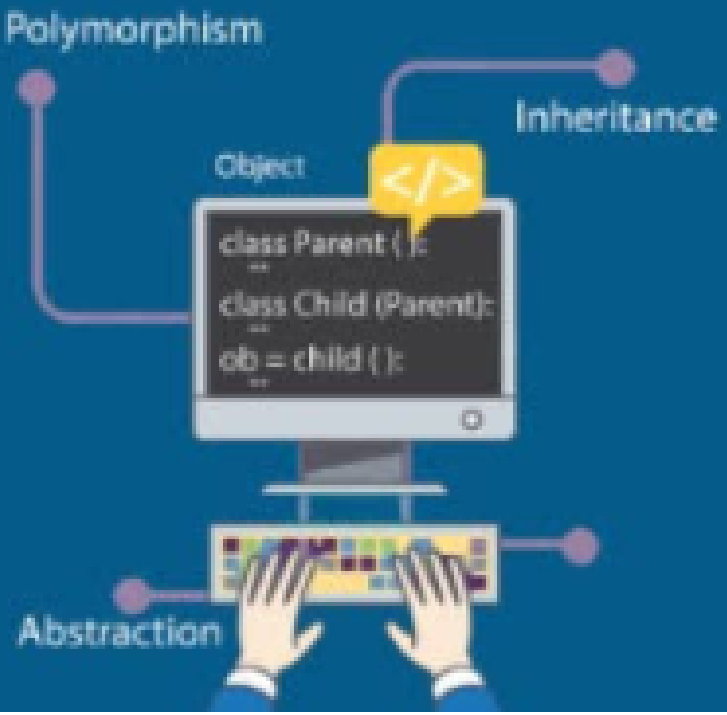
Python

Python Classes and Objects



Python Classes and 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.

To understand the need for creating a class and object in Python let's consider an example, let's say you wanted to track the number of dogs that 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's the exact need for classes.

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:

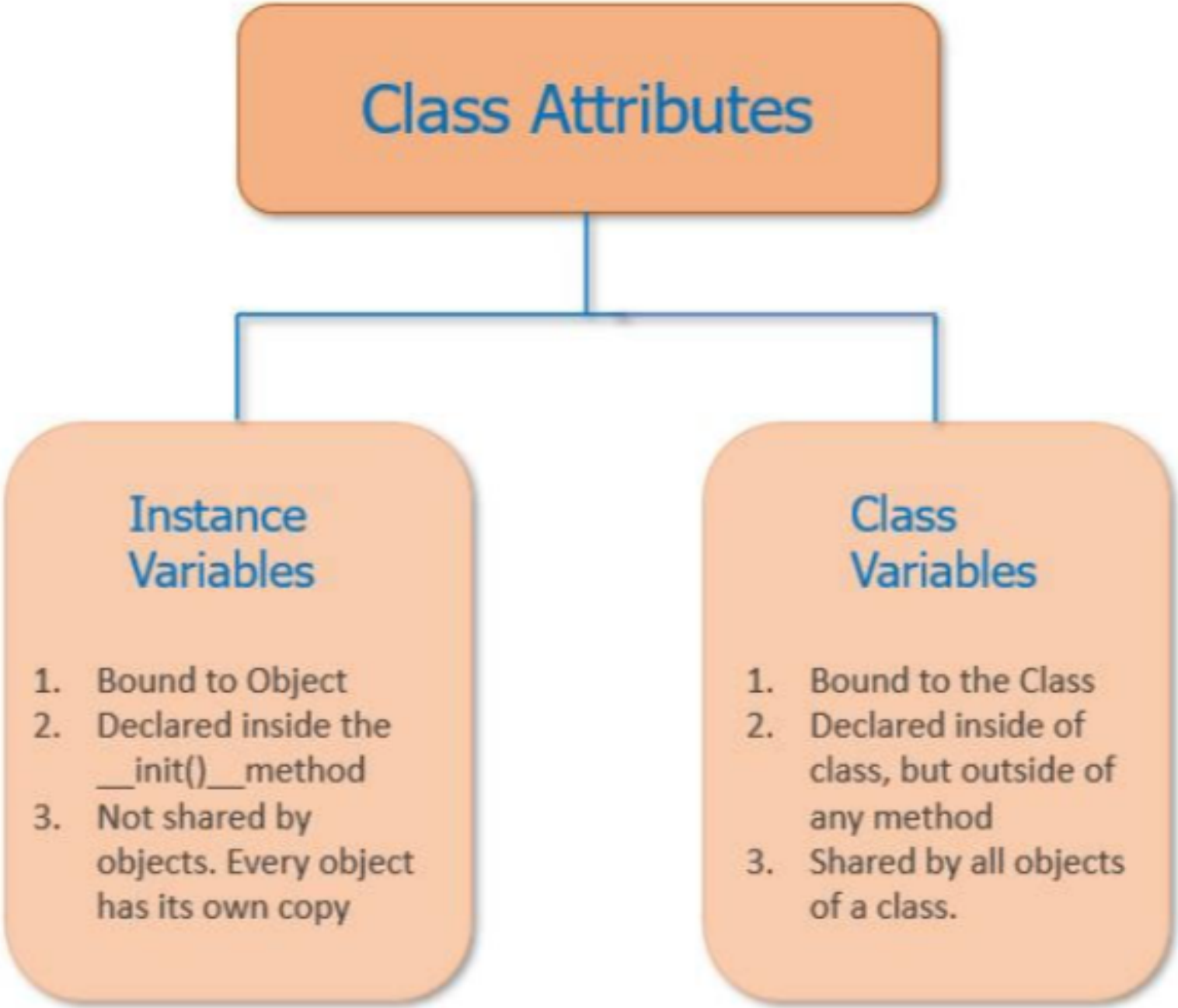
- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Eg.: My class.Myattribute

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"
```

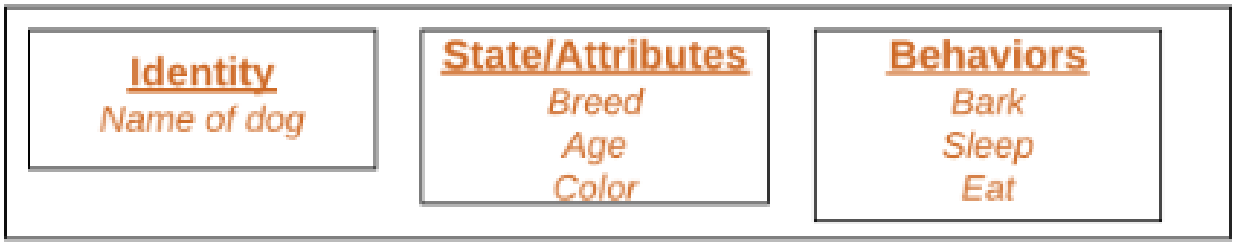

Class Attributes
Instance Variables
1. Bound to Object
2. Declared inside the __init()__method
3. Not shared by objects. Every object has its own copy
Class Variables
1. Bound to the Class
2. Declared inside of class, but outside of any method
3. Shared by all objects of a class.

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:

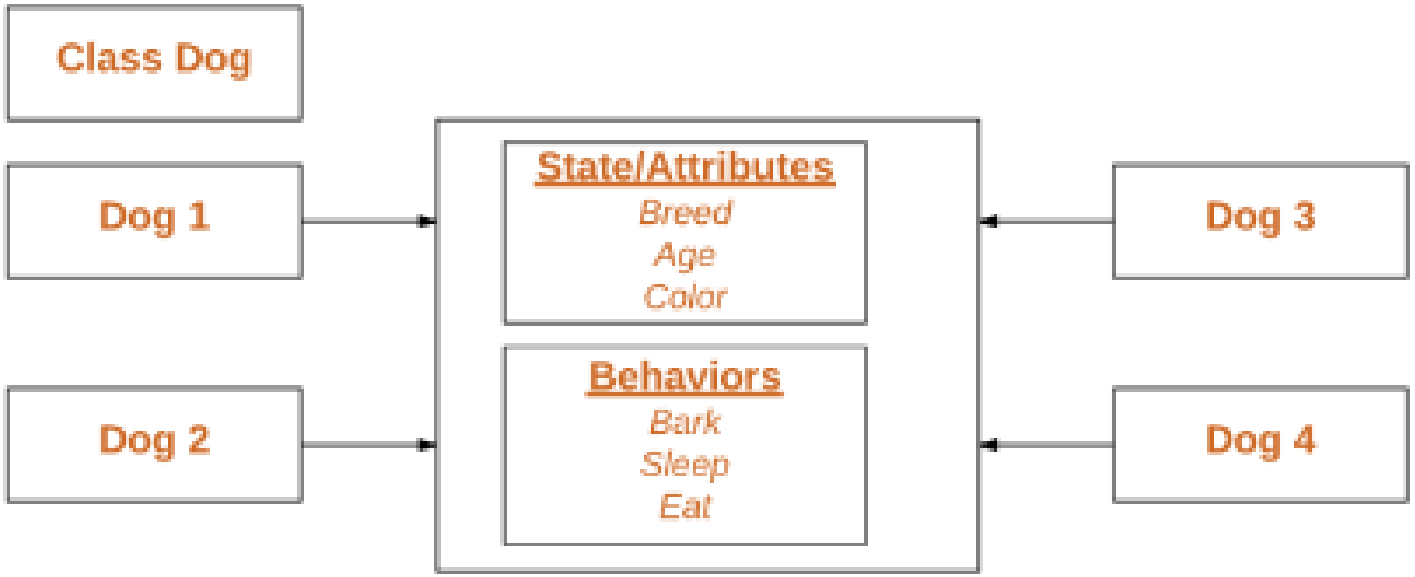
- **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
- **Behavior:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one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



Declaring Claas 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'
    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
```

In the above example, an object is created which is basically a dog named Rodger. This class only has two class attributes that tell us that Rodger is a dog and a mammal.

Explanation :

In this example, we are creating a Dog class and we have created two class variables **attr1 and attr2**. We have created a method named **fun()** which returns the string "I'm a, {attr1}" and I'm a, {attr2}. We have created an object of the Dog class and we are printing at the **attr1** of the object. Finally, we are calling the **fun()** function.

Self Parameter

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.

Python3

```
class GFG:
    def __init__(self, name, company):
        self.name = name
        self.company = company

    def show(self):
        print("Hello my name is " + self.name+" and I" +
              " work in "+self.company+".")

obj = GFG('John', 'GeeksForGeeks')
obj.show()
```

The Self Parameter does not call it to be Self, You can use any other name instead of it. Here we change the self to the word someone and the output will be the same.

Python3

```
class GFG:
    def __init__(somename, name, company):
        somename.name = name
        somename.company = company

    def show(somename):
        print("Hello my name is " + somename.name +
              " and I work in "+somename.company+".")

obj = GFG("John", 'GeeksForGeeks")
obj.show()
```

Output: Output for both of the codes will be the same.

```
Hello my name is John and I work in GeeksForGeeks.
```

Explanation:

In this example, we are creating a GFG class and we have created the **name, and company** instance variables in the constructor. We have created a method named **say_hi()** which returns the string "Hello my name is " + {name} +" and I work in "+{company}+"."."We have created a person class object and we passing the name **John and Company** GeeksForGeeks to the instance variable. Finally, we are calling the **show()** of the class.

Pass Statement

The program's execution is unaffected by the **pass** statement's inaction. It merely permits the program to skip past that section of the code without doing anything. It is frequently employed when the syntactic constraints of Python demand a valid statement but no useful code must be executed.

Python3

```
class MyClass:
    pass
```