

# Object Oriented Programming (Python)

It is a programming paradigm - A way to write programs

**Class** is a template and Object is **Instance** of that class.

*We can have multiple instances of a class*

```
In [2]: 1 class Student: # Class named Student
        2     pass
```

```
In [3]: 1 s1 = Student() # s1 is instance of "Student" class.
```

```
In [7]: 1 print(type(s1))

<class '__main__.Student'>
```

**s1 belongs to the class Student**

```
In [8]: 1 l = list()
        2 print(type(l))

<class 'list'>
```

/ belongs to the class list()

**Infact everything in python is a class**

## Attributes in Class

```
In [4]: 1 s1.name = "Rajat"
```

```
In [5]: 1 s1.__dict__
```

```
Out[5]: {'name': 'Rajat'}
```

```
In [ ]: 1 s2 = Student()
        2 s2.name = 'Rajat'
        3 s2.rollno = '123'
```

**name and rollno are attributes(data) of the object s2 of class Student**

```
In [7]: 1 s2.__dict__
```

```
Out[7]: {'name': 'Rajat', 'rollno': '123'}
```

Those attributes are stored as key-value pairs as dict.

object.key = value

```
In [8]: 1 hasattr(s1, 'name') # check if s1 have the attribute named "name".
```

Out[8]: True

```
In [15]: 1 hasattr(s1, 'rollno')
```

Out[15]: False

```
In [9]: 1 getattr(s1, 'name') # Alternate of object.key
```

Out[9]: 'Rajat'

```
In [17]: 1 getattr(s2, 'rollno')
```

Out[17]: '123'

```
In [18]: 1 getattr(s1, 'rollno')
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-18-03a6c151f88f> in <module>
----> 1 getattr(s1, 'rollno')

AttributeError: 'Student' object has no attribute 'rollno'
```

```
In [19]: 1 # dont want an error
        2 getattr(s1, 'rollno', 'not found')
```

Out[19]: 'not found'

```
In [25]: 1 # delete attribute
        2 delattr(s1, 'name')
```

```
In [26]: 1 s1.name
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-26-bfa1764dcdbc> in <module>
----> 1 s1.name

AttributeError: 'Student' object has no attribute 'name'
```

```
In [27]: 1 class student:
        2     name = 'Rajat Bansal'
        3     pp = 80
        4
        5     def isPassed(self):
        6         self.percentage = 90
        7         if self.percentage > self.pp:
        8             return True
```

## Class and Instance Methods

```
In [49]: 1 class Student:
2         passingper = 40
3         def StudentDetails(self): # Instance Method
4             self.name = 'Rajat Bansal'
5             print(self.name)
6
7             self.percentage = 96
8             print(self.percentage)
9
10        pass
11
12        def isPassed(self): # Instance Method
13            if self.percentage >= self.passingper: # or can use Student.passingper
14                print("passed")
15            else:
16                print("Not Passed")
```

```
In [50]: 1 s1 = Student()
2         s1.StudentDetails()
```

```
Rajat Bansal
96
```

```
In [51]: 1 s1.isPassed()
```

```
passed
```

**Instance Methods** are those which do require the properties of object and we need to pass the object itself to them.

## Static Methos

These methods are which which do not require properties from the object itself so we do not need to pass 'self' to them. But this can be done by **@staticmethod** decorator. this decorator ignores the first argument from the method.

```
In [74]: 1 class Student:
2         passingper = 40
3         name = 'my name'
4         def StudentDetails(self): # Instance Method
5             self.name = 'Rajat Bansal'
6             print(self.name)
7
8             self.percentage = 96
9             print(self.percentage)
10
11        pass
12
13        def isPassed(self): # Instance Method
14            if self.percentage >= self.passingper: # or can use Student.passingper
15                print("passed")
16            else:
17                print("Not Passed")
18
19        ## Static Method
20        @staticmethod # this ignores the first argument in the function
21        def welcometoschool():
22            print('Welcome to School')
```

```
In [69]: 1 s1 = Student()
```

```
In [67]: 1 s1.welcometoschool()
```

Welcome to School

```
In [70]: 1 s1.name
```

```
Out[70]: 'my name'
```

```
In [72]: 1 s1.StudentDetails() # name updated by the method
```

Rajat Bansal  
96

```
In [73]: 1 s1.name
```

```
Out[73]: 'Rajat Bansal'
```

## "init method"

```
In [75]: 1 class Student:
2
3     def __init__(self, name, rollno):
4         self.name = name
5         self.rollno = rollno
6
7     passingper = 40
8     name = 'my name'
9     def StudentDetails(self): # Instance Method
10         self.name = 'Rajat Bansal'
11         print(self.name)
12
13         self.percentage = 96
14         print(self.percentage)
15
16         pass
17
18     def isPassed(self): # Instance Method
19         if self.percentage >= self.passingper: # or can use Student.passingper
20             print("passed")
21         else:
22             print("Not Passed")
23
24     ## Static Method
25     @staticmethod # this ignores the first argument in the function
26     def welcometoschool():
27         print('Welcome to School')
```

```
In [77]: 1 s1 = Student('Rajat', 123)
```

## Classmethods

**Classmethods** manipulates the input and return an object of any class.

In [90]:

```
1 from datetime import date
2 class Student:
3
4     def __init__(self, name, age, rollno):
5         self.name = name
6         self.age = age
7         self.rollno = rollno
8
9     # classmethod
10    @classmethod
11    def fromBithYear(cls, name, year, rollno):
12        return cls(name, date.today().year - year, rollno)
13
14
15
16    def StudentDetails(self): # Instance Method
17        self.name = 'Rajat Bansal'
18        print(self.name)
19        print("age", self.age)
20        self.percentage = 96
21        print(self.percentage)
22
23        pass
24
25    def isPassed(self): # Instance Method
26        if self.percentage >= self.passingper: # or can use Student.passingper
27            print("passed")
28        else:
29            print("Not Passed")
30
31    ## Static Method
32    @staticmethod # this ignores the first argument in the function
33    def welcometoschool():
34        print('Welcome to School')
```

In [82]:

```
1 s1 = Student.fromBithYear('Rajat', 2001, 25)
```

In [83]:

```
1 s1.StudentDetails()
```

```
Rajat Bansal
age 20
96
```

## Public and Private Modifiers

```

In [28]: 1 from datetime import date
          2 class Student:
          3     __passingpercentage = 40 # private Variable
          4
          5     def __init__(self, name, age, rollno):
          6         self.__name = name
          7         self.age = age
          8         self.rollno = rollno
          9
         10     # classmethod
         11     @classmethod
         12     def fromBithYear(cls, name, year, rollno):
         13         return cls(name, date.today().year - year, rollno)
         14
         15
         16
         17     def StudentDetails(self): # Instance Method
         18         print(self.__name)
         19         print("age", self.age)
         20         self.percentage = 96
         21         print(self.percentage)
         22
         23         pass
         24
         25     def isPassed(self): # Instance Method
         26         if self.percentage >= self.passingper: # or can use Student.passingper
         27             print("passed")
         28         else:
         29             print("Not Passed")
         30
         31     ## Static Method
         32     @staticmethod # this ignores the first argument in the function
         33     def welcometoschool():
         34         print('Welcome to School')

```

```

In [29]: 1 s1 = Student('Rajat', 20, 251)

```

```

In [30]: 1 s1.StudentDetails()

```

```

Rajat
age 20
96

```

```

In [31]: 1 s1.name = "Rohan"

```

```

In [32]: 1 s1.name

```

```

Out[32]: 'Rohan'

```

```

In [33]: 1 s1.__name

```

```

-----
AttributeError                                Traceback (most recent call last)
<ipython-input-33-04dad977536f> in <module>
----> 1 s1.__name

AttributeError: 'Student' object has no attribute '__name'

```

```
In [25]: 1 # access outside
         2 # called Name Bangling Syntax: object._className__varName
         3
         4 s1._Student__name
```

Out[25]: 'Rajat'

```
In [72]: 1 class Fraction:
         2
         3     def __init__(self, num = 0, den = 1):
         4         if den == 0:
         5             # Thorow error
         6             den = 1
         7         self.num = num
         8         self.den = den
         9
        10     def print(self):
        11         if self.num == 0:
        12             print(0)
        13         elif self.den == 1:
        14             print(self.num)
        15         else:
        16             print(self.num, "/" , self.den)
        17     def simplyfy(self):
        18         if self.num == 0:
        19             self.den = 1
        20             return
        21         curr = min(self.num, self.den)
        22         while curr > 1:
        23             if self.num % curr == 0 and self.den % curr == 0:
        24                 break
        25             curr -= 1
        26         self.num = self.num // curr
        27         self.den = self.den // curr
        28
        29     def add(self, other):
        30         newNum = other.den * self.num + other.num * self.den
        31         newDen = other.den * self.den
        32
        33         self.num = newNum
        34         self.den = newDen
        35
        36         self.simplyfy()
        37
        38     def multiply(self, other):
        39         self.num = other.num * self.num
        40         self.den = other.den * self.den
        41
        42         self.simplyfy
        43
        44
        45
        46
        47
```

```
In [77]: 1 f1 = Fraction(2,3)
         2 f2 = Fraction(1,3)
```

```
In [ ]: 1 f1.add(f2)
         2 f1.print()
```

```
In [78]: 1 f1.multiply(f2)
        2 f1.print()
```

2 / 9

```
In [61]: 1 f.__dict__
```

```
Out[61]: {'num': 0, 'den': 5}
```

```
In [62]: 1 f.print()
```

0

```
In [63]: 1 f.simplyfy()
```

```
In [64]: 1 f.print()
```

0

## Complex Numbers

```
In [128]: 1 class Complex:
        2     def __init__(self, a = 0, b = 0):
        3         self.real = a
        4         self.im = b
        5         print(self.real, "+ i" , self.im, sep = "")
        6
        7     def print(self):
        8         print(self.real, "+ i" , self.im, sep = "")
        9
       10     def plus(self, other, inplace = False):
       11         a = self.real + other.real
       12         b = self.im + other.im
       13         if inplace == True:
       14             self.real = a
       15             self.im = b
       16             self.print()
       17         else:
       18             return (Complex(a,b))
       19
       20
       21     def mul(self, other):
       22         a = (self.real* other.real) - (self.im * other.im)
       23         b = (self.real * other.im) + (self.im * other.real)
       24         self.real = a
       25         self.im = b
       26         self.print()
       27
```

```
In [129]: 1 c1 = Complex(4,5)
        2 c2 = Complex(6,7)
        3
```

4+ i5

6+ i7



```
In [133]: 1 n = c1.plus(c2, inplace = True)
```

```
10+ i12
```

```
In [134]: 1 c1.print()
```

```
10+ i12
```

```
In [116]: 1 c1.mul(c2)
```

```
-11+ i58
```

```
In [121]: 1 n.print()
```

```
10+ i12
```

## Inheritance

use **super()** for to inherit attributes from another(parent) class.

```
In [1]: 1 class vehicle:
2       def __init__(self, color, maxSpeed):
3           self.color = color
4           self.maxSpeed = maxSpeed
5
6       class car(vehicle):
7           def __init__(self, color, maxSpeed, numGears, isConvertible):
8               super().__init__(color, maxSpeed)
9               self.numGears = numGears
10              self.isConvertible = isConvertible
11
12          def printCar(self):
13              print(self.color, self.maxSpeed, self.numGears, self.isConvertible)
```

```
In [2]: 1 c1 = car('Red', 120, 4, False)
2       c1.printCar()
```

```
Red 120 4 False
```

## Inheritance and Private Members

```
In [3]: 1 class vehicle:
2       def __init__(self, color, maxSpeed):
3           self.color = color
4           self.__maxSpeed = maxSpeed
5
6       class car(vehicle):
7           def __init__(self, color, maxSpeed, numGears, isConvertible):
8               super().__init__(color, maxSpeed)
9               self.numGears = numGears
10              self.isConvertible = isConvertible
11
12          def printCar(self):
13              print(self.color, self.__maxSpeed, self.numGears, self.isConvertible)
```

In [4]:

```
1 c1 = car('Red', 120, 4, False)
2 c1.printCar()
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-4-8e60d7e7b8f7> in <module>
      1 c1 = car('Red', 120, 4, False)
----> 2 c1.printCar()

<ipython-input-3-48e9c4c1f1c2> in printCar(self)
     11
     12     def printCar(self):
--> 13         print(self.color, self.__maxSpeed, self.numGears, self.isConvertible)

AttributeError: 'car' object has no attribute '_car__maxSpeed'
```

We can not access private variables from the parent class directly. But we can use functions to return the values of those private variables.

In [5]:

```
1 class vehicle:
2     def __init__(self, color, maxSpeed):
3         self.color = color
4         self.__maxSpeed = maxSpeed
5
6     def getMaxSpeed(self):
7         return self.__maxSpeed
8
9     def setMaxSpeed(self, newSpeed):
10        self.__maxSpeed = newSpeed
11
12 class car(vehicle):
13     def __init__(self, color, maxSpeed, numGears, isConvertible):
14         super().__init__(color, maxSpeed)
15         self.numGears = numGears
16         self.isConvertible = isConvertible
17
18     def printCar(self):
19         print(self.color, self.getMaxSpeed(), self.numGears, self.isConvertible)
20
21     def change_speed()
```

```
File "<ipython-input-5-5f437f05f6aa>", line 21
    def change_speed()
        ^
```

**SyntaxError:** invalid syntax

```
In [6]: 1 c1 = car('Red', 120, 4, False)
        2 c1.printCar()
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-6-8e60d7e7b8f7> in <module>
      1 c1 = car('Red', 120, 4, False)
----> 2 c1.printCar()

<ipython-input-3-48e9c4c1f1c2> in printCar(self)
     11
     12     def printCar(self):
--> 13         print(self.color, self.__maxSpeed, self.numGears, self.isConvertible)

AttributeError: 'car' object has no attribute '_car__maxSpeed'
```

```
In [15]: 1 class vehicle:
        2     def __init__(self, color, maxSpeed):
        3         self.color = color
        4         self.__maxSpeed = maxSpeed
        5
        6     def getMaxSpeed(self):
        7         return self.__maxSpeed
        8
        9     def setMaxSpeed(self, newSpeed):
       10         self.__maxSpeed = newSpeed
       11
       12     def Print(self):
       13         print(self.color, self.__maxSpeed)
       14
       15 class car(vehicle):
       16     def __init__(self, color, maxSpeed, numGears, isConvertible):
       17         super().__init__(color, maxSpeed)
       18         self.numGears = numGears
       19         self.isConvertible = isConvertible
       20
       21     def printCar(self):
       22         self.Print() # self.print() also works fine
       23         super().Print()
       24         print(self.numGears, self.isConvertible)
       25
```

self.Print() also works as super().Print()

```
In [16]: 1 c1 = car('Red', 120, 4, False)
        2 c1.printCar()
```

```
Red 120
Red 120
4 False
```

In [9]:

```
1 class vehicle:
2     def __init__(self, color, maxSpeed):
3         self.color = color
4         self.__maxSpeed = maxSpeed
5
6     def getMaxSpeed(self):
7         return self.__maxSpeed
8
9     def setMaxSpeed(self, newSpeed):
10        self.__maxSpeed = newSpeed
11
12    def Print(self):
13        print(self.color, self.__maxSpeed)
14
15 class car(vehicle):
16     def __init__(self, color, maxSpeed, numGears, isConvertible):
17         super().__init__(color, maxSpeed)
18         self.numGears = numGears
19         self.isConvertible = isConvertible
20
21     def printCar(self):
22         self.Print() #self.Print() will also work as this function is inherited fr
23         print(self.numGears, self.isConvertible)
24
```

In [10]:

```
1 c1 = car('Red', 120, 4, False)
2 c1.printCar()
```

Red 120

4 False

**Methods and properties are inherited from the parent class.**

## Polymorphism

### Method Overwriting

Ability to take multiple forms.

```
In [20]: 1 class vehicle:
2         def __init__(self, color, maxSpeed):
3             self.color = color
4             self.__maxSpeed = maxSpeed
5
6         def getMaxSpeed(self):
7             return self.__maxSpeed
8
9         def setMaxSpeed(self, newSpeed):
10            self.__maxSpeed = newSpeed
11
12        def Print(self):
13            print(self.color, self.__maxSpeed)
14
15    class car(vehicle):
16        def __init__(self, color, maxSpeed, numGears, isConvertible):
17            super().__init__(color, maxSpeed)
18            self.numGears = numGears
19            self.isConvertible = isConvertible
20
21        def Print(self):
22            # self.Print() #self.Print() will also work as this function is inherited
23            print(self.numGears, self.isConvertible)
```

Both Parent(vehicle) and Child(Car) classes have the method **Print** But when called firstly the method from child class is being used.

```
In [22]: 1 c1 = car('Red', 120, 4, False)
2         c1.Print()
```

4 False

```
In [23]: 1 class vehicle:
2         def __init__(self, color, maxSpeed):
3             self.color = color
4             self.__maxSpeed = maxSpeed
5
6         def getMaxSpeed(self):
7             return self.__maxSpeed
8
9         def setMaxSpeed(self, newSpeed):
10            self.__maxSpeed = newSpeed
11
12        def Print(self):
13            print(self.color, self.__maxSpeed)
14
15    class car(vehicle):
16        def __init__(self, color, maxSpeed, numGears, isConvertible):
17            super().__init__(color, maxSpeed)
18            self.numGears = numGears
19            self.isConvertible = isConvertible
20
21        # def Print(self):
22        #     self.Print() #self.Print() will also work as this function is inherited
23        #     print(self.numGears, self.isConvertible)
24
```

```
In [24]: 1 c1 = car('Red', 120, 4, False)
         2 c1.Print()
```

Red 120

**First it will search in the child class for the method if not found then it will go for the Parent Class**

```
In [25]: 1 class vehicle:
         2     def __init__(self, color, maxSpeed):
         3         self.color = color
         4         self.__maxSpeed = maxSpeed
         5
         6     def getMaxSpeed(self):
         7         return self.__maxSpeed
         8
         9     def setMaxSpeed(self, newSpeed):
        10         self.__maxSpeed = newSpeed
        11
        12     def Print(self):
        13         print(self.color, self.__maxSpeed)
        14
        15 class car(vehicle):
        16     def __init__(self, color, maxSpeed, numGears, isConvertible):
        17         super().__init__(color, maxSpeed)
        18         self.numGears = numGears
        19         self.isConvertible = isConvertible
        20
        21     def Print(self):
        22         self.Print() #self.Print() will also work as this function is inherited fr
        23         print(self.numGears, self.isConvertible)
        24
```

```
In [26]: 1 c1 = car('Red', 120, 4, False)
         2 c1.Print()
```

-----  
**RecursionError** Traceback (most recent call last)

<ipython-input-26-75875184239f> in <module>

```
      1 c1 = car('Red', 120, 4, False)
----> 2 c1.Print()
```

<ipython-input-25-2ec174f2f597> in Print(self)

```
    20
    21     def Print(self):
--> 22         self.Print() #self.Print() will also work as this function is inherit
ed from the parent class
    23         print(self.numGears, self.isConvertible)
    24
```

... last 1 frames repeated, from the frame below ...

<ipython-input-25-2ec174f2f597> in Print(self)

```
    20
    21     def Print(self):
--> 22         self.Print() #self.Print() will also work as this function is inherit
ed from the parent class
    23         print(self.numGears, self.isConvertible)
    24
```

**RecursionError**: maximum recursion depth exceeded

**Max recursion error because the Print function calls itself again and again because it finds the**

function **Print()** in itself

To avoid from this we need **Super()**. *Super()* ensures that function is being called from Parent class.

In [28]:

```
1 class vehicle:
2     def __init__(self, color, maxSpeed):
3         self.color = color
4         self.__maxSpeed = maxSpeed
5
6     def getMaxSpeed(self):
7         return self.__maxSpeed
8
9     def setMaxSpeed(self, newSpeed):
10        self.__maxSpeed = newSpeed
11
12    def Print(self):
13        print(self.color, self.__maxSpeed)
14
15 class car(vehicle):
16     def __init__(self, color, maxSpeed, numGears, isConvertible):
17         super().__init__(color, maxSpeed)
18         self.numGears = numGears
19         self.isConvertible = isConvertible
20
21     def Print(self):
22         super().Print() #self.Print() will also work as this function is inherited
23         print(self.numGears, self.isConvertible)
24
```

In [29]:

```
1 c1 = car('Red', 120, 4, False)
2 c1.Print()
```

Red 120

4 False

## Protected Members

Protected members are just public members but a underscore( `_` ) indicates that it is inherited from a parent class we should not access or modify this.

```
In [33]: 1 class vehicle:
2         def __init__(self, color, maxSpeed):
3             self.color = color
4             self._maxSpeed = maxSpeed
5
6         def getMaxSpeed(self):
7             return self._maxSpeed
8
9         def setMaxSpeed(self, newSpeed):
10            self._maxSpeed = newSpeed
11
12        def Print(self):
13            print(self.color, self._maxSpeed)
14
15    class car(vehicle):
16        def __init__(self, color, maxSpeed, numGears, isConvertible):
17            super().__init__(color, maxSpeed)
18            self.numGears = numGears
19            self.isConvertible = isConvertible
20
21        def Print(self):
22            super().Print() #self.Print() will also work as this function is inherited
23            print(self.numGears, self.isConvertible)
24
```

```
In [34]: 1 c1 = car('Red', 120, 4, False)
2
```

```
In [35]: 1 c1._maxSpeed
```

Out[35]: 120

## Class "object"

Every class inherit from **object** class.

**object** class have 3 methods.

1. --new--
2. --init--
3. --str--

```
In [36]: 1 class Circle:
2         def __init__(self, radius):
3             self.radius = radius
```

```
In [37]: 1 c = Circle()
2         print(c)
```

```
<__main__.Circle object at 0x00000214500627F0>
```

**--str--** function prints the location of the object. But we can override this function to print some useful information about the class.



```
In [38]: 1 class Circle:
2         def __init__(self,radius):
3             self.radius = radius
4
5         def __str__(self):
6             return "This is a class Circle which takes radius as an argument"
```

```
In [40]: 1 c = Circle()
2         print(c) # overriding the __str__() function and print the custom information.
```

This is a class Circle which takes radius as an argument

## Multiple Inheritance

```
In [41]: 1 class mother:
2         def print(self):
3             print("print of mother called")
4
5         class father:
6             def print(self):
7                 print("print of father called")
8
9         class child(mother, father):
10            def __init__(self,name):
11                self.name = name
12            def printChild(self):
13                print("name is", self.name)
```

```
In [42]: 1 a = child("Rohan")
2         a.printChild()
```

name is Rohan

```
In [43]: 1 a.print()
```

print of mother called

print of mother is called as we have written child(mother, father) is mother first.

**as we wrote father first the print of father is called**

In [52]:

```
1 class mother:
2     def __init__(self):
3         self.name = "Manju"
4
5
6     def print(self):
7         print("print of mother called")
8
9 class father:
10    def __init__(self):
11        self.name = "Ajay"
12
13    def print(self):
14        print("print of father called")
15
16 class child(father, mother):
17    def __init__(self):
18        super().__init__()
19
20
21    def printChild(self):
22        print("name is", self.name)
```

In [54]:

```
1 a = child()
2 a.printChild()
```

name is Ajay

## Method Resolution Order

This tells about the hierarchy order of the classes properties and methods for multiple parent classes.

In [2]:

```
1 class mother:
2     def __init__(self):
3         self.name = "Manju"
4
5
6     def print(self):
7         print("print of mother called")
8
9 class father:
10    def __init__(self):
11        self.name = "Ajay"
12
13    def print(self):
14        print("print of father called")
15
16 class child(father, mother):
17    def __init__(self):
18        super().__init__()
19
20
21    def print(self):
22        print("name is", self.name)
```

In [3]:

```
1 c = child()
2 c.print()
```

name is Ajay

```
In [5]: 1 # print the method resolution order
        2 print(child.mro())
```

```
[<class '__main__.child'>, <class '__main__.father'>, <class '__main__.mother'>, <class 'object'>]
```

**mro is child > father > mother > object class**

```
In [9]: 1 class father:
        2     def __init__(self):
        3         self.name = "Ajay"
        4         super().__init__()
        5
        6     def print(self):
        7         print("print of father called")
        8
        9
       10 class mother:
       11     def __init__(self):
       12         self.name = "Manju"
       13
       14
       15
       16     def print(self):
       17         print("print of mother called")
       18
       19 class child(father, mother):
       20     def __init__(self):
       21         super().__init__()
       22
       23
       24     def print(self):
       25         print("name is", self.name)
```

```
In [10]: 1 c = child()
        2 c.print()
```

name is Manju

**using super() we go to one superior class to the present class**

In this case as father was super class of child and mother is super class of father we used init() in the father class that changes the order to mother which is its super class.

father is super class of child and mother is super class of father.

## Operator Overloading

```
In [28]: 1 import math
2 class Point:
3     def __init__(self,x,y):
4         self.__x = x
5         self.__y = y
6     def __str__(self):
7         return "This point is at " + str(self.__x) + "," + str(self.__y)
8
9     def __add__(self, point_obj): # overloading + operator
10        return Point(self.__x + point_obj.__x , self.__y + point_obj.__y)
11
12    def __lt__(self, po): #overloading < operator
13        return math.sqrt(self.__x**2 + self.__y**2) < math.sqrt(po.__x**2 + po.__y**2)
14
```

```
In [29]: 1 p1 = Point(2,3)
2 p2 = Point(4,5)
3 print(p1+p2) # + operator overloading
```

This point is at 6,8

```
In [31]: 1 print(p2<p1)
```

False

## Abstract Classes

**Abstract Classes contains abstract methods**

Those methods which are declared but not implemented. They will be implemented by some other class.

```
In [14]: 1 from abc import ABC, abstractmethod
2 class automobile(ABC):
3
4     def __init__(self):
5         print("automobile created")
6
7
8     def start(self):
9         pass
10
11
12    def drive(self):
13        pass
14
15
16    def stop(self):
17        pass
```

```
In [15]: 1 c = automobile()
```

automobile created

```
In [16]: 1 from abc import ABC, abstractmethod
2 class automobile(ABC):
3
4     def __init__(self):
5         print("automobile created")
6
7     @abstractmethod
8     def start(self):
9         pass
10
11    @abstractmethod
12    def drive(self):
13        pass
14
15    @abstractmethod
16    def stop(self):
17        pass
```

```
In [17]: 1 c = automobile()
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-17-33fbd8d848ba> in <module>
----> 1 c = automobile()
```

**TypeError:** Can't instantiate abstract class automobile with abstract methods drive, start, stop

**can't create object of the class which have any abstract methods**

```
In [18]: 1 class Car(automobile):
2
3     def __init__(self,name):
4         print("car created")
5         self.name = name
6
```

```
In [19]: 1 a = Car("Honda")
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-19-5935755da3ab> in <module>
----> 1 a = Car("Honda")
```

**TypeError:** Can't instantiate abstract class Car with abstract methods drive, start, stop

**We can't initiate an object of Car without using the abstract methods of automobile class**

```
In [21]: 1 class Car(automobile):
2
3     def __init__(self,name):
4         print("car created")
5         self.name = name
6
7     def start(self):
8         pass
9
10    def stop(self):
11        pass
12
13    def drive(self):
14        pass
15
```

```
In [22]: 1 c = Car("Honda")
```

car created

### Important Points about Abstract Methods

1. Objects of abstract class can not be created.
2. Implement all the abstract methods in the child class.

```
In [28]: 1 from abc import ABC, abstractmethod
2 class automobile(ABC):
3
4     def __init__(self):
5         print("automobile created")
6
7     @abstractmethod
8     def start(self):
9         print("start of automobile called")
10        pass
11
12    @abstractmethod
13    def drive(self):
14        pass
15
16    @abstractmethod
17    def stop(self):
18        pass
19
20 class Car(automobile):
21
22     def __init__(self,name):
23         print("car created")
24         self.name = name
25
26     def start(self):
27         super().start()
28         print("start of car called")
29         pass
30
31     def stop(self):
32         pass
33
34     def drive(self):
35         pass
```

```
In [29]: 1 c = Car("Maruti")
```

car created

```
In [30]: 1 c.start()
```

start of automobile called  
start of car called

```
In [33]: 1 from abc import ABC, abstractmethod
2 class automobile(ABC):
3
4     def __init__(self, no_of_wheels):
5         self.wheels = no_of_wheels
6         print("automobile created")
7
8     @abstractmethod
9     def start(self):
10        print("start of automobile called")
11        pass
12
13    @abstractmethod
14    def drive(self):
15        pass
16
17    @abstractmethod
18    def stop(self):
19        pass
20
21    class Car(automobile):
22
23        # def __init__(self, name):
24        #     print("car created")
25        #     self.name = name
26
27        def start(self):
28            super().start()
29            print("start of car called")
30            pass
31
32        def stop(self):
33            pass
34
35        def drive(self):
36            pass
```

```
In [34]: 1 b = Car(4)
```

automobile created