Inspiring Excellence

Course Title: Programming Language II

Course Code: CSE 111

Semester: Summer 2020

Topic: Object-Oriented Programming (OOP)

Table of Contents

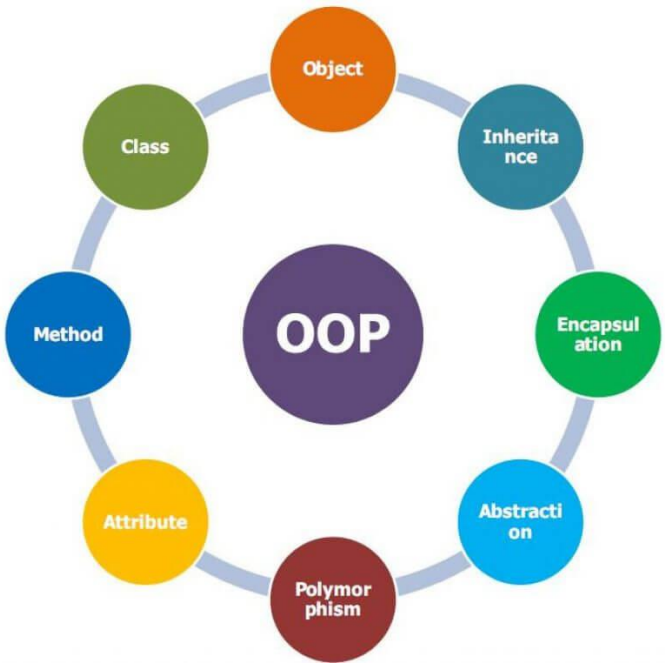
1. Object Oriented Programming (OOP):

Python is a multi-paradigm programming language. It supports different programming approaches. One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

Object-oriented programming is a programming paradigm that provides a means of structuring programs so that properties and behaviors are bundled into individual objects. OOP reflects the real world behavior of how things work and make visualization easier because it is closest to real world scenarios. We can reuse the code through inheritance, this saves time, and shrinks our project and there are flexibility through polymorphism.

Application of OOP:

Object-Oriented Programming has multiple applications-



2. Class:

A class in object-oriented programming serves as a blueprint for the object. A class can be considered as a map for the house. We can get an idea of what the house looks like by simply seeing the map. However, a class itself is nothing. For instance, a map is not a house, it only explains how the actual house will look.

Each class contains some data definitions (called fields), together with methods to manipulate that data. When the object is instantiated from the class, an instance variable is created for each field in the class.

3. Object:

Earlier, we said that a class provides a blueprint. However, to actually use the objects and methods of a class, we need to create an object out of that class. Object are the basic run time entities in an object oriented system. Objects are the variables of the type class.

An object is also called an instance; therefore, the process of creating an object of a class is called instantiation. In Python, to create an object of a class we simply need to write the class name followed by opening and closing parenthesis.

```
class Dog:

    def __init__(self, name, age):
        self.name = name
        self.age = age
ozzy = Dog("Ozzy", 2)

print(ozzy.name)

print(ozzy.age)
```

Output:

Ozzy

2

In the above program, we created a class with the name **Dog**. Then, we define attributes. The attributes are a characteristic of an object.

These attributes are defined inside the `__init__` method of the class. It is the initializer method that is first run as soon as the object is created. Then, we create instances of the **Dog** class.

3.1 Instance variable:

Instance variables are owned by instances of the class. This means that for each object or instance of a class, the instance variables are different. Unlike class variables, instance variables are defined within methods.

```
class Shark:
    def __init__(self, name, age):
        self.name = name
        self.age = age

new_shark = Shark("Sammy", 5)
print(new_shark.name)
print(new_shark.age)
```

Output:

Sammy

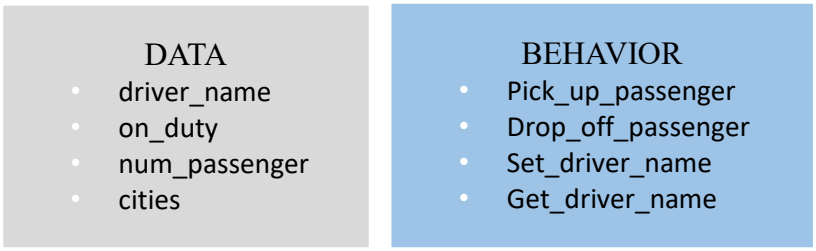
5

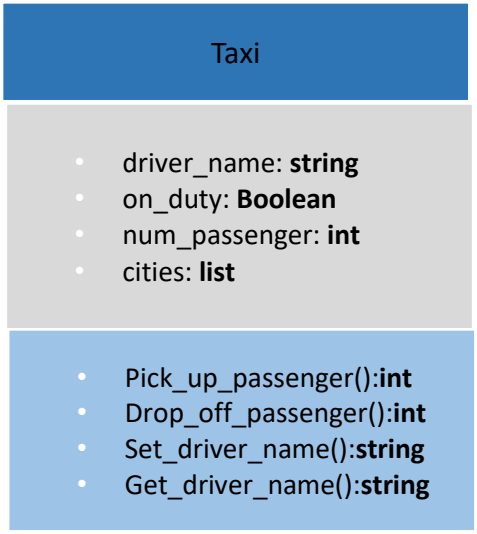
The output we receive is made up of the values of the variables that we initialized for the object instance of `new_shark`.

3.2 Components of Object:

Every object has two main components-

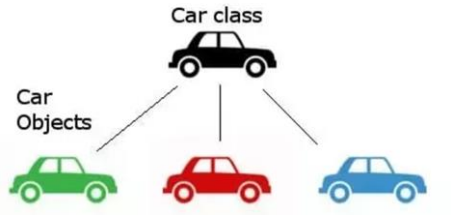
1. Data (the attributes about it)
2. Behavior(the methods)





Here for the Taxi class specific attributes and methods mentioned.

So far we know that, a class is the blueprint for the objects created from that class and objects are the variables of the type class. Here following pictures shows that how objects are create from the class.



CLASS

	General Attributes and Behaviors
Name	Car
Attributes	*Color *Brand *Fuel
Behaviors	*Display()

OBJECT	**OBJECT**	**OBJECT**
Specific Attributes and Behaviors	**Specific Attributes and Behaviors**	**Specific Attributes and Behaviors**
Mustang Car	Prius Car	Golf Car
*Green *Ford *Gasoline	*Red *Toyota *Electricity	*Blue *Volkswagen *Diesel
*Display()	*Display()	*Display()

4. Constructor:

Python Constructor in object-oriented programming is a special kind of method/function we use to initialize instance members of that class. It is used for initializing the instance members when we create the object of a class. If we create four objects, the class constructor is called four times. Every class has a constructor, but it's not required to explicitly define it. Every class must have a constructor, even if it simply relies on the default constructor.

```
class DemoClass:
    # constructor
    def __init__(self):
        # initializing instance variable
        self.num=100
    # a method
    def read_number(self):
        print(self.num)
# creating object of the class. This invokes constructor
obj = DemoClass()
# calling the instance method using the object obj
obj.read_number()
```

Output:

100

5.1 Python __init__:

"__init__" is a reserved method in python classes. It is known as a constructor in OOP concepts. This method called when an object is created from the class and it allows the class to initialize the attributes of a class. It accepts the self-keyword as a first argument which allows accessing the attributes or method of the class. We can pass any number of arguments at the time of creating the class object, depending upon the __init__() definition. It is mostly used to initialize the class attributes.

```
class Employee:
    def __init__(self, name, id):
        self.id = id
        self.name = name

    def display(self):
        print("ID: %d \nName: %s" % (self.id, self.name))
emp1 = Employee("John", 101)
emp2 = Employee("David", 102)

# accessing display() method to print employee 1 information
emp1.display()
# accessing display() method to print employee 2 information
emp2.display()
```

Output:

ID: 101

Name: John

ID: 102

Name: David

5.2 Types of Constructor:

Constructors can be of two types.

1. **Parameterized Constructor:**

   The parameterized constructor has multiple parameters along with the `self`. When we declare a constructor in such a way that it accepts the arguments during object creation then such type of constructors are known as Parameterized constructors.

   In the following example we can see that with such type of constructors we can pass the values (data) during object creation, which is used by the constructor to initialize the instance members of that object.

```
class DemoClass:

    # parameterized constructor
    def __init__(self, data):
        self.num = data

    # a method
    def read_number(self):
        print(self.num)

# creating object of the class
# this will invoke parameterized constructor
obj = DemoClass(55)

# calling the instance method using the object obj
obj.read_number()

# creating another object of the class
obj2 = DemoClass(66)

# calling the instance method using the object obj
obj2.read_number()
```

Output:

55

66

2. Non-parameterized Constructor (Default constructor):

When we do not include the constructor in the class or forget to declare it, then that becomes the default constructor. It does not perform any task but initializes the objects

In the following example, we do not have a constructor but still we are able to create an object for the class. This is because there is a default constructor implicitly injected by python during program compilation, this is an empty default constructor that looks like this.

```
class DemoClass:


    # a method
    def read_number(self):
        print(101)


# creating object of the class
obj = DemoClass()

# calling the instance method using the object obj
obj.read_number()
```

Output:

101
