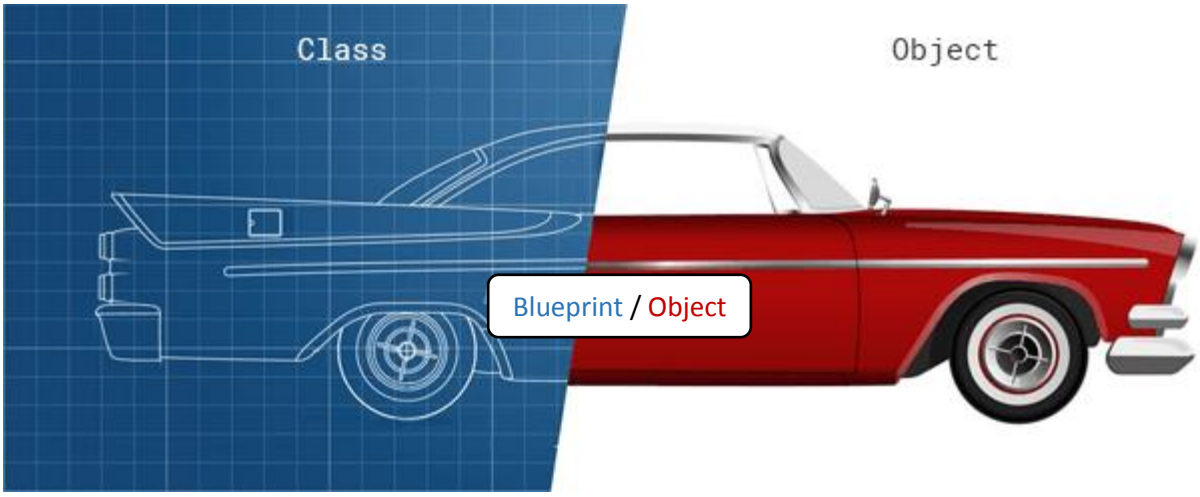
Python Classes and Objects

Classes and objects are the two main aspects of object-oriented programming.

A class is the blueprint from which individual objects are created. In the real world, for example, there may be thousands of cars in existence, all of the same make and model.



Each car was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your car is an instance (object) of the class Car.

Did you know?

In Python, everything is an object – integers, strings, lists, functions, even classes themselves.

However, Python hides the object machinery with the help of special syntax.

For example, when you type `num = 42`, Python actually creates a new object of type integer with the value 42, and assign its reference to the name `num`.

Create a Class

To create your own custom object in Python, you first need to define a class, using the keyword `class`.

Suppose you want to create objects to represent information about cars. Each object will represent a single car. You'll first need to define a class called Car.

Here's the simplest possible class (an empty one):

```
class Car:
  pass
```

Here the `pass` statement is used to indicate that this class is empty.

The __init__() Method

`__init__()` is the special method that initializes an individual object. This method runs automatically each time an object of a class is created.

The `__init__()` method is generally used to perform operations that are necessary before the object is created.

```
class Car:


  # initializer
  def __init__(self):
    pass
```

When you define `__init__()` in a class definition, its first parameter should be `self`.

The self Parameter

The `self` parameter refers to the individual object itself. It is used to fetch or set attributes of the particular instance.

This parameter doesn't have to be called `self`, you can call it whatever you want, but it is standard practice, and you should probably stick with it.

self should always be the first parameter of any method in the class, even if the method does not use it.

Attributes

Every class you write in Python has two basic features: attributes and methods.

Attributes are the individual things that differentiate one object from another. They determine the appearance, state, or other qualities of that object.

In our case, the 'Car' class might have the following attributes:

- Style: Sedan, SUV, Coupe
- Color: Silver, Black, White
- Wheels: Four

Attributes are defined in classes by variables, and each object can have its own values for those variables.

There are two types of attributes: Instance attributes and Class attributes.

Instance Attribute

The instance attribute is a variable that is unique to each object (instance). Every object of that class has its own copy of that variable. Any changes made to the variable don't reflect in other objects of that class.

In the case of our Car() class, each car has a specific color and style.

```
# A class with two instance attributes
class Car:


    # initializer with instance attributes
    def __init__(self, color, style):
        self.color = color
        self.style = style
```

Class Attribute

The class attribute is a variable that is same for all objects. And there's only one copy of that variable that is shared with all objects. Any changes made to that variable will reflect in all other objects.

In the case of our Car() class, each car has 4 wheels.

```
# A class with one class attribute
class Car:


    # class attribute
    wheels = 4


    # initializer with instance attributes
    def __init__(self, color, style):
        self.color = color
        self.style = style
```

So while each car has a unique style and color, every car will have 4 wheels.

Create an Object

You create an object of a class by calling the class name and passing arguments as if it were a function.

```
# Create an object from the 'Car' class by passing style and color
class Car:


    # class attribute
    wheels = 4


    # initializer with instance attributes
    def __init__(self, color, style):
        self.color = color
        self.style = style
```

```
c = Car('Sedan', 'Black')
```

Here, we created a new object from the Car class by passing strings for the style and color parameters. But, we didn't pass in the `self` argument.

This is because, when you create a new object, Python automatically determines what self is (our newly-created object in this case) and passes it to the `__init__` method.

Access and Modify Attributes

The attributes of an instance are accessed and assigned to by using dot `.` notation.

```
# Access and modify attributes of an object
class Car:

    # class attribute
    wheels = 4

    # initializer with instance attributes
    def __init__(self, color, style):
        self.color = color
        self.style = style

c = Car('Black', 'Sedan')

# Access attributes
print(c.style)
# Prints Sedan
print(c.color)
# Prints Black
```

```
# Modify attribute
c.style = 'SUV'
print(c.style)
# Prints SUV
```

Methods

Methods determine what type of functionality a class has, how it handles its data, and its overall behavior. Without methods, a class would simply be a structure.

In our case, the ‘Car’ class might have the following methods:

- Change color
- Start engine
- Stop engine
- Change gear

Just as there are instance and class attributes, there are also instance and class methods.

Instance methods operate on an instance of a class; whereas class methods operate on the class itself.

Instance Methods

Instance methods are nothing but functions defined inside a class that operates on instances of that class.

Now let’s add some methods to the class.

- showDescription() method: to print the current values of all the instance attributes
- changeColor() method: to change the value of ‘color’ attribute

```
class Car:

    # class attribute
    wheels = 4

    # initializer / instance attributes
    def __init__(self, color, style):
        self.color = color
        self.style = style

    # method 1
    def showDescription(self):
        print("This car is a", self.color, self.style)

    # method 2
    def changeColor(self, color):
        self.color = color

c = Car('Black', 'Sedan')

# call method 1
c.showDescription()
# Prints This car is a Black Sedan

# call method 2 and set color
c.changeColor('White')

c.showDescription()
# Prints This car is a White Sedan
```

Class Methods

Example:1

```
# Class Method w/o Parameter
class Mobile:
    @classmethod
    def show_model(cls):          # Class Method
        print("RealMe X")

realme = Mobile()
Mobile.show_model()          # Calling Class Method
```

Example-2

```
# Class Method w/o Parameter
class Mobile:
    fp = 'Yes'                    # Class Variable

    @classmethod                  # decorator
    def show_model(cls):    # Class Method
        # Accessing Class Variable
        print("Fingerprint Scaner:", cls.fp)

realme = Mobile()
Mobile.show_model()                    # Calling Class
Method
```

```
Example - 3
class myclass:
    num = 10

    @classmethod
    def fun(cls):
        print("This is class variable : ",cls.num)

obj = myclass() # creating object of the class
myclass.fun() # accessing class method
```

```
print(myclass.num) # accessing class attributes
myclass.num = 50   # updating class attribute value
outside the class
print(myclass.num)  # printing updated value
```

Example - 4

```
class myclass:
    num = 10
    def __init__(self):
        self.a = 100

    def add(self):
        b = 50  # local variable
        print('This is instance variable :
',self.a, '\n This is local variable : ',b)

    # def sum(self):
    #     print(b)  # Can't access b here because
its a local variable of add()

    @classmethod
    def fun(cls):
        print("This is class variable : ",cls.num)

obj = myclass() # creating object of the class
obj.add()
myclass.fun() # accessing class method
print(myclass.num) # accessing class attributes
myclass.num = 50   # updating class attribute value
outside the class
print(myclass.num)  # printing updated value
```

Example - 5

```
class myclass:
    num = 10
    def __init__(self):
        self.a = 100

    def add(self):
        b = 50  # local variable
        print('This is instance variable :
',self.a, '\n This is local variable : ',b)

    # def sum(self):
    #     print(b)  # Can't access b here because
its a local variable of add()

    @classmethod
    def fun(cls):
        print("This is class variable : ",cls.num)

obj = myclass() # creating object of the class
# ************* Accessing methods ****************
# obj.add() # accessing instance method through
instance
# obj.fun() # accessing class method through
instance
# myclass.fun() # accessing class method through
class
# myclass.add(self=obj) # accessing instance method
through class
# myclass.add(obj) # accessing instance method
through class

# ************* Accessing variables
****************
# print(obj.a)  # accessing instance variable
through instance
# print(obj.b) # can't access because its local
variable
```

```
# print(obj.num) # accessing class variable through
instance
# print(myclass.a) # can't access because its
instance variable
# print(myclass.num) # accessing class variable
through class

# ************* Updating variables ****************
# myclass.num = 50   # updating class attribute
value using class
# print(myclass.num)  # printing updated value
# obj.num = 100      # updating class attribute
value through instance
# print(obj.num)     # printing updated value
```

Static Variable/Methods

```
# Static Method w/o Parameter
class Mobile:
    @staticmethod
    def show_model():          # Static Method
        print("RealMe X")

realme = Mobile()
Mobile.show_model()              # Calling Static
Method
```

Example- 2

```
# Static Method w/o Parameter
class Mobile:
    fp = 'Yes'                  # Class Variable

    @staticmethod
    def show_model():           # Static Method
```

```
        # Accessing Class Variable
        print("Fingerprint Scaner:", Mobile.fp)

realme = Mobile()
Mobile.show_model()                    # Calling Static
Method
```

Example – 3

```
# Static Method with Parameter
class Mobile:

    @staticmethod
    def show_model(m, p):              # Static Method
        model = m
        price = p
        print("Model:", model, "Price", price)

realme = Mobile() # we can access static methout
without creating objects also
Mobile.show_model('RealMe X', 1000)        # Calling
Static Method
```

Delete Attributes and Objects

To delete any object attribute, use the del keyword.

```
del c.color
```

You can delete the object completely with del keyword.

```
del c
```