Python Programming

- Presentation By Uplatz
- Contact us: https://training.uplatz.com
- Email: info@uplatz.com
- Phone: +44 7836 212635

Uplatz

CLASSES AND OBJECTS

Learning outcomes:

What is an Object?

What is a Class?

Creating a Class

Creating an object

Self in Python

__init__ method

Examples

What is an *object*?

Python is an object oriented programming language. Unlike procedure oriented programming, where the main emphasis is on functions, object oriented programming stresses on objects.

- An object is simply a collection of data (variables) and methods (functions) that act on those data.
- Physical entity
- An Object is an instance of a Class.

What is an *object*?

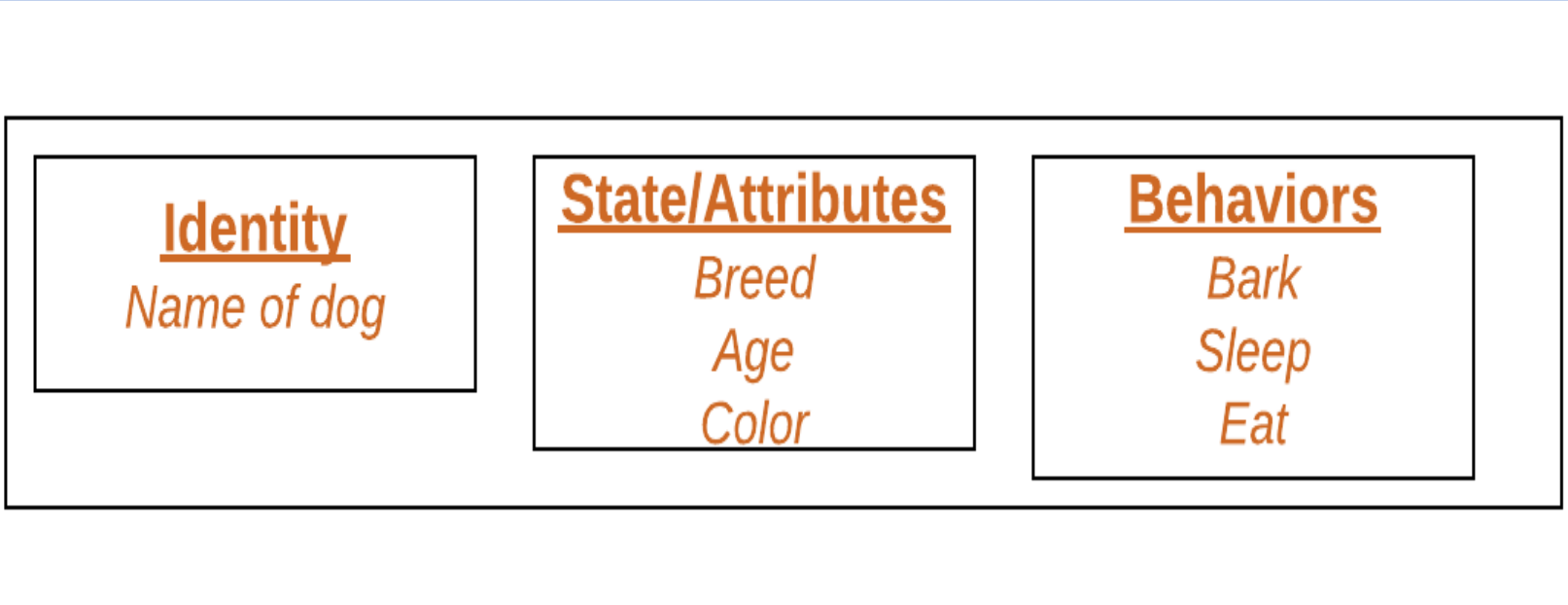
An object consists of :

State : It is represented by attributes of an object. It also reflects the properties of an object.

Behavior : It is represented by methods of an object. It also reflects the response of an object with other objects.

Identity : It gives a unique name to an object and enables one object to interact with other objects.

What is an *object*?



What is a *class*?

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

What is a *class*?

Compared with other programming languages, Python's class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in C++ and Modula-3. Python classes provide all the standard features of Object Oriented Programming

What is a *class*?

- A **blueprint** for the object and also can be defined as a collection of objects.
- A template to create an object
- Created using the keyword **class**
- It supports Camel Case Letters
- Logical Entity that has some specific attributes.

Creating a *class*

The syntax for creating a class :

```
class className:

    class documentation

    class_suite
```

Creating a *class*

Class Members:

Brand

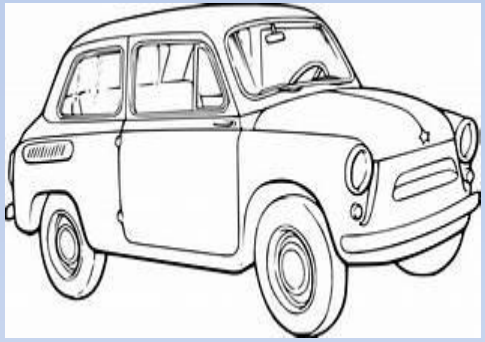
Color

Class Functions:

Apply Break()

Increasing speed()

Decreasing speed()



Class

Creating a *class*

How to write a class?

```
class Car:
        //Class Members
        //Functions
```

Class name should start with Capital letter

How will you access the Functions??

By creating a object

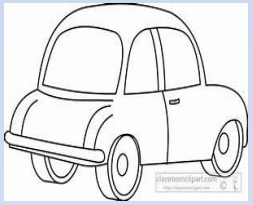
Uplatz

Creating an *object*

- The syntax for creating a object :

  ob=MyClass

OBJ 1



OBJ 2

OBJ 3

Uplatz

Example

Python program to demonstrate instantiating a class :

```
class Animals:
    # A simple class attribute
    attr1 = "mamal"
    attr2 = "dog"

    # A sample method
    def fun(self):
        print("This is a", self.attr1)
        print("This is a", self.attr2)

# Driver code Object instantiation
obj = Animals()
# Accessing class attributes and method through objects
print(obj.attr1)
obj.fun()
```

Self in Python

The self is used to represent the instance of the class. With this keyword, you can access the attributes and methods of the class in python. It binds the attributes with the given arguments. Python decided to do methods in a way that makes the instance to which the method belongs be passed automatically, but not received automatically: the first parameter of methods is the instance the method is called on.

Self in Python

Class methods must have an extra first parameter in method definition. We do not give a value for this parameter when we call the method, Python provides it.
If we have a method which takes no arguments, then we still have to have one argument.
When we call a method of this object as **myobject.method(arg1, arg2)**, this is automatically converted by Python into **MyClass.method(myobject, arg1, arg2)** – this is all the special self is about.

__init__ method

"**__init__**" is a reseved method in python classes. It is called as a constructor in object oriented terminology. This method is called when an object is created from a class and it allows the class to initialize the attributes of the class.

Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

Example

```
class Employee:
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount = Employee.empCount + 1
   def displayCount(self):
      print ("Total Employee %d" % Employee.empCount)
   def displayEmployee(self):
      print ("Name : ", self.name, ", Salary: ", self.salary)
#Creating Instance Objects
emp1 = Employee("Zara", 2000)
emp2 = Employee("Manni", 5000)
#Accessing Attributes
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee = %d" % Employee.empCount)
```

Example

The variable empCount is a class variable whose value is shared among all instances of a this class. This can be accessed as Employee.empCount from inside the class or outside the class.

The first method __init__() is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.

You declare other class methods like normal functions with the exception that the first argument to each method is self. Python adds the self argument to the list for you; you do not need to include it when you call the methods.

Example

Creating Instance Objects :

To create instances of a class, you call the class using class name and pass in whatever arguments its __*init*__ method accepts.

"This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

"This would create second object of Employee class"

emp2 = Employee("Manni", 5000)

Example

Accessing Attributes

You access the object's attributes using the dot operator with object. Class variable would be accessed using class name as follows:

```
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" %
Employee.empCount)
```