What is a Python Class ?

A class is a user-defined blueprint or prototype from which objects are created.

Why do we use Classes?

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.

To understand the need for creating a class, let's consider an example. Let's say you wanted to track the number of dogs which may have different attributes like breed and age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it demonstrates the need for classes.

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

How do we create Classes?

Here are simple rules to create a class in Python:

- Classes are created by keyword **class**.
- Attributes are the variables that belong to class.
- Attributes are always public and can be accessed using dot (.) operator. Eg.: Myclass.Myattribute
- Attributes can be made not directly visible by adding a double underscore prefix to their name. Eg.: Myclass.__Hiddenattribute

Here is the syntax of a class in Python:

```
class ClassName:
    # Statement-1
    .
    .
    .
    # Statement-N
```

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

```
class Dog:

    # Class Variable
    animal = 'dog'

    # The init method or constructor
    def __init__(self, breed):

        # Instance Variable
        self.breed = breed

    # Adds an instance variable
    def setColor(self, color):
        self.color = color

    # Retrieves instance variable
    def getColor(self):
        return self.color

# Driver Code
Rodger = Dog("pug")
Rodger.setColor("brown")
print(Rodger.getColor())
```

The **init** method is a constructor. Constructors are used to initialize the object's state. Like methods, a constructor also contains a collection of statements (i.e. instructions) that are executed at 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.

Often, the first argument of a method is called self. This is nothing more than a convention: the name self has absolutely no special meaning to Python. The **self** represents the instance of the class. By using the **self** keyword we can access the attributes and methods of the class in python.

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 what the special **self** is about.

What is a Python Object?

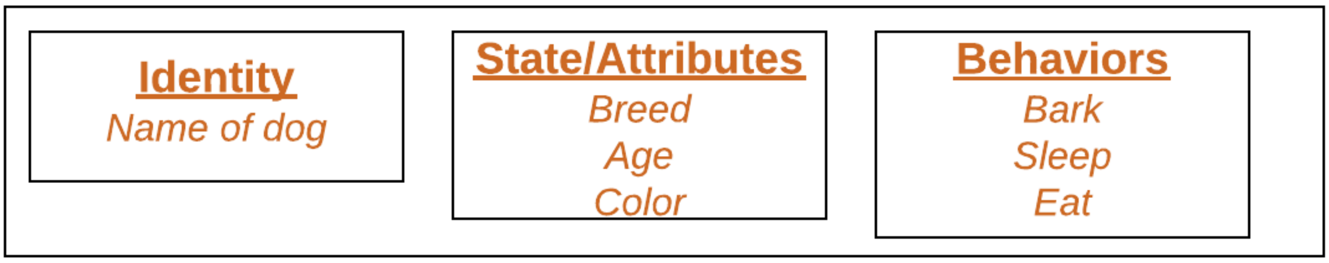
An Object is an instance of a Class.

Why do we use an Object?

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 that has the breed of pug and is 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 :

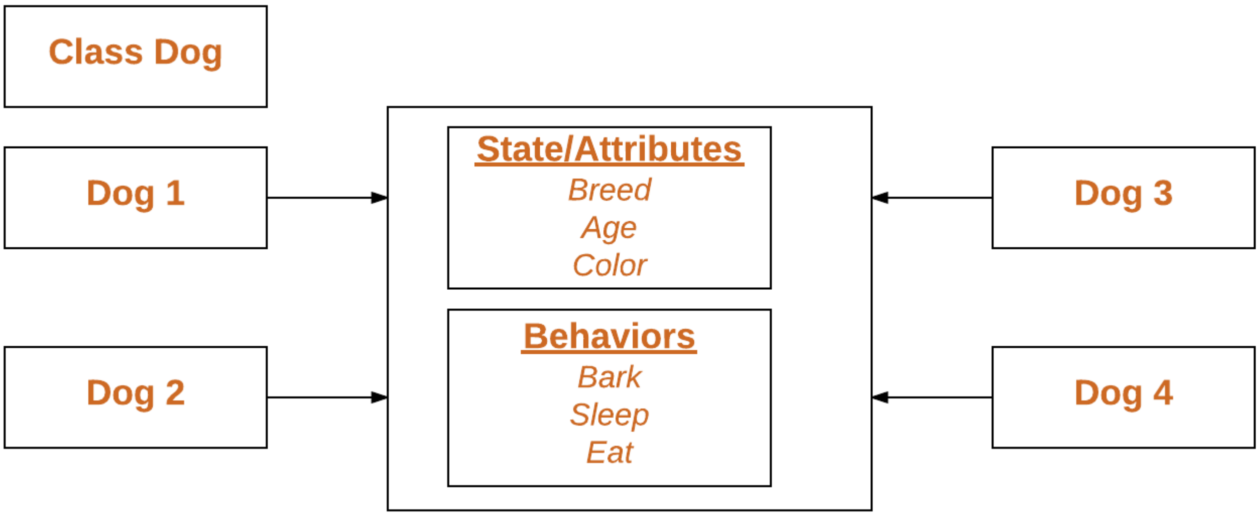
- State : It is represented by attributes of an object. It also reflects the properties of an object.
- Behavior : It is represented by methods of an object. It also reflects the response of an object with other objects.
- Identity : It gives a unique name to an object and enables one object to interact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



How do we create 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 states are unique for each object. A single class may have any number of instances.

Instance variables are for data that is unique to each instance. Class variables are for attributes and methods shared by all instances of the class. Instance variables are variables whose value is assigned inside a constructor or method with **self** whereas class variables are variables whose value is assigned in the class.



Here is an example:

```
class Dog:

    # A simple class
    # attribute
    attr1 = "mammal"
    attr2 = "dog"

    # init method or constructor
    def __init__(self, name):
        self.name = name

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

# Driver code
# Object instantiation
first_dog = Dog('Justin')

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

When the above code is executed, it produces the following result.

```
mammal
I'm a mammal
I'm a dog
My name is Justin
```

In the above example, an object (**first_dog**) is created which is basically a dog. This class only has two **class** attributes that tell us that first_dog is a dog and a mammal and one **instance** attribute that tells us that the dog is named Justin.

Start coding or generate with AI.