Points to remember

1. Methods of base class are automatically inherited within derived class.

Example:

```
class A: # Base Class/Super Class
    def m1(self): # public method
        print("m1 of A")
    def __m3(self):
        print("m3 of A")
    def _m4(self): # protected method
        print("m4 of A")

class B(A): # Sub Class/Derived Class
    def m2(self):
        print("m2 of B")
        self._m4()

objb=B()
objb.m1()
objb.m2()
```

Output

```
m1 of A
m2 of B
m4 of A
m4 of A
```

2. Data members or variables of base class are automatically inherited within derived class

Example:

```
class A:
    def __init__(self):
        self.x=100


class B(A):
    pass
```

```
objb=B()
print(objb.x)
```

Output
100

Example:
```
class A:
    def __init__(self):
        self.x=100

class B(A):
    def __init__(self):
        super().__init__()
        self.y=200


objb=B()
print(objb.x,objb.y)
```

Output
100 200

super()
The super() function in Python is a built-in function that allows you to call methods from a parent class within a subclass. It is primarily used in the context of inheritance, where a subclass inherits properties and methods from its parent class.

super() enables a subclass to call methods defined in its parent class. This is particularly useful when a subclass overrides a method from the parent but still needs to use the parent's implementation.

Syntax: super().<member-name>

Example:
```
class A:
    def m1(self):
        print("m1 of A")
```

```
class B(A):
    def m2(self):
        self.m1()
        super().m1()
        print("m2 of B")

objb=B()
objb.m2()
```

Output

```
m1 of A
m1 of A
m2 of B
```

In order to inherit data members of base class within derived class, it is required call constructor of base class within constructor of derived class.

Single Level Inheritance

In single level inheritance there is one base class and derived class.

Example:

```
class Person:
    def __init__(self):
        self.__name=None
    def set_name(self,n):
        self.__name=n
    def get_name(self):
        return self.__name

class Employee(Person):
    def __init__(self):
        super().__init__()
        self.__job=None
    def set_job(self,j):
        self.__job=j
    def get_job(self):
        return self.__job

emp1=Employee()
```

```
emp1.set_name("NARESH")
emp1.set_job("HR")
print(f'{emp1.get_name()},{emp1.get_job()}')
```

Output

```
NARESH,HR
```

Multilevel Inheritance

If class is derived from another derived class, it is called multilevel inheritance (OR) more than one level of inheritance is called multi level inheritance.

Example:

```
class Person:
    def __init__(self):
        self.__name=None
    def set_name(self,n):
        self.__name=n
    def get_name(self):
        return self.__name

class Employee(Person):
    def __init__(self):
        super().__init__()
        self.__job=None
    def set_job(self,j):
        self.__job=j
    def get_job(self):
        return self.__job

class SalariedEmployee(Employee):
    def __init__(self):
        super().__init__()
        self.__salary=None
    def set_salary(self,s):
        self.__salary=s
    def get_salary(self):
        return self.__salary

emp1=SalariedEmployee()
```

```
emp1.set_name("naresh")
emp1.set_job("manager")
emp1.set_salary(54000)
print(f'{emp1.get_name()},{emp1.get_job()},{emp1.get_salary()}')
```

Output

```
naresh,manager,54000
```

Multiple Inheritance

A class derived from more than one base class, it is called multiple inheritance.

Example:

```
class A:
    def m1(self):
        print("m1 of A")

class B:
    def m2(self):
        print("m2 of B")

class C(A,B):
    def m3(self):
        print("m3 of C")

objc=C()
objc.m1()
objc.m2()
objc.m3()
```

Output

```
m1 of A
m2 of B
m3 of C
```

Example:

```
class A:
    def __init__(self):
        self.x=100
```

```
class B:
    def __init__(self):
        self.y=200

class C(A,B):
    def __init__(self):
        super().__init__()
        B.__init__(self)
        self.z=300
objc=C()
print(objc.x,objc.y,objc.z)
```

Output

100 200 300

Example:

```
class Person:
    def __init__(self):
        self.__name=None
    def set_name(self,n):
        self.__name=n
    def get_name(self):
        return self.__name

class Account:
    def __init__(self):
        self.__accno=None
        self.__balance=None
    def set_accno(self,a):
        self.__accno=a
    def set_balance(self,b):
        self.__balance=b
    def get_accno(self):
        return self.__accno
    def get_balance(self):
        return self.__balance

class SavingAccount(Person,Account):
    def __init__(self):
```

```
        super().__init__()
        Account.__init__(self)
        self.__cheqfac=None
    def set_cheqfac(self,c):
        self.__cheqfac=c
    def get_cheqfac(self):
        return self.__cheqfac

acc1=SavingAccount()
acc1.set_name("naresh")
acc1.set_accno(10111234)
acc1.set_balance(54000)
acc1.set_cheqfac(True)
print(f"'{acc1.get_name()}
{acc1.get_accno()}
{acc1.get_balance()}
{acc1.get_cheqfac()}'")
```

Output

```
naresh
10111234
54000
True
```

Hierarchical Inheritance

If more than one class derived from same base class, it is called hierarchical inheritance.

Example:

```
class Person:
    def __init__(self):
        self.__name=None
    def set_name(self,n):
        self.__name=n
    def get_name(self):
        return self.__name

class Student(Person):
    def __init__(self):
        self.__course=None
```

```
    def set_course(self,c):
        self.__course=c
    def get_course(self):
        return self.__course

class Employee(Person):
    def __init__(self):
        self.__job=None
    def set_job(self,j):
        self.__job=j
    def get_job(self):
        return self.__job

class Player(Person):
    def __init__(self):
        self.__country=None
    def set_country(self,c):
        self.__country=c
    def get_country(self):
        return self.__country

stud1=Student()
stud1.set_name("ramesh")
stud1.set_course("python")
print(f"'{stud1.get_name()},{stud1.get_course()}'")
p1=Player()
p1.set_name("virat")
p1.set_country("ind")
print(f"'{p1.get_name()},{p1.get_country()}'")
emp1=Employee()
emp1.set_name("suresh")
emp1.set_job("hr")
print(f'{emp1.get_name()},{emp1.get_job()}')
```

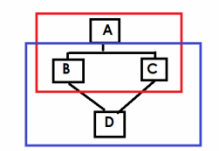
Output

```
ramesh,python
virat,ind
suresh,hr
```

Hybrid Inheritance

If classes are organized using more than one type of inheritance, it is called hybrid inheritance.

Hybrid inheritance in Python is a combination of multiple and hierarchical inheritance. It occurs when a class inherits from multiple classes, and at least one of those parent classes is itself derived from another class. This creates a complex inheritance structure involving multiple inheritance paths.



Example:

```
class A:
    def m1(self):
        print("m1 if A")

class B(A):
    def m2(self):
        print("m2 of B")

class C(A):
    def m3(self):
        print("m3 of C")

class D(B,C):
    def m4(self):
        print("m4 of D")

objd=D()
objd.m1()
objd.m2()
objd.m3()
objd.m4()
```

Output

```
m1 if A
m2 of B
m3 of C
```

m4 of D

MRO