yes**\'** to continue'

' and reload from file.'

' Type anything else to cancel the reload: ')

**if** strYesNo.lower() == 'yes':

print('**\n**reloading...**\n**')

lst\_Inventory = FileIO.load\_inventory(strFileName, lst\_Inventory)

**else**:

input('Canceling...Inventory data NOT reloaded.'

' Press [ENTER] to continue to the menu.')

IO.show\_inventory(lst\_Inventory)

**continue**

# let user add data to the inventory

**elif** strChoice == 'a':

cd\_id, cd\_title, cd\_artist = IO.get\_new\_CD()

newCD = CD(cd\_id, cd\_title, cd\_artist)

lst\_Inventory.append(newCD)

IO.show\_inventory(lst\_Inventory)

**continue**

# show user current inventory

**elif** strChoice == 'i':

print()

print('======= The Current Inventory: =======')

print('ID**\t**CD Title (by: Artist)**\n**')

**for** CD **in** lst\_Inventory:

print(f'{CD.cd\_id}**\t**{CD.cd\_title} (by:{CD.cd\_artist})')

print('======================================')

print()

**continue**

# let user save inventory to file

**elif** strChoice == 's':

IO.show\_inventory(lst\_Inventory)

strYesNo = input('Save inventory to file? [y/n] ').strip().lower()

**if** strYesNo == 'y':

FileIO.save\_inventory(strFileName, lst\_Inventory)

print('Inventory saved!**\n**')

**else**:

input('The inventory was NOT saved to file.'

' Press [ENTER] to return to the menu.**\n**')

**continue**

**else**:

print('General Error')

1. <http://hilite.me/> retrieved 2021-Mar-10 [↑](#footnote-ref-1)
2. <https://www.youtube.com/watch?v=qiSCMNBIP2g> retrieved 2021-Mar-08 [↑](#footnote-ref-2)