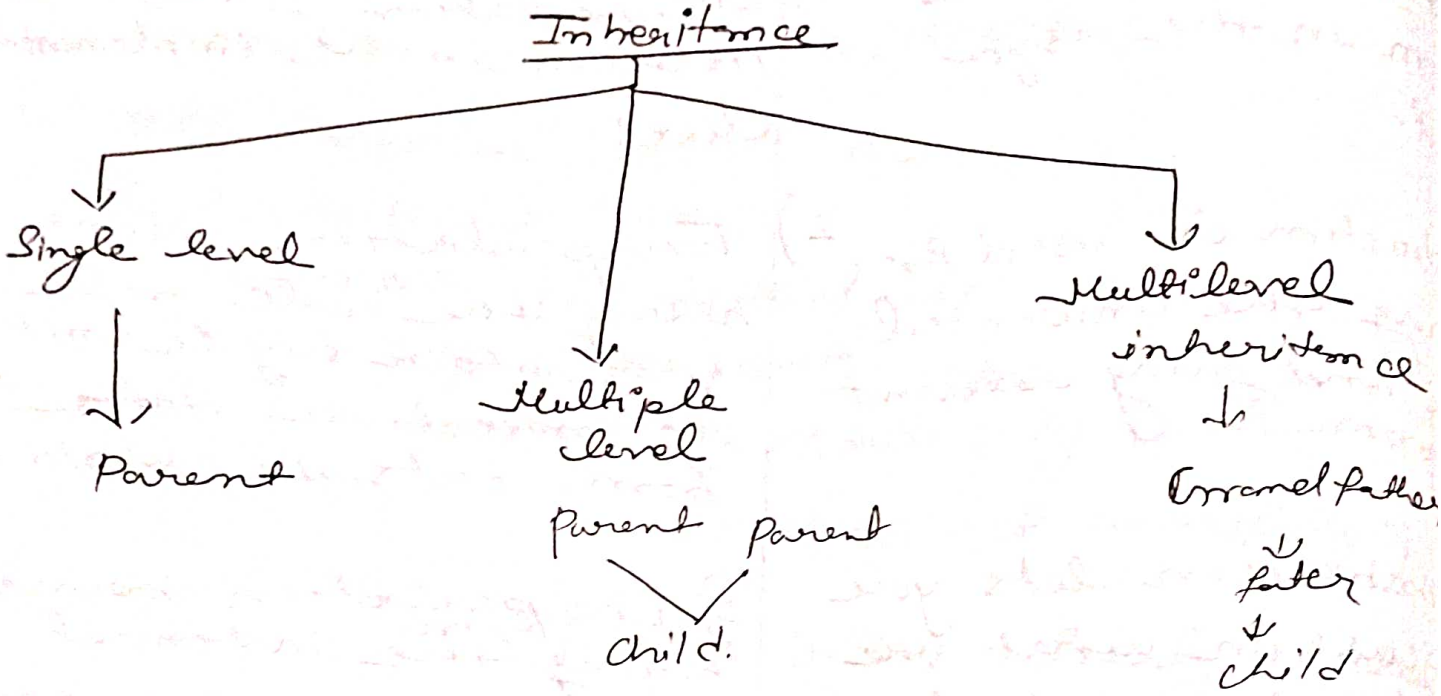
Inheritance



Inheritance

1)
```
class User:
    def login(self):
        print("Login")
    def registration(self):
        print("Registration")
class Student(User):
    def enroll(self):
        print("Enroll")
    def review(self):
        print("Review")
```

create object

```
stu = Student()
stu.enroll()
```

O/P Enroll

```
stu.registration()
```

O/P registration

2)
```
class Phone:
    def __init__(self, price, brand, camera):
        print("Inside phone constructor")
        self.price = price
        self.brand = brand
        self.camera = camera
    def buy(self):
        print("Buying a phone")
    def return_phone(self):
        print("Returning phone")
class Smartphone(Phone):
    pass
```

```
s = Smartphone(20000, "Vivo", 15)

Inside phone Constructor
```

Inheriting private members
Inheriting Constructor.

```
class Phone:
    def __init__(self, price, brand, camera):
        print("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera
    def buy(self):
        print("Buying a phone")
    def return_phone(self):
        print("Returning phone")
class Smartphone(Phone):
    pass

v = Smartphone(45000, "OPPO", 20px)
```

O/P
Inside phone constructor

~~v.__p~~ v._Phone__price

O/P 45000

Inheritance Constructor

```
class Phone:
    def __init__(self, price, brand, camera):
        print("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera
```

```
def buy(self):
    print("Buying a phone")

def return_phone(self):
    print("Returning phone")

class Smartphone(Phone):
    def buy(self):
        print("Buying a smartphone")
```

Create object

```
v = Smartphone(40000, "OPPO", "20px")
```

O/P Inside phone Constructor

```
v.buy()
```

O/P Buying a smartphone

single level inheritance

```
class Parent:
    def __init__(self, num):
        self.__num = num

    def get_num(self):
        return self.__num

class Child(Parent):
    def show(self):
        print("Child Class")
```

Create object

```
son = Child(55)
print(son.__num)   (It shows error because num attribute is private)

son.show()
```

Child Class

Super Keyword

```
class phone:
    def __init__(self, price, brand, camera):
        print("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera
    def buy(self):
        print("Buying a phone")
    def return_phone(self)
        print("Returning phone")
class Smartphone(Phone):
    def buy(self):
        print("Buying a smartphone")
        super().buy()
```

Create Object

```
v = Smartphone(55000, "Vivo", "30px")
```

O/P Inside phone Constructor)

call

```
v.buy()
```

O/P Buying a Smartphone
Buying a phone.

1)

```
class Parent:
    def __init__(self, num):
        self.num = num
    def get_num(self):
        return self.num
class child(Parent):
    def __init__(self, val, num):
        self.__val = val
    def get_val(self):
        return self.__val
```

Object

daughter = chil

daughter ge

O/P too

daughter.

Error - 'cl

class phone

def -

def

class S

de

S

Object

S = S

O/P

Inside

Inside

prt

O/P 15

prin

O/P P

Super keyword

```
class Phone:
    def __init__(self, price, brand, camera):
        print("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera
    def buy(self):
        print("Buying a phone")
    def return_phone(self)
        print("Returning phone")
class Smartphone(Phone):
    def buy(self):
        print("Buying a smartphone")
        super().buy()
```

create object

```
v = Smartphone(55000, "Vivo", "30px")
```

O/P Inside phone constructor

call

```
v.buy()
```

O/P Buying a smartphone
Buying a phone.

1)

```
class Parent:
    def __init__(self, num):
        self.num = num
    def get_num(self):
        return self.num
class child(Parent):
    def __init__(self, val, num):
        self.__val = val
    def get_val(self):
        return self.__val
```

Super Keyword

```
class Phone:
    def __init__(self, price, brand, camera):
        print("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera
    def buy(self):
        print("Buying a phone")
    def return_phone(self)
        print("Returning phone")
class Smartphone(Phone):
    def buy(self):
        print("Buying a smartphone")
        super().buy()
```

Create Object

```
v = Smartphone(55000, "Vivo", "30px")
```

O/P Inside phone Constructor

call

```
v.buy()
```

O/P Buying a Smartphone
Buying a phone.

1)

```
class Parent:
    def __init__(self, num):
        self.num = num
    def get_num(self):
        return self.num
class Child(Parent):
    def __init__(self, val, num):
        self.__val = val
    def get_val(self):
        return self.__val
```

Object

daughter = child(100, 50)

daughter.get_val()

O/P 100

daughter.get_num()

Error – 'child' object has no attribute 'num'

2) class phone:

```
def __init__(self, price, brand, camera):
    print("Inside phone constructor")
    self.price = price
    self.brand = brand
    self.camera = camera
def buy(self):
    print("buying a phone")
class Smartphone(Phone):
    def __init__(self, price, brand, camera,
                 OS, ram):
        super().__init__(price, brand, camera)
        self.os = os
        self.ram = ram
        print("Inside smartphone constructor")
```

Object

S = Smartphone(20000, "Apple", 15, "Android", 2)

O/P

Inside phone constructor

Inside smartphone constructor.

print(S.camera)

O/P 15

print(S.brand)

O/P Apple

Multilevel Inheritance

```
class Product:
    def review(self):
        print("Product customer review")
class Phone(Product):
    def __init__(self, price, brand, camera):
        print("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera
    def buy(self):
        print("Buying a phone")
Class Smartphone(Phone):
    pass
```

Object

```
s = Smartphone(20000, "Apple", 15)
```

O/P Inside phone constructor

```
s.review()
```

O/P Product customer review

Create Object

```
p = Phone(1000000, "VIVO", 12)
```

O/P Inside phone constructor

```
p.review()
```

O/P Product customer review.

Multiple Inheritance

```
class phone:
    def __init__(self, price, brand, camera):
        print("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera
    def buy(self):
        print("Buying a phone")

class Product:
    def review(self):
        print("Product customer review")

class Smartphone(phone, Product):
    pass

p = Smartphone(1000000, "VIVO", 12)
```

O/P
Inside phone constructor

```
p.review()
```

O/P
Product customer review

Static Variable

```
class Atm:
    def __init__(self):
        self.__pin = ""
        self.__balance = 0
        self.sno = 0
        self.sno += 1
    def menu(self):
        pass
```

```
sbi = Atm()
unb = Atm()
```

```
sbi = Atm()
unb.sno
```

O/P 1

Example of Aggregation (Symbol - ◇—)

```
class Customer:
    def __init__(self, name, gender, address):
        self.name = name
        self.gender = gender
        self.address = address
class Address:
    def __init__(self, city, pinno, state):
        self.city = city
        self.pinno = pinno
        self.state = state
```

Object

```
a = Address("Kolkata", 700006, "West Bengal")
c = Customer("Priya", "Female", a)
```

Output

```
print(c.address)
```


Address object at 0x0000...

Polymorphism

The word polymorphism means having many forms. In programming, polymorphism means the same function name (but different signatures) being used for different types.

i) Method Overriding
ii) Method Overloading
~~iii) Method Overloaded~~

Method Overriding

```
class A:
    def m1(self)
        return 20

class B(A):
    def m1(self):
        return 30
    def m2(self):
        return 40

class C(B):
    def m2(self):
        return 20

obj1 = A()
obj2 = B()
obj3 = C()
print(obj1.m1() + obj3.m1() + obj3.m2())
```

O/P — 70

method overloading

```
class Shape:
    def area(self, r):
        return 3.14*r*r
    def area(self, a, b):
        return a*b
```

object

```
p = Shape()
p.area(2)
```

Error (Python doesn't support overloading)

Another approach

```
class Shape:
    def area(self, a, r=0):
        if r==0:
            return 3.14*a*a
        else:
            return a*r

s = Shape()
print(s.area(2))
```