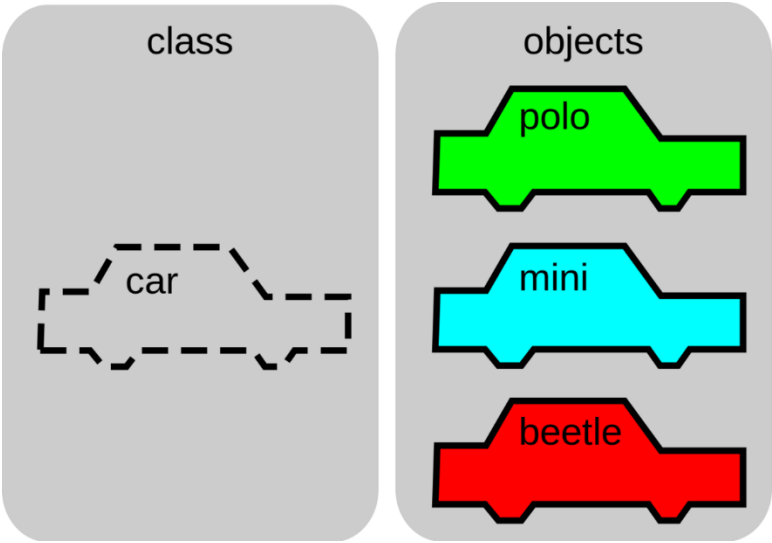
Getting Started with Python Programming with Object Oriented Approached

What is Class?

Class is the building block of object-Oriented programming. Which holds its attributes / properties and methods (Some Action), **For Example**: Consider the Class of Cars. There may be many cars with **different names and brands** but all of them will share some common properties like all of them will have **4 wheels**, **Speed Limit**, **Mileage range**, etc. So here, Car is the class, and wheels, speed limits, and mileage are their properties.



Procedure-Oriented Programming	Object-Oriented Programming
It's like following a recipe with a series of steps. You organize code around functions that perform tasks, like cooking steps.	It's like building with LEGO blocks. You create objects that combine data (the blocks) with actions (how the blocks interact) to model real-world entity.

Class

```
class Classname(object) :
    def __init__(self):
        self.variable_name = value
        self.variable_name = 'value'
    def method_name(self):
        Body of Method
```

Method

Attributes

```
class Classname :
    def __init__(self):
        self.variable_name = value
        self.variable_name = 'value'
    def method_name(self):
        Body of Method
```

- class - class keyword is used to create a class
- object - object represents the base class name from where all classes in Python are derived. This class is also derived from object class. This is optional.
- __init__() – This method is used to initialize the variables. This is a special method. We do not call this method explicitly.
- self – self is a variable which refers to current class instance/object.

How to Create Class

```
class Classname :
    def __init__(self):
        self.variable_name = value
        self.variable_name = 'value'

    def method_name(self):
        Body of Method

    def method_name(self, f1, f2):
        Body of Method
```

Formal Argument

```
class Classname :
    def __init__(self, f1, f2):
        self.variable_name = value
        self.variable_name = 'value'

    def method_name(self):
        Body of Method

    def method_name(self, f1, f2):
        Body of Method
```

Formal Argument

Formal Argument

Object

Object is class type variable or class instance. To use a class, we should create an object to the class.

Instance creation represents allotting memory necessary to store the actual data of the variables.

Each time you create an object of a class a copy of each variables defined in the class is created.

In other words you can say that each object of a class has its own copy of data members defined in the class.

Syntax: -

object_name = class_name()

object_name = class_name(arg)

self Variable

self is a default variable that contains the memory address of the current object.

This variable is used to refer all the instance variable and method.

When we create object of a class, the object name contains the memory location of the object.

This memory location is internally passed to *self*, as *self* knows the memory address of the object so we can access variable and method of object.

self is the first argument to any object method because the first argument is always the object reference. This is automatic, whether you call it *self* or not.

```
def __init__(self):

def show_model(self):
```

Calling Class Method with Argument

Syntax:- Classname.method_name(Actual_argument)

Ex:- Mobile.show_model('4GB')

```
class Mobile:
    fp = 'Yes'
    @classmethod
    def show_model(cls, r):
        cls.ram = r
        print(cls.fp, cls.ram)

realme = Mobile( )
Mobile.show_model(101)
```

Calling Method with argument

Class Method with Parameter

Class Variable

Decorator

Defining Method with parameter

```
class Mobile:
    fp = 'Yes'
    @classmethod
    def show_model(cls, r):
        cls.ram = r
        print(cls.fp, cls.ram)

realme = Mobile( )
```

Class Method without Parameter

```
class Mobile:
    @classmethod
    def show_model(cls):
        print("RealMe X")

realme = Mobile( )
```

Decorator

Class Method

```
class Mobile:
    fp = 'Yes'
    @classmethod
    def show_model(cls):
        print(cls.fp)

realme = Mobile( )
```

Class Variable

Decorator

Class Method

Accessing Class variable Inside Class Method

Class Method without Parameter

```
class Mobile:
    @classmethod
    def show_model(cls):
        print("RealMe X")

realme = Mobile( )
```

Decorator

Class Method

```
class Mobile:
    fp = 'Yes'
    @classmethod
    def show_model(cls):
        print(cls.fp)

realme = Mobile( )
```

Class Variable

Decorator

Class Method

Accessing Class variable Inside Class Method

Calling Static Method with Argument

Syntax:- Classname.method_name(Actual_argument)

Ex:- Mobile.show_model(1000)

```
class Mobile:
    @staticmethod
   def show_model(m, p):
        model = m
        price = p
        print(model, price)
realme = Mobile( )
Mobile.show_model('RealMe X', 1000)
```

Calling Method with argument

Static Method with Parameter

```
class Mobile:
    @staticmethod
   def show_model(m, p):
        model = m
      price   = p
        print(model, price)
realme = Mobile( )
```

Decorator

Defining Method with parameter

Static Method without Parameter

```
class Mobile:
   @staticmethod
   def show_model():
        print("RealMe X")

realme = Mobile( )
```

Decorator

Static Method

```
class Mobile:
    fp = 'Yes'
   @staticmethod
   def show_model():
        print(Mobile.fp)

realme = Mobile( )
```

Static Method

Calling Static Method without Argument

Syntax:- Classname.method_name()

```
class Mobile:
    @staticmethod
    def show_model():
        print("RealMe X")

realme = Mobile( )
Mobile.show_model()
```

Calling Static Method w/o Argument

Nested Class Concept

```
class Army:
    def __init__(self):
        self.name = 'Rahul'
        self.gn = self.Gun()
    def show(self):
        print(self.name)
    class Gun:
        def __init__(self):
            self.name = 'AK47'
            self.capacity = '75 Rounds'
            self.length = '34.3 in'
        def disp(self):
            print(self.name, self.capacity, self.length)
a = Army()
```

Outer Class

Inner Class Object

Inner Class

Outer Class Object