super() function

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 parent class.

Syntax:
super()
super(child-class,self)

super() function is used with single level inheritance.

Example:

```
# Single Level Inheritance

class Person:
    def __init__(self):
        self.__name=None
    def setName(self,n):
        self.__name=n
    def getName(self):
        return self.__name


class Student(Person):
    def __init__(self):
        super().__init__()
        self.__course=None
    def setCourse(self,c):
        self.__course=c
    def getCourse(self):
        return self.__course

stud1=Student()
stud1.setName("naresh")
```

```
stud1.setCourse("python")
name=stud1.getName()
course=stud1.getCourse()
print(name,course)
```

Output

naresh python

Multi Level Inheritance

If a class is derived from another derived class, it is called multi level inheritance.

Example:

```
class A:
    def __init__(self):
        print("constructor of A")

class B(A):
    def __init__(self):
        super().__init__()
        print("constructor of B")

class C(B):
    def __init__(self):
        super().__init__()
        print("constructor of C")



objc=C()
```

Output

constructor of A
constructor of B
constructor of C

Example:

```
class A:
    def __init__(self):
        self.x=100
        print("constructor of A")

class B(A):
    def __init__(self):
        super().__init__()
        self.y=200
        print("constructor of B")

class C(B):
    def __init__(self):
        super().__init__()
        self.z=300
        print("constructor of C")



objc=C()
print(objc.x,objc.y,objc.z)
```

Output

```
constructor of A
constructor of B
constructor of C
100 200 300
```

Example:

```
class Person:
    def __init__(self):
        self.__name=None
    def setName(self,n):
```

```
        self.__name=n
    def getName(self):
        return self.__name

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

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


emp1=SalariedEmployee()
emp1.setName("naresh")
emp1.setJob("manager")
emp1.setSalary(50000)
print(emp1.getName(),emp1.getJob(),emp1.getSalary())
```

Output

naresh manager 50000

Multiple Inheritance

If a class is derived from more than one base class or parent class is called multiple inheritance.

Example:

```
class A:
    def __init__(self):
        print("constructor of A")

class B:
    def __init__(self):
        print("constructor of B")


class C(A,B):
    def __init__(self):
        super().__init__()
        B.__init__(self)
        print("constructor of C")


objc=C()
```

Output

constructor of A
constructor of B
constructor of C

Example:

```
class A:
    def __init__(self):
        self.x=100
        print("constructor of A")

class B:
    def __init__(self):
        self.y=200
        print("constructor of B")
```

```
class C(A,B):
    def __init__(self):
        super().__init__()
        B.__init__(self)
        self.z=300
        print("constructor of C")


objc=C()
print(objc.x,objc.y,objc.z)
```

Output

constructor of A
constructor of B
constructor of C
100 200 300

Hierarchical Inheritance

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

Example:

```
class Employee:
    def __init__(self,n,j):
        self.__name=n
        self.__job=j
    def getName(self):
        return self.__name
    def getJob(self):
        return self.__job

class SalariedEmployee(Employee):
    def __init__(self,n,j,s):
        super().__init__(n,j)
        self.__salary=s
```

```
    def getSalary(self):
        return self.__salary

class Worker(Employee):
    def __init__(self,n,j,w):
        super().__init__(n,j)
        self.__wage=w
    def getWage(self):
        return self.__wage


se1=SalariedEmployee("naresh","manager",5000)
w1=Worker("suresh","sales",1000)
print(se1.getName(),se1.getJob(),se1.getSalary())
print(w1.getName(),w1.getJob(),w1.getWage())
```

Output

naresh manager 5000

suresh sales 1000

Hybrid Inheritance

If there is more than one type of inheritance to create a class is called hybrid inheritance.

```
class A:
    def __init__(self):
        print("Constructor of A")

class B(A):
    def __init__(self):
        super().__init__()
        print("Constructor of B")

class C(A):
    def __init__(self):
        super().__init__()
```

```
        print("Constructor of C")

class D(B,C):
    def __init__(self):
        super().__init__()
        C.__init__(self)
        print("Constructor of D")


objd=D()
```

Output

Constructor of A
Constructor of C
Constructor of B
Constructor of A
Constructor of C
Constructor of D

Method Overriding (Polymorphism)

What is polymorphism?

"poly" means "many" and "morphism" is "forms", defining one thing in many forms is called polymorphism.

Python support two types of polymorphism

1. Method overriding
2. Operator overloading