Types of oops

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

# 1.Inheritance:

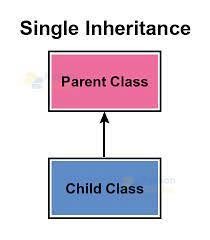
There are 4 types of inheritance.

1. Single inheritance
2. Multi - level inheritance
3. Multiple inheritance
4. Hierarchical inheritance

# Single inheritance:

Single inheritance enables a derived class to inherit properties from a single parent class.

Thus, enabling code reusability and the addition of new features to existing code.



# Example:

class parent ():

def display(self):

print ("this is parent class")

class child(parent):

def child1(self):

print ("this is child class")

object=child ()

object. display ()

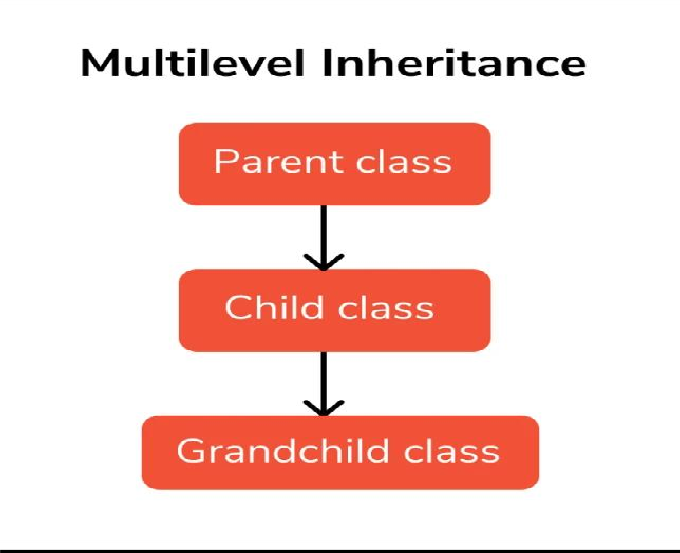
object. child1 ()

# Output:

this is parent class

this is child class

# Multi - level inheritance:



Multilevel Inheritance in Python is a type of Inheritance that involves inheriting a class that has already inherited some other class.

That means the derived/subclass class inherits the features of the base class/parent class, and the new derived class inherits

the features of the derived class.

# Example:

class Grandfather():

def vanaja(self):

print ("this is grandfather class")

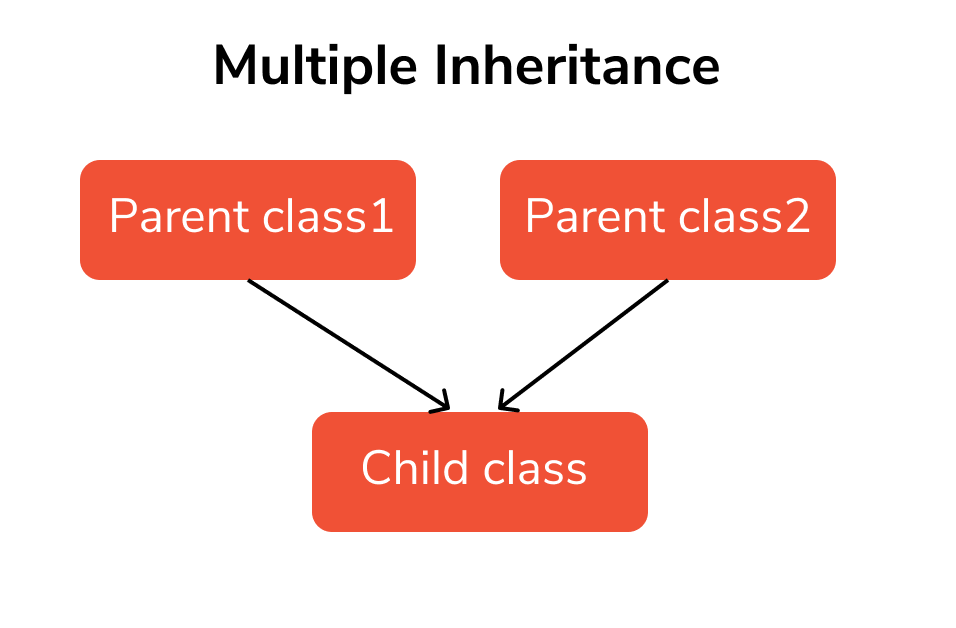
class father (Grandfather):

def divya (self):

print ("this is father class")

class child(father):

def sandhya(self):



print (“this is child class”)

Ramya= child ()

Ramya. Vanaja ()

Ramya. divya ()

Ramya. sandhya ()

# Output:

this is Grandfather class

this is father class

this is child class

# Multiple inheritance:

If a child class inherits from more than one class, i.e. this child class is derived from multiple classes,

we call it multiple inheritance in Python.

# Example:

class Father ():

def display(self):

print ("this is father class")

class Mother ():

def display1(self):

print ("this is mother class")

class child (Father, Mother):

def display2(self):

print ("this is child class")

obj =child ()

obj. display ()

obj. display1 ()

obj. display2 ()

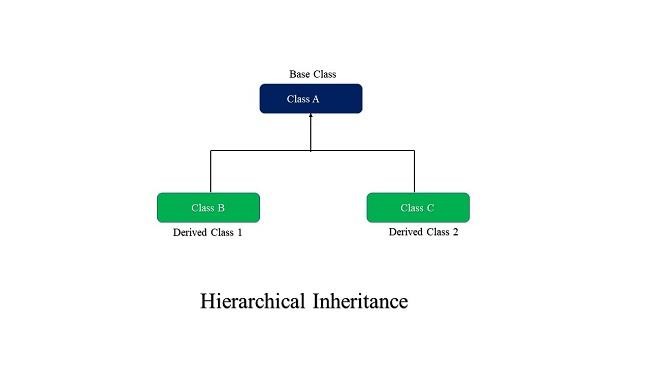
# Output:

this is father class

this is mother class

this is child class

# Hierarchical inheritance:



If multiple derived classes are created from the same base, this kind of Inheritance is known as hierarchical inheritance. In this instance, we have two base classes as a parent (base) class as well as two children (derived) classes.

# Example:

class Father ():

def vanaja(self):

print ("this is father class")

class child1(Father):

def ram(self):

print ("this is child1 class")

class child2(Father):

def syam (self):

print ("this is child2 class")

raju=child1()

raju. vanaja ()

raju. ram ()

remo =child2 ()

remo. vanaja ()

remo. syam ()

# Output:

this is father class

this is child1 class

this is father class

this is child2 class

# 2.Polymorphism:

Polymorphism is another important concept of object-oriented programming. It simply means more than one form.

That is, the same entity (method or operator or object) can perform different operations in different scenarios.

# Example:

1. method overloading:

class Methodoverload():

def display (self, a=None, b=None):

print (a, b)

object=Methodoverload ()

object. display (1,2)

object. display (1)

object. display ()

1. method overriding:

class Methodoverride():

def result(self):

print ("this is method overriding")

class vanaja (Methodoverride):

def result(self):

print ("this is vanaja class")

super (). result ()

object= vanaja ()

object. result ()

# Output:

1 2

1 None None None

this is vanaja class

this is method overriding

**3.Encapsulation:**

Encapsulation is one of the key features of object-oriented programming. Encapsulation refers to the bundling of attributes and methods inside a single class.

It prevents outer classes from accessing and changing attributes and methods of a class. This also helps to achieve **data hiding**

.

In Python, we denote private attributes using underscore as the

prefix i.e single or double . For example,

\_

class Computer:

**Example:**

class Grandfather(): def init (self, a):

self.\_y=a print(self.\_y)

class Father (Grandfather): def display1(self):

print ("Father", self.\_y)

class child (Father):

def display2(self):

print ("child", self.\_y)

obj=child (5)

obj. display1()

obj. display2()

# Output:

5

Father 5

child 5

# 4. Abstraction:

classes in Python Abstract class can be inherited by the subclass and abstract method gets its definition in the subclass.

Abstraction classes are meant to be the blueprint of the other class. An abstract class can be useful when we are designing large functions.

# Example:

from abc import ABC, abstractmethod

class Car (ABC):

@abstractmethod

def colour (self):

pass

class Tesla (Car):

def colour (self):

print ("the colour is blue")

class Suziki (Car):

def colour (self):

print ("the colour is red")

class ford (Car):

def colour(self):

print ("the colour is black")

t=Tesla ()

t. colour ()

s=suziki ()

s. colour ()

f=ford ()

f. colour ()

# Output:

the colour is blue

the colour is red

the colour is black