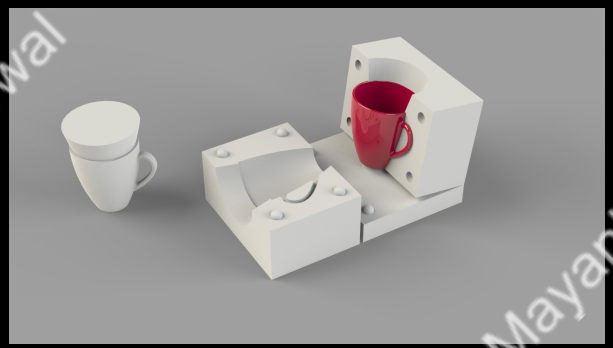
OOPS in Python

Object oriented Programming System

1. Procedural/ Functional approach is not a lot helpful for real world use cases.

let us understand via example :

It is always helpful if we could create a blueprint of what we want and then create multiple copies via it.



So, OOPS help us in doing below things.

1. Define our own data type
2. Generality to Specificity

OOPS is thus a programming paradigm which help us to write better code.

Which is the most complex machine?

Humans

OOPS

1. Class
2. Object
3. Encapsulation
4. Inheritance
5. Polymorphism
6. Abstraction

More focus on practical implementation so no issues are faced.

Class

1. Blueprint
2. Self-defined datatype

Let us see some example of known datatype.

All data types are built in classes in python.

Class

data / attributes / property

Behaviour / method

Color
brand
name
topspeed

accelerate()
brake()
start()

Syntax :-

```
Class <ClassName>:
    ____
    ____
    ____
    ____
    ____
```

Object

An Object is an instance of our class

'Everything in Python is an object'

Let us create object of our class.

Python gives us 'Object literal' for built in types.

Creating our first Class

Class Student :

data members.

- name
- roll no.
- percentage

methods

- study()
- play()

Constructor & self

__ init __

Whenever we create/construct an object of a class, there is an inbuilt method (__init__) which is called known as constructor.

Special method called itself.

Advantage / Application of Consructor :-

Background check in companies

Internet check for apps

Configuration reloted checks before starting.

It's like a receptionist which checks and add / gives all information required.

Self

→ Self is the reference to the object created of our class.

→ It is passed whenever we are calling any ~~function~~ method.

→ only an object can access or helpful in connecting Attributes and methods inside our class.

Let us see few examples :-

Complex Number Class :

3 + 5

2 + 3 + 5

5 + 5

__add__ (self, other)

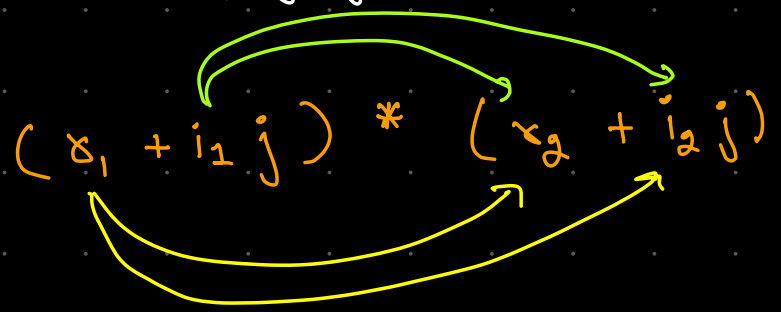
c1 + c2

$c_1 = 3 + 4i$

$c_2 = 4 \pm 5i$

$7 + 9i$

Multiplying Complex Numbers



$j = \sqrt{-1}$

$j^2 = -1$

$$= x_1 x_2 + x_1 i_2 j + x_2 i_1 j + i_1 i_2 (j)^2$$

$$= (x_1 x_2 - i_1 i_2) + j(x_1 i_2 + x_2 i_1)$$

$$(a + ib)(c + id) = (ac - bd) + i(ad + bc)$$

Division of Complex numbers

$$\frac{x_1 \pm i_1 j}{x_2 \pm i_2 j} * 1$$

$$= \frac{x_1 + i_1 j}{x_2 + i_2 j} * \frac{x_2 - i_2 j}{x_2 - i_2 j}$$

$$= \frac{(x_1 + i_1 j) * (x_2 - i_2 j)}{x_2^2 + i_2^2}$$

$$(a+ib)(a-ib) = a^2 + b^2 i^2$$

$$\frac{a+bi}{c+di} = \frac{(a+bi)(c-di)}{(c+di)(c-di)}$$

$$= \frac{(ac+bd)+(bc-ad)i}{c^2+d^2}$$

$$= \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2} i$$

Magic methods (dunder)

→ They are special methods in python which are defined inside class.

→ They begin and end with double underscore (__)

→ They have a special where in they get called themselves.

creation → __init__

print → __str__

+ → __add__

Instance Variable

```
s1 = Student ('Mayank', 29)
s2 = Student ('Ram', 25)
```

s1.name ——— s2.name ———

s1 and s2 above are instance variable which have diff. values of attribute.

Access modifier in python

We can chose to provide access to our variable by making them public or private.

Normally, we make all the data members private.

public

private

✯ Once we make our data member private using __marks, then everywhere we have to make sure to access them using __marks.

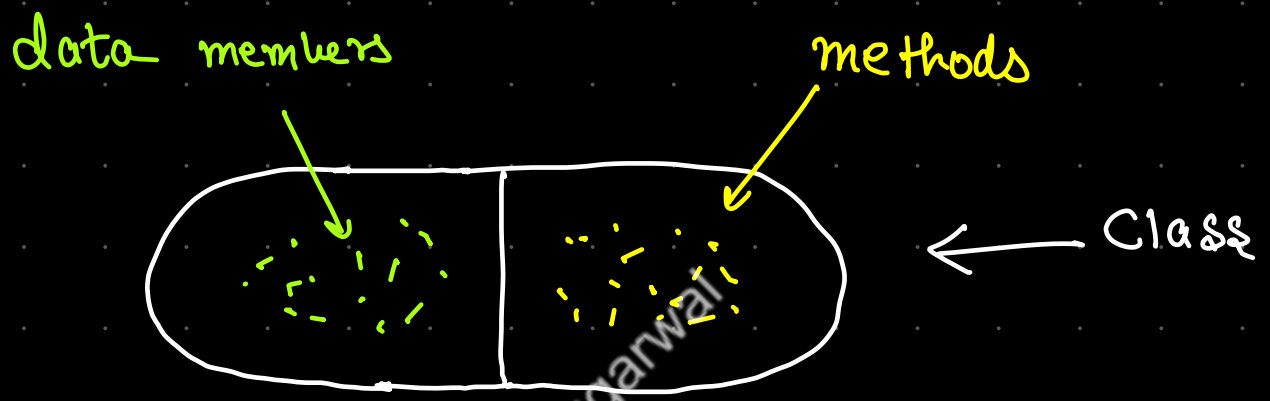
We can also make our methods private as well.

What if we actually need to access some data member or change their classes?

We make getter & setters for the same use case.

* This binding together of attributes and methods is very useful in achieving security and checks.

Encapsulation



The binding of our data members/attribute and methods in a single unit is known as **Encapsulation**.

Static Property / Variables.

For many use cases, we need static properties instead of instance.

→ They are defined outside methods at class level.
→ own by class majorly but accessible by objects

→ We can have static methods as well decorated with @ Static method.

Inheritance in OOPS

→ followed in nature

→ parent
↓
child

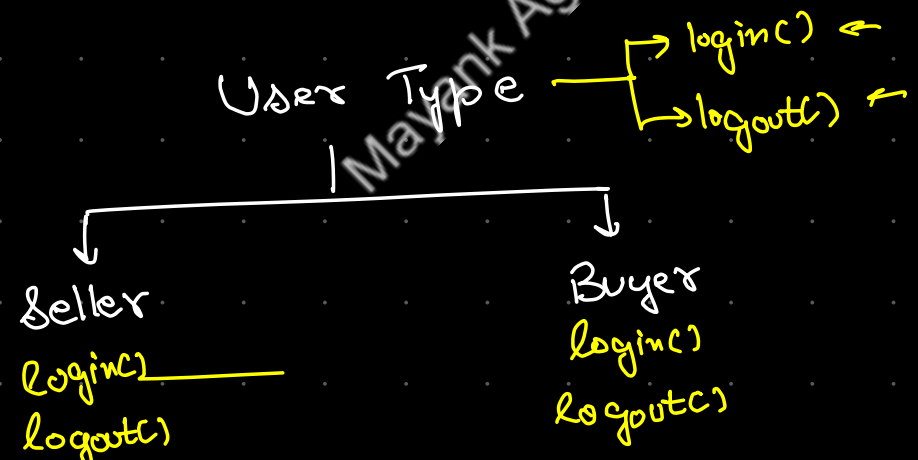
→ gives us major benefit of code reusability.

Student

- login()
- logout()
- study()

Teacher

- login()
- logout()
- teach()



We inherit

- → non-private attributes
- → non-private methods
- → Constructor [other magic methods]

→ We cannot access private attributes or methods of our parent class.

→ Parent cannot inherit from child but only child can inherit from parent.

Super Keyword

super() Keyword is used to access methods of parent from our child class.

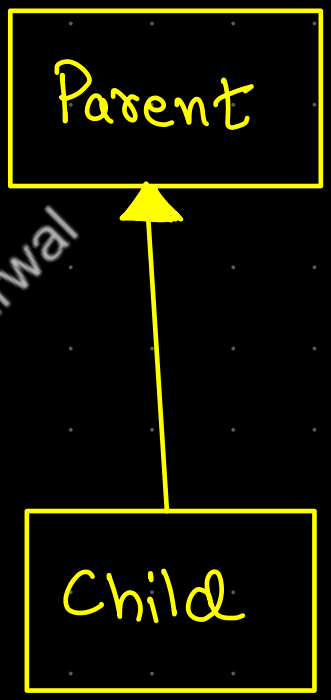
→ Cannot be used outside

→ Can only access methods and not attribute.

Types of Inheritance

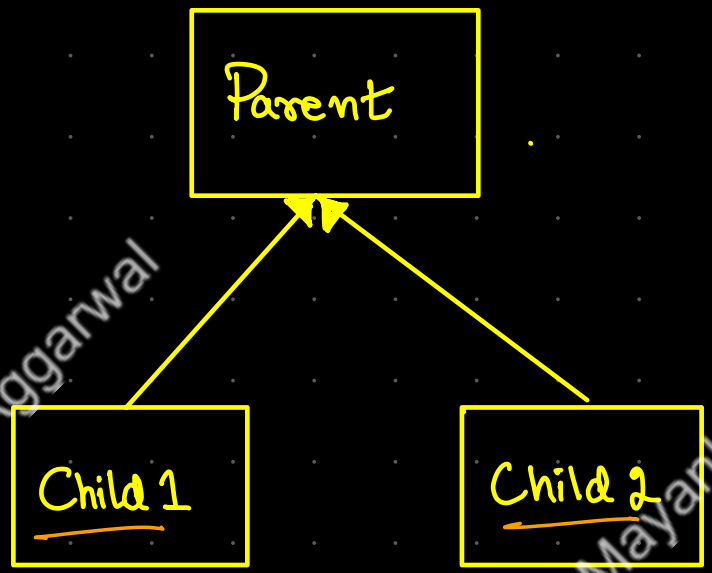
- Simple
- Heirarichal
- Multilevel
- Multiple
- Hybrid

Simple



→ non private data members and methods

Heirarchial

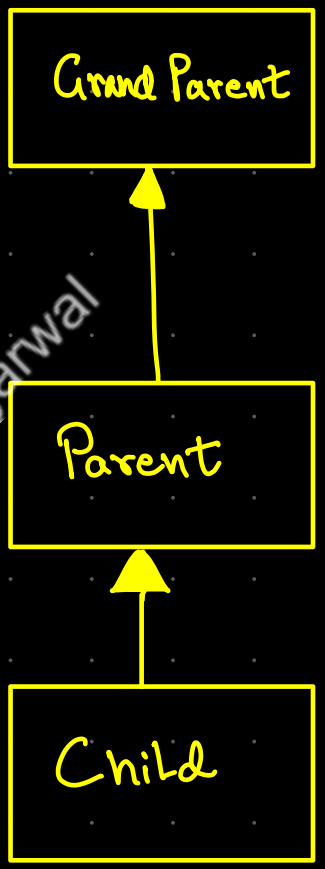


→ Both children get properties of parent

→ Children are not related to each other.

≡

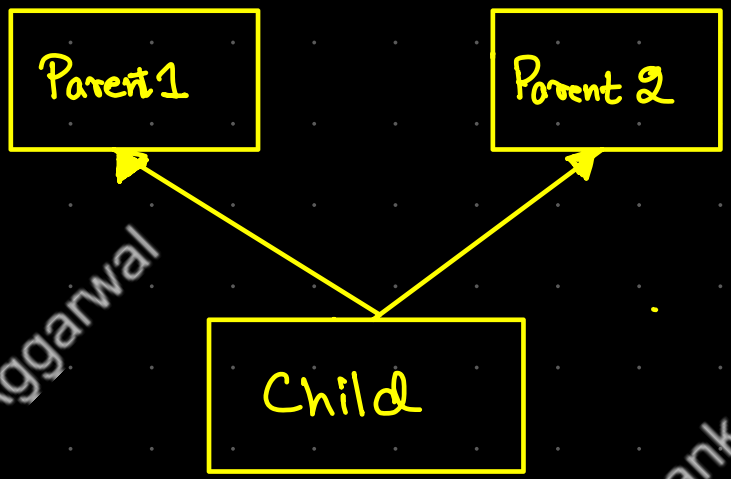
Multilevel



→ lower levels gets properties from above levels.

→ super() of each class can reach out to just the parent.

Multiple



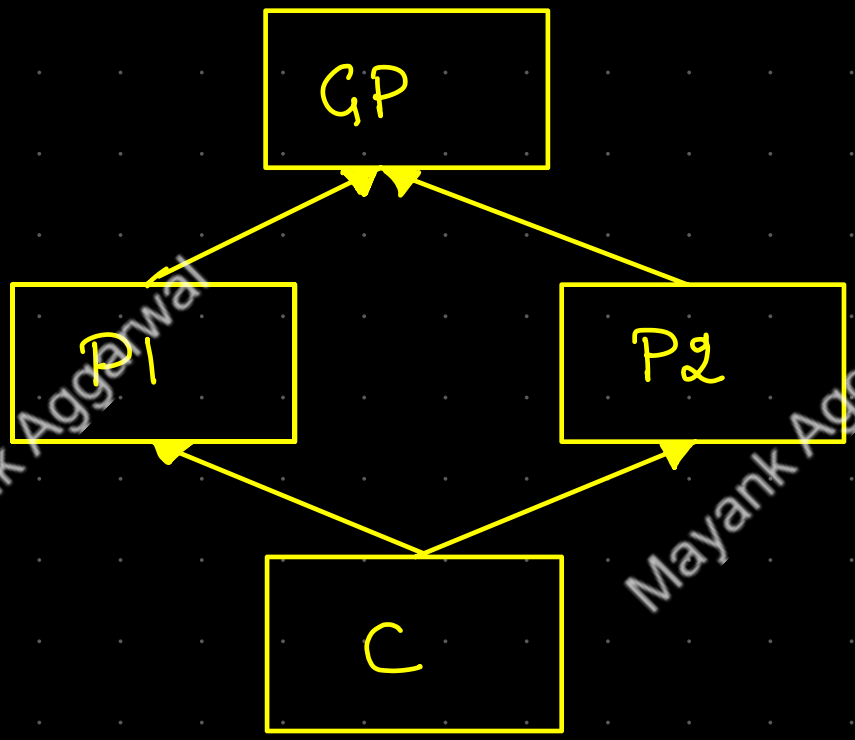
Class C (P1, P2):

→ not allowed in Java

→ ambiguity solved in the order of inheritance. Get properties of both parent

→ MRO (method resolution order) followed in resolving ambiguity.

Hybrid Inheritance



Diamond Problem

→ mix of inheritance types

Let's solve some Questions :

Now let us solve some questions to test our understanding of polymorphism.

⭐⭐ Please make sure to pause the video and answer first.

Polymorphism

poly → multiple / many

morph → shape / form

In simple words, polymorphism is when a single entity can take multiple form.

In python, we achieve polymorphism by below :-

1. Method Overriding
2. Method overloading
3. Operator overloading

→ major benefit is that code becomes a lot clean to read and work upon.

Abstraction in Python

Abstract ⟶ Abstract art

Abstract = hidden

fundamental concept of OOPS that involve hiding complex implementation details & showing only essential features.

eg. Phone

Similarly, we have a lot of examples in real life.

- mobile
- human body
- TV remote

Abstract Method & Class

```
from abc import ABC, abstractmethod
```

We have to ensure 2 things

1. Inherit abc class
2. Should have an abstract method.

```
Class Animal (abc):
    @ abstract method
    def sound (self):
        pass
```

```
Class Dog (Animal):
    def sound (self):
        'Bark'
```

```
class Cat (Animal):
    def sund (self):
        'meow'
```

So abstract classes are useful when we have a group of related objects that should share a common features

but should / will have different implementation.

⇒ Help in preparing a blueprint for other classes.

Now, let us understand the same using a real life object as well.

A User in an app.