CLASS & OOP in PYTHON

1. CLASS :-

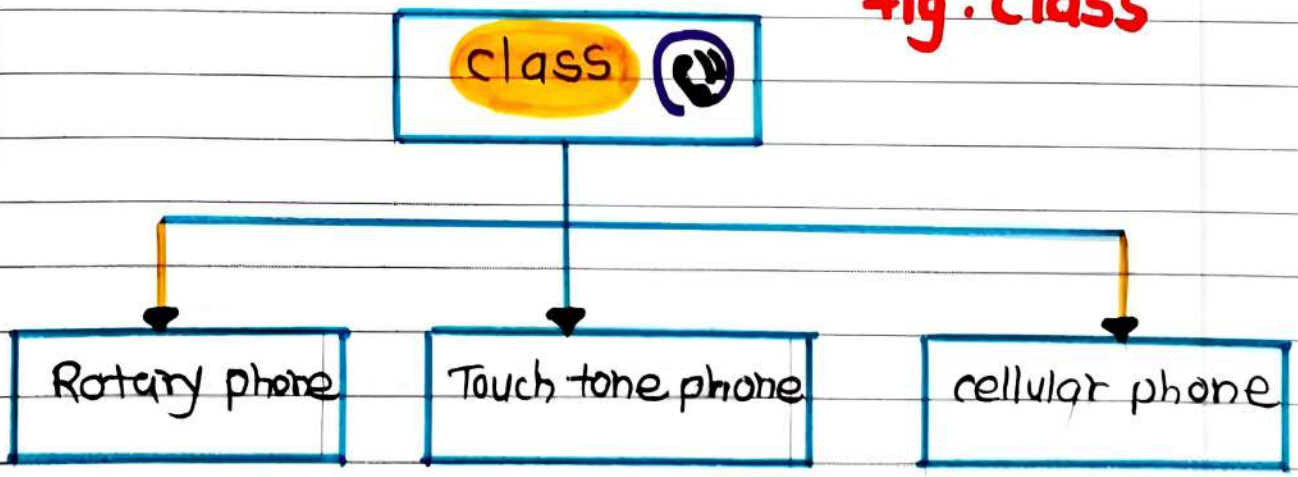
A class is like an object constructor, or a blueprint for creating objects.

TO create a class:

to create a class, use the keyword class:

```
Class Myclass :
   x = 5 ;
```

fig : Class



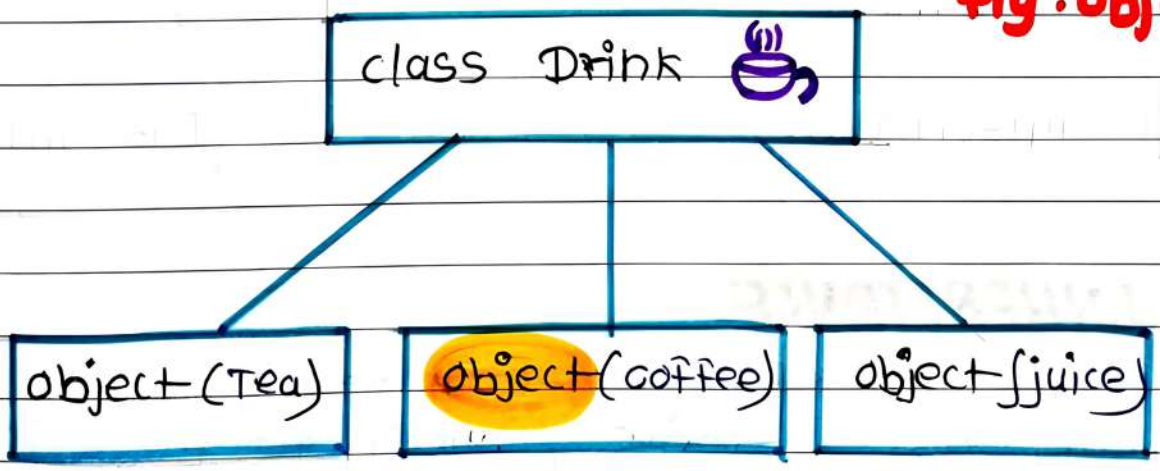
2. OBJECT :-

An object is an instance of a class. A class is an blueprint while an instance is a copy of the class with actual values.

to create an object :-

```
class Cars:                                    ← class variable
    def __init__(self, m, p):                 ← init method.
        self.model = m
        self.price = p                         } instance variable
Audi = Cars("R8", 1000000)                     ← object of class
print(Audi.model)
print(Audi.price)
```

fig : object

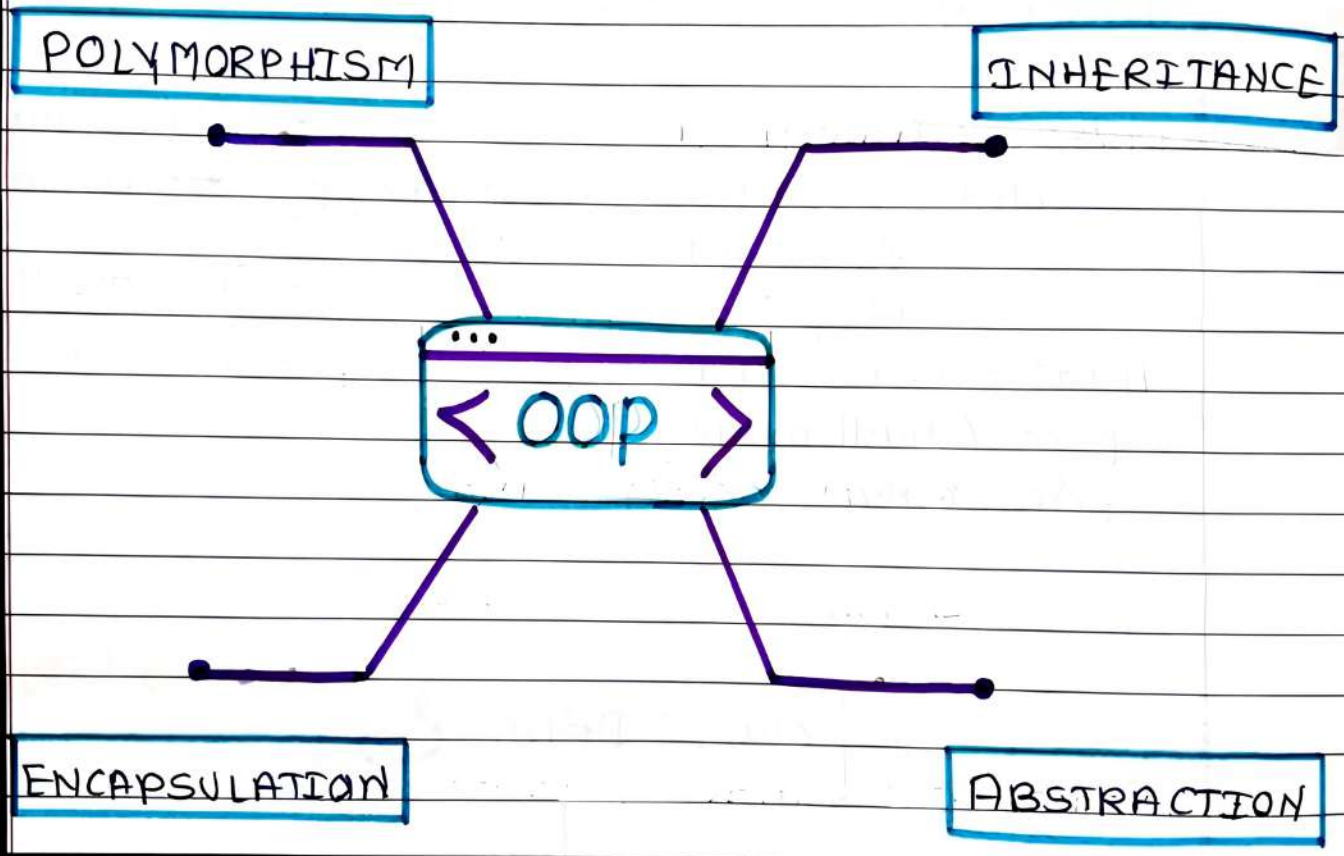


3. OOP :-

In python, object oriented programming (oops) is a programming that paradigm that uses object and classes in programming.

The main concept of OOPs is to bind the data and the functions that work on it on that together as a single unit.

fig: OOP concepts



4. INHERITANCE

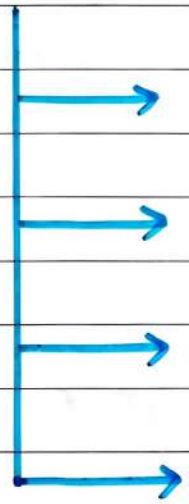
Inheritance is the capability of the one class to derive / inherit the properties from another class.

derived / child class :- The class that derived properties from the parent class is called as derived class.

base class / parent class :- The class from which the properties are being derived is called the base class / parent class.

* provides reusability of a code.

Types of Inheritance :-



1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance.
4. Multiple Inheritance

1. single level Inheritance :-
enables a derived class to inherit characteristics from a single-parent class.

2. Multilevel Inheritance :-
enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent.

3. Hierarchical Inheritance :-
this enables more than one derived class to inherit properties from a parent.

4. Multiple Inheritance :-

this enables one derived class to inherit properties from more than one base class.

fig: Single Inheritance

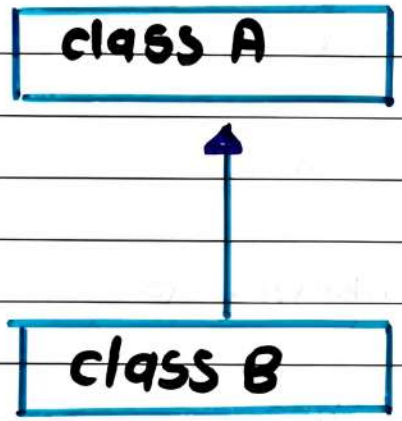


fig Multiple Inheritance

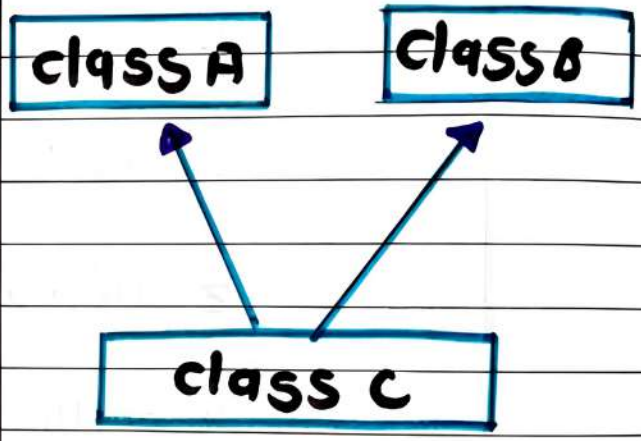


fig: Multilevel Inheritance

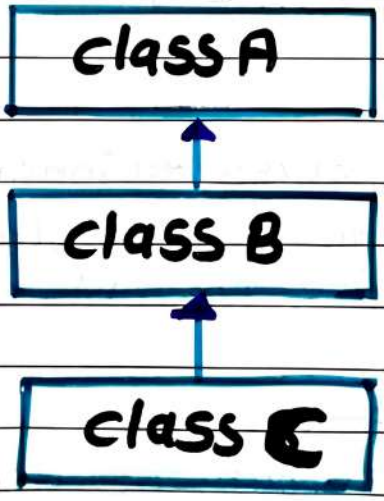
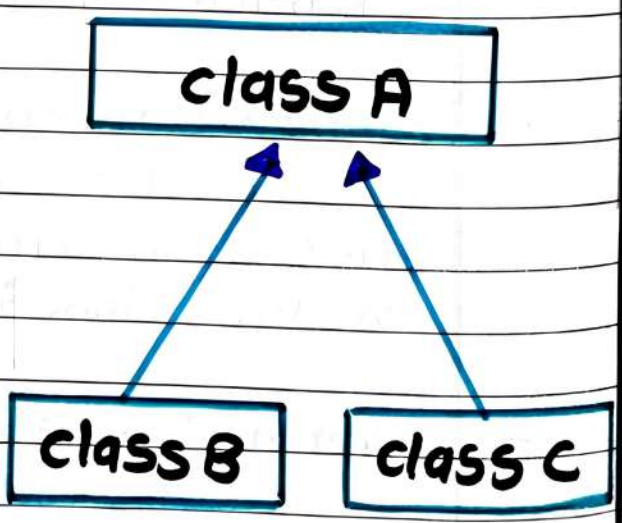


fig: Hierarchical Inheritance



5. POLYMORPHISM

Polymorphism simply means having many forms. For example we need to determine if the given species of birds fly or not, using polymorphism we can do this using a single function.

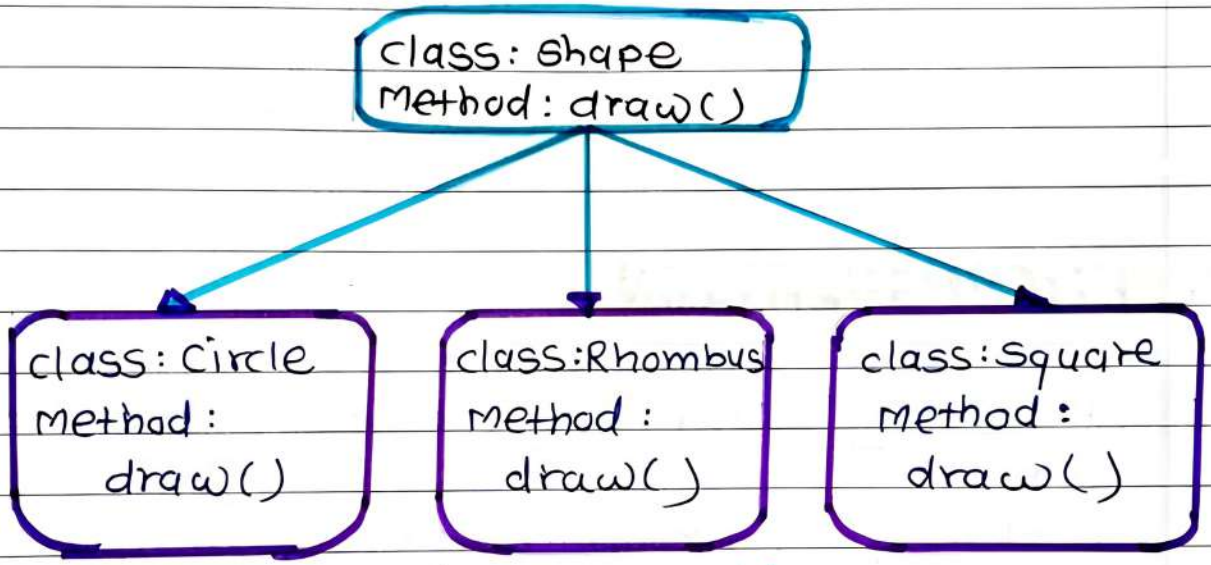


fig : polymorphism

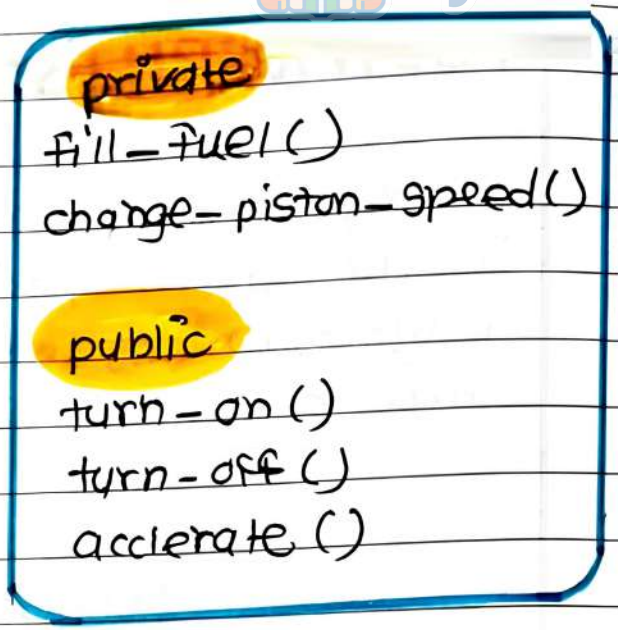
6. ABSTRACTION

Abstraction is defined as a process of handling complexity by hiding unnecessary information from the user.

User is familar with :

how it does ?
(don't know)

What function does ?
(know)



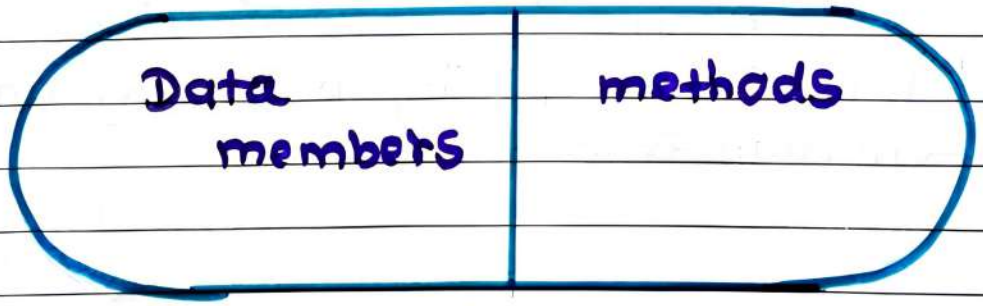
Car

Fig: Abstraction

6. ENCAPSULATION

Encapsulation describes the concept of bundling data and methods within a single unit.

A class is an example of encapsulation as it binds all the data members (instance variable) and methods into a single unit.



class.

fig: Encapsulation

* securing data by hiding implementation details

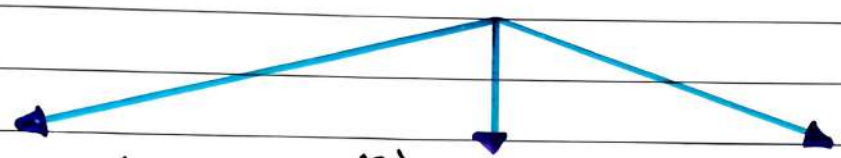
to the user.

7. ACCESS MODIFIERS

Data hiding can be achieved by using :

Access modifiers in python :-
This are used to modify the default scope of the variables.

There are three types of access modifiers :-



i). public ii) private iii). protected

i) public access modifier :-
easily accessible from any part of the program.
python uses '—' symbol to determine the access control for a specific data member.

ii) private access modifier :-
accessible within a class only.
by adding double underscore symbol '__'.

iii) protected access modifier :-
only accessible to a class derived from it.
by adding a single underscore symbol '_'.