INHERITANCE

Inheritance is the capability of one class to derive or inherit the properties from another class.

Parent class is the class being inherited from, also called **base class**.

Child class is the class that inherits from another class, also called **derived class**.

The benefits of inheritance are:
1. It represents real-world relationships well.
2. It provides **reusability** of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
3. It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Syntax:

```
class parent_class:
        ---
        ---
class derived_class(parent_class):
        ---
        ---
```

e.g 1.
```
class A:   → parent class
    x = 10
    def m1(self):
        print('m1 -- A')

class B(A): → child class
    y = 20
    def m2(self):
        print('m2 -- B')

a = A()
print(a.x)  #10
a.m1()  #m1 -- A
b = B()
```

```
print(b.x,b.y)   #10 20
b.m1() #m1 -- A
b.m2() #m2 -- B
```

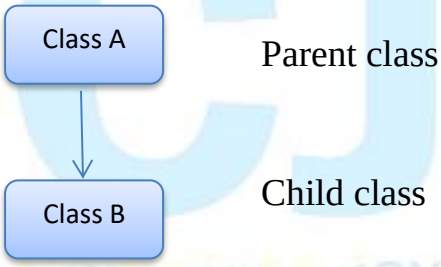
In above program, class B inherits class A so by object of class B i.e b we can access ‘x’ variable and ‘m1’ function of class A also.

Types of Inheritance:

1. Single level inheritance
2. Multilevel inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

1.Single level inheritance:

When a child class inherits from only one parent class, it is called single inheritance.

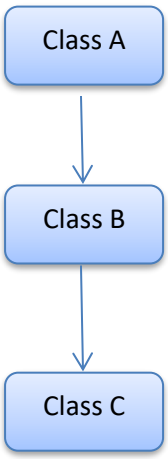


e.g .

```
class A:
        pass
class B(A):
        pass
```

2.Multilevel inheritance:

Multi-level inheritance is archived when a derived class inherits another derived class. There is no limit on the number of levels up to which, the multi-level inheritance is archived in python.



Parent class

Child class

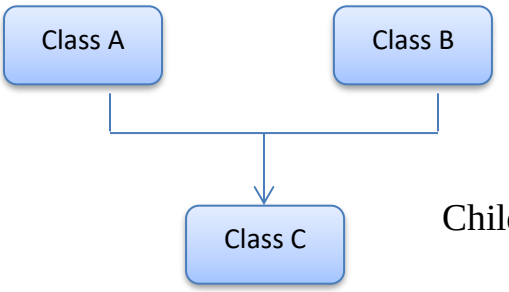
Grandchild class

e.g.

```
class A:
   pass

class B(A):
   pass

class C(B):
   pass
```

3.Multiple inheritance:

When a child class inherits from multiple parent classes, it is called multiple inheritance.



Parent class

Child class

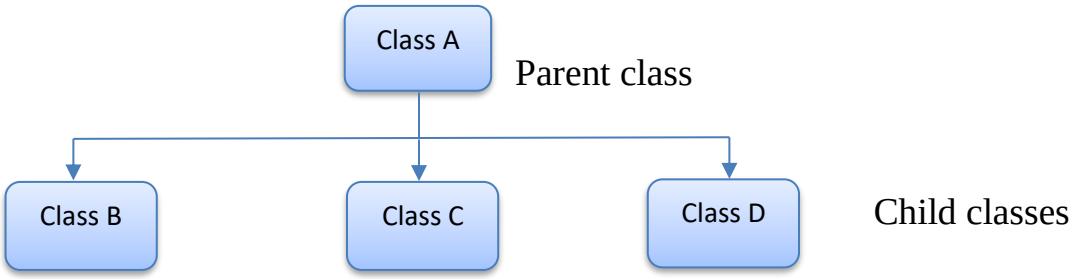
e.g.

```
class A:
   pass

class B:
```

```
    pass

class C(A,B):
    pass
```

4.Hierarchical inheritance:

More than one derived classes are created from a single base.

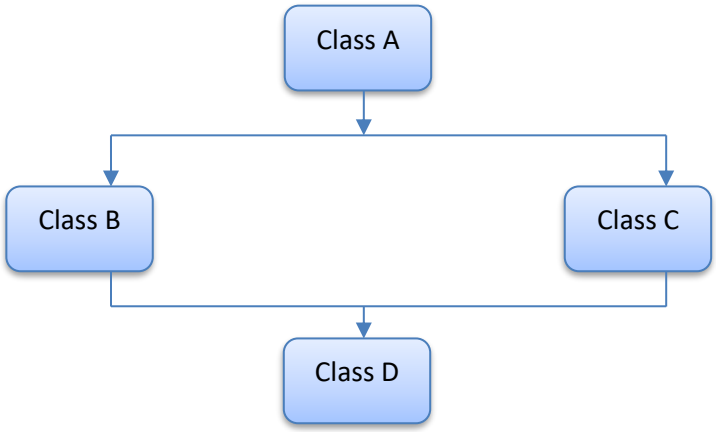


e.g

```
class A:
    pass

class B(A):
    pass

class C(A):
    pass

class D(A):
    pass
```

5.Hybrid inheritance:

This form combines more than one form of inheritance. Basically, it is a blend of more than one type of inheritance like blend of multiple and hierarchical inheritance.



e.g.

```
class A:
   pass

class B(A):
   pass

class C(A):
   pass

class D(B, C):
        pass
```

super() in class:

The super() function is used to give access to methods and properties of a parent or sibling class. The super() function returns an object that represents the immediate parent class.

```
class Parent:
        def __init__(self,txt):
                self.message=txt

        def printmessage(self):
                print(self.message)
```

```
class Child(Parent):
        def __init__(self, txt):
                super().__init__(txt)

x=Child("Hello,and welcome!”)
x.printmessage()
```

OUTPUT:
Hello, and welcome!

Diamond problem:

The hybrid inheritance is considered as diamond problem [Refer the hybrid inheritance figure]. It refers to an ambiguity that arises when two classes Class B and Class C inherit from a superclass Class A and Class D inherits from both Class B and Class C. If there is a method **“m”** which is an present in Class B and Class C then the ambiguity arises which of the method “m” of which class should Class D inherit.

The solution for this problem is linearization that finds the order which is to be followed using a set of rules known as **MRO(Method Resolution Order).**

```
class A:
   def m(self):
      print('m -- A')

class B(A):
   def m(self):
      print('m -- B')
      super().m()

class C(A):
   def m(self):
      print('m -- C')
      super().m()

class D(B,C):
   def m(self):
      print('m -- D')
      super().m()
```

```
print('MRO of class D --> ')
print(D.mro())

d = D()
d.m()
```

OUTPUT:
MRO of class D -->
[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>]

m -- D
m -- B
m -- C
m – A

The issubclass(sub,sup) method:

The issubclass(sub, sup) method is used to check the relationships between the specified classes. It returns true if the first class is the subclass of the second class, and false otherwise.

e.g

```
class A:
    pass

class B(A):
    pass

class C:
    pass

print(issubclass(B,A)) #True
print(issubclass(C,A)) #False
```

The isinstance (obj, class) method:

The isinstance() method is used to check the relationship between the objects and classes. It returns true if the first parameter, i.e., obj is the instance of the second parameter, i.e., class.

e.g.

```
class A:
    pass

class B(A):
    pass

a = A()
b = B()
print(isinstance(a,A)) #True
print(isinstance(b,A)) #True (through inheritance)
print(isinstance(a,B)) #False
```