# Overerving en inheritance

Vijf soorten overerving in python

1. Single Inheritance
2. Multi level Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance
6. Cyclic Inheritance

## Single Inheritance

bevat één ouder en één kind klasse

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()

## Multi level Inheritance

Het concept van het één na één overerven van leden van meerdere klassen naar één kindklasse wordt MULTIPLE LEVEL INHERITANCE genoemd.

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

## Hierarchical Inheritance

één parent-klasse meerdere kindklassen

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

## Multiple Inheritance

Meerdere ouderklassen en enkele kindklassen

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()

### Hybrid Inheritance

De combinatie van alle bovenstaande overervingen wordt bepaald door het MRO Method Resolution Order Algorithm. Zelfs als twee soorten overerving worden gebruikt, wordt het hybride overerving genoemd.

### 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()

# 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()

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()

### 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)

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")

* —–> **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)

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 in first print function we used "e"(argument in print function) reference variable first  
 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 in first print function we used "t"(argument in print function) reference variable first  
 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)

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)

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)

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)