Python Classes and Objects

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.

- 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 their class) for modifying their state.

Syntax: Class Definition

```
class ClassName:
    # Statement
```

Syntax: Object Definition

```
obj = ClassName()
print(obj.atrr)
```

The class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Eg.: My class.Myattribute

Creating a Python Class

Here, the class keyword indicates that you are creating a class followed by the name of the class (Dog in this case).

Python3

```
class Dog:
    sound = "bark"
```

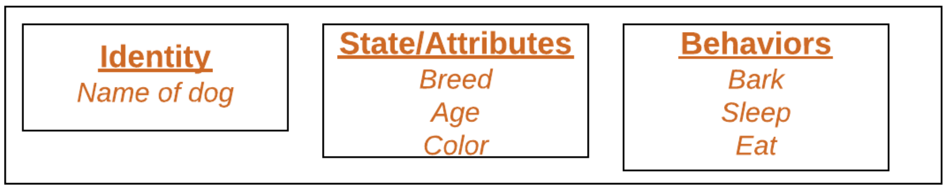
Object of Python Class

An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with *actual values*. It's not an idea anymore, it's an actual dog, like a dog of breed pug who's seven years old. You can have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information is required.

An object consists of:

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

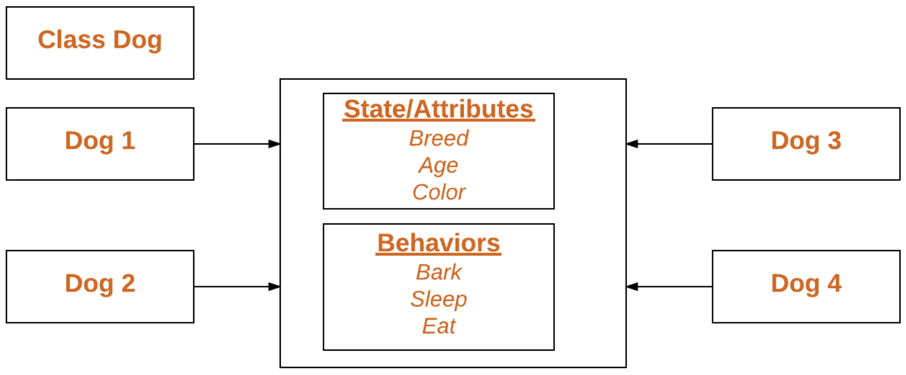
- **Behavior:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.



Declaring Class Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

Example:



Example of Python Class and object

Creating an object in Python involves instantiating a class to create a new instance of that class. This process is also referred to as object instantiation.

Python3

```
# Python3 program to
# demonstrate instantiating
# a class
class Dog:

    # A simple class
    # attribute
    attr1 = "mammal" // Class member (out side the function or inside the class)
    attr2 = "dog"

    # A sample method
    def fun(self):
        print("I'm a", self.attr1)
        print("I'm a", self.attr2)


# Driver code
# Object instantiation
```

```
Rodger = Dog()

# Accessing class attributes
# and method through objects
print(Rodger.attr1)
Rodger.fun()
```

Output:

```
mammal
I'm a mammal
I'm a dog
```

The self Parameter

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named `self` , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words *mysillyobject* and *abc* instead of *self*:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name  // instance variable (inside the function)
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

OUTPUT

```
Hello my name is John
```

The __init__() Function (like constructors)

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

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

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

OUTPUT

```
John
36
```

The __str__() Function

The __str__() function controls what should be returned when the class object is represented as a string.

If the __str__() function is not set, the string representation of the object is returned:

Example

The string representation of an object WITHOUT the __str__() function:

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

p1 = Person("John", 36)

print(p1)
```

output

```
<__main__.Person object at 0x15039e602100>
```

Example

The string representation of an object WITH the __str__() function:

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

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)
```

Output

```
John(36)
```