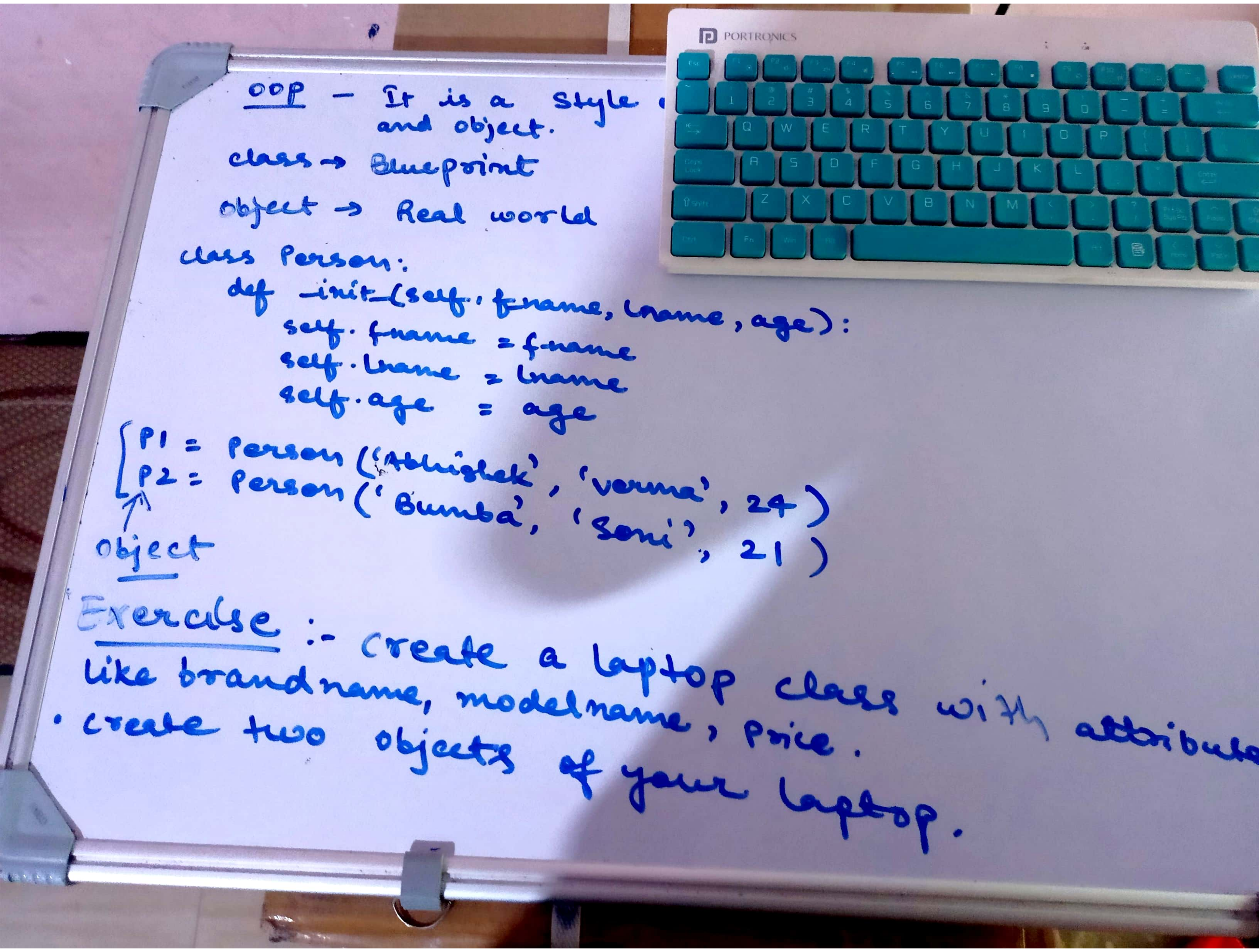


OOP - It is a style and object.

class → Blueprint

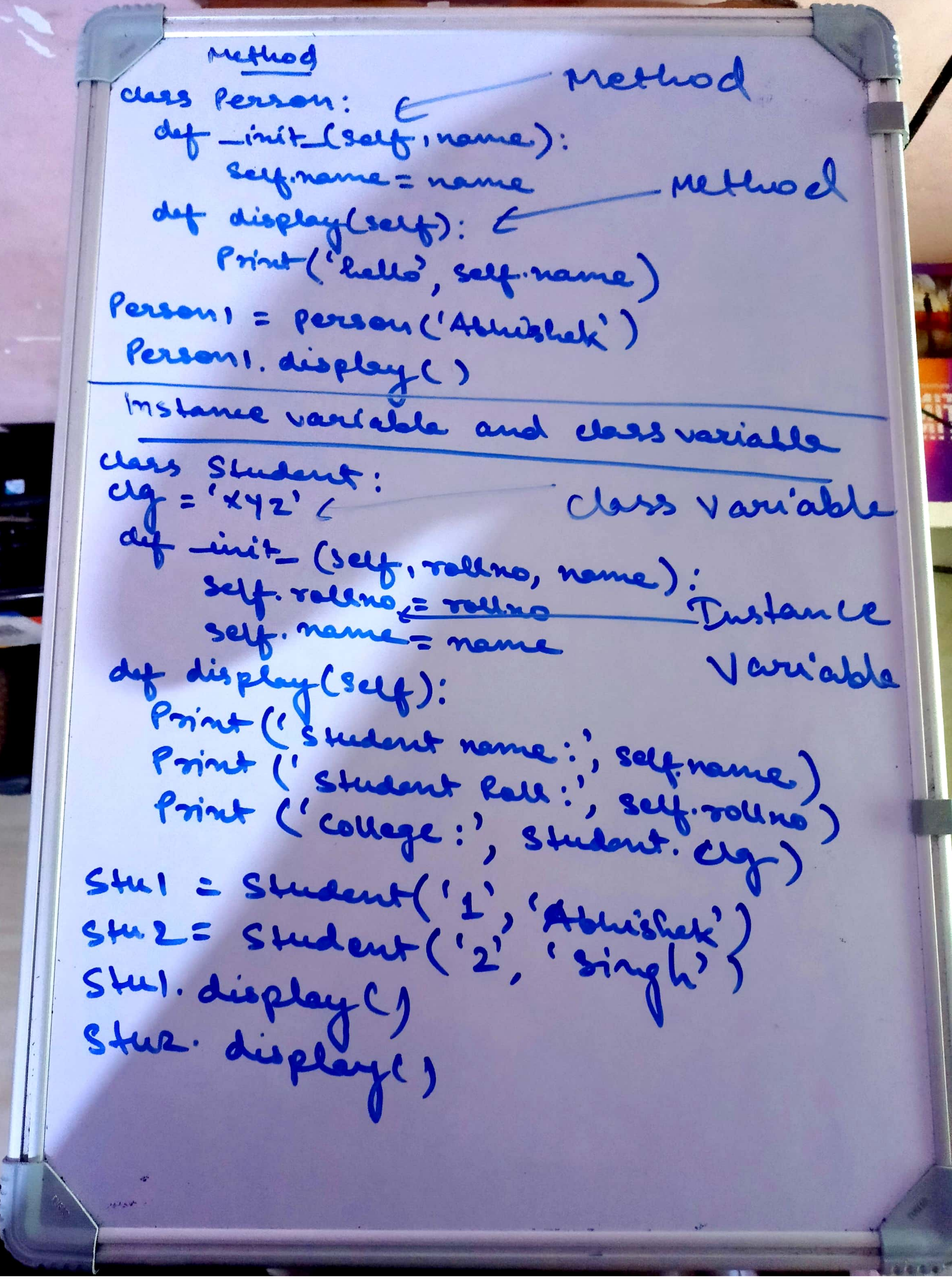
object → Real world

```
class Person:
    def _init_(self, fname, lname, age):
        self.fname = fname
        self.lname = lname
        self.age = age
P1 = Person('Abhishek', 'verma', 24)
P2 = Person('Bumba', 'Soni', 21)
```

object

Exercise :- create a laptop class with attribute like brandname, modelname, price.

- create two objects of your laptop.



Method

```
class Person:                          Method
  def _init_(self,name):
     self.name = name
  def display(self):                   Method
     Print('hello', self.name)
Person1 = person('Abhishek')
 Person1. display()
```

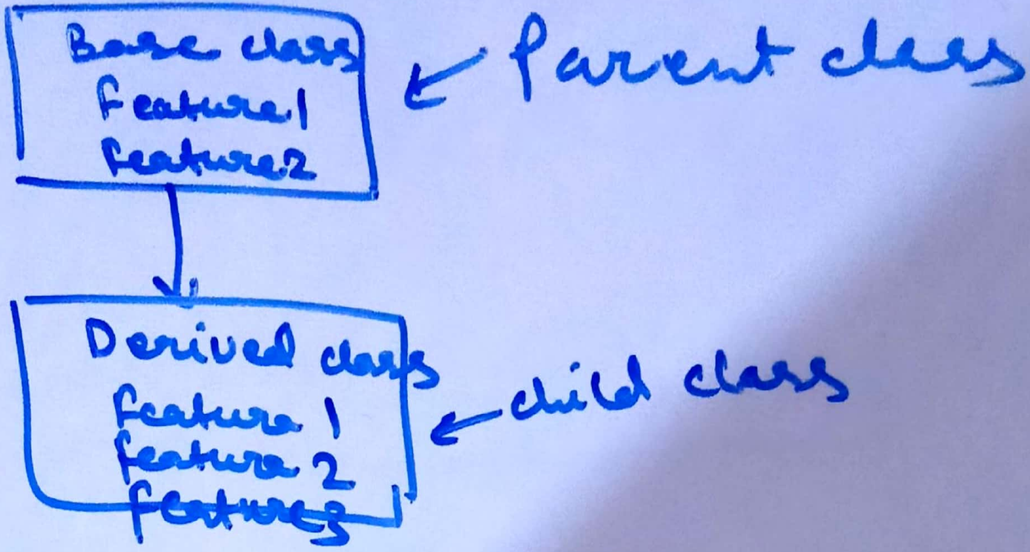
Instance variable and class variable

```
class Student:
clg = 'XYZ'                            class variable
 def _init_ (self, rollno, name):
     self.rollno = rollno              Instance
     self.name = name                  Variable
def display(self):
  Print('student name:', self.name)
  Print ('student Roll:', self.rollno)
  Print ('college:', Student.clg)
Stu1 = Student('1', 'Abhishek')
Stu2= Student('2', 'singh')
Stu1. display()
Stu2. display()
```

features of oop

Single

① Inheritance


Base class
feature1
feature2

Parent class

Derived class
feature 1
feature 2
feature3

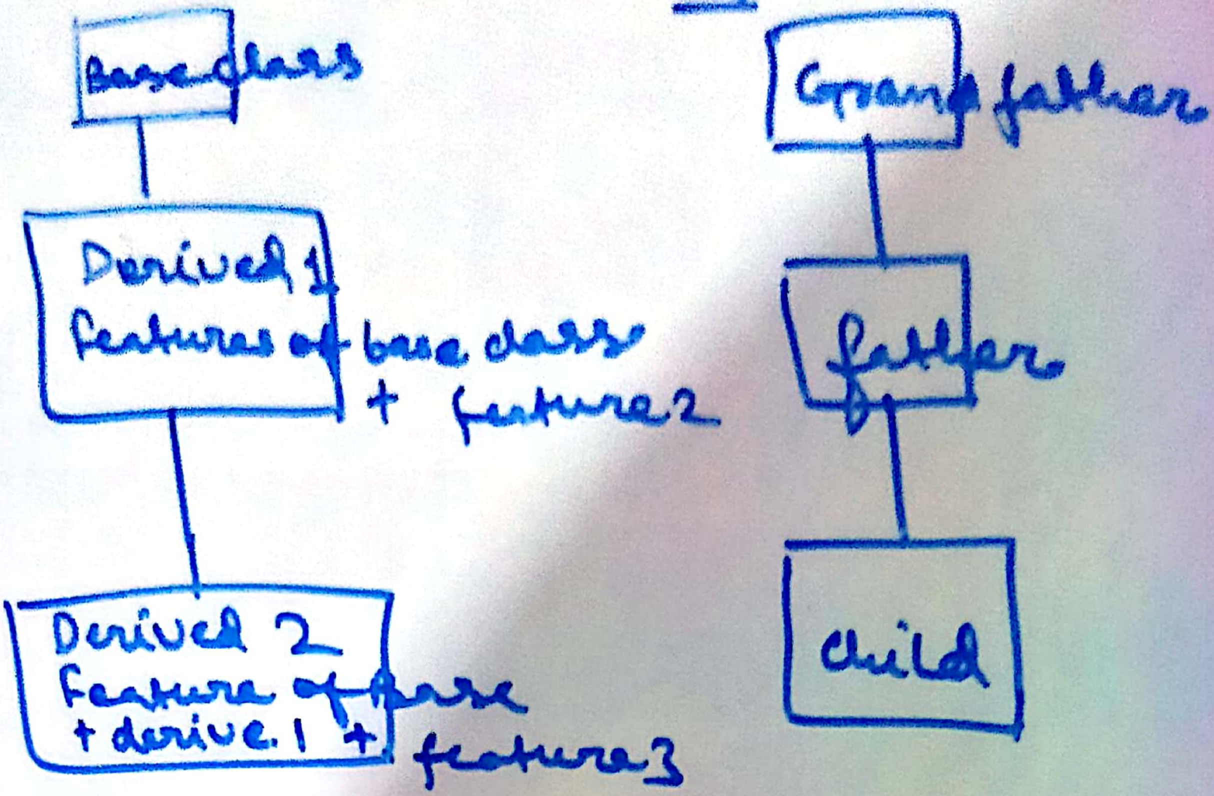
child class

inheritance helps us to reuse the Parent features,

```
class animal:
  def eating(self):
    Print('eating')
class dog(animal):        Remove
                          animal to
  def bark (self):        see
    Print('bark')
 d = dog
 d.eating()
 d.bark()
```

Multi level Inheritance

ex

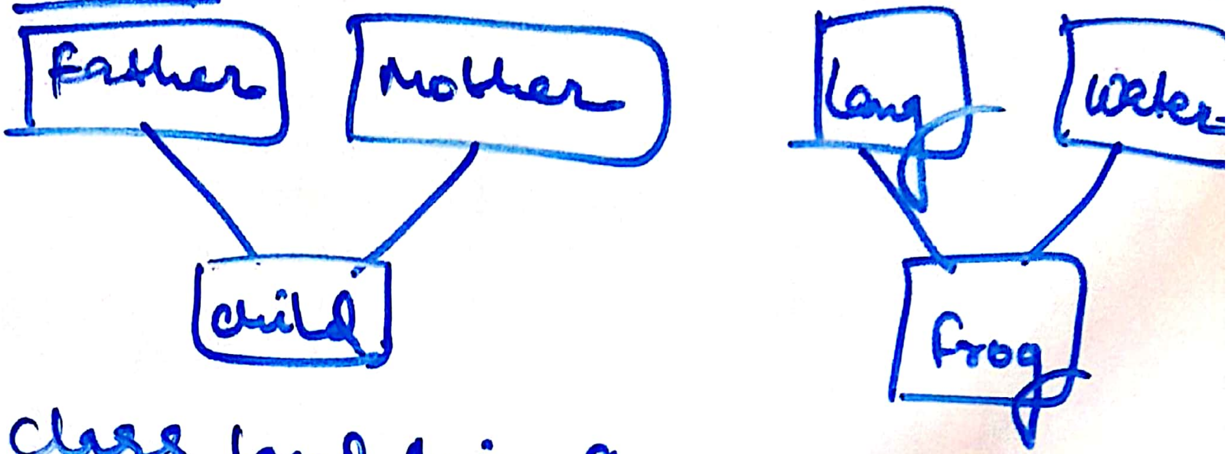


```
class Person:
  def display(self):
     print('hello, this is class person')
class Employee(person):
    def Printing(self):
        Print("helo, this is derived class employee")
class programmer(Employee):
   def show(self):
      Print ('hello, this is 2nd derived class programmer')

P1 = Programmer()
P1.display()
P1. Printing()
P1. show()
```

Multiple inheritance

example



```
class land_Animal:
    def printing(self):
        print('this animal lives on land'
class water_animal:
    def display(self):
        print('this animal lives on wat

class frog(land_Animal, water_animal
    Pass

f1 = frog()
f1.printing()
f1.display()
```

Method overriding

```
classA:
 def display(self):
   Print('method belongs to A')

class B(A):
  def display(Self):
    Print('method B')

b1 = B()

b1.display().
```

Encapsulation :- protection of data methods we use Encapsulation.

```
class car:
  def __init__(self):
    self.__updateSoftware()
  def printt(self)
     Print('Driving')
  def __update.software():
       Print('Anthing')

P = car()
P.printt() ✓
P.__update.software ✗
```

we can't call Private func^n outside of class

Encapsulation

```
class car:
 __max_speed =0
 __name = ""
  def __init__(self):
     self.__max_speed = 200
     self.__name = 'supercar'
  def drive(self):
    Print('driving')
    Print(self.__max speed)
  def setspeed(self, speed):
      self.__maxspeed = speed
      Print(self.__max_speed )

s = car()
s. drive()  ——> 200
s. setspeed(100) ——> 100
s. __max_speed =50 X → 100
```

Encapsulation

```
class car:
 __max_speed =0
 __name =""
 def __init__(self):
   self.__max_speed = 200
   self.__name = 'supercar'
 def drive(self):
  Print('driving')
  Print(self.__maxspeed)
 def setspeed(self, speed):
    self.__maxspeed = speed
    Print(self.__max_speed )
s = car()
s.drive()          → 200
s.setspeed(100) → 100
s.__maxspeed =50 X → 100
```

Polymorphisms
↓ Many
↓ forms

① Method overridin
② method overloading