

OBJECTS and CLASSES

What we learnt in Module 8

In this module, we were formally introduced to Object Oriented Programming (OOP): we learned about *Classes* and *Objects* of these Classes, *Constructors*, private and public Fields, Attributes and Methods, Typehints and how to write DocStrings for Classes. We also learnt more about got and GitHub. We focused on how to apply Classes to our code and use **getters** and **setters** to access and assign / set values to the properties, respectively. Setters are also the right places to enforce the apt data type for a property of a Class. We continued to focus on the use of exceptions to catch both anticipated and unforeseen errors. We continued to use *Spyder* as our IDE and stored code in *GitHub*.

Assignment 08

For this assignment, we were provided with a pseudocode that we had to fill with the relevant code from our script that saw us apply Classes, Constructors, Properties, Methods including Getters, Setters and Static methods. The theme of the assignment continued to be CD Inventory but we reverted to using a Text file. We applied Exception handling especially wherever there is user interaction and type casting or file access operations.

1. On the pseudocode that was provided, I re-used code from Assignment 06, pulled-in exception handling pieces from Assignment 07 and created the .py file for assignment 08 that essentially was restructured to use Classes and objects.
2. I created appropriate classes to help define the required objects.
3. I wrote methods and created attributes for objects that were instantiated from the afore-mentioned classes
4. I used Python's built-in exception classes.
5. While writing the code, I created / updated DocStrings and applied typehints when defining Methods, as required
6. Apart from re-testing last week's test cases to ensure that the changes in code did not adversely affect the program's intent, the exceptions were also duly tested. Snips of code execution are listed under the **Code Execution** section below – in both **Spyder** and on the **Terminal**. The code is detailed in the **Appendix** section.

Code Execution in Spyder

Code was **tested** for the following:

1. Adding entries to the CD Inventory
2. Saving entries to file
3. Loading data
4. Displaying the inventory
5. Exiting the program

Then, the following was further tested for the exception handling written in the script:

Upon first execution, if the file did not exist, then the exception was handled - as follows:

```
In [43]: runfile('C:/FP_Python/Mod_08/CD_Inventory.py', wdir='C:/FP_Python/Mod_08')
File not found.
File not found! Creating a new file.
Menu

[l] Load Inventory from file
[a] Add CD
[i] Display Current Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, s or x]:
```

Figure 1: File not found exception

If the input for CD ID was not convertible to an integer, then the exception handled was as follows:

```
In [41]: runfile('C:/FP_Python/Mod_08/CD_Inventory.py', wdir='C:/FP_Python/Mod_08')
Menu

[1] Load Inventory from file
[a] Add CD
[i] Display Current Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [1, a, i, s or x]: a

Enter ID: rt
The CD ID entered cannot be converted to an integer!
invalid literal for int() with base 10: 'rt'
Menu

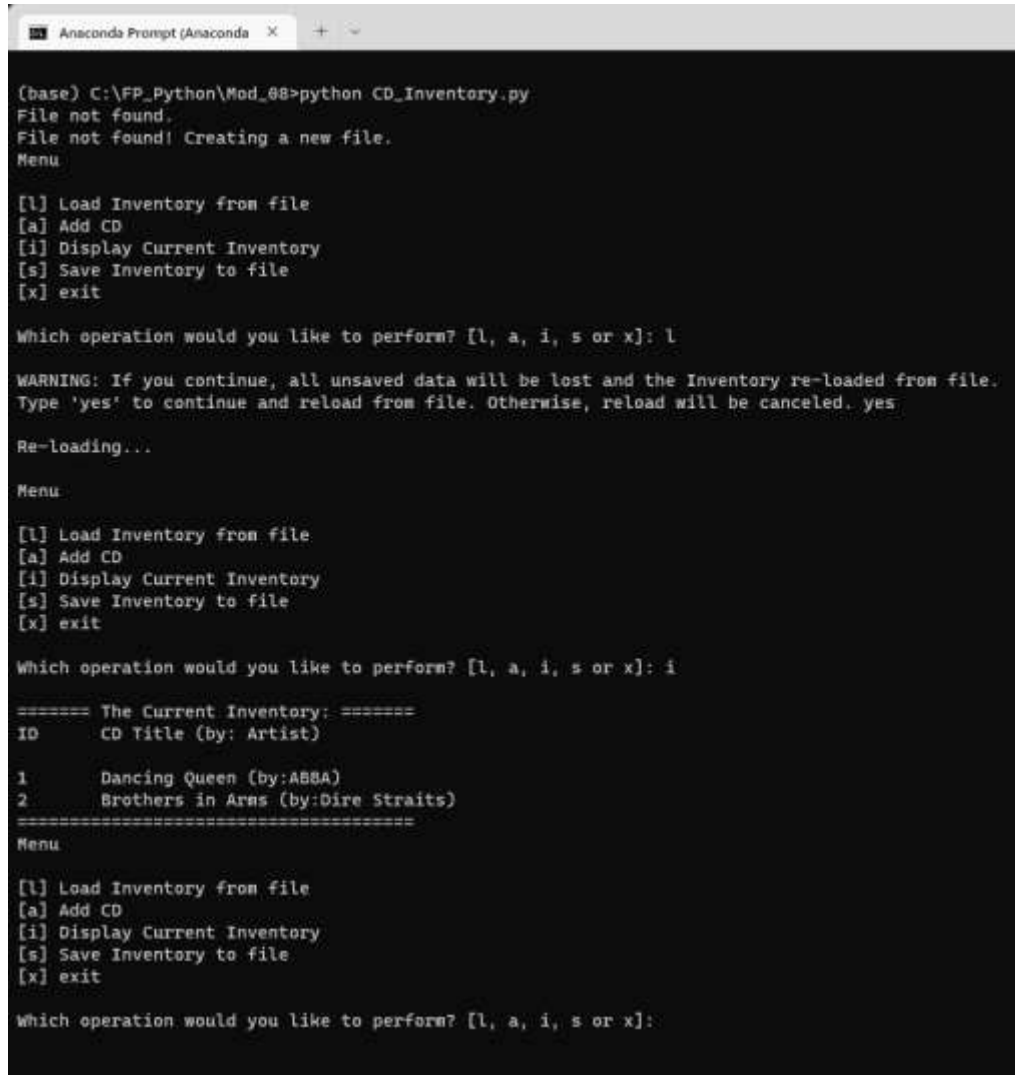
[1] Load Inventory from file
[a] Add CD
[i] Display Current Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [1, a, i, s or x]:
```

Figure 2: Integer typecasting error

Execution on Terminal

The following image captures a successful execution of Assignment 08's script on the Terminal –



```
(base) C:\FP_Python\Mod_08>python CD_Inventory.py
File not found.
File not found! Creating a new file.
Menu

[l] Load Inventory from file
[a] Add CD
[i] Display Current Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, s or x]: l

WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.
Type 'yes' to continue and reload from file. Otherwise, reload will be canceled. yes

Re-loading...

Menu

[l] Load Inventory from file
[a] Add CD
[i] Display Current Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, s or x]: i

===== The Current Inventory: =====
ID      CD Title (by: Artist)

1       Dancing Queen (by:ABBA)
2       Brothers in Arms (by:Dire Straits)
=====
Menu

[l] Load Inventory from file
[a] Add CD
[i] Display Current Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, s or x]:
```

Figure 3: Execution on Terminal for the same exceptions handled as in Spyder

The following is the link to my GitHub repository for this assignment:

https://github.com/PadmaBham/Assignment_08

Appendix

1. My **Python script** for the assignment is as follows:

```
#-----#
# Title: CD_Inventory.py
# Desc: Assignment 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
# PBhamidipati, 2022-Dec-04, Updated title to CD_Inventory.py
# PBhamidipati, 2022-Dec-04, Updated code to reflect the given pseudocode in
# order to address Assignment 08 - implementing objects & classes
#-----#

from typing import List

# -- DATA -- #
strFileName = 'cdInventory.txt'
lstOfCDObjects = []

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:
```

```
"""
```

```
# TODone Add Code to the CD class
```

```
#---Fields---#
```

```
#---Constructor---#
```

```
def __init__(self, trackId = 0, trackTitle = '<None chosen>', trackArtist = '<Not applicable>'):
```

```
    #--Attributes--#
```

```
    self.__cd_id = trackId
```

```
    self.__cd_title = trackTitle
```

```
    self.__cd_artist = trackArtist
```

```
#---Properties---#
```

```
@property
```

```
def cd_id(self):
```

```
    return self.__cd_id
```

```
@cd_id.setter
```

```
def cd_id(self, intID):
```

```
    if not str(intID).isnumeric():
```

```
        raise Exception('An integer value is required for the CD ID field!')
```

```
    else:
```

```
        self.__cd_id = intID
```

```
@property
```

```
def cd_title(self):
```

```
    return self.__cd_title
```

```
@cd_title.setter
```

```
def cd_title(self, strTitle):
```

```
    self.__cd_title = str(strTitle)
```

```
@property
```

```
def cd_artist(self):
```

```
    return self.__cd_artist
```

```
@cd_artist.setter
```

```

def cd_artist(self, strArtist):
    self.__cd_artist = str(strArtist)

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

    properties:

    methods:
        save_inventory(file_name, lst_Inventory): -> None

        load_inventory(file_name): -> (a list of CD objects)

    """
    # TODO Add code to process data from a file
    def Load_inventory(file_name) -> List[CD]:
        """ Loads the current CD inventory from file to a list

        Args:
            file_name (str): name of the text file that contains the CD inventory that's to be loaded

        Returns:
            list: list of CD objects
        """

        lstTemp = []
        try:
            objFile = open(file_name, 'r')
            for line in objFile:
                data = line.strip().split(',')
                cdTemp = CD()
                cdTemp.cd_id = int(data[0])
                cdTemp.cd_title = data[1]
                cdTemp.cd_artist = data[2]
                lstTemp.append(cdTemp)
        except ValueError as e:
            print(e.__doc__)
            print('Bad Data found! The CD ID in the file cannot be converted to an integer.')
        except FileNotFoundError as e:

```

```

        print(e.__doc__)
        print('File not found! Creating a new file.')
        objFile = open(file_name, 'wb')
    except IOError as e:
        print(e.__doc__)
        print('Unhandled error while reading file!')
        raise(e)
    finally:
        objFile.close()

```

```

return lstTemp

```

TODO Add code to process data to a file

def save_inventory(file_name, lst_inventory) -> None:

""" Writes the current CD inventory from list to the file

Args:

file_name (str): name of the text file that contains the CD inventory that's to be loaded

lst_inventory (list): list of CD objects / inventory details

Returns:

None

"""

try:

objFile = open(file_name, 'w')

for cd in lst_inventory:

lstValues = [str(cd.cd_id), cd.cd_title, cd.cd_artist]

objFile.write(','.join(lstValues) + '\n')

except IOError as e:

print(e.__doc__)

print('Unhandled error while reading file!')

raise(e)

finally:

objFile.close()

-- PRESENTATION (Input/Output) --

class IO:

TODO add docstring

"""Handling Input / Output"""


```

# TODone add code to show menu to user
@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')
    print('[s] Save Inventory to file\n[x] exit\n')

# TODone add code to captures user's choice
@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 l, a, i, s or x

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

# TODone add code to display the current data on screen
def show_inventory(listInventory):
    """Displays current inventory

    Args:
        listInventory (list of CD class objects): 2D data structure

```

(list of CD objects) that holds the data during runtime.

Returns:

None.

"""

```
print('==== The Current Inventory: =====')
print('ID\tCD Title (by: Artist)\n')
for cd in listInventory:
    dictValues = {'ID': cd.cd_id, 'Title': cd.cd_title, 'Artist': cd.cd_artist}
    print('{}\t{} (by: {})'.format(*dictValues.values()))
print('=====')
```

TODO add code to get CD data from user

@staticmethod

def ask_input_addCD():

"""Gets user input for the CD's details that are to be added to the inventory

Args:

None.

Returns:

None.

"""

try:

```
    strID = input('Enter ID: ').strip()
    intID = int(strID)
    strTitle = input('What is the CD\'s title? ').strip()
    strArtist = input('What is the Artist\'s name? ').strip()
```

```
    cd = CD()
    cd.cd_id = intID
    cd.cd_title = strTitle
    cd.cd_artist = strArtist
```

```
    lstOfCDObjects.append(cd)
```

```
    IO.show_inventory(lstOfCDObjects)
```

```
except ValueError as e:  
    print('The CD ID entered cannot be converted to an integer!')  
    print(e)
```

@staticmethod

def save_data_list():

"""Displays the latest inventory from the in-memory list and confirms that the list is to be saved to the file. If confirmed, the list is written to the file, otherwise it informs that the list was NOT saved and asks the user to go back to the menu list

Args:

None.

Returns:

None.

"""

IO.show_inventory(lstOfCDObjects)

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

Process choice

if strYesNo == 'y':

save data

TODone move processing code into function

FileIO.save_inventory(strFileName, lstOfCDObjects)

else:

input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')

@staticmethod

def load_from_file():

"""Ascertain from the user if the CD inventory is to be loaded from the file or the in-memory list variable, displaying suitable warning messages as to what would happen for each of their choices, and proceeds to load or cancel loading as may be the user's choice

Args:

None.

Returns:

list: current list of CD objects loaded from file

```

"""
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. Otherwise, reload will be canceled. ')
lstTemp = []
if strYesNo.lower() == 'yes':
    print('\nRe-loading...\n')
    lstTemp = FileIO.Load_inventory(strFileName)
else:
    input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.\n')

return lstTemp

# -- Main Body of Script -- #
# TODone Add Code to the main body

# Load data from file into a list of CD objects on script start
lstOfCDObjects = FileIO.Load_inventory(strFileName)

# Display menu to user
while True:
    # Display Menu to user and get choice
    IO.print_menu()
    strChoice = IO.menu_choice()

    # Process menu selection
    # process exit first
    if strChoice == 'x':
        break

    # show user current inventory (display)
    if strChoice == 'i':
        IO.show_inventory(lstOfCDObjects)
        continue # start loop back at top.

    # let user add data to the inventory (add)
    elif strChoice == 'a':
        # Ask user for new ID, CD Title and Artist
        IO.ask_input_addCD()
        continue # start loop back at top.

```

```
# let user save inventory to file (save)
elif strChoice == 's':
    # Display current inventory and ask user for confirmation to save
    IO.save_data_list()
    continue # start loop back at top.

# let user load inventory from file (load)
elif strChoice == 'l':
    lstOfCDOObjects = IO.load_from_file() # call method to load data from file to in-memory list
    continue # start loop back at top.

# catch-all should not be possible, as user choice gets vetted in IO, but to be save:
else:
    print('General Error')
```

2. Reference material to learning this module has been all the reading documentation and videos demonstrations that were provided in this course.
3. Search strings: none