2. Multilevel inheritance
Child → Parent → Grandparent

```
Class A:
    def greet(self):
        Print("Hello")
Class B(A):
    pass
class C(B)
    Pass.

D = C()
D.greet() → Hello
```

3 Multiple inheritance
Parent 1 > Child
Parent 2

```
Class A:
    def greet(self):
        Print("Hello")
Class B:
    def name(self):
        Print("Ayush")

Class C(A,B):
    Pass

d = C()
d.greet() → Hello
d.name() → Ayush
```

4. Hierarchical inheritance
Parent → Child 1
→ Child 2

```
Class A:
    def greet(self):
        Print("Hello")

Class B(A):
    Pass
Class C(A):
    Pass.
B().greet() → Hello
C().greet() → Hello
```

5. Hybrid inheritance

Mix of two or more type we seen above.

like Multiple + multi-level

Date
Page

Super Keyword:

It is use to Set Parent Constructor to child Constructor.

```
Class Child:
    def __init__(self, breed):
        super().__init__()
        self.breed = breed
```

→ Now we can attribute of Parent Constructor.

1. Super can only be used inside class
2. Used only inside child class
3. ~~He cannot access attribute only constructure~~

Type of Inheritance

→ Single inheritance → One child inherits from one parent.

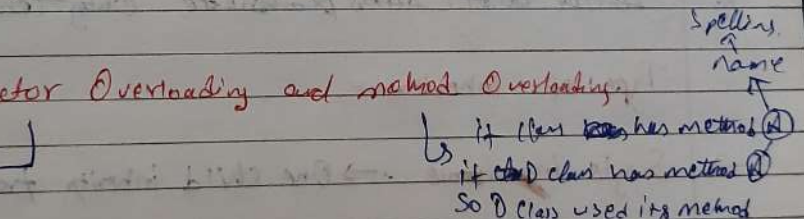
```
Class A:
    def greet(self):
        Print("hello")

Class B(A):
    Pass

c = B()
c.greet() → hello
```

* Inheritance → Code in my git hub.

→ Why inheritance is need?
→ Code reusability.
→ Child has access to parent but Parent has no access to child.

What Can be ~~inherit~~ inherit
→ Constructor
→ Non-Private Attribute / Method
...

* Constructor Overloading and method Overloading.



~~if case~~
→ We Made a Class A
→ Made a Class B and herited class A
→ Case 1: ~~If~~ class B has no Constructure So it will use Parent Class Constructor.
→ Case 2: Class B has Constructure now Parent one will be nulify and class B will its own Constructor → it is Called Constructor Overloading.

* Static Method

No need to ~~get~~ give self, cls

let count user_id mean how many user we created

```
class employee:
    __user_id = 0
    def __init__(self):
        employee.__userid += 1

    @staticmethod
    def get_usid():
        return employee.__userid

Ayush = employee()
Print(Ayush.get_usid()) → by object Access through getter

employee.__userid → directly access
```

* ~~Static~~ Inst

FREEMIND

Date

Page

→ We can also create attribute outside a class.

~~Rajesh Sethi~~ obj.name = 'Ayush'

** Encapsulation, getter and setter, static method

→ Encapsulation

Hiding the internal state of an object and only allowing access via method

Eg:- Normal Attribute.

emp(id) —Can be access→ obj.id

Protected/Private Attribute

emp(--name) —Can't be access→ obj.--name

↳ to access → obj._ClassName__name

Alert: So as you can see ~~Attribute~~ are not fully Private, we can access them.

In Python it's not 100% Private because python thinks you are adult and will not open Private attribute.

→ Getter and Setter

Used to read (get) and update (Set) private attribute

```
def get_name(self):        } Getter
    return self.__name
```

~~def get Set_name(self):~~

```
def set_name(self, value):    } Setter
    self.__name = value
```

Date
Page

Advantages of OOPs:

→ We can create our own datatypes.
→ Code reusability
→ Easy to collab
→ debugging.

→ Made a Mini-Project from OOPs
↳

→ Method Vs function.
↓ ↳ Normal Block of code
Of function that eg:- len().
is called using
an object.
Eg obj.method()

→ dunder / Magic Method
* Starts and ends with double Underscore
* A __init__ constructor get auto-called when Class runs.

→ Self
if we try to print Memory address of Self (id(self)) and our obj (id(obj)) they are at same memory address not possible, this implies that Self is nothing but the object and in the function with Parameter Self, here only object is allowed to access because Self is object itself.

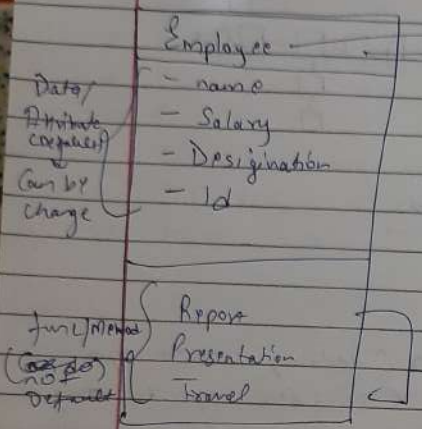
Quote ↳ Hey Python, whenever I talk about Self, I mean the exact obj. being invoked

FREEMIND

Date

Page

OOPs



1. Once class is called attribute automatically are Executed.

2. what is use of __init__
   ↳ It gets automatically Executed once the class is called.
   ↳ Eg → in Production, User don't manually connect the data with Cloud, the Constructor auto do this.

Que: Why is Python a OOPs language?

Ans: Every datatype is made up to class you can print their type.

Everything in Python is object.