Michael List

March 15, 2020

Foundations of Programming: Python

Assignment 08

Objects and Classes

Introduction

This document is about the eight module and assignment of the Foundations of Programming: Python course. It is an introduction to object oriented programming (OOP) and conveys the very basics of OOP: Objects and classes and how to work with them.

In the second part the gained knowledge is demonstrated by modifying a new starter script of CDInventory.py file. The starter script has three empty classes outlined that need to be defined as well as the main body of the script that needs to be written as well.

Part 1 – Knowledge

Part 1 of this document explains key concepts learned in this module as outlined in the introduction.

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

A class is a blueprint that is being used to create objects based on the class's definition. A class itself can have attributes and methods and other elements. Every time a new object is created (initiated) based on a class, the object is inheriting these features form the class and makes them available for the object.

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

A class is made up of the following items:

1. Fields
2. Constructor
3. Attributes
4. Properties
5. Methods

These components will be further explained in the rest of this document.

What is the purpose of a class constructor?

A class constructor is a specific method that gets automatically called upon the initiation of an object. Arguments that are being passed to the class upon the object initiation can be assigned to attributes (variables) of the created object. The constructor also allows the execution of any kind of statement upon the initiation of an object.

When do you use the keyword "self"?

The 'self' keyword is always used as the first parameter in every method defined within an object. Theoretically any keyword can be used, however, 'self' is an agreed upon standard that is expected by other programmers. This keyword is the first parameter in a function and its sole purpose is to establish a reference between the object that is created and the method of the class.

When do you use the keyword "@staticmethod"?

‘@staticmethod’ is a decorator that is used before the definition of a class-wide method. Methods can be either defined within an object where the method references to its own object (this is where the keyword “self” is being used) or on a the class level where the function is declared with the decorator “@staticmethod” and can be used by any object of the class. Generally static methods are used for processing data as the additional abstraction layer of an instance is not necessary.

How are fields and attributes and property functions related?

Fields, attributes and property functions are all part of a class definition.

1. **Fields**

   Fields are like variables that exist within a class and are available for objects of a class to be called. Fields exist outside of objects and are independent of objects. Any value changes of a field happen on a class level (and not an object level). If an object calls a field, it will reference the same value (independent of which object is calling the field)

2. **Attributes**
   Attributes are variables that exist on an object level rather than a class level. This means that the attribute is directly linked to the object itself. If there are multiple objects, they can all contain an attribute with the same name. To call an attribute associated to an object the object needs to be referenced in the call as well. To add an additional level of abstraction attributes can, and often should, be defined as private attributes which does not allow the access of attributes outside of a class

3. **Property functions**
   Property functions help the programmer to regulate the read and write access of attributes. Because of this, there are usually two types of properties defined for each attribute in a class:
   a) “Setter”: This property function regulates the write (“set”) access for (private) attributes. This can be especially useful to validate arguments passed into a class before they are assigned to an attribute
   b) “Getter”: This property function regulates the read (“get”) access for (private) attributes. A “Getter” can be additionally used to format the returned attribute value as well.

What is the difference between a property and a method?

Properties are methods that have the decorator ‘@property’ before the method definition. The purpose of the property methods is to solely regulate the read and write access for attributes within and outside of the object. The purpose of a method within an object or class are the same as functions in the main body of a script: to group together statements to add an abstraction layer and improve the separation of concerns.

Why do you include a docstring in a class?

Docstrings should be part of a class as they are for any function as discussed in the previous modules of this course. A docstring for a class will explain the purpose of the class and will provide information about the attributes, properties and methods of a class.

Part 2 – Modify CD_Inventory.py

The coding assignment portion of this assignment was to modify a provided starter script. The starter script only included pseudocode. The first part of the assignment was to understand the pseudocode and then add code to create a functional program, including error handling and docstrings.My submission of this assignment can also be found on Github: https://github.com/List-Michael/Assignment_08[1]

Out of all the coding assignments of this course this was by the far the most challenging for me. This was caused by the additional layer of abstractions introduced with OOP as well as classes interacting with each. I also attempted to write the CD_Inventory.py file from scratch script with minimal pseudocode as guidance instead of modifying a starter.

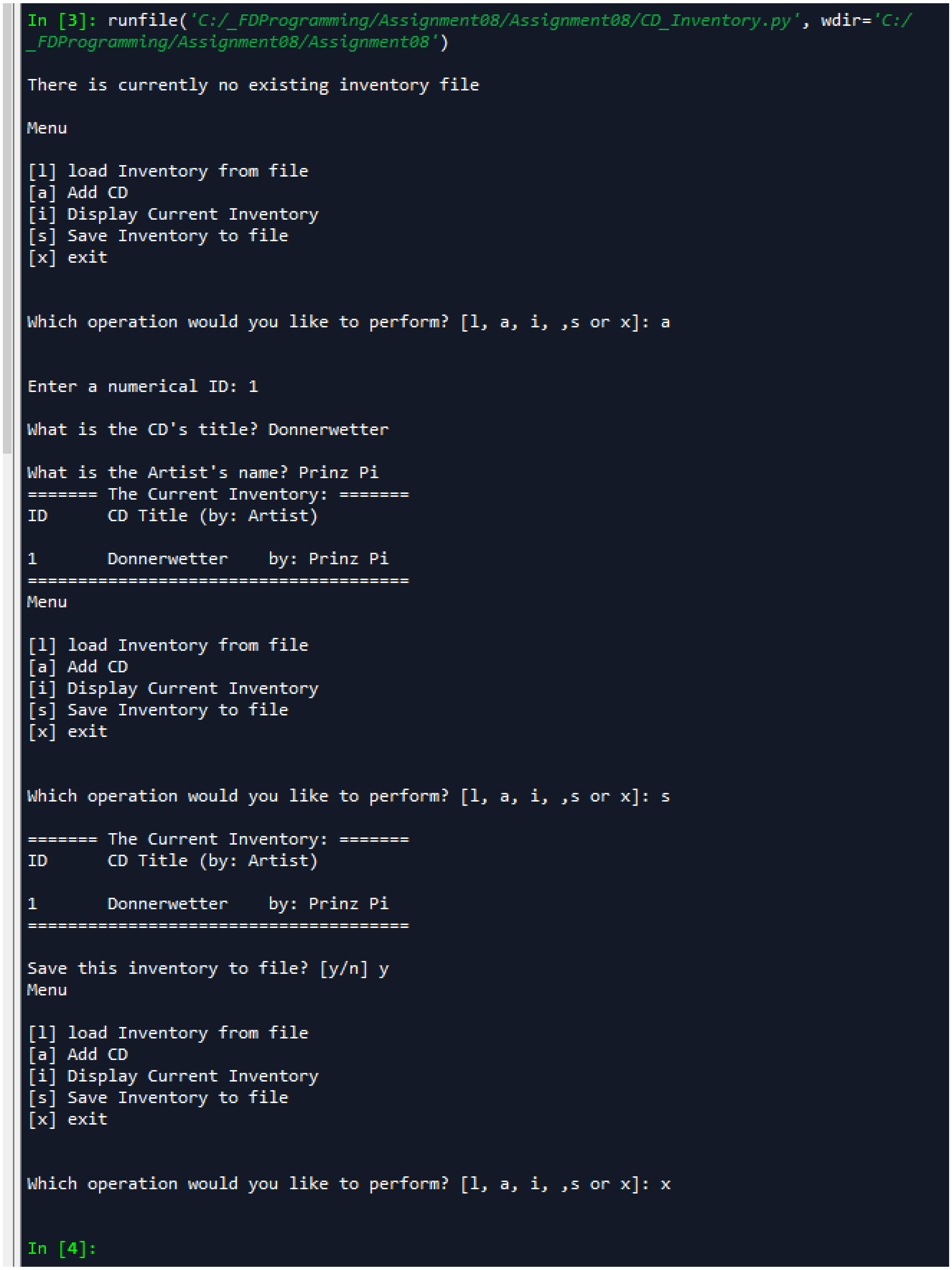
Learnings and struggles:

- It was challenging figuring out how I can reuse some code from assignment 07. I had to logically think through the methods from assignment 07 to figure out if they can still be used, and if so, how they would need to be modified. For example, a lot of the IO class from assignment 07 was usable without alterations as no objects will be created with the IO. Any input from the IO class can be passed onto the CD class for object initiation. Doing so also allows us to check if the entered ID is an integer in the IO class (in a while loop) and have a separation of concerns established.
- There was no pseudocode for deleting a CD object. I assume this will be part of assignment 09 as we have yet to cover this concept
- To complete this assignment, I had to write out a lot in pseudocode and think the problem through before starting coding. I also experimented with starter script and had to review some code of the labs as well as previous assignments to produce some functional code with CD objects.
- Display inventory function: It was Interesting to learn that displaying the current inventory can simply be done by printing the class if the "__str__ method" of the class has been defined
- A good challenge was to find a way how to read and load a list of objects from and to a text file. I believe it would have been easier to use the pickle module instead. By using objects, I realized that dictionaries are not necessary for storing any CD information as the objects themselves have attributes that represent the keys of a dictionary.
- I initially struggled with the object creation as I thought I would need to create a unique object name for each CD. However, even when removing this logic (which I was not sure was even working) my code executed just fine.
- It was interesting to learn that I was able to create an object while being in the load_inventory() method. I did not expect this behavior and was anticipating that I would have to separate these two issues. The give away was the starter code. However, it might not be a bad idea to separate the reading of an existing text file from the object creation to improve the SoC

I think overall part of my struggles were also caused by overthinking how to interact with CD objects as there were multiple surprises during the coding assignment.

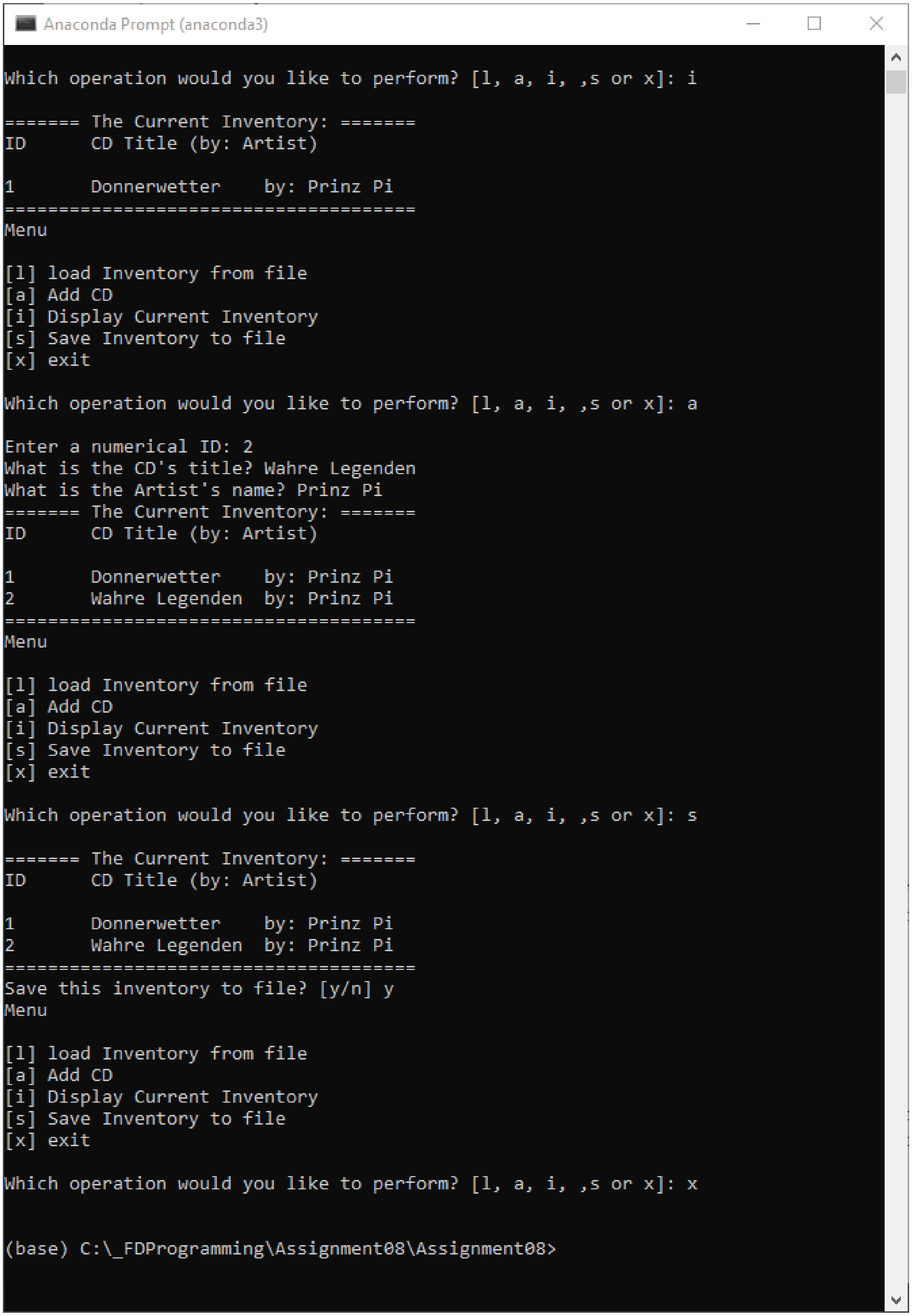
The three following figures show the script executed in Spyder, Anaconda prompt and the resulting text file.

[1] Created and Retrieved on 2020-Mar-14



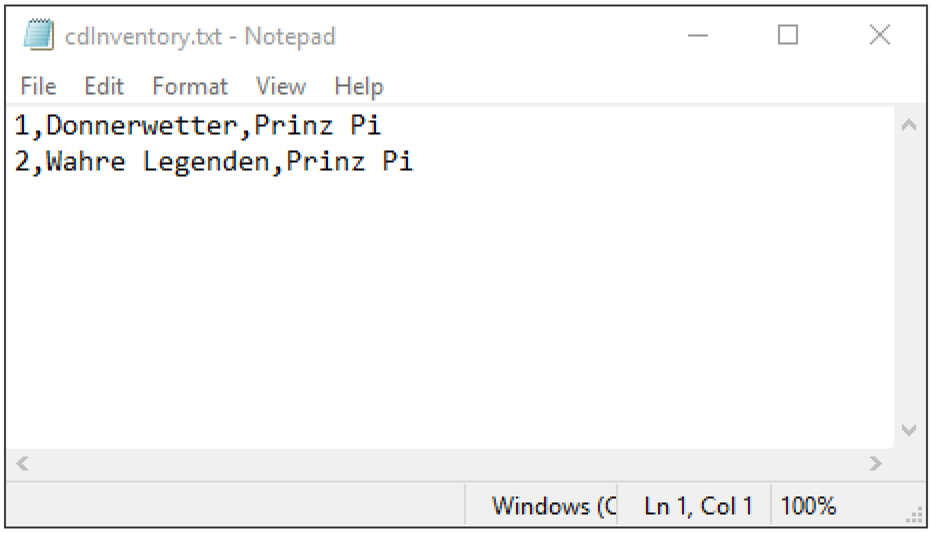
```
In [3]: runfile('C:/_FDProgramming/Assignment08/Assignment08/CD_Inventory.py', wdir='C:/
_FDProgramming/Assignment08/Assignment08')

There is currently no existing inventory 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]: a


Enter a numerical ID: 1

What is the CD's title? Donnerwetter

What is the Artist's name? Prinz Pi
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Donnerwetter    by: Prinz Pi
======================================
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]: s

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

1       Donnerwetter    by: Prinz Pi
======================================

Save this inventory to file? [y/n] y
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]: x


In [4]:
```

Figure 1 - CD_Inventory.py executed in Spyder


Anaconda Prompt (anaconda3)

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

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

1       Donnerwetter    by: Prinz Pi
======================================
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]: a

Enter a numerical ID: 2
What is the CD's title? Wahre Legenden
What is the Artist's name? Prinz Pi
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Donnerwetter    by: Prinz Pi
2       Wahre Legenden  by: Prinz Pi
======================================
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]: s

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

1       Donnerwetter    by: Prinz Pi
2       Wahre Legenden  by: Prinz Pi
======================================
Save this inventory to file? [y/n] y
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]: x


(base) C:\_FDProgramming\Assignment08\Assignment08>
```

Figure 2 - CD_Inventory.py executed in Anaconda Prompt



```
1,Donnerwetter,Prinz Pi
2,Wahre Legenden,Prinz Pi
```

Figure 3 - cdInventory.txt after execution

Summary

In this assignment the students were introduced to object-oriented programming. While we were warned that this subject is a rather complex matter, I did not expect that many challenges while executing the code part. I think I got a good idea what object-oriented programming is about while also realizing that there is a lot logic too it that will take a while to sink in.

Appendix

Listing 1 – CD_Inventory.py

```
#------------------------------------------#
# Title: CD_Inventory.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
# Mlist, 2020-March-14, added fucntions from previous assignments
# Mlist, 2020-March-15, build out all functionalities and tested
# Mlist, 2020-March-15, updated docstrings, cleaned up code, added comments
#------------------------------------------#

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

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

    properties:
        position: (int) with CD ID
        album: (string) with the title of the CD
        artist: (string) with the artist of the CD
    methods:
        append_cd_inventory_memory_list(objCD, table): -> None
    """
    # -- Fields -- #
    # -- Constructer -- #
    def __init__(self):
        self.__intPosition = 1
        self.__strAlbum = ""
        self.__strArtist = ""
```

```python

    @property
    def position(self):
        return self.__intPosition
    @position.setter
    def position (self, ps):
        self.__intPosition = ps

    @property
    def album (self):
        return self.__strAlbum
    @album.setter
    def album (self, alb):
        self.__strAlbum = alb

    @property
    def artist (self):
        return self.__strArtist
    @artist.setter
    def artist (self, ar):
        self.__strArtist = ar

    def __str__(self):
        return str(self.__intPosition) + "\t" + str(self.__strAlbum) + "\tby: " + str(self.__strArtist)

    @staticmethod
    def append_cd_inventory_memory_list(objCD, table):
        """Function to append a newly created CD object (user input or file reading) to the global list of CD objects

        Uses passed arguements of CD information to create a new CD entry as a dictionary and adds dictionary as a new line to global 2D table

        Args:
            objCD (object): CD Object that needs to be added to the list
            table (list of objects): 2D data structure (list of objects) that holds the data during runtime

        Returns:
            None.
        """
        table.append(objCD)
# -- PROCESSING -- #

class FileIO:
    """Processes data to and from file:

    properties:

    methods:
        write_file(file_name, table): -> None
        load_inventory(file_name): -> cdObjLst (a list of CD objects)

    """
    # -- Methods --#
    @staticmethod
    def write_file(file_name, table):
        """Function to write added data to file

        Reads the data of objects stored in 2D table.
        Casts each object into a string, adds a new line at the end of each string and overrides any data in file.

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

        Returns:
            None.
        """
        objFile = open(file_name, 'w')
        for obj in table:
                    cd_string = str(obj.position)+ ',' + obj.album + ',' + obj.artist
                    objFile.write(cd_string + '\n')
                objFile.close()

            @staticmethod
            def load_inventory(file_name):

                """Function to manage data ingestion from file to a list of dictionaries

                Reads the data from file identified by file_name into a global 2D table
                (list of dicts) table one line in the file represents one dictionary row in table.

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

                Returns:
                    cdObjLst (list): list of objects
                """
                cdObjLst = []
                try:
                    with open(file_name, 'r') as objFile:
                        for line in objFile:
                            cdObjName = CD()
                            data = line.strip().split(',')
                            cdObjName.position = data[0]
                            cdObjName.album = data[1]
                            cdObjName.artist = data[2]
                            cdObjLst.append(cdObjName)
                except IOError:
                    print('\nThere is currently no existing inventory file\n')
                return cdObjLst

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

        class IO:
            """Presents menu, data and requests data from the user:

            properties:

            methods:
                cd_user_input(): -> CD (list)
                print_menu(): -> None
                menu_choice(): -> choice(string)
                show_inventory(table): -> None
            """
            # -- Methods --#
            @staticmethod
            def cd_user_input():
                """Gets user input to add new CD

                Catches ValueError in case the entered ID for the CD is not an integer.
                If this exception is caught the program keeps asking the user to enter a numerical ID
                until an integer has been entered.

                Args:
                    None.

                Returns:
                    CD (list): Returns list with the 3 values: ID, Title and Artist

                """
                while True:
```

```python
            try:
                strID = input('Enter a numerical ID: ').strip()
                intID = int(strID)
                break
            except ValueError:
                print ('The entered ID is not an integer. Please enter a number')
        strTitle = input('What is the CD\'s title? ').strip()
        strArtist = input('What is the Artist\'s name? ').strip()
        CD = [intID, strTitle, strArtist]
        return CD

    @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')


    @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

    @staticmethod
    def show_inventory(table):
        """Displays current inventory table


        Args:
            table (list of CD 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 obj in table:
            print(obj) #printing out __str__ for each CD object
        print('======================================')

# -- Main Body of Script -- #

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

```python
        initial_Load = FileIO.load_inventory(strFileName)
        #assigning loaded data to list of CD objects
        lstOfCDObjects = initial_Load

        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. otherwise reload will be canceled: ')
                if strYesNo.lower() == 'yes':
                    print('reloading...')
                    #FileProcessor.read_file(strFileName, lstTbl)
                    lst = FileIO.load_inventory(strFileName)
                    lstOfCDObjects = lst
                    IO.show_inventory (lstOfCDObjects)
                else:
                    input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
                    IO.show_inventory (lstOfCDObjects)
                continue  # start loop back at top.
            # let user add data to the inventory
            elif strChoice == 'a':
                UserInputLst = IO.cd_user_input() #requests the user input of the new CD object and temprorarly stores it into a list object
                cdObjName = CD() # initiate an empty CD object
                #Writing the user input into the initialized CD object
                cdObjName.position = UserInputLst[0]
                cdObjName.album = UserInputLst[1]
                cdObjName.artist = UserInputLst[2]
                #Appending the newly created CD to the CD object list
                CD.append_cd_inventory_memory_list(cdObjName, lstOfCDObjects)
                IO.show_inventory (lstOfCDObjects)
            # show user current inventory
            elif strChoice == 'i':
                IO.show_inventory (lstOfCDObjects)
                #print('Length of CD object list :', len(lstOfCDObjects))
                continue
            # let user save inventory to file
            elif strChoice == 's':
                # Display current inventory and ask user for confirmation to save
                IO.show_inventory(lstOfCDObjects)
                strYesNo = input('Save this inventory to file? [y/n] ').strip().lower() #No Error handling required
                # Process choice
                if strYesNo == 'y':
                    # save data
                    FileIO.write_file(strFileName, lstOfCDObjects)
                else:
                    input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')

                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')
```