Martin Stallworth

2 September 2020

Foundations of Programming, Python

Assignment 08

Working with Object Oriented Programming and Classes

Introduction

During this week's assignment we got an introduction into object oriented programming. In order to work with these objects we also had to delve deeper into classes. We then further broke down step by step how to use classes as well as fields, constructors, attributes, properties, and methods to work with objects. Of course, we learned how to do all this in the lens of using python.

Getting the Background

We started out by learning that classes are actually critical holding the data and the functionality of an object. The module and the textbook describe a class as the blueprint for an object. The textbook was very helpful of my understanding of the relationship between an object and class. Especially when the textbook compared classes and objects to building houses with the same blueprint (Dawson 219). The real python webpage on object oriented programming was also helpful throughout for supplement. There seems to be a lot of versatility when creating classes for objects. You can create an object by defining fields, constructors and attributes but not necesarilly all at the same time.

After setting up the class we first defined the object using fields. Fields are the data stores of a class and they seem to be similar to creating a variable. Since this was similar to creating a variable this concept was not difficult to grasp as we have been creating variables since the beginning of the class. We then moved into using constructors. When a new object is created a constructor is automatically invoked. They also make sure the right data types and defaults of values are in the fields. This was a unique concept for me but it involved using functions which made it easier for me to understand. The "Constructor Critter Program" from the textbook was also a great supplement with the Saravji's Hut Module 8 Part 2 video with understanding how to properly set up constructions (Dawson 224). Within the constructors are attributes which are internal fields that can be automatically created and used after it is called through the constructor method.

We then began to work with properties and methods. With what we just discussed the attributes have been public. This means that they can accessed or called by a client. We can use properties to make attributes private and enforce the interaction. They can do this because properties are special types of methods that have a control mechanism built in. It seems that typically each attribute with having a property for getting and setting the attribute. In python an attribute is privatized by putting a double underscore in front of its name. This was also similar to the work we have been doing with functions so it was not too hard for me to get a grasp of. The final part of the class are methods. These methods are similar to functions in scripts that we have been learning about throughout the course. It allows you to organize statements and then call them using the name of the method. One specific method that we learned about was the "__str__" method. This method is relevant because most of the time objects are returned as strings. This method is automatically done by python, but you can change it to make it return more useful information like attributes. Methods have a lot of versatility because they can be used on the class level and instance level as well as they can be private themselves.

Working through Labs and Assignment

I did not have too much difficulty working through the labs this week once I got some help with the first lab. Based on the example provided in the module the first lab did not seem like it would take time to complete. When I initially tried it out I got the gist of it but there were some things I needed to fix to make my script more ideal which was pointed out by Dirk. I was still pre-pending the variable type to the variable name which is not something that is done with the fields in the class. Once I fixed this issue I was able to work through the rest of the labs fairly quickly. I was initially nervous about creating a constructor for three fields instead of the one that was in the module example but I just went with what I thought the logical solution was and it worked out exactly how I wanted it to. I was able to fit all of the fields into one function instead of having to do three different functions which is what I was worried about. Additionally, I kind of stumbled upon lab C in my code for lab B so I was able to work with constructors and attributes at the same time. An example of my constructor and attributes is provided below.

```python
    # -- Constructor -- #
    def __init__(self, pos, ttl, leng):
        # -- Attributes -- #
        self.Position = pos
        self.Title = ttl
        self.Length = leng
```

Listing 1: Ex of creating a constructor and attributes

I also thought that lab D looked intimidating but I was to work through it fairly quickly once I got through a small snag. I was able to create the getters and setters using the module example but since one of my objects was an integer I had to make an adjustment. I initially used the ".title()" for the integer and Doug had to point out that this was only used for strings. Once I fixed this the function worked how I wanted. I found it interesting that it worked without the need for a field. An example of my properties are provided below.

```python
     # -- Properties -- #
    @property
    def position(self):
        return self.__position
    @position.setter
    def position(self, value):
        if int(value).isnumeric():
              self.__position = value
        else:
            raise Exception('The position can\'t be cryptic')

    @property
    def title(self):
        return self.__title.title()
    @title.setter
    def title(self, value):
        if str(value).isnumeric():
            raise Exception('The title can\'t be cryptic')
        else:
            self.__title = value

    @property
    def length(self):
        return self.__length.title()
    @length.setter
    def length(self, value):
        if str(value).isnumeric():
            raise Exception('The length can\'t be cryptic')
        else:
            self.__length = value
```

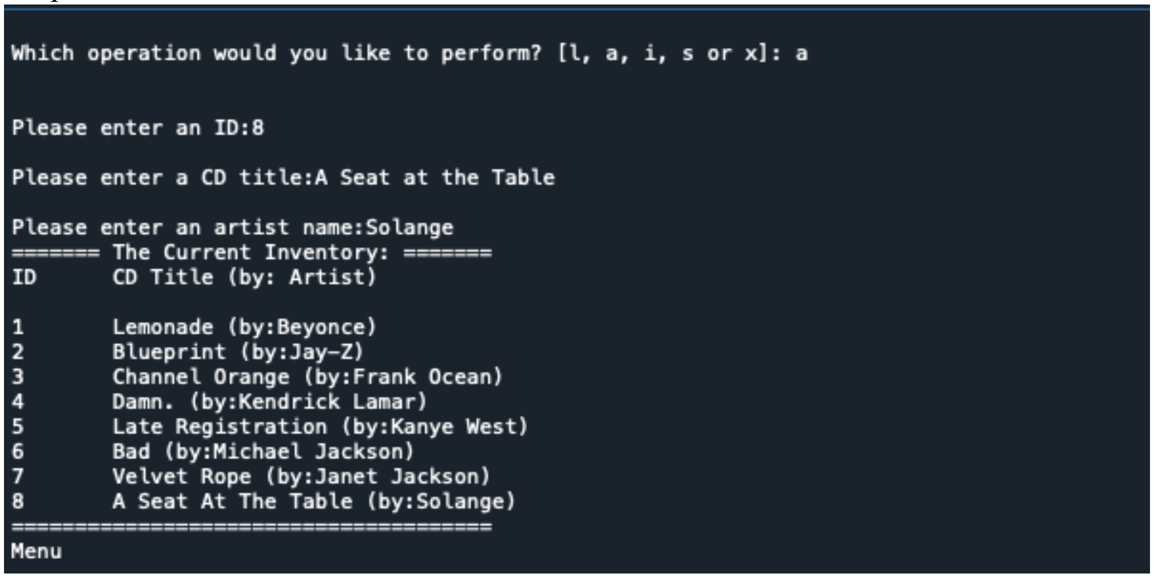
Listing 2: Ex of creating properties

The last lab that I worked on this week was lab E. I was somewhat confused about what the "__str__" did but I realized that this just formats the data how you would like it to be seen. This proved to be very useful once I got into the assignment. Since the last few weeks of assignments have been similar it was not too difficult to set up the bulk of the code. The biggest difference was that I had to adjust to using objects instead of dictionaries and I had to use a class to create the objects that would be added into the 2D list. My biggest difficulty was with my add CD function and how to translate the user inputs to my class. This was solved when I fixed my "__str__" method to the format that I wanted my data to appear in. An example of this method is provided below.

```
    def __str__(self):
        return f'{self.cd_id}\t{self.cd_title} (by:{self.cd_artist})'
```

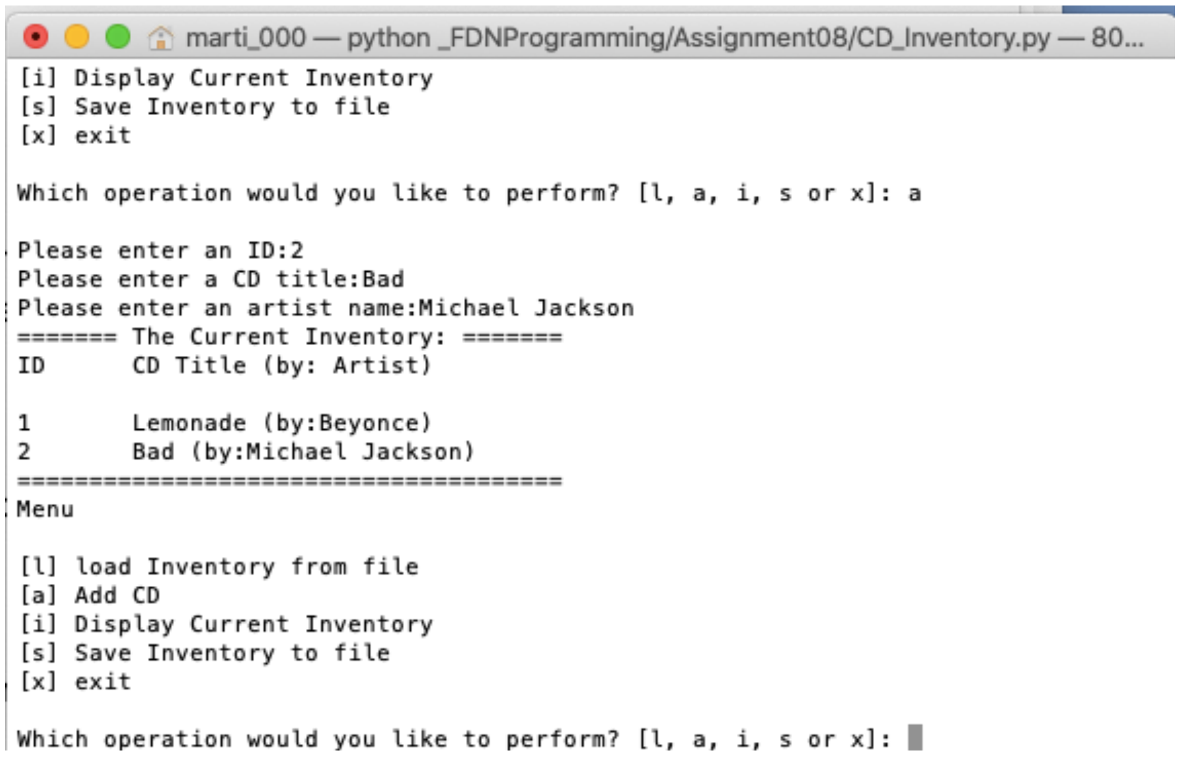
Listing 3: Ex of __str__ method

I then had to adjust all of the other functions to react to objects and not dictionaries which were a couple of small adjustments. After I added some error handling and updated my docstrings I felt that my code was adequate for the assignment. I tested all of my functions and they worked as I intended and I took screenshots of my add function working in both Spyder and the terminal. Lastly, I posted my script to GitHub. The screenshots are provided below.



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


Please enter an ID:8

Please enter a CD title:A Seat at the Table

Please enter an artist name:Solange
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Lemonade (by:Beyonce)
2       Blueprint (by:Jay-Z)
3       Channel Orange (by:Frank Ocean)
4       Damn. (by:Kendrick Lamar)
5       Late Registration (by:Kanye West)
6       Bad (by:Michael Jackson)
7       Velvet Rope (by:Janet Jackson)
8       A Seat At The Table (by:Solange)
======================================
Menu
```

Figure 1: Ex of Add CD function working in Spyder



```
[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

Please enter an ID:2
Please enter a CD title:Bad
Please enter an artist name:Michael Jackson
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Lemonade (by:Beyonce)
2       Bad (by:Michael Jackson)
======================================
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 2: Ex of add function working in terminal

Summary

In this assignment, we continued to work with classes but we used them for a different purpose than we had previously. We used them to work as the blueprint for objects as we have begun to dive into object oriented programming. We also worked on the different parts of this blueprint step-by-step. These parts include fields, constructors and attributes, properties, and methods. It appears that there is a lot of versatility to create these objects because all of these parts do not need to be utilized at the same time. They all serve different purposes depending on what you want to do and how you want to initialize the object. We then worked to integrate objects with the work that we have been doing for the past few weeks to show how there are many different ways to store data. I am interested to see what more can be done with OOP in the next module.

Appendix

Using planetb's webpage (external reference). Retrieved 2020-Sep-2
Using Saravji's Hut Youtube Page (external reference). Retrieved 2020-Sep-2
Using Saravji's Hut Youtube Page (external reference). Retrieved 2020-Sep-2
Using Real Python's webpage (external reference). Retrieved 2020-Sep-2
Dawson, Michael. Python Programming for the Absolute Beginner: 3rd Edition. Kindle ed., Cengage Learning, 2010

Listing for LAB08_B.py

```
#----------------------------------------------#
# Title: LAB08_B.py
# Desc: TrackInfo demonstrator class 2
# Change Log: (Who, when, what)
# MStallworth, 2020-Aug-30, Created File
# MStallworth, 2020-Aug-30, created a constructor and added code to populate fields
#----------------------------------------------#

# -- DATA -- #


# -- PROCESSING -- #

```

```python
class TrackInfo():
    # -- Fields -- #
    Position = int()
    Title = ''
    Length = ''

    # -- Constructor -- #
    def __init__(self, pos, ttl, leng):
        # -- Attributes -- #
        self.Position = pos
        self.Title = ttl
        self.Length = leng

    # -- Properties -- #

    # -- Methods -- #
objTrack1 = TrackInfo(4, 'Thriller', '5.58')
objTrack2 = TrackInfo(6, 'Billie Jean', '4.54')

print('Position: {}, Title: {}, Length: {}'.format(objTrack1.Position, objTrack1.Title, objTrack1.Length))
print('Position: {}, Title: {}, Length: {}'.format(objTrack
```

Listing for LAB08_D.py

```python
#---------------------------------------------#
# Title: LAB08_D.py
# Desc: TrackInfo demonstrator class 4
# Change Log: (Who, when, what)
# MStallworth, 2020-Aug-31, added code to create properties for attributes
# MStallworth, 2020-Aug-31, Created File.
# MStallworth, 2020-Aug-31, added code to create properties for attributes
#---------------------------------------------#

# -- DATA -- #


# -- PROCESSING -- #

class TrackInfo():
    # -- Fields -- #

    # -- Constructor -- #
    def __init__(self, pos, ttl, leng):
        # -- Attributes -- #
        self.__position = pos
        self.__title = ttl
        self.__length = leng

    # -- Properties -- #
    @property
    def position(self):
        return self.__position
    @position.setter
    def position(self, value):
        if int(value).isnumeric():
              self.__position = value
        else:
            raise Exception('The position can\'t be cryptic')

    @property
    def title(self):
        return self.__title.title()
    @title.setter
    def title(self, value):
        if str(value).isnumeric():
            raise Exception('The title can\'t be cryptic')
        else:
            self.__title = value

    @property
    def length(self):
        return self.__length.title()
    @length.setter
    def length(self, value):
        if str(value).isnumeric():
            raise Exception('The length can\'t be cryptic')
        else:
            self.__length = value
    # -- Methods -- #
objTrack1 = TrackInfo(4, 'Thriller', '5.58')
objTrack2 = TrackInfo(6, 'Billie Jean', '4.54')

print('Position: {}, Title: {}, Length: {}'.format(objTrack1.position, objTrack1.title, objTrack1.length))
print('Position: {}, Title: {}, Length: {}'.format(objTrack2.position, objTrack2.title, objTrack2.length))
```

Listing for CD_Inventory.py

```python
#------------------------------------------#
# 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
# MStallworth, 2020-Sep-1, added code to CD class, loading and saving inventory
# MStallworth, 2020-Sep-1, added code to main body and various menu functions under IO class
# MStallworth, 2020-Sep-2, added __str__ function to CD class and fixed docstrings
# MStallworth, 2020-Sep-2, added structured error handling where applicable
# MStallworth, 2020-Sep-2, fixed function for adding CD
# MStallworth, 2020-Sep-2, tested code and all functions
#------------------------------------------#

# -- DATA -- #
import pickle
strFileName = 'cdInventory.dat'
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, cd_id, cd_title, cd_artist):
        # -- Attributes -- #
        self.__cd_id = int(cd_id)
        self.__cd_title = cd_title
        self.__cd_artist = cd_artist

    # -- Properties -- #
    @property
    def cd_id(self):
        return self.__cd_id
    @cd_id.setter
    def cd_id(self, value):
        self.__cd_id = value
    @property
    def cd_title(self):
        return self.__cd_title.title()
    @cd_title.setter
    def cd_title(self, value):
        self.__cd_title = value
    @property
    def cd_artist(self):
        return self.__cd_artist.title()
    @cd_artist.setter
    def cd_artist(self, value):
        self.__cd_artist = value


    # -- Methods -- #
    def __str__(self):
        return f'{self.cd_id}\t{self.cd_title} (by:{self.cd_artist})'

# -- 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)

    """
    # TODone Add code to process data from a file
    @staticmethod
    def load_inventory(file_name):
        """Function to manage data ingestion from file to a list of objects

        Reads the data from 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 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.
        """
        with open(file_name, 'rb') as objFile:
            data = pickle.load(objFile)
            return data


    # TODone Add code to process data to a file
    @staticmethod
    def save_inventory(file_name, lst_Inventory):
        """Function to write in memory data into a file
        Saves the data into identified file name from a 2D table


        Args:
            file_name (string): name of the file used to save the data
            lst_Inventory: 2D data structure used to save the in memory data


        Returns:
            None.
        """
        with open(file_name, 'wb') as objFile:
            pickle.dump(lst_Inventory, objFile)


# -- PRESENTATION (Input/Output) -- #
class IO:
    # TODone add docstring
    """ Handling Input/Output

    properties:

    methods:
        print_menu(): -> None
        menu_choice(): -> choice (letter corresponding to menu option)
        show inventory(table): -> None
        add entry(): -> cd_id(ID for data entry), cd_title(input for CD title, cd_artist(input for artist for CD))
    """
    # 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
    @staticmethod
    def show_inventory(table):
        """Displays current inventory table


        Args:
            table (list of objects): 2D data structure (list of objects) that holds the data during runtime.

        Returns:
            None.

        """
        print('======= The Current Inventory: =======')
        print('ID\tCD Title (by: Artist)\n')
        for row in table:
            print(row)
        print('======================================')
    # TODone add code to get CD data from user
    @staticmethod
    def add_entry():
        """Adds user input into the in memory inventory
        using .add_data() from Data Processing Class


        Args:
            None.


        Returns:
            cd_id (string): ID for data entry
            cd_title (string): Input for CD Title
            cd_artist (string): Input for Artist Name
        """
        cd_id = input('Please enter an ID:').strip()
        cd_title = input('Please enter a CD title:').strip()
        cd_artist = input('Please enter an artist name:').strip()
        return cd_id, cd_title, cd_artist


# -- Main Body of Script -- #
# TODone Add Code to the main body
# Load data from file into a list of CD objects on script start
try:
    lstOfCDObjects.clear()
    CDfile = FileIO.load_inventory(strFileName)
    lstOfCDObjects.extend(CDfile)
    IO.show_inventory(lstOfCDObjects)
    print('Welcome back!')
except FileNotFoundError:
    print('\nThere is no file currently avaiable!\n')
    print('Hello. Welcome to the program! Creating new file for you...\n')
    FileIO.save_inventory(strFileName, lstOfCDObjects)
    print('File has been created. You may now enter the program!\n')
# Display menu to user
while True:
    IO.print_menu()
    strChoice = IO.menu_choice()
    # let user exit program
    if strChoice == 'x':
        print('\nGoodbye.')
        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...')
            lstOfCDObjects.clear()
            CDfile = FileIO.load_inventory(strFileName)
            lstOfCDObjects.extend(CDfile)
            IO.show_inventory(lstOfCDObjects)
         else:
            input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
            IO.show_inventory(lstOfCDObjects)
            continue
    # let user add data to the inventory
    elif strChoice == 'a':
        ID, Title, Artist = IO.add_entry()
        try: # error handling to make sure ID is an integer
```

```
                    cd_id = int(ID)
                except ValueError:
                    print('\nYour ID must be an integer!\n')
                    print('This entry was not added to the inventory. Try again.\n')
                    continue
                new_cd = CD(ID, Title, Artist)
                lstOfCDObjects.append(new_cd)
                IO.show_inventory(lstOfCDObjects)
                continue
            # show user current inventor
            elif strChoice == 'i':
                IO.show_inventory(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()
                # Process choice
                if strYesNo == 'y':
                    # save data
                    FileIO.save_inventory(strFileName, lstOfCDObjects)
                    print('\nYour inventory was successfully saved to the file!\n')
                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')
```