[5] Object Oriented Programming

July 4, 2022

1 Procedural and Object-Oriented Programming

Concept: Procedural programming is a method of writing software. It is a programming practice centered on the procedures or actions that take place in a program. Object-oriented programming is centered on objects. Objects are created from abstract data types that encapsulate data and functions together.

There are primarily two methods of programming in use today: procedural and objectoriented. The earliest programming languages were procedural, meaning a program was made of one or more procedures. You can think of a procedure simply as a function that performs a specific task such as gathering input from the user, performing calculations, reading or writing files, displaying output, and so on. The programs that you have written so far have been procedural in nature.

This section explains how to use Classes and Objects in Python. Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects. This is a Jupyter notebook. It is useful because you can both compose and run code in the same place. This is a "markdown block" it is use to provide none code information. The next block is a "code" block, it contains runnable python code.

[1]:
```
#A very basic class would look something like this:
class MyClass:
    variable = "GIFT University"

    def function(self):
        print("This is a message inside the class.")
```

[2]:
```
# We'll explain why you have to include that "self" as a parameter a little bit␣
 ↪later.
# First, to assign the above class(template) to an object you would do the␣
 ↪following:

class MyClass:
    variable = "GIFT University"

    def function(self):
        print("This is a message inside the class.")

```

```
myobjectx = MyClass()

# Now the variable "myobjectx" holds an object of the class "MyClass" that␣
↪contains the
# variable and the function defined within the class called "MyClass".
```

2 Accessing Object Variables

[3]:
```
# So for instance the below would output the string "blah":
class MyClass:
    variable = "GIFT University"

    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()

print(myobjectx.variable)
```

```
GIFT University
```

[4]:
```
# You can create multiple different objects that are of the same class(have the␣
↪same variables and functions defined).
# However, each object contains independent copies of the variables defined in␣
↪the class.
# For instance, if we were to define another object with the "MyClass" class␣
↪and then change the string in the
# variable above:

class MyClass:
    variable = "GIFT University"

    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()
myobjecty = MyClass()

myobjecty.variable = "Gujranwala"

# Then print out both values
print(myobjectx.variable)
print(myobjecty.variable)
```

```
GIFT University
Gujranwala
```

3 Accessing Object Functions

[5]:
```
# To access a function inside of an object you use notation similar to␣
 ↪accessing a variable:
class MyClass:
    variable = "GIFT University"

    def function(self):
        print("Hello, Welcome to AI Lab")

myobjectx = MyClass()

myobjectx.function()

# The above would print out the message, "This is a message inside the class."
```

```
Hello, Welcome to AI Lab
```

4 Unserstanding initializer method and self parameter

Lets Consider a class named **Coin** that simulates a coin that can be flipped

[6]:
```
import random

# The Coin class simulates a coin that can
# be flipped.

class Coin:
    # The _ _init_ _ method initializes the
    # sideup data attribute with 'Heads'.

    def __init__(self):
        self.__sideup = 'Heads'
        # The toss method generates a random number
        # in the range of 0 through 1. If the number
        # is 0, then sideup is set to 'Heads'.
        # Otherwise, sideup is set to 'Tails'.

    def toss(self):
        if random.randint(0, 1) == 0:
            self.__sideup = 'Heads'
        else:
            self.__sideup = 'Tails'

    # The get_sideup method returns the value
    # referenced by sideup.
    def get_sideup(self):
        return self.__sideup
```

Take a closer look at the header for each of the method definitions (lines 10, 17, and 26) and notice that each method has a parameter variable named **self**:

```
Line 10: def _ _init_ _(self):
Line 17: def toss(self):
Line 26: def get_sideup(self):
```

In object-oriented programming that a method operates on a specific object's data attributes. When a method executes, it must have a way of knowing which object's data attributes it is supposed to operate on. That's where the self parameter comes in. When a method is called, Python makes the self parameter reference the specific object that the method is supposed to operate on.

Let's look at each of the methods. The first method, which is named **__ *init* __**, is defined in lines 10 through 11:

```
def __init__(self):
        self.__sideup = 'Heads'
```

Most Python classes have a special method named _ *init , which is automatically executed when an instance of the class is created in memory. The init* _ method is commonly known as an initializer method because it initializes the object's data attributes.

5 Private Variable

"Private" instance variables that cannot be accessed except from inside an object don't exist in Python. Look at the above code line no 11

```
 def __init__(self):
        self.__sideup = 'Heads'
```

___sideup variable is a private variable of class Coin

6 main function for Coin Class

[7]:
```
# The main function.
def main():
    # Create an object from the Coin class.
    my_coin = Coin()
    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

    # Toss the coin.
    print('I am going to toss the coin ten times:')
    for count in range(5):
        my_coin.toss()
        print(my_coin.get_sideup())
```

```
# Call the main function.
main()
```

```
This side is up: Heads
I am going to toss the coin ten times:
Heads
Tails
Heads
Tails
Tails
```

7 The __ *str* __ method

Quite often we need to display a message that indicates an object's state. An object's state is simply the values of the object's attributes at any given moment.

Lets add this method to the Employeee class

[8]:
```
class Employee:

     # The _ _init_ _ method initializes the
    # attributes of the employee
    def __init__(self):

        self.__emp_id = 12345
        self.__name = "Ali"
        self.__salary = 5000.50

    #The __str__ method to display the employee
    # object state
    def __str__(self):

        return " The id of emplyee is:  {} \n The name of employee is: {} \n␣
↪The salary of employee is: {}".format(self.__emp_id,self.__name,self.
↪__salary)

def main():
    emp = Employee()
    print(emp)
if "__name__ == main":

    main()
```

```
 The id of emplyee is:  12345
 The name of employee is: Ali
 The salary of employee is: 5000.5
```

8 Inheritance

[9]:
```
# The Automobile class holds general data
# about an automobile in inventory.

class Automobile:


    # The _ _init_ _method accepts arguments for the
    # make, model, mileage, and price. It initializes
    # the data attributes with these values.
    def __init__(self, make, model, mileage, price):
            self.__make = make
            self.__model = model
            self.__mileage = mileage
            self.__price = price
    # The following methods are mutators for the
    # class's data attributes.

    def set_make(self, make):
        self.__make = make

    def set_model(self, model):
        self.__model = model

    def set_mileage(self, mileage):
        self.__mileage = mileage

    def set_price(self, price):
        self.__price = price

     # The following methods are the accessors
     # for the class's data attributes.

    def get_make(self):
        return self.__make

    def get_model(self):
        return self.__model

    def get_mileage(self):
        return self.__mileage

    def get_price(self):
        return self.__price
```

[10]:
```
# The Car class represents a car. It is a subclass
# of the Automobile class.

class Car(Automobile):

    # The _ _init_ _ method accepts arguments for the
    # car's make, model, mileage, price, and doors.

    def __init__(self, make, model, mileage, price, doors):


        # Call the superclass's _ _init_ _ method and pass
        # the required arguments. Note that we also have
        # to pass self as an argument.
        Automobile.__init__(self, make, model, mileage, price)

        # Initialize the _ _doors attribute.
        self.__doors = doors

    # The set_doors method is the mutator for the
    # _ _doors attribute.

    def set_doors(self, doors):
            self.__doors = doors

     # The get_doors method is the accessor for the
     # _ _doors attribute.
    def get_doors(self):
         return self.__doors
```

[11]:
```
def main():
    # Create an object from the Car class.
    # The car is a 2007 Audi with 12,500 miles, priced
    # at $21,500.00, and has 4 doors.
    used_car = Car('Audi', 2007, 12500, 21500.00, 4)

    # Display the car's data.
    print('Make:', used_car.get_make())
    print('Model:', used_car.get_model())
    print('Mileage:', used_car.get_mileage())
    print('Price:', used_car.get_price())
    print('Number of doors:', used_car.get_doors())

    # Call the main function.
main()
```

```
Make: Audi
```

```
Model: 2007
Mileage: 12500
Price: 21500.0
Number of doors: 4
```

9 Exercise

9.1 Employee Class

Write a class named Employee that holds the following data about an employee in attributes: name, ID number, department, and job title.

Once you have written the class, write a program that creates three Employee objects to hold the following data:

Name	ID	Number	Department	Job Title
Susan Meyers	47899		Accounting	Vice President
Mark Jones	39119		IT	Programmer
Joy Rogers	81774		Manufacturing	Engineer

he program should store this data in the three objects and then display the data for each employee on the screen.

9.2 RetailItem Class

Write a class named RetailItem that holds data about an item in a retail store. The class should store the following data in attributes: item description, units in inventory, and price.

Once you have written the class, write a program that creates three RetailItem objects and stores the following data in them:

	Description	Units in Inventory	Price
Item #1	Jacket	12	59.95
Item #2	Designer Jeans	40	34.95
Item #3	Shirt	20	24.95

9.3 Cash Register

This exercise assumes that you have created the RetailItem class for Programming Exercise 2. Create a CashRegister class that can be used with the RetailItem class. The CashRegister class should be able to internally keep a list of RetailItem objects. The class should have the following methods:

- A method named **purchase_item** that accepts a RetailItem object as an argument. Each time the purchase_item method is called, the **RetailItem object** that is passed as an argument should be added to the list.
- A method named **get_total** that returns the total price of all the RetailItem objects stored in the CashRegister object's internal list.

- A method named **show__items** that displays data about the RetailItem objects stored in the CashRegister object's internal list.
- A method named **clear** that should clear the CashRegister object's internal list.

Demonstrate the **CashRegister** class in a program that allows the user to select several items for purchase. When the user is ready to check out, the program should display a list of all the items he or she has selected for purchase, as well as the total price.

9.4 Employee and ProductionWorker Classes

Write an Employee class that keeps data attributes for the following pieces of information: * Employee name * Employee number

Next, write a class named ProductionWorker that is a subclass of the Employee class. The ProductionWorker class should keep data attributes for the following information:

- Shift number (an integer, such as 1, 2, or 3)
- Hourly pay rate

The workday is divided into two shifts: day and night. The shift attribute will hold an integer value representing the shift that the employee works. The day shift is shift 1 and the night shift is shift 2. Write the appropriate accessor and mutator methods for each class.

Once you have written the classes, write a program that creates an object of the ProductionWorker class and prompts the user to enter data for each of the object's data attributes. Store the data in the object and then use the object's accessor methods to retrieve it and display it on the screen.

9.5 Person and Customer Classes

Write a class named Person with data attributes for a person's name, address, and telephone number. Next, write a class named Customer that is a subclass of the Person class. The Customer class should have a data attribute for a customer number and a Boolean data attribute indicating whether the customer wishes to be on a mailing list. Demonstrate an instance of the Customer class in a simple program.