Method Overriding

Redefining of base class instance method or object level method within derived class is called method overriding.
Defining method inside derived class with same name and arguments of method exists in base class is called method overriding.
Method is override in order extend or modify functionality of base class method within derived class.
Method is override in order provide different implementation of base class method within derived class.

Example:

```
class A:
    def m1(self):  # overriden method
        print("m1 of A")
    def m2(self):
        print("m2 of A")
class B(A):
     def m3(self):
         print("m3 of B")
     def m1(self): # overriding
         print("overriding method")
def main():
   objb=B()
   objb.m1()
   objb.m2()
   objb.m3()
main()
```

Output:

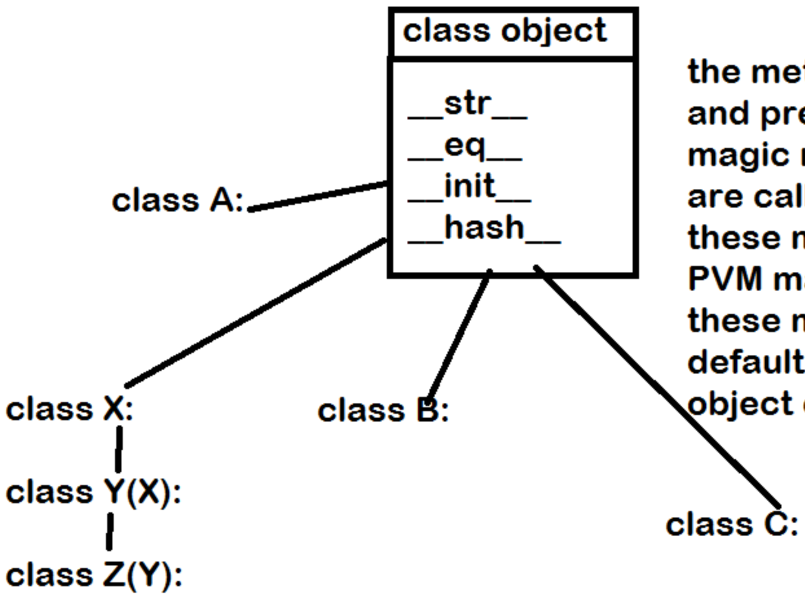
```
overriding method
m2 of A
m3 of B
>>>
```

```
class Parent:
    def eat(self):   # overriden method
        print("veg")

class Child(Parent):
    def eat(self):   #overriding method
        print("non veg and veg")
```

```
class Amithab:
    def acting(self): # overriden
        print("Very Good in acting")

class Abhishek(Amithab):
    def acting(self): # overriding
        print("Very bad in acting")
```

object class

Every class in python inherits properties and methods of object class
Every class in python is object.



the methods which are sufix and prefix with __ are called magic methods, these methods are called automatically.
these methods are used by PVM managing objects
these methods are having default implementation in object class.

__str__ : this method is called automatically whenever we print object. Object is printed in string format. All predefined class overrides __str__ method.

```
>>> class Point:
        def __init__(self):
                self.x=100
                self.y=200
```

```
>>> p1=Point()
>>> print(p1)
<__main__.Point object at 0x000000CC20282F40>
>>> l1=list()
>>> print(l1)
[]
>>> print(l1.__str__())
[]
>>> print(p1.__str__())
<__main__.Point object at 0x000000CC20282F40>
>>> class Point:
        def __init__(self):
                self.x=100
                self.y=200
        def __str__(self): # overriding method
                return f'x={self.x},y={self.y}'

>>> p1=Point()
>>> print(p1)
x=100,y=200
>>>
```

Example:

```
class Student:
    def __init__(self,r,n):
        self.__rollno=r
        self.__name=n
    def __str__(self):
        return f'rollno={self.__rollno},name={self.__name}'

def main():
    stud1=Student(1,"naresh")
    print(stud1) # stud1.__str__()
    print(stud1.__str__())
    comp1=complex(1.5,2.5)
    print(comp1)
    print(comp1.__str__())

main()
```

Output:
```
rollno=1,name=naresh
rollno=1,name=naresh
(1.5+2.5j)
(1.5+2.5j)
>>>
```

__eq__() : this method is called automatically when ever two objects are compared using == operator.

```
class Point:
    def __init__(self,x,y):
        self.__x=x
        self.__y=y
    def __eq__(self,other): # overriding method
        if self.__x==other.__x and self.__y==other.__y:
            return True
        else:
            return False

def main():
    p1=Point(100,200)
    p2=Point(100,200)
    x=p1==p2 # p1.__eq__(p2)
    print(x)

main()
```

Output:
```
True
>>>
```

Example:
```
class Employee:
    def __init__(self):
        self.__empno=None
        self.__ename=None
    def read_emp(self): # overriden method
```

```
        self.__empno=int(input("EmployeeNo:"))
        self.__ename=input("EmployeeName:")
    def print_emp(self):
        print(f'EmployeeNo {self.__empno}')
        print(f'EmployeeName {self.__ename}')

class SalariedEmployee(Employee):
        def __init__(self):
            super().__init__()
            self.__salary=None
        def read_emp(self): # overriding method
            super().read_emp() # calling method of super class
            self.__salary=float(input("Salary :"))
        def print_emp(self):
            super().print_emp()
            print(f'Salary :{self.__salary}')

def main():
    emp1=SalariedEmployee()
    emp1.read_emp()
    emp1.print_emp()
main()
```

Output:

```
 EmployeeNo:1
EmployeeName:naresh
Salary :90000
EmployeeNo 1
EmployeeName naresh
Salary :90000.0
>>>
```

Abstract classes and abstract methods

“abc” module of python provides classes and methods for working with abstract classes and methods.