Overerving en inheritance

Fives types of inheritance in python 1) Single Inheritance 2) Multi level Inheritance 3) Hierarchical Inheritance 4) Multiple Inheritance 5) Hybrid Inheritance 6) Cyclic Inheritance

Single Inheritance :- contains single parent and single child class

```
class Parent:
    def m1(self):
        print("Parent class Method")


class Child(Parent):
    def m2(self):
        print("Child class method")


c = Child()
c.m1()
c.m2()
```

```
Parent class Method
Child class method
```

Multi level Inheritance :- The concept of inheriting members from multiple classes to a single child class one after other is called MULTIPLE LEVEL INHERITANCE

```
class Parent:
    def m1(self):
        print("Parent class object")


class Child(Parent):
    def m2(self):
        print("Child class object")


class Child2(Child):
    def m3(self):
        print("Child2 class object")


c = Child2()
```

```
c.m1()
c.m2()
c.m3()
# Any(multiple) number of levels possible
```

```
Parent class object
Child class object
Child2 class object
```

Hierarchical Inheritance :- single parent class multiple child clases

```
class Parent:
    def m1(self):
        print("Parent class object")


class Child1(Parent):
    def m2(self):
        print("Child1 class object")


class Child2(Parent):
    def m3(self):
        print("Child2 class object")


c = Child1()
c.m1()
c.m2()
# Here c.m3() will get error
c = Child2()
c.m1()
c.m3()
# Here c.m2() will get error
```

```
Parent class object
Child1 class object
Parent class object
Child2 class object
```

Multiple Inheritance :- Multiple parent classes and single chils class

```
class Parent1:
    def m1(self):
        print("Parent1 class object")
```

```
class Parent2():
    def m2(self):
        print("Parent2 class object")


class Child(Parent1, Parent2):
    def m3(self):
        print("Child class object")


c = Child()
c.m1()
c.m2()
c.m3()
```

```
Parent1 class object
Parent2 class object
Child class object
```

Hybrid Inheritance :- combination of all the above Inheritances order will be decided by MRO Method Resolution Order Algorithm. Even two types of inheritance is used itis called Hybrid Inheritance.

Cyclic Inheritance

```
class Person:
    def __init__(self, name, age, height, weight):
        self.name = name
        self.age = age
        self.height = height
        self.weight = weight

    def display(self):
        print("Name", self.name)
        print("Age", self.age)
        print("Height", self.height)
        print("Weight", self.weight)


class Student(Person):
    def __init__(self, name, age, height, weight, rollno, marks):
        self.name = name
        self.age = age
        self.height = height
        self.weight = weight
```

```
        self.rollno = rollno
        self.marks = marks

    def display(self):
        print("Name", self.name)
        print("Age", self.age)
        print("Height", self.height)
        print("Weight", self.weight)
        print("Rollno", self.rollno)
        print("Marks", self.marks)


s = Student("Raj", 25, 5.6, 75, 587, 90)
s.display()
```

```
Name Raj
Age 25
Height 5.6
Weight 75
Rollno 587
Marks 90
```

```
# Now by using super() method
class Person:
    def __init__(self, name, age, height, weight):
        self.name = name
        self.age = age
        self.height = height
        self.weight = weight

    def display(self):
        print("Name", self.name)
        print("Age", self.age)
        print("Height", self.height)
        print("Weight", self.weight)


class Student(Person):
    def __init__(self, name, age, height, weight, rollno, marks):
        super().__init__(name, age, height, weight)
        self.rollno = rollno
        self.marks = marks

    def display(self):
        super().display()
        print("Rollno", self.rollno)
        print("Marks", self.marks)
```

```
s = Student("Raj", 25, 5.6, 75, 587, 90)
s.display()
```

```
Name Raj
Age 25
Height 5.6
Weight 75
Rollno 587
Marks 90
```

```
class P:
    def __init__(self):
        print("Parent constructor")

    def m1(self):
        print("Parent Instance method")

    @classmethod
    def m2(cls):
        print("parent class method")

    @staticmethod
    def m3():
        print("Parent static method")


class C(P):
    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()


c = C()
```

```
Parent constructor
Parent Instance method
parent class method
Parent static method
```

POLYMORPHISM

```
class Book:
    def __init__(self, pages):
```

```
        self.pages = pages

    def __add__(self, other):
        total_pages = self.pages + other.pages
        return total_pages


b1 = Book(200)
b2 = Book(300)
print(b1 + b2)
```

```
500
```

```
class Book:
    def __init__(self, pages):
        self.pages = pages

    def __add__(self, other):
        total_pages = self.pages + other.pages
        return total_pages


b1 = Book(200)
b2 = Book(300)
b3 = Book(500)
print(b1 + b2)
print(b1 + b3)
print(b2 + b3)
print(10 + 20)
print("POLY" + "MORPHISM")
```

```
500
700
800
30
POLYMORPHISM
```

- ——> **add**()
- ——> **sub**()
- ——> **mul**() / ——> **div**() // ——> **floordiv**() ** ——> **pow**() % ——-> **mod**()+= ——> **iadd**() -= ——> **isub**() *= ——> **imul**() /= ——> **idiv**() //= ——> **ifloordiv**() **= ——> **ipow**() %= ——-> **imod**()< ——> **lt**() > ——> **gt**() <= ——> **le**() >= ——> **ge**() == ——> **eq**() != ——> **ne**()

```
class Student:
    def __init__(self, name, marks):
        self.name = name
```

```
        self.marks = marks

    def __lt__(self, other):
        result = self.marks < other.marks
        return result

    def __le__(self, other):
        result = self.marks <= other.marks
        return result


s1 = Student("one", 100)
s2 = Student("two", 200)
s3 = Student("three", 50)

print(s1 < s2)
print(s2 < s3)
print(s3 <= s1)
```

```
True
False
True
```

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def __mul__(self,
                other):  ## Here in Employee function we used magic function(mul) because it
        result = self.salary * other.days
        return result


class TimeSheet:
    def __init__(self, name, days):
        self.name = name
        self.days = days

    def __mul__(self,
                other):  ## Here in TimeStamp function we used magic function(mul) because t
        result = self.days * other.salary
        return result


e = Employee("one", 1000)
t = TimeSheet("two", 25)
```

```
print("This month salary", e * t)
print("This month salary", t * e)
```

```
This month salary 25000
This month salary 25000
```

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def __str__(self):
        return self.name
        #return self.marks # Error int type


s1 = Student("one", 90)
s2 = Student("two", 95)

print(s1)
print(s2)
```

```
one
two
```

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def __str__(self):
        return "Student with Name:{},Marks:{}".format(self.name, self.marks)


s1 = Student("one", 90)
s2 = Student("two", 95)

print(s1)
print(s2)
```

```
Student with Name:one,Marks:90
Student with Name:two,Marks:95
```

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks
```

```
    def __str__(self):
        return str(self.marks)


s1 = Student("one", 90)
s2 = Student("two", 95)
print(s1)
print(s2)
```

```
90
95
```