OOPS

What is OOPS?

*OOPS stands for **Object Oriented Programming System**. It is a machanism which is used to write programs using classes and Object's.*

What are Python OOPs Concepts?

Major concepts in Python OOPs include Class, Object, Method, Inheritance, Polymorphism, Data Abstraction, and Encapsulation.

Key Points to Remember:

- *Object-Oriented Programming makes the program easy to understand as well as efficient.*
- *Since the class is sharable, the code can be reused.*
- *Data is safe and secure with data abstraction.*
- *Polymorphism allows the same interface for different objects, so programmers can write efficient code.*

What is OOPL?

OOPL stands for **Object Oriented Programming Language**.

Any programming language which supports Object Oriented Programming Principles is called as Object Oriented Programming Language.

Principles of OOPS

According to Object Oriented Analysis and Design OOAD we have 4 principles.

1. **Abstraction**
2. **Encapsulation**
3. **Inheritance**
4. **Polymorphism**

What is Abstraction?

- ***It is a process of getting necessary Data and hiding unnecessary Data.***

Abstraction in OOP is a process of hiding the real implementation of the methods by only showing a method signature. In Python, using **ABC** (abstraction class) is a class from the **abc** module in Python. When annotate any method with an **@abstractmethod** decorator, then it is an abstract method in Python.

EXAMPLE-1:

```
from abc import abstractmethod, ABC
```

```
class Vehicle(ABC):

    def __init__(self, speed, year):

        self.speed = speed
        self.year = year

    def start(self):
        print("Starting engine")

    def stop(self):
        print("Stopping engine")

    @abstractmethod
    def drive(self):
        pass

class Car(Vehicle):

    def __init__(self, canClimbMountains, speed, year):
        Vehicle.start(self)
        Vehicle.__init__(self, speed, year)
        self.canClimbMountains = canClimbMountains

def drive(self):
print("Car is in drive mode")
print("speed is:",self.speed)

c = 'yes'
s = 150
y = 25
ob = Car(c,s,y)
ob.drive()
```

OUTPUT:

```
Starting engine
Car is in drive mode
speed is: 150
```

EXAMPLE-2:

```
# Python program showing
# abstract base class work

from abc import ABC, abstractmethod

class Animal(ABC):
# in this class,if only one method is available then,
# no need to write a decorator.
    #@abstractmethod
    def move(self):
        pass

class Human(Animal):

    def move(self):
        print("I can walk and run")

class Snake(Animal):

    def move(self):
        print("I can crawl")

class Dog(Animal):

    def move(self):
        print("I can bark")

class Lion(Animal):

    def move(self):
        print("I can roar")

# Driver code
R = Human()
R.move()

K = Snake()
```

```
K.move()

R = Dog()
R.move()

K = Lion()
K.move()
```

OUTPUT:

I can walk and run

I can crawl

I can bark

I can roar

EXAMPLE-3:

```
# Python program invoking a
# method using super()

from abc import ABC, abstractmethod

class R(ABC):

    def rk(self):
        print("Abstract Base Class")

class K(R):

    def rk(self):
        super().rk()
        print("subclass ")
```

```
# Driver code
r = K()
r.rk()
```

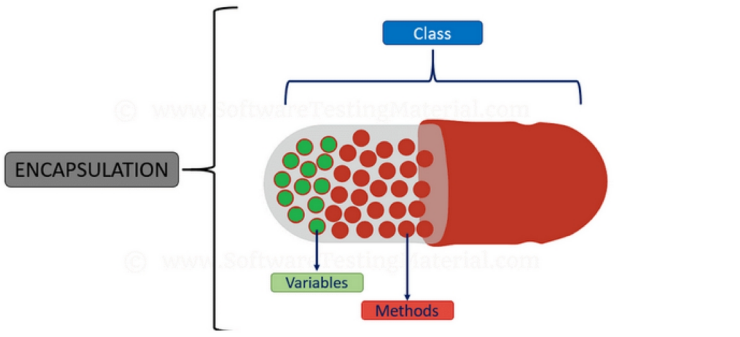
Output:

Abstract Base Class

subclass

What is Encapsulation?

- ***It is a process of binding state(variables) and behavior(methods) in a single container.***



Using OOP in Python, we can restrict access to variables and methods and variables. This prevents data from direct modification which is called encapsulation. In Python, we denote private attributes using underscore as the prefix i.e single_or double__.

Example: Data Encapsulation in Python;

```
class Computer:
def __init__(self):
self.__maxprice = 900

def sell(self):
print("Selling Price: {}".format(self.__maxprice))

def setMaxPrice(self, price):
self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

OUTPUT:
selling price : 900
selling price : 900
selling price : 1000

In the above program, we defined a computer class. We used __init__() method to store the maximum selling price of computer. We tried to modify the price. However, we can't change it because

Python treats the as private attributes. As shown, to change the value, we have to use a setter function i.e setMaxPrice() which takes price as a parameter.

What is Inheritance?

Inheritance is a powerful feature in object oriented programming.

It is a process of getting properties from one class to other class.

The class which is providing properties is called as a **super class** or **parent class** or **base class.**

The class which is taking the properties is called as **sub class** or **child class** or derived class.

Swipe up

Types of Inheritance in Python Programming:

Types of inheritance:

There are five types of inheritance in python programming;

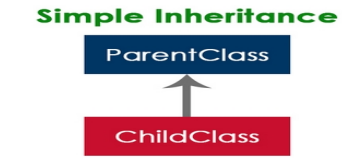
1.Single inheritance

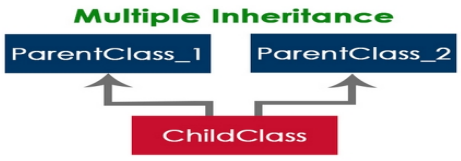
2. Multiple inheritance

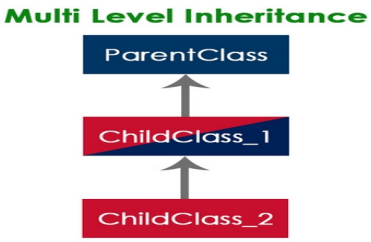
3. Multi-level inheritance

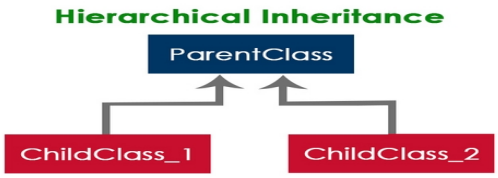
4. Hierarchical inheritance

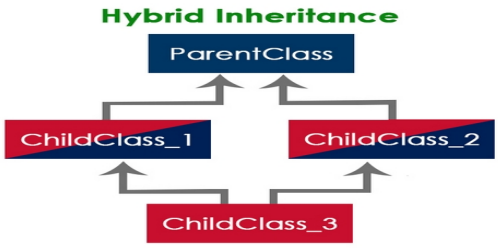
5.Hybrid inheritance



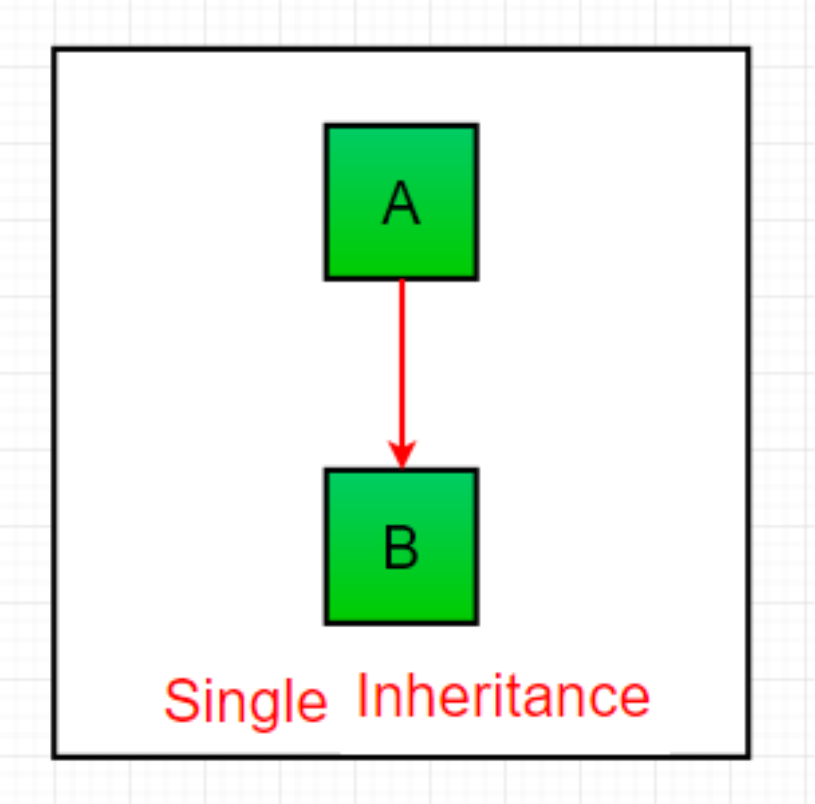








1.Single inheritance: When one child class is derived from only one parent class. This is called single inheritance.



Syntax of single inheritance:

```
#syntax_of_single_inheritance
class class1:              #parent_class
      pass
class class2(class1):      #child_class
      pass
obj_name = class2()
```

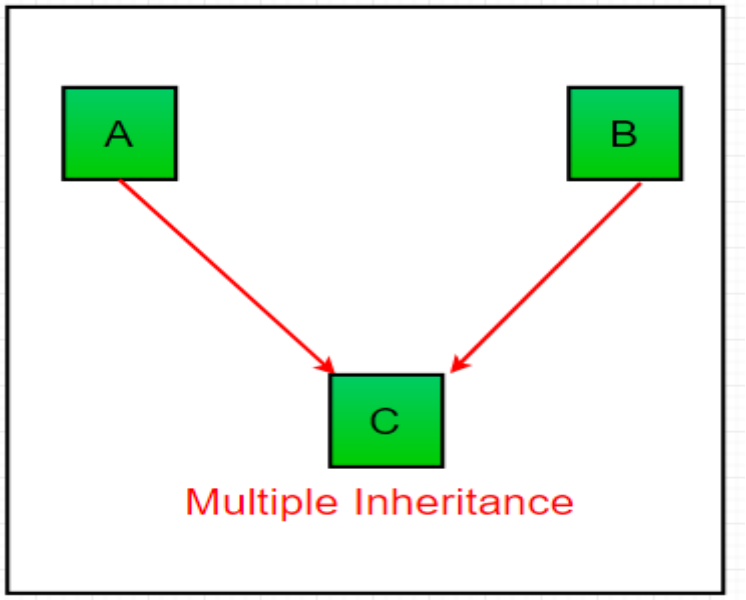
Example:

```
class Brands:               #parent_class
      brand_name_1 = "Amazon"
      brand_name_2 = "Ebay"
      brand_name_3 = "OLX"

class Products(Brands):      #child_class
      prod_1 = "Online Ecommerce Store"
      prod_2 = "Online Store"
      prod_3 = "Online Buy Sell Store"

obj_1 = Products()          #Object_creation
print(obj_1.brand_name_1+" is an "+obj_1.prod_1)
print(obj_1.brand_name_2+" is an "+obj_1.prod_2)
print(obj_1.brand_name_3+" is an "+obj_1.prod_3)
```

Output:

```
#Output
#Amazon is an Online Ecommerce Store
#Ebay is an Online Store
#OLX is an Online Buy Sell Store
```

2. Multiple inheritance: When one child class is derived or inherited from the more than one parent class. This is called multiple inheritance.



Syntax of multiple inheritance:

```
#syntax_of_multiple_inheritance
class parent_1:
    pass
class parent_2:
    pass
class child(parent_1,parent_2):
    pass
```

```
obj = child()
```

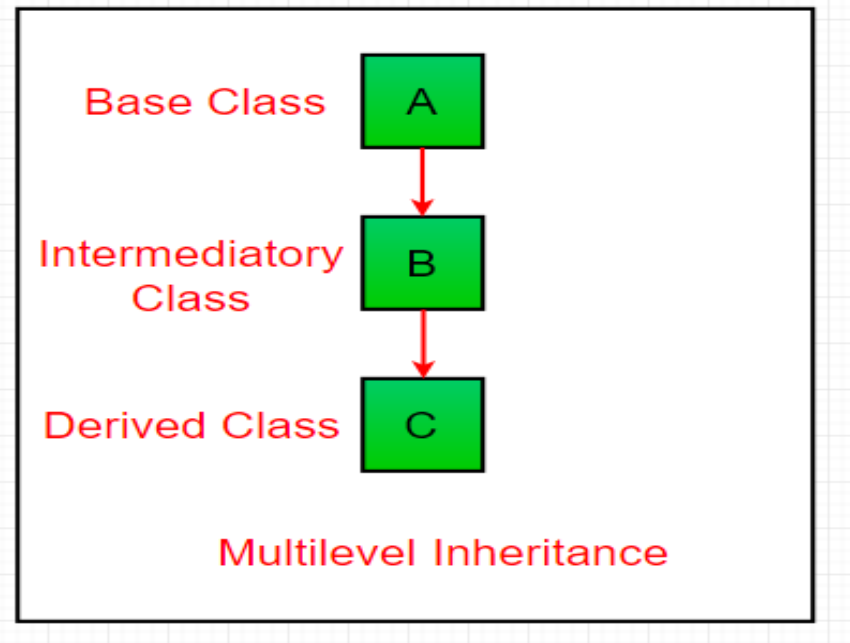
Example:

```
#example_of_multiple_inheritance
class Brands:                #parent_class
      brand_name_1 = "Amazon"
      brand_name_2 = "Ebay"
      brand_name_3 = "OLX"
class Products:              #child_class
      prod_1 = "Online Ecommerce Store"
      prod_2 = "Online Store"
      prod_3 = "Online Buy Sell Store"
class Popularity(Brands,Products):
      prod_1_popularity = 100
      prod_2_popularity = 70
      prod_3_popularity = 60

obj_1 = Popularity()         #Object_creation
print(obj_1.brand_name_1+" is an "+obj_1.prod_1)
print(obj_1.brand_name_2+" is an "+obj_1.prod_2)
print(obj_1.brand_name_3+" is an "+obj_1.prod_3)
```

Output

```
#Output
#Amazon is an Online Ecommerce Store popularity of 100
#Ebay is an Online Store popularity of 70
#OLX is an Online Buy Sell Store popularity of 60
```

3. Multi-level inheritance: In multilevel inheritance; one parent class and one child class, that child class is derived or inherited from that parent class. And that child class derives or inherits a new-child class (grand child class) is called as multilevel inheritance.



Syntax of multilevel inheritance:

```
#Syntax_of_multilevel_inheritance
class A:
```

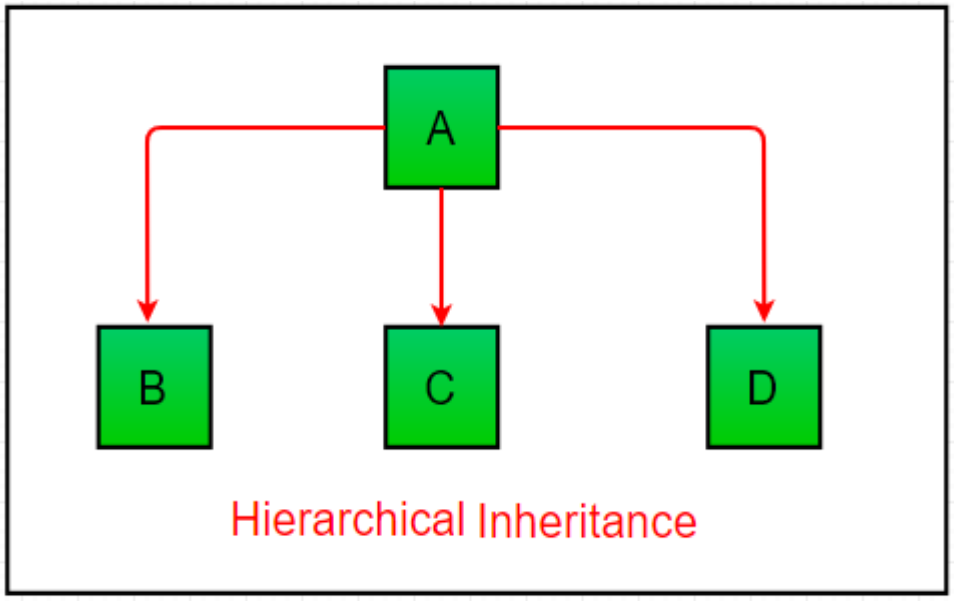
```
    pass

class B(A):
    pass

class C(B):
    pass

obj = C()
```

Example:

```
#example_of_multilevel_inheritance
class Brands:                   #parent_class
    brand_name_1 = "Amazon"
    brand_name_2 = "Ebay"
    brand_name_3 = "OLX"
class Products(Brands):         #child_class
    prod_1 = "Online Ecommerce Store"
    prod_2 = "Online Store"
    prod_3 = "Online Buy Sell Store"


class Popularity(Products):     #grand_child_class
    prod_1_popularity = 100
    prod_2_popularity = 70
    prod_3_popularity = 60



obj_1 = Popularity()        #Object_creation
```

```
print(obj_1.brand_name_1+" is an "+obj_1.prod_1+" popularity of "+str(obj_1.prod_1_popularity))
print(obj_1.brand_name_2+" is an "+obj_1.prod_2+" popularity of "+str(obj_1.prod_2_popularity))
print(obj_1.brand_name_3+" is an "+obj_1.prod_3+" popularity of "+str(obj_1.prod_3_popularity))


#Output
#Amazon is an Online Ecommerce Store popularity of 100
#Ebay is an Online Store popularity of 70
#OLX is an Online Buy Sell Store popularity of 60
```

4. Hierarchical inheritance: One Parent class derives or inherits more than one child class; maybe two or three or more child classes. Then this type of inheritance is called hierarchical inheritance.



Syntax of hierarchical inheritance:

```
#syntax_of_hierarchical_inheritance


class A:                #parent_class
    pass

class B(A):             #child_class
    pass


class C(A):             #child_class
    pass


class D(A):             #child_class
    pass


obj_1 = B()     #Object_creation
obj_2 = C()
obj_3 = D()
```

Example:

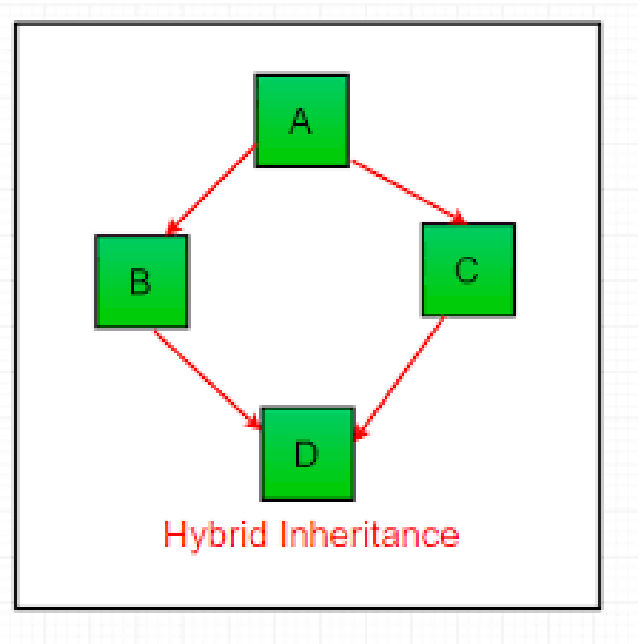
```
#example


class Brands:                   #parent_class
    brand_name_1 = "Amazon"
    brand_name_2 = "Ebay"
    brand_name_3 = "OLX"
```

```
class Products(Brands):          #child_class
      prod_1 = "Online Ecommerce Store"
      prod_2 = "Online Store"
      prod_3 = "Online Buy Sell Store"


class Popularity(Brands):       #grand_child_class
      prod_1_popularity = 100
      prod_2_popularity = 70
      prod_3_popularity = 60


class Value(Brands):
      prod_1_value = "Excellent Value"
      prod_2_value = "Better Value"
      prod_3_value = "Good Value"

obj_1 = Products()         #Object_creation
obj_2 = Popularity()
obj_3 = Value()
print(obj_1.brand_name_1+" is an "+obj_1.prod_1)
print(obj_1.brand_name_1+" is an "+obj_1.prod_1)
print(obj_1.brand_name_1+" is an "+obj_1.prod_1)
```

Output:

```
#Output
#Amazon is an Online Ecommerce Store
#Ebay is an Online Store
#OLX is an Online Buy Sell Store
```

5.Hybrid_inheritance: Inheritance which involves multiple types of inheritance in a single program is called hybrid inheritance.



Syntax of hybrid inheritance:

```
#Syntax_Hybrid_inheritance
class PC:
        pass
class Laptop(PC):
        pass
class Mouse(Laptop):
        pass
class Student3(Mouse, Laptop):
        pass
# Driver's code
obj = Student3()
```

Note: There is no sequence in Hybrid inheritance that which class will inherit which particular class. You can use it according to your requirements.

Example:

```
#Example_Hybrid_inheritance
class PC:
def fun1(self):
print("This is PC class")

class Laptop(PC):
def fun2(self):
print("This is Laptop class inheriting PC class")

class Mouse(Laptop):
def fun3(self):
print("This is Mouse class inheriting Laptop class")

class Student(Mouse, Laptop):
def fun4(self):
print("This is Student class inheriting PC and Laptop")
# Driver's code
obj = Student()
obj1 = Mouse()
obj.fun4()
obj.fun3()
```

What is polymorphism?

Poly means many and Morphs means form or types
Polymorphism means many forms or many types.

Polymorphism is an ability (in OOP) to use a common interface for **multiple forms** (or **data types**).

Suppose, we need to color a shape, there are multiple shape options (rectangle, square, circle). However we could use the same method to color any shape. This concept is called Polymorphism.

To use the above principles in python we use class and object.

Example: Using Polymorphism in Python;

```
class Parrot:
def fly(self):
print("Parrot can fly")
def swim(self):
print("Parrot can't swim")

class Penguin:

def fly(self):
print("Penguin can't fly")
def swim(self):
print("Penguin can swim")

# common interface
def flying_test(bird):
bird.fly()

#instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
```

```
flying_test(blu)
flying_test(peggy)
```

```
OUTPUT:

parrot can fly

penguin can’t fly
```

In the above program, we defined two classes *parrot* and *penguin*. Each of them have a common *fly()* method. However, their functions are different.

To use polymorphism, we created a common interface i.e *flying_test()* function that takes any object and calls the object's *fly()* method. Thus, when we passed the *blu* and *peggy* objects in the *flying_test* function, it ran effectively.

WE HAVE TWO TYPES OF POLYMORPHISM;

1.Method Overloading
2.Method Overriding

Method Overloading;

We can overload a Static Method / Instance Method / Constructor.

What is Overloading?

Overloading is a process of defining multiple methods with **same name** but with **different parameters**.It is used in single class.

Note : In python overloading is done automatically using **default parameters** .

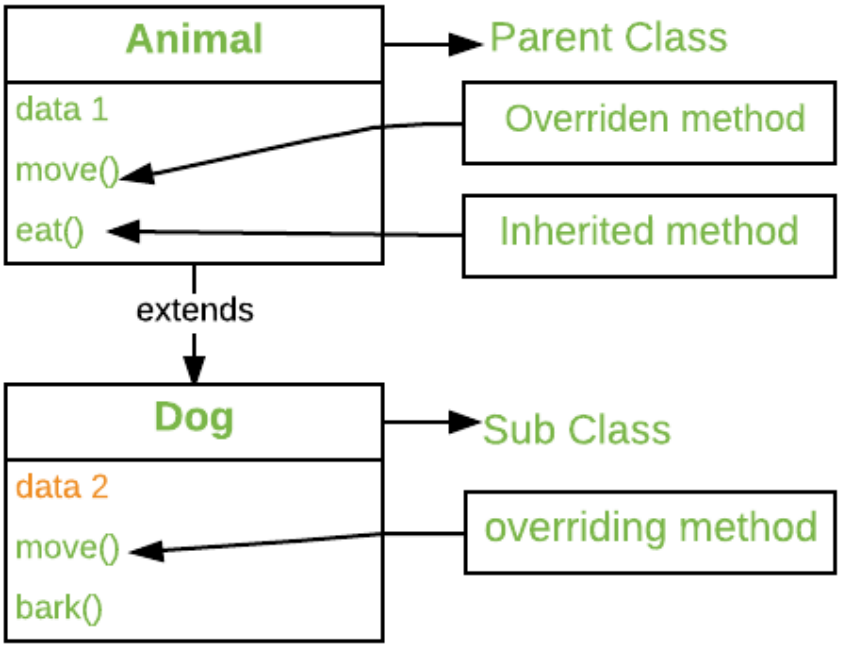
EXAMPLE:

```
class Person:

    def Hello(self, name=None):

        if name is not None:
            print('Hello ' + name)
        else:
            print('Hello ')

# Create instance
obj = Person()

# Call the method
obj.Hello()

# Call the method with a parameter
obj.Hello('MOHANXQUOTES')
```

OUTPUT:
Hello
Hello MOHANXQUOTES

Method Overriding;

Overriding is the property of a class to change the implementation of a method provided by one of its parent classes.



In Python method; overriding occurs by simply defining in the ***child class*** a method with the ***same name*** of a method in the ***parent class.***

Method Overriding is the method having the same name and with the same arguments.

It is implementated with inheritance also.

It mostly used for memory reducing processes.

EXAMPLE-1:

```
# Python program to demonstrate
# calling the parent's class method
# inside the overridden method

class Parent():
def show(self):
print("Inside Parent")
class Child(Parent):
def show(self):
# Calling the parent's class
# method
Parent.show(self)
print("Inside Child")
# Driver's code
obj = Child()
obj.show()
```

OUTPUT:
Inside Parent
Inside Child

EXAMPLE-2:

```
# Python program to demonstrate
# overriding in multilevel inheritance


class Parent():

    # Parent's show method
    def display(self):
```

```
    print("Inside Parent")


# Inherited or Sub class (Note Parent in bracket)
class Child(Parent):

    # Child's show method
   def show(self):
   print("Inside Child")


# Inherited or Sub class (Note Child in bracket)
class GrandChild(Child):

   # Child's show method
   def show(self):
   print("Inside GrandChild")



# Driver code
g = GrandChild()
g.show()
g.display()
```

```
OUTPUT:
Inside GrandChild
Inside Parent
```

What is a class?

Class is a collection of **variables and methods.**

Syntax:

class(keyword) **class_name**:

    variables

    variables

        functions

        functions

Static variables.

The variables which are declared inside the class and outside the method are called as **static variables.**

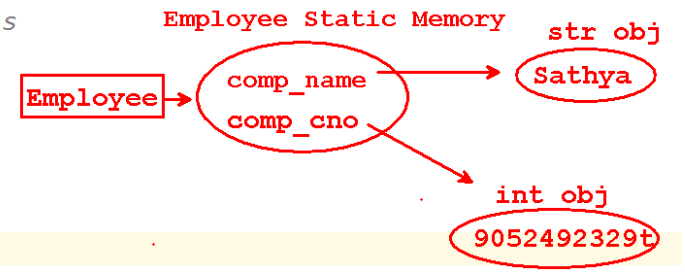
To call static variables we use class name.

The scope of static variables is anywhere but we need to use class name.

The static variables will get memory at class loading time.

The static variable will get Memory only 1 time

```
1
2   # Example on Static Variables
3
4   class Employee:
5       comp_name = "Sathya"
6       comp_cno = 9052492329
7
8
9
10  print(Employee.comp_name)
11  print(Employee.comp_cno)
```



Static Methods

To declare a static method we use **"@staticmethod"** decorator.

Static method is used to Perform Operations on static variables.

To call static methods we use class name.

The scope of static method is anywhere but we need to use class name.

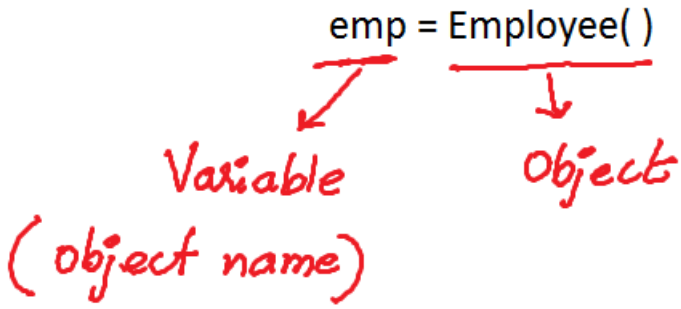
```
class Employee:
    comp_name = "Sathya"
    comp_cno = "123456798"

    @staticmethod
    def display_company_details():
        print(Employee.comp_name)
        print(Employee.comp_cno)



Employee.display_company_details()
```

Note : To Work with **instance variables** or **instance methods** we need an **object**.

What is Object ?

Object is an instance of a class.

Example:



What is an Instance ?

Instance means allocating required memory for instance variables.

Syntax

variable = Class_name( )

Instance Variables

The variables which are declared inside the method using **"self"** word are called as **instance variables.**

To call instance variables we use Object or Object name(Object reference variable).

The scope of instance variables is anywhere but we need to use Object or Object name(Object reference variable).

The instance variables will get memory after creating an object.

*The instance variable will get Memory for every object creation.

(it means if we create 5 objects the instance variables will get memory for 5 times)

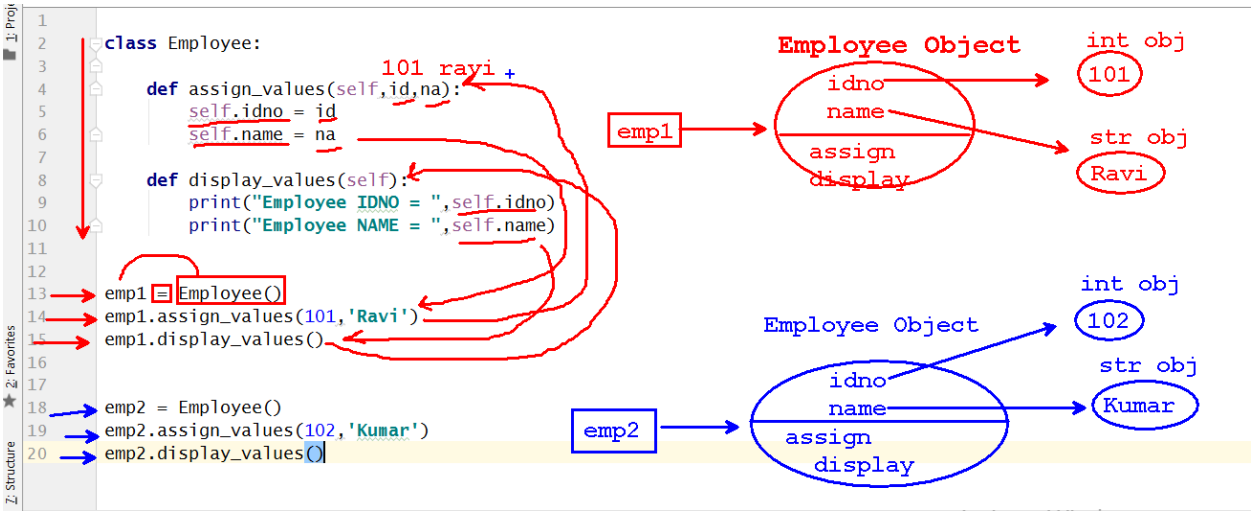
Instance Methods

If a method takes **"self"** as a default parameter we can call it as **instance method.**

Instance method is used to Perform Operations on Instance variables.

To call Instance methods we use Object or Object name(Object reference variable).

The scope of instance methods is anywhere but we need to use Object or Object name(Object reference variable).



Constructor

Constructor is a special kind of method used to **Initialize instance variables of a class.**

The constructor name must be **"__init__( )".**

Constructor will take **"self"** as a default parameter.

Note : These kind of methods also called as **Dunder methods**. Dunder means Double underscore.

No need to call constructor manually because constructor is called automatically at the time of object creation.

Difference between instance method and constructor.

Instance Method	Constructor
Instance method is used to perform operations on instance variables	Constructor is used to initialize instance variables
Instance method need to be called manually using object or object ref variable	Constructor is called automatically at the time of object creation.
Instance method name can be any thing	Constructor name must be __init__ only.
One class can have any number of instance methods.	Once class can have only 1 constructor.