

# Object-Oriented Programming

---

## Introduction

This assignment will investigate the principles of object-oriented programming (OOP) through a version of the CD Inventory programming that has well-developed classes including constructors, attributes, properties, and methods. The purpose and function of these components will be explained and demonstrated, showing the utility of classes.

## Overview of Classes

Classes are the foundation of OOP. This is because objects simply cannot exist without the class structures that underlie them. Every type of object in Python has a parent class which serves as blueprint of sorts, for determining the object's attributes (or fields), properties, and methods.<sup>1</sup> In other words, the difference between a class and an object is that a class packages the data and functionality that constitutes a specific type of object—and each individual object of that type is merely an instantiation of the class.<sup>2</sup>

```
23.     def __init__(self, ID=None, title="...", artist="..."):
24.         self.__cd_id = ID
25.         self.__cd_title = title
26.         self.__cd_artist = artist
27.
28.     @property
29.     def cd_id(self):
30.         return self.__cd_id
31.
32.     @cd_id.setter
33.     def cd_id(self, t_ID):
34.         if str(t_ID).isalpha():
35.             raise Exception("Numeric value required for ID.")
36.         else:
37.             self.__cd_id = t_ID
```

*Listing 1 - Excerpt, Appendix A: Components of a class (constructor, attributes/fields, and properties)*

```
74.     @staticmethod
75.     def add_cd(addID, addtitle, addartist, table):
...
87.         newCD = CD(addID, addtitle, addartist)
88.         table.append(newCD)
89.         print()
```

*Listing 2 - Excerpt, Appendix A: Components of a class (methods)*

Above, Listings 1 and 2 collectively contain the standard components of a class construct (although the two Listings are excerpted from different classes within Appendix A). These are the constructor (line 23), the attributes (lines 24-26), the

---

<sup>1</sup> "Chapter 7 Software Objects: The Critter Caretaker Program", Python Programming for the Absolute Beginner Third Edition, Michael Dawson, p219-220.

<sup>2</sup> "Module 08 of 10", *Foundations of Programming (Python)*, Dirk Biesinger, p2 (accessed August 31, 2020).

a property (lines 28-37), all part of class CD, and a method (lines 74-89) that is part of class DataProcessor which acts on data provided by CD. This interconnected relationship between two classes exhibiting different components of the class construct is not unusual, and in fact represents appropriate organization of operational sections of code per the principles of OOP. Class CD is constructed to build objects which collate user-input data characterizing a CD to be recorded in a file, and the `add_cd()` method within DataProcessor creates an instantiation of the CD class—a CD object—that is then stored in a runtime data structure. The partition between preparation and creation of a CD object across the two classes is clear.

### Constructor

The class constructor is a highly utilized component of the class construct which has multiple uses. First and foremost, constructors are used to establish default values for the attributes of an object. They may also be used to perform operations which are executed concurrently with the object's creation—for example, a `print()` function containing a message which announces that the object was created.<sup>3</sup> What all of the constructor's uses have in common is the fact that they are limited to running one time, that being when the object is created—which is when the constructor runs.<sup>4</sup>

### Fields, Attributes, and Properties

Fields and attributes are components which provide two approaches for satisfying the same common operating requirement of a class: storage of data within objects instantiated by the class.<sup>5</sup> The difference is that fields are invoked separately from the point of object creation by the class constructor, whereas attributes are created as a byproduct of object creation, and so must be provided values when the constructor runs. Listing 1 provides an example of attributes, three of which are initialized on lines 24-26.

Properties have an equivalent relationship to both fields and attributes. They intermediate between user input of data for object parameters and the instantiation of an object in order to allow error handling and formatting as necessary. Moreover, in the case of attributes which are created implicitly during object instantiation and may be rendered “private” outside of the constructor, properties are essential to accessing and manipulating their private data.<sup>6</sup>

### The ‘self’ Keyword

Notice that the keyword ‘self’ is invoked repeatedly in Listing 1. It is not an official Python keyword, and each mention of it could be replaced with an arbitrary term so long as it was consistently applied—but the use of self in this context has become a universal norm. This is because self is used to signify the first parameter of every class method, and the first parameter of a method serves as a reference to the object it is acting on.<sup>7</sup> Thus, it makes the word self is used to refer to the object “itself”. Perhaps the most straightforward example in Listing 1 of how this works is on lines 24-26, where the parameter self is used to initiate the three attributes associated with the constructor. This instructs Python that user input data for those three attributes are to be passed along as parameters to the CD object presently being created.

### The ‘@staticmethod’ Keyword

On the other hand, the keyword ‘@staticmethod’ on line 74 in Listing 2 is used for almost diametrically opposed reasons to ‘self’, albeit with different implementation. The fundamental difference was alluded to in describing how the DataProcessor class, to which the `add_cd()` method belongs, acts on the CD class—take note that `add_cd()` never references self. This is because the method processes data in the context of the overall class rather than its instantiations in the form of objects. Labelling a method with `@staticmethod` indicates that it is

---

<sup>3</sup> “Chapter 7”, Michael Dawson, p222-224.

<sup>4</sup> “Module 08 of 10”, Diek Biesinger, p4 (accessed August 31, 2020).

<sup>5</sup> Ibid, p3 & p5 (accessed August 31, 2020).

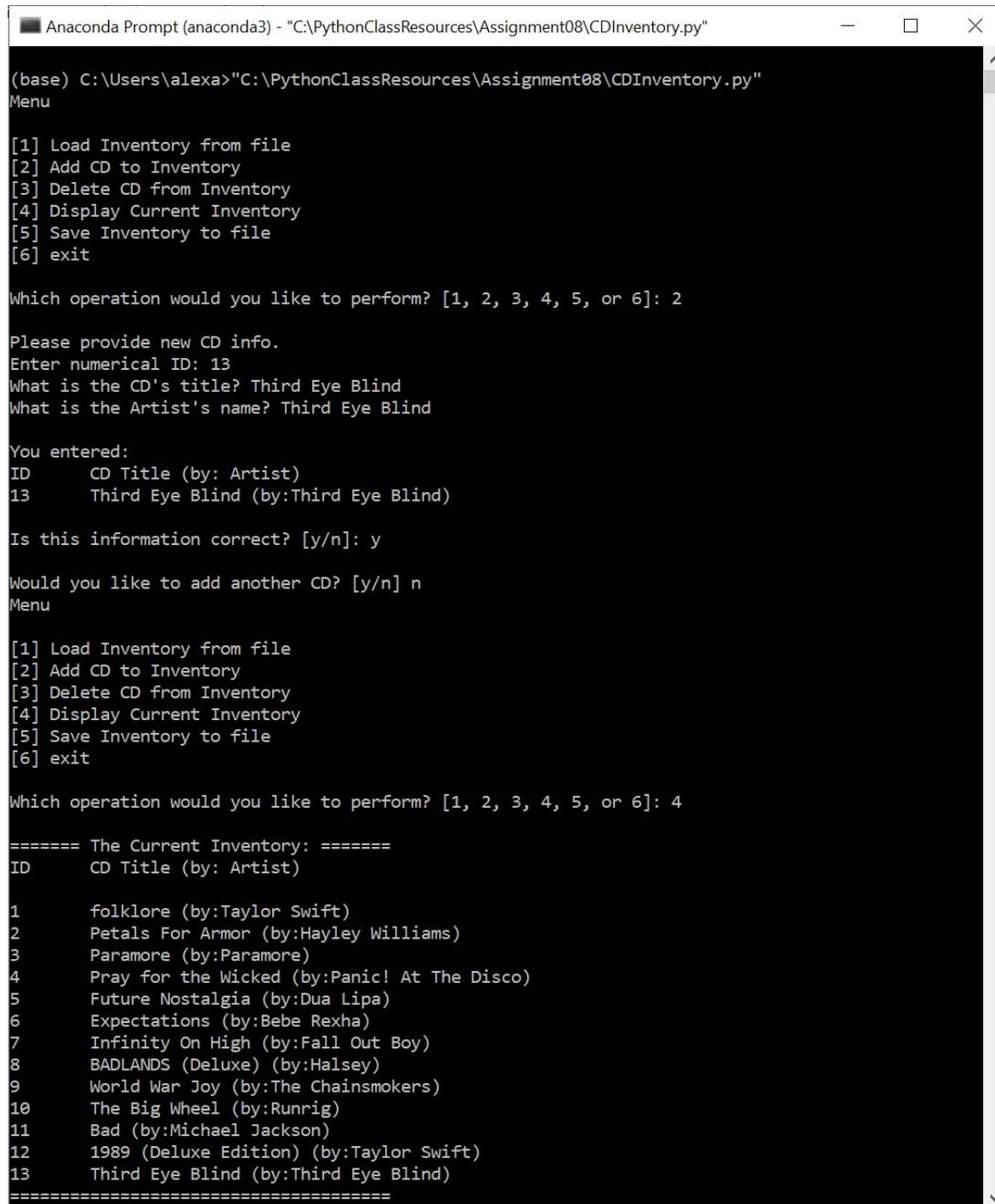
<sup>6</sup> Ibid, p6-7 (accessed August 31, 2020).

<sup>7</sup> Ibid, p5 (accessed August 31, 2020).

supposed to behave this way. In the case of `add_cd()`, the way that this relationship works is that it takes CD objects already formed by the CD class and inserts them into a runtime data structure.<sup>8</sup>

The nature of this relationship also demonstrates the fundamental difference between methods and properties. Where methods only act on a whole class of objects, manipulating their data after instantiation, properties are conversely limited to controlling data on an individual object level at the time the object is instantiated.

Although the totality of these components as parts of a functioning class may appear complicated, classes easily fulfill the requirements of a program like CD Inventory while providing a user-friendly experience. See figure 1 below.



```
(base) C:\Users\alexa>"C:\PythonClassResources\Assignment08\CDInventory.py"
Menu

[1] Load Inventory from file
[2] Add CD to Inventory
[3] Delete CD from Inventory
[4] Display Current Inventory
[5] Save Inventory to file
[6] exit

Which operation would you like to perform? [1, 2, 3, 4, 5, or 6]: 2

Please provide new CD info.
Enter numerical ID: 13
What is the CD's title? Third Eye Blind
What is the Artist's name? Third Eye Blind

You entered:
ID      CD Title (by: Artist)
13      Third Eye Blind (by:Third Eye Blind)

Is this information correct? [y/n]: y

Would you like to add another CD? [y/n] n
Menu

[1] Load Inventory from file
[2] Add CD to Inventory
[3] Delete CD from Inventory
[4] Display Current Inventory
[5] Save Inventory to file
[6] exit

Which operation would you like to perform? [1, 2, 3, 4, 5, or 6]: 4

===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       folklore (by:Taylor Swift)
2       Petals For Armor (by:Hayley Williams)
3       Paramore (by:Paramore)
4       Pray for the Wicked (by:Panic! At The Disco)
5       Future Nostalgia (by:Dua Lipa)
6       Expectations (by:Bebe Rexha)
7       Infinity On High (by:Fall Out Boy)
8       BADLANDS (Deluxe) (by:Halsey)
9       World War Joy (by:The Chainsmokers)
10      The Big Wheel (by:Runrig)
11      Bad (by:Michael Jackson)
12      1989 (Deluxe Edition) (by:Taylor Swift)
13      Third Eye Blind (by:Third Eye Blind)
=====
```

Figure 1 - Using class structures to build new CD entries and store them in runtime

<sup>8</sup> Ibid, p13-14 (accessed August 31, 2020).

## Docstrings for Classes

As should be clear by now, classes are sophisticated constructs that can contain a large number of interoperable code blocks performing distinct operations which may themselves be tied to different classes entirely. For that reason, it behooves programmers to write class-level docstrings which summarize the purpose of classes and how their components work with reference to the program as a whole. By doing so the programmer satisfies the chief concern of OOP: clear, common sense structure that can be understood by other programmers.

## Summary

Learning to use the conceptual framework of OOP through class structures was satisfying. I think this experience was furthered by the fact that the majority of the code used to build the program was recycled from previous iterations of the CD Inventory, the modular implementation of these code blocks serving to accentuate the modular nature of classes as the building blocks of OOP.

## Appendix

### A – CD Inventory Code

```
11. # -- DATA -- #
12. strFileName = 'CDInventory.txt'
13. lstOfCDObjects = []
14.
15. class CD:
16.     """Stores data about a CD:
17.
18.     properties:
19.         cd_id: (int) with CD ID
20.         cd_title: (string) with the title of the CD
21.         cd_artist: (string) with the artist of the CD
22.     """
23.     def __init__(self, ID=None, title="...", artist="..."):
24.         self.__cd_id = ID
25.         self.__cd_title = title
26.         self.__cd_artist = artist
27.
28.     @property
29.     def cd_id(self):
30.         return self.__cd_id
31.
32.     @cd_id.setter
33.     def cd_id(self, t_ID):
34.         if str(t_ID).isalpha():
35.             raise Exception("Numeric value required for ID.")
36.         else:
37.             self.__cd_id = t_ID
38.
39.     @property
40.     def cd_title(self):
41.         return self.__cd_title
42.
43.     @cd_title.setter
44.     def cd_title(self, t_tit):
45.         if str(t_tit).isnumeric():
46.             raise Exception("That doesn't sound like a track title.")
47.         else:
48.             self.__cd_title = t_tit
49.
50.     @property
51.     def cd_artist(self):
52.         return self.__cd_artist
53.
54.     @cd_artist.setter
```

```

55.     def cd_artist(self, t_art):
56.         if str(t_art).isnumeric:
57.             raise Exception("That's doesn't sound like an artist name.")
58.         else:
59.             self.__cd_artist = t_art
60.
61.     def __str__(self):
62.         return '{} | {} | {}'.format(self.__cd_id, self.__cd_title, self.__cd_artist)
63.
64. # -- PROCESSING -- #
65. class DataProcessor:
66.     """Processes data within the runtime
67.
68.     properties:
69.
70.     methods:
71.         add_cd: processes user input values into a new CD object
72.         delete_cd processes user input ID values through current inventory table and deletes matches
by cd_id attribute
73.     """
74.     @staticmethod
75.     def add_cd(addID, addtitle, addartist, table):
76.         """Converts user input data into a new CD object and appends to current inventory table
77.
78.         Args:
79.             addID (string): numerical identification for the new CD
80.             addtitle (string): album title of the new CD
81.             addartist (string): artist name of the new CD
82.             table (list of CD objects): 2D data structure (list of CD objects) that holds the data du
ring runtime
83.
84.         Returns:
85.             None.
86.         """
87.         newCD = CD(addID, addtitle, addartist)
88.         table.append(newCD)
89.         print()
90.
91.     @staticmethod
92.     def delete_cd(delID, table):
93.         """Deletes CD objects by cd_id attribute from current inventory table
94.
95.         Args:
96.             delID (list of strings): holds one or more ID values designated for deletiton
97.             table (list of CD objects): 2D data structure (list of CD objects) that holds the data du
ring runtime
98.
99.         Returns:
100.            None.
101.        """
102.        remove_count = 0
103.        for row in delID: delID = row.strip().split(',')
104.
105.        for _id in delID:
106.            try:
107.                _id = int(_id)
108.            except:
109.                print('','',_id,'',' is not valid ID input. Removing from delete list and re
turning to main menu.', sep='')
110.                delID.remove(_id)
111.            pass
112.            for row in table:
113.                if row.cd_id == _id:
114.                    remove_count += 1
115.                    table.remove(row)
116.
117.        if remove_count == 0: print('No matching IDs')
118.        else: print(f'{remove_count} total IDs removed out of {len(delID)}')

```

```

119.
120.
121.     class FileIO:
122.         """Processes data to and from file:
123.
124.         properties:
125.
126.         methods:
127.             save_inventory(file_name, lst_Inventory): Saves 2D data structure in runtime to file
128.             load_inventory(file_name): Loads 2D data structure into runtime from file
129.         """
130.         @staticmethod
131.         def save_inventory(fileName, table):
132.             """Function to manage transcription of data from list of CD objects in
133.             current inventory to file
134.
135.             Args:
136.                 fileName (string): name of file used to wrtie the data to
137.                 table (list of CD objects): 2D data structure (list of CD objects) that holds the
data during runtime
138.
139.             Returns:
140.                 None.
141.             """
142.             tempTbl = []
143.             FileIO.load_inventory(strFileName, tempTbl)
144.
145.             try:
146.                 if str(table) != str(tempTbl):
147.                     objFile = open(fileName, 'w')
148.                     for row in table:
149.                         lstValues = [row.cd_id, row.cd_title, row.cd_artist]
150.                         lstValues[0] = str(lstValues[0])
151.                         objFile.write(','.join(lstValues) + '\n')
152.                     objFile.close()
153.             except:
154.                 print('There are no new CDs in Inventory to save.')
155.
156.         @staticmethod
157.         def load_inventory(fileName, table):
158.             """Function to manage data ingestion from file to a list of CD objects
159.
160.             Reads the data from file identified by fileName into a 2D table
161.             (list of CD objects) table one line in the file represents one CD object row in table.
162.
163.             Args:
164.                 fileName (string): name of file used to read the data from
165.                 table (list of dict): 2D data structure (list of dicts) that holds the data during
runtime
166.
167.             Returns:
168.                 None.
169.             """
170.             table.clear()
171.
172.             try:
173.                 objFile = open(fileName, 'r')
174.             except (IOError):
175.                 print('CD Inventory file not found. Creating now. Please restart program.')
176.                 objFile = open(fileName, 'w+')
177.                 objFile.close()
178.                 sys.exit()
179.
180.             for line in objFile:
181.                 try:
182.                     if line != {}: pass
183.                 except (TypeError, ValueError):

```

```

184.         print('Corrupt or missing data.')
185.         sys.exit()
186.         data = line.strip().split(',')
187.         data[0] = int(data[0])
188.         inventoryCD = CD(data[0], data[1], data[2])
189.         table.append(inventoryCD)
190.     objFile.close()
191.
192. # -- PRESENTATION (Input/Output) -- #
193. class IO:
194.     """Handling Input / Output
195.
196.     properties:
197.
198.     methods:
199.         print_menu(): Displays program options for user
200.         menu_choice(): Gets user input based on print_menu() choices
201.         show_inventory(table): Displays current inventory in runtime from list of CD objects
202.         add_cd_input(table): Gets user input for new CD entries
203.     """
204.     @staticmethod
205.     def print_menu():
206.         """Displays a menu of choices to the user
207.
208.         Args:
209.             None.
210.
211.         Returns:
212.             None.
213.         """
214.
215.         print('Menu\n\n[1] Load Inventory from file\n[2] Add CD to Inventory\n[3] Delete CD fr
om Inventory')
216.         print('[4] Display Current Inventory\n[5] Save Inventory to file\n[6] exit\n')
217.
218.     @staticmethod
219.     def menu_choice():
220.         """Gets user input for menu selection
221.
222.         Args:
223.             None.
224.
225.         Returns:
226.             choice (string): a lower case sting of the users input out of the choices l, a, i,
d, s or x
227.         """
228.         options = ["1", "2", "3", "4", "5", "6"]
229.         while True:
230.             choice = input('Which operation would you like to perform? [1, 2, 3, 4, 5, or 6]:
').strip()
231.             try:
232.                 if options.count(choice) == 1: break
233.             except:
234.                 continue
235.             print()
236.             return choice
237.
238.     @staticmethod
239.     def show_inventory(table):
240.         """Displays current inventory table
241.
242.
243.         Args:
244.             table (list of CD objects): 2D data structure (list of CD objects) that holds the
data during runtime.
245.
246.         Returns:
247.             None.

```

```

248.         """
249.         print('==== The Current Inventory: =====')
250.         print('ID\tCD Title (by: Artist)\n')
251.         for row in table:
252.             print('{ }\t{ } (by:{ })'.format(row.cd_id, row.cd_title, row.cd_artist))
253.         print('=====')
254.
255.     def add_cd_input(table):
256.         """Gets user input for new CD information
257.
258.         Args:
259.             table (list of CD objects): 2D data structure (list of CD objects) that holds the
data during runtime.
260.
261.         Returns:
262.             ID (string): numerical identification for the new CD
263.             title (string): album title of the new CD
264.             artist (string): artist name of the new CD
265.         """
266.         used_ids = []
267.         for row in table:
268.             used_ids.append(row.cd_id)
269.
270.         while True:
271.             ID = input('Enter numerical ID: ').strip()
272.             try:
273.                 ID = int(ID)
274.                 break
275.             except:
276.                 continue
277.
278.         while ID in used_ids:
279.             print('That ID already exists. Please enter a new ID.')
280.             while True:
281.                 ID = input('Enter numerical ID: ').strip()
282.                 try:
283.                     ID = int(ID)
284.                     break
285.                 except:
286.                     continue
287.
288.         while True:
289.             title = input('What is the CD\'s title? ').strip()
290.             try:
291.                 if (len(title)) > 0: break
292.             except:
293.                 continue
294.
295.         while True:
296.             artist = input('What is the Artist\'s name? ').strip()
297.             try:
298.                 if (len(artist)) > 0: break
299.             except:
300.                 continue
301.
302.         return ID, title, artist
303.
304.
305.     # -- Main Body of Script -- #
306.     import sys
307.     FileIO.load_inventory(strFileName, lstOfCDObjects)
308.
309.     while True:
310.         IO.print_menu()
311.         menuChoice = IO.menu_choice()
312.
313.         if menuChoice == "6":
314.             break

```



```

315.
316.         if menuChoice == "1":
317.             print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-
loaded from file.')
318.             overwrite_verification = input('Type \'yes\' to continue and reload from file. Otherwi
se reload will be canceled.\n')
319.             if overwrite_verification.lower() == 'yes':
320.                 print('reloading...')
321.                 FileIO.load_inventory(strFileName, lstOfCDObjects)
322.                 IO.show_inventory(lstOfCDObjects)
323.             else:
324.                 input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the
menu.')
325.                 IO.show_inventory(lstOfCDObjects)
326.                 continue
327.
328.         elif menuChoice == "2":
329.             print("Please provide new CD info.")
330.
331.             add_verification = 'y'
332.             while add_verification == 'y':
333.                 newID, newTitle, newArtist = IO.add_cd_input(lstOfCDObjects)
334.
335.                 print('\nYou entered:')
336.                 print('ID\tCD Title (by: Artist)')
337.                 print(f'{newID}\t\t{newTitle} (by:{newArtist})\n')
338.                 input_validation = input("Is this information correct? [y/n]: ").lower()
339.                 if input_validation == "y":
340.                     pass
341.                 elif input_validation == "n":
342.                     continue
343.                 DataProcessor.add_cd(newID, newTitle, newArtist, lstOfCDObjects)
344.                 while True:
345.                     add_verification = input('Would you like to add another CD? [y/n] ').lower()
346.                     try:
347.                         if add_verification == 'y': break
348.                         elif add_verification == 'n': break
349.                     except:
350.                         continue
351.                 continue
352.
353.         elif menuChoice == "3":
354.             IO.show_inventory(lstOfCDObjects)
355.
356.             while True:
357.                 cdIDdel = [input('Enter one or more IDs to delete, separated by commas (example: "
1,2,3)": ').strip()]
358.                 try:
359.                     if cdIDdel != ['']: break
360.                 except:
361.                     continue
362.
363.                 DataProcessor.delete_cd(cdIDdel, lstOfCDObjects)
364.                 continue
365.
366.         elif menuChoice == "4":
367.             IO.show_inventory(lstOfCDObjects)
368.             continue
369.
370.         elif menuChoice == "5":
371.
372.             while True:
373.                 save_verification = input('Save current inventory to file? [y/n] ').strip().lower(
)
374.                 try:
375.                     if save_verification == 'y':
376.                         FileIO.save_inventory(strFileName, lstOfCDObjects)
377.                         break

```

```
378.                 elif save_verification == 'n':
379.                     input('The inventory was NOT saved to file. Press [ENTER] to return to the
menu.')
```

```
380.                 break
381.             except:
382.                 continue
383.         continue
384.
385.     else:
386.         print("Error.")
```

## B – Github Link

[https://github.com/Alex-Angelico/Assignment\\_08](https://github.com/Alex-Angelico/Assignment_08)