Method Overriding (Polymorphism)

What is polymorphism?

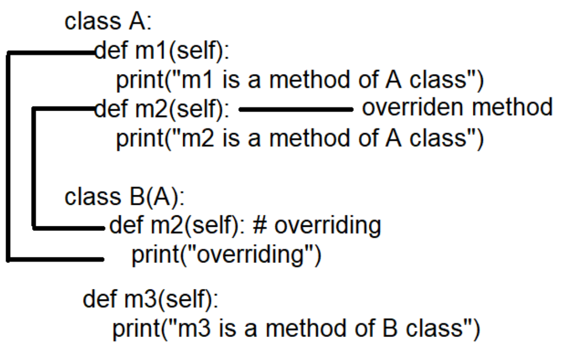
“poly” means “many” and “morphism” is “forms”, defining one thing in many forms is called polymorphism.

Python support two types of polymorphism

1. Method overriding
2. Operator overloading

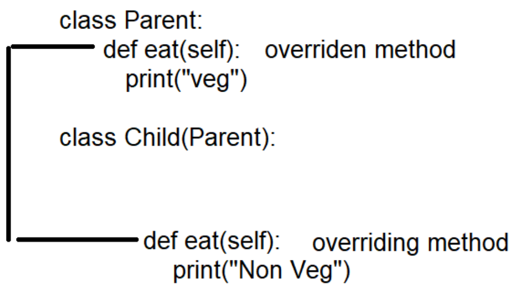
Method Overriding

Defining method inside derived class with same name and parameters exists in base class is called method overriding.
Method is override to modify or extend functionality of base class method within derived class.
If derived class wants to provide different implementation of method exists in base class is called method overriding.



```
class A:
    def m1(self):
      print("m1 is a method of A class")
    def m2(self):          overriden method
      print("m2 is a method of A class")

class B(A):
    def m2(self): # overriding
      print("overriding")
   def m3(self):
      print("m3 is a method of B class")
```



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

class Child(Parent):


        def eat(self):    overriding method
           print("Non Veg")
```

objjb=B()
objb.m1() -- m1 of A
objb.m2() -- ~~m2 of A~~ overriding
objb.m3() -- m3 of B

Example:
```
class Person:
    def __init__(self):
        self.__name=None
    def read(self): # Overriden method
        self.__name=input("Enter Name ")
    def print_info(self):
        print(f'Name {self.__name}')

class Customer(Person):
    def __init__(self):
        self.__creditLimit=None
    def read(self): # Overriding method
        super().read()
        self.__creditLimit=float(input("Enter Credit Limit"))
    def print_info(self): # Overriding method
        super().print_info()
        print(f'Credit Limit {self.__creditLimit}')


cust1=Customer()
cust1.read()
cust1.print_info()
```

Output:
```
Enter Name naresh
Enter Credit Limit50000
Name naresh
Credit Limit 50000.0
```

Object class

Every class in python automatically inherits object class.
Every class in python is object type.
Every class inherits the methods and properties of object class.
Methods of object class are used by PVM for managing objects.

Methods of object class are magic methods. These methods get executed automatically. Any method which is prefix and suffix with __ is called magic method.

How to find method of object class?
dir() is a predefined function, which return attributes of type(class). These attributes can be variables and methods.

```
a=dir(object)
print(a)
b=dir(list)
print(b)
c=dir(dict)
print(c)
```

Methods and Variables of object class

1. __class__
2. __delattr__
3. __dir__
4. __doc__
5. __eq__
6. __format__
7. __ge__
8. __getattribute__
9. __getstate__
10. __gt__
11. __hash__
12. __init__
13. __init_subclass__
14. __le__
15. __lt__
16. __ne__
17. __new__

18. __reduce__
19. __reduce_ex__
20. __repr__
21. __setattr__
22. __sizeof__
23. __str__
24. __subclasshook__

Example

```
class A:
    pass


x=dir(A)
print(x)
```

Output

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
```

Example

```
class A:
    pass

class B(A):
    pass

class C:
    pass

obja=A()
b1=issubclass(A,(object))
```

```
print(b1)
b2=issubclass(B,(A))
print(b2)
b3=issubclass(B,(object))
print(b3)
b4=issubclass(B,(C))
print(b4)
```

Output

```
True
True
True
False
```

Example:

```
class Employee:
    '''This is Employee class or data type'''
    pass


def fun1():
    '''this is function1'''

print(Employee.__doc__)
print(list.__doc__)
print(set.__doc__)
print(print.__doc__)
print(fun1.__doc__)
```

Output

```
This is Employee class or data type
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
```

The argument must be an iterable if specified.
set() -> new empty set object
set(iterable) -> new set object

__str__() method of object class

This method represents string representation of object.
This method converts object into string type. This method is called when the following methods/functions invoked.

1. Print()
2. Str()

The __str__() method returns a human-readable, or informal, string representation of an object. This method is called by the built-in print() , str() , and format() functions. If you don't define a __str__() method for a class, then the built-in object implementation calls the __repr__() method instead.

Example:

```
class Employee:
    def __init__(self,eno,en):
        self.__empno=eno
        self.__ename=en
    def __str__(self):
        return f'{self.__empno},{self.__ename}'




emp1=Employee(101,"naresh")
print(emp1) # emp1.__str__
```

```
comp1=complex(1.5,1.7)
print(comp1)
```

Output

```
101,naresh
(1.5+1.7j)
```

__repr__() method of object class

The __repr__() method returns a more information-rich, or official, string representation of an object. This method is called by the built-in repr() function.

Example:

```
class Employee:
    def __init__(self,eno,en):
        self.__empno=eno
        self.__ename=en
    def __str__(self):
        return f'{self.__empno},{self.__ename}'
    def __repr__(self):
        return f'{self.__class__},{self.__empno},{self.__ename}'



emp1=Employee(101,"naresh")
print(emp1) # emp1.__str__
comp1=complex(1.5,1.7)
print(comp1)
print(repr(emp1))
```

Output

```
101,naresh
(1.5+1.7j)
<class '__main__.Employee'>,101,naresh
```

In general, the `__str__()` string is intended for users and the `__repr__()` string is intended for developers.