Polymorphism
"Poly" means many and "morphism" is forms, defining one thing in many forms is called polymorphism.

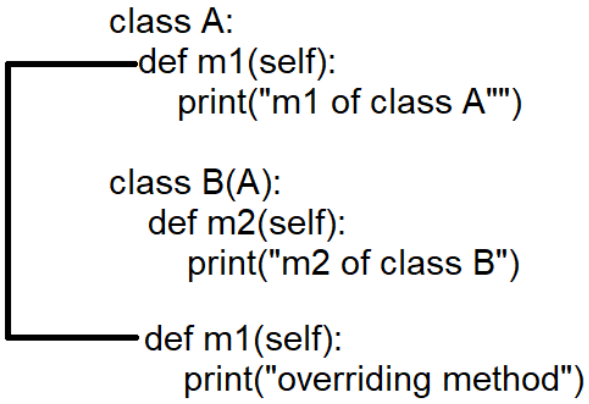
In python polymorphism is implemented using,

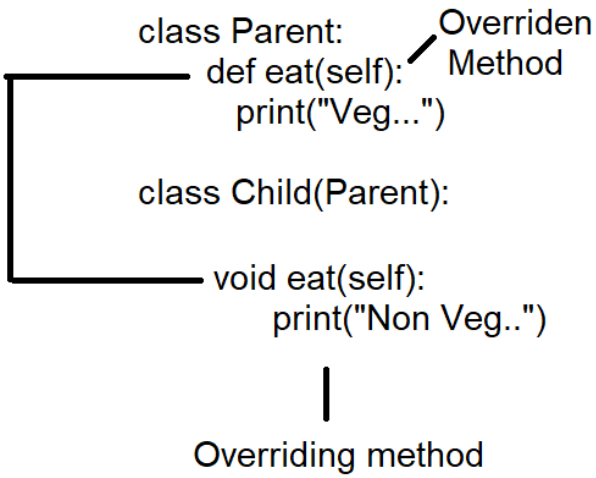
1. Method Overriding
2. Abstract classes and abstract methods

Advantage of polymorphism is reusability and extensibility.

Method Overriding

Defining instance method in derived class with same name and number of parameters/arguments of method exists in base class is called method overriding.
Method overriding allows providing different implementation of base class method within derived class (OR) it allows modifying or extending functionality of base class method within derived class.





Example:

```
class A:
    def m1(self):
        print("m1 of A class")
    def m2(self):
        print("m2 of A class")
class B(A):
```

```
    def m3(self):
        print("m3 of B class")
    def m1(self):
        super().m1()
        print("Overriding method")



objb=B()
objb.m1()
objb.m2()
objb.m3()
```

Output:

```
m1 of A class
Overriding method
m2 of A class
m3 of B class
```

Example:

```
class Employee:
    def __init__(self):
        self.__ename=None
        self.__job=None
    def read(self):
        self.__ename=input("Enter Employee Name ")
        self.__job=input("Enter Employee Job")
    def print_info(self):
        print(f'EmployeeName {self.__ename}')
        print(f'EmployeeJob {self.__job}')

class SalariedEmployee(Employee):
    def __init__(self):
        super().__init__()
        self.__salary=None
    def read(self):
        super().read()
        self.__salary=float(input("Enter Employee Salary "))
```

```
    def print_info(self):
        super().print_info()
        print(f'Employee Salary {self.__salary}')

emp1=SalariedEmployee()
emp1.read()
emp1.print_info()
```

Output:

```
Enter Employee Name naresh
Enter Employee Jobmanager
Enter Employee Salary 50000
EmployeeName naresh
EmployeeJob manager
Employee Salary 50000.0
```

object

object is a base class or parent class for all class in python.
Predefined or user defined classes are inherited from object class.
In python objects are managed by PVM (Python Virtual Machine).
PVM manage objects using methods of object class.
The methods exists within object class are magic methods. A method which is prefix and suffix with __ is called magic method. This method is executed by PVM automatically.

__str__(self)

__str__ method of object class return string representation of object.
This method is called by PVM, whenever programmer print variable.

```
class A:
    '''This is smaple class'''
    def __init__(self):
        self.x=100
        self.y=200
    def __str__(self):
        return f'{self.x},{self.y}'
```

```
obja=A()
print(obja) # PVM calls __str__ method
print(obja.__str__()) # Explicit

list1=list(range(10,60,10))
print(list1)
```

Output:

```
100,200
100,200
[10, 20, 30, 40, 50]
```

Example:

```
class Integer:
    def __init__(self):
        self.__value=0
    def __str__(self):
        return f'{self.__value}'

a=Integer()
print(a)
b=int()
print(b)
```

Output:

```
0
0
```

Example:

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

studList=[]
```

```
studList.append(Student(101,"naresh"))
studList.append(Student(102,"suresh"))
studList.append(Student(103,"kishore"))
for s in studList:
    print(s)
```

Output:

```
101,naresh
102,suresh
103,kishore
```

Example:

```
class Employee:
    def __init__(self,e,n):
        self.__empno=e
        self.__ename=n
    def __str__(self):
        return f'{self.__empno},{self.__ename}'
    def __eq__(self, other):
        return self.__empno==other.__empno
    def __hash__(self):
        return self.__empno

A={10,20,30,40,50,10,20,30,40,50}
print(A)
B=set()
B.add(Employee(101,"naresh"))
B.add(Employee(102,"suresh"))
B.add(Employee(103,"kishore"))
B.add(Employee(101,"ramesh"))
for e in B:
    print(e)
```

Output:

```
{50, 20, 40, 10, 30}
101,naresh
102,suresh
103,kishore
```

Abstract classes and abstract methods (abc module)
This module provides the infrastructure for defining abstract base classes (ABCs) in Python. It is a default module which comes with python software.