Josephine Gnanaraj
03\07\2021
Foundations Of Programming: Python
Assignment_08

Introduction

In this week's module we learnt working with the basic but 'new' concept that everything is an object. That Python is a pure Object-Oriented Language and took first steps into understanding OOP. Learnt about creating Classes to define Objects, write methods and create attributes for objects, instantiate Objects (from classes) and restrict Access to an object's attributes.

Object-Oriented Programming (OOP) in Python

Object-oriented programming (OOP) is a method of structuring a program by bundling related properties and behaviors into individual objects. Conceptually, objects are like the components of a system. Think of a program as a factory assembly line of sorts. At each step of the assembly line a system component processes some material, ultimately transforming raw material into a finished product. An object contains data, like the raw or preprocessed materials at each step on an assembly line, and behavior, like the action each assembly line component performs.

Working with Classes:

A class is a code template for creating objects. Objects have member variables and have behavior associated with them. In python a class is created by the keyword class.

It is a blueprint for how something should be defined. It doesn't actually contain any data. The Dog class specifies that a name and an age are necessary for defining a dog, but it doesn't contain the name or age of any specific dog.

```
class Dog: pass
```

The body of the Dog class consists of a single statement:
the pass keyword. pass is often used as a placeholder indicating where code will eventually go. It allows you to run this code without Python throwing an error.

Working with Attributes:

A class by itself is of no use unless there is some functionality associated with it. Functionalities are defined by setting attributes, which act as containers for data and functions related to those attributes.

There are a number of properties that we can choose from, including name, age, coat color, and breed. The properties that all Dog objects must have are defined in a method called .__init__(). Every time a new Dog object is created, .__init__() sets the initial state of the object by assigning the values of the object's properties. That is,._init__() initializes each new instance of the class.
You can give .__init__() any number of parameters, but the first parameter will always be a variable called self. When a new class instance is created, the instance is automatically passed to the self-parameter in .__init__() so that new attributes can be defined on the object.

```
class Dog: def __init__(self, name, age): self.name = name self.age = ag
e
```

In the body of .__init__(), there are two statements using the self-variable:

1. self.name = name creates an attribute called name and assigns to it the value of the name parameter.
2. self.age = age creates an attribute called age and assigns to it the value of the age parameter.

Attributes created in .__init__() are called instance attributes. An instance attribute's value is specific to a particular instance of the class. All Dog objects have a name and an age, but the values for the name and age attributes will vary depending on the Dog instance. On the other hand, class attributes are attributes that have the same value for all class instances. You can define a class attribute by assigning a value to a variable name outside of .__init__().

Working with Constructors:

Constructors are generally used for instantiating an object. The task of **constructors** is to initialize (assign values) to the data members of the class when an object of class is created. In **Python** the __init__() method is called the **constructor** and is always called when an object is created.

Types of constructors:

- **default constructor:** The default constructor is simple constructor which doesn't accept any arguments. Its definition has only one argument which is a reference to the instance being constructed.
- **parameterized constructor: constructor** with parameters is known as parameterized constructor. The parameterized constructor take its first argument as a reference to the instance being constructed

known as self and the rest of the arguments are provided by the programmer.

Syntax of constructor declaration:
def __init__(self): # body of the constructor

Working with properties:

In Python, the main purpose of Property() function is to create property of a class. Return: Returns a property attribute from the given getter, setter and deleter. Note: If no arguments are given, property() method returns a base property attribute that doesn't contain any getter, setter or deleter.

In Python, the main purpose of **Property()** function is to create property of a class.
Syntax: property(fget, fset, fdel, doc)
Parameters:
fget() – used to get the value of attribute
fset() – used to set the value of attribute
fdel() – used to delete the attribute value
doc() – string that contains the documentation (docstring) for the attribute
Return: Returns a property attribute from the given getter, setter and deleter.

Summary

In this assignment\module we worked with exploring Object-Oriented Programming. We created classes to define objects, wrote methods and create attributes for objects, instantiated objects (from classes) and restricted access to an object's attributes.
We used Spyder as our IDE.

CDInventory.py

```
#------------------------------------------#
# 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
# Jgnanaraj, 2021-Feb-28, Modified File and added requirements
#------------------------------------------#

# -- 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
    #----Constructor--------
    def __init__(self,cd_id,cd_title,cd_artist):
        self.__cd_id = 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):
    if type(value) == int:
        self.__cd_id = value
    else:
        raise Exception("ID has to be numeric!")


@property
def cd_title(self):
    return self.__cd_title

@cd_title.setter
def cd_title(self, value):
    if type(value) == str:
        self.__cd_title = value
    else:
        raise Exception("Title has to be a string!")

@property
def cd_artist(self):
    return self.__cd_artist

@cd_artist.setter
def cd_artist(self, value):
    if type(value) == str:
        self.__cd_artist = value
    else:
        raise Exception("Artist has to be a string!")
```

```
# -- 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, lstOfCDs):
        """ Loads inventory data from the text file into a list in object
format."""
        objFile = None
        lstOfCDs.clear()
        objFile = open(file_name, 'r')

        try:
            for line in objFile:
                data = line.strip().split(',')
                cdObj = CD(data[0],data[1],data[2])
                lstOfCDs.append(cdObj)
        except:
            ("Something happened here - maybe there is nothing in the file
yet?")
        objFile.close()
```

```
    # TODOne Add code to process data to a file
    @staticmethod
    def save_inventory(file_name, lst_Inventory):
        """Saves data from a list of objects to an inventory file"""
        new_line = ""
        objFile = None
        counter = 0

        while counter < len(lst_Inventory):
            str_cd_id = str(lst_Inventory[counter].cd_id)
            new_line = new_line + str_cd_id + ","
            new_line = new_line + lst_Inventory[counter].cd_title + ","
            new_line = new_line + lst_Inventory[counter].cd_artist + ","
            new_line = new_line[:-1] + "\n"
            counter += 1

        objFile = open(strFileName, "w")
        objFile.write(new_line)
        objFile.close()
        print("Data saved!")


#-- PRESENTATION (Input/Output) --#
class IO:
    #TODOne add docstring
    """Displays data to the screen:

    properties: None

    methods:
        print_menu()
        menu_choice()
        show_inventory()
```

```
        ask_user_data()

    """

    #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('Select the operation 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(lstObj):
        """ Prepares the CD inventory for proper display and formatting to the screen

        Args: the inventory table of CD objects
        Returns: None

        """

        counter = 0

        print('======= The Current Inventory: =======')
        print('ID\tCD Title (by: Artist)\n')

        while counter < len(lstObj):
            print('{}\t{} (by:{})'.format(lstObj[counter].cd_id, lstObj[counter].cd_title, lstObj[counter].cd_artist))
            counter += 1
        print('======================================')

    #TODOne add code to get CD data from user
    @staticmethod
    def ask_user_data():
        """Asks for user data - the ID of the new CD, the Title and the Artist
```

```
        Args: None
        Returns: The ID, the CD Title and the Artist of the title
        """

        #catching errors like entering non-numeric entries
        try:
            ID = int(input('Enter ID: ').strip())
            Title = input('What is the CD\'s title? ').strip()
            Artist = input('What is the Artist\'s name? ').strip()
            return ID, Title, Artist
        except ValueError as e:
            print("Only numbers allowed for ID")
            print("Error info: ")
            print(type(e),e,e.__doc__, sep="\n")


#-- Main Body of Script --#

#TODOne Add Code to the main body

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

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...')
            FileIO.load_inventory(strFileName, lstOfCDObjects)
            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
    if strChoice == 'a':
        #catching errors when erroneous data is not passed
from IO.ask_user_data()
        try:
            intID, strTitle, stArtist = IO.ask_user_data()

        except TypeError as e:
            print("Error in data entry.")
            print("Error info: ")
            print(type(e),e, sep="\n")
            continue

        newObj = CD(intID, strTitle, stArtist)
        lstOfCDObjects.append(newObj)
        IO.show_inventory(lstOfCDObjects)
        continue
```

```
    #show user current inventory
    if strChoice == 'i':
        IO.show_inventory(lstOfCDObjects)
        continue

    #let user save inventory to file
    if strChoice == 's':
        IO.show_inventory(lstOfCDObjects)
        strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()

        if strYesNo == 'y':
            FileIO.save_inventory(strFileName, lstOfCDObjects)
        else:
            input('Inventory NOT saved to file. Press [ENTER] to return to the
menu.')
        continue
```

Created a script and Compiled to verify output:

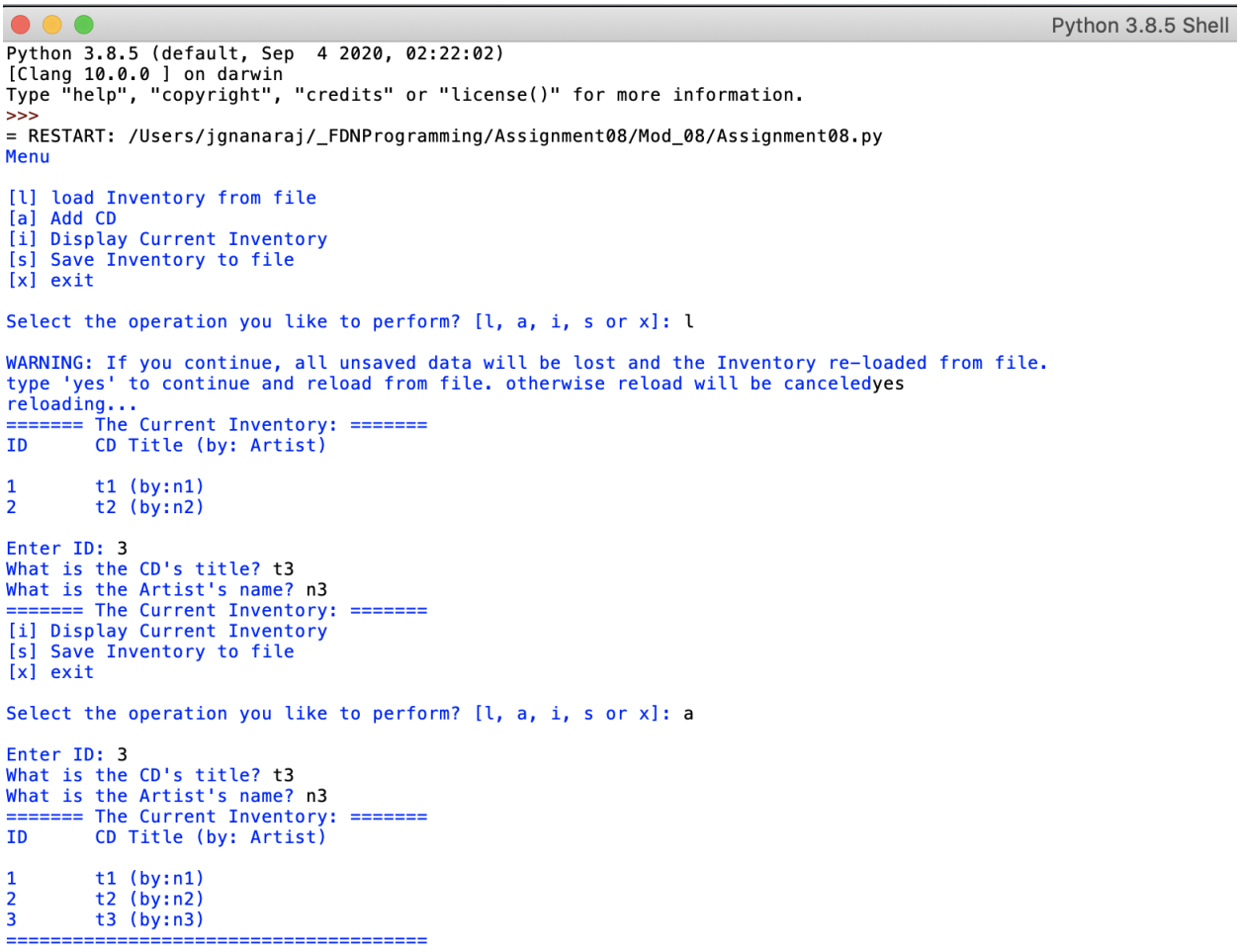

Figure 1 - Screen capture

```
Menu

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

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

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

1       t1 (by:n1)
2       t2 (by:n2)
3       t3 (by:n3)
======================================
Menu

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

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

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

1       t1 (by:n1)
2       t2 (by:n2)
3       t3 (by:n3)
======================================
Save this inventory to file? [y/n] y
Data saved!
Menu

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

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

>>> |
```

Figure 2 - Screen capture

References:

https://realpython.com/python3-object-oriented-programming/
https://www.hackerearth.com/practice/python/object-oriented-programming/classes-and-objects-i/tutorial/
Python OOP Tutorial 1: Classes and Instances
https://www.pythonmorsels.com/topics/what-is-self/
https://www.geeksforgeeks.org/constructors-in-python/
https://www.geeksforgeeks.org/python-property-function/
FDN_Py_Module_08.pdf