

# OOP

## Object Oriented Programming ↓ based on

1 → Inheritance

2 → encapsulation

3 → polymorphism

4 → Abstraction

Pillars of  
OOP

Syntax → `class class_name:`  
                  member of class

ex → `class Person:`  
          `def greet(self):`  
              `print("Hello")`

`p = Person()` # creating object of class  
`p.greet()` # calling the function from class

```
class Math:
    def add(a,b):
        return a+b
    def square(a,b):
        return a**b
```

```
ob=Math()
sum=ob.add(3,4)
print(sum)
```

```
print(ob.square(6,7))
```

Instance variable  
self.name  
self.age

```
class Person:
    # constructor
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def show(self):
        print(self.name)
        print(self.age)

p = Person('Anup', 26)
p.show()
```

Anup

local variable

local variable  
↓  
name age

we are  
storing  
local variable  
value in instance variable

Inheritance

Parent → child

Super class → Sub class

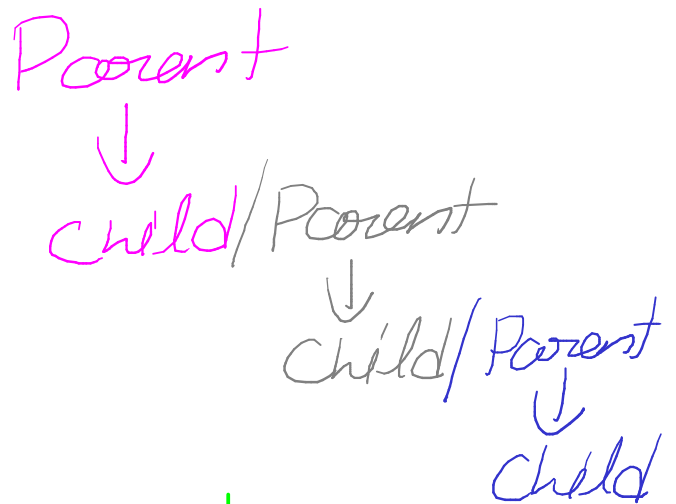
Base class → derived class

class Animal: → Parent class  
def eat(self):  
 print('I cant')

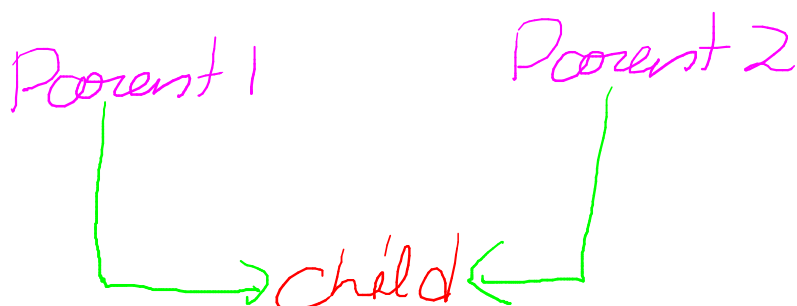
class Dog(Animal): → child class  
def bark(self):  
 print('I can bark')

d = Dog()  
d.eat() # inherited Method  
d.bark()

## Multiple



## Multiple inheritance



class A: class B:  
↓ ↓  
class C(A,B):

# Access Modifiers

public → name

⇒ Accessible everywhere

Protected → \_name

⇒ Accessible inside class & sub class

Private → \_\_name

⇒ Accessible inside class only

## ex of Access modifiers

Public  
↓

class demo:

def \_\_init\_\_(self):

self.name = "Anup"

d = Demo()  
print(d.name)

Protected  
↓

class demo:

def \_\_init\_\_(self):  
self.\_name = "Anup"

class subdemo(demo):

def show(self):  
print(self.\_name)

s = subdemo()  
s.show()

\_\_name



get

set method

Create a class BankAccount that represents a user's bank account. It should:

- Have private attributes: \_\_account\_number, \_\_balance
- Have:
- A method deposit(amount) to add money
- A method withdraw(amount) to deduct money (only if balance is enough)
- A method get\_balance() to return current balance

✓ You should use encapsulation to protect the balance from direct modification.

Abstraction →

Abstraction means hiding the internal details and showing only the essential features to the user

#### ◆ How to Use Abstraction in Python

1. Use the abc module
2. Use @abstractmethod decorator
3. Cannot create object of abstract class
4. Child class must implement all abstract methods

try:

# code that might raise an exception

except:

# code that runs if an exception occurs