1. Object Oriented Programming

- class :-
  - it is nothing but a blueprint to create object
  - it is an blueprint of objects that holds the variable and methods
- object :-
  - it is a variable name that we assign to a class
    - e.g. variable = ClassName()
  - in OPP's we called it "INSTANCE" of a class
  - object :- it is a REAL LIFE ENTITY (e.g. mobial, laptop, headphones, chair, fan, house....etc)
- when we create a class we define number of fuction which is called nothing but "METHODS" in OOP's.
- it provieds more readibility, reusablity and provides more structured way

In [ ]:
```
1
```

syntax:-

```
1 class ClassName():
2     def __init__(self):  # this is where our attributes is going to be
3         pass
4     def Method1(self):
5         pass
6     def Method2(self):
7         pass
8     .
9     .
```

```
    .
    statements_n
```

- Attribute - attributes are those that the things they have such as name, colors, hands, brain etc...
- methods - methods are those the things they can do or perform. such as walking, eating, jumping, playing etc

e.g

- cow :- is a class name
- attributes :- four legs, colour, two horns etc...
- methods :- cow can walk, eat, sleep, sound etc...

2. The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class

- and throughout the class we need to specify the same name instead of self

e.g the instance we are creating will be passed or reffering to self in this way our method will get executed

- The instance itself passing into the method itself and then it gets executed

Self

- Self is nothing but a keyword, We can use any keword name, but you have to follow the same keyword name in entire class
- It allows to access the variables and methods of each object
- Self keyword is used to represent an instance of a class

- It helps to distinguish between ***local variables and instance variables***

NOTE:- class name variable can not be convert into global variable Or It can not be access globally. it can be only called inside of a class

In [ ]:
```
1
```

```
var1 = 5000                          # Global Variable

class ClassName():                   # Class Name
    x = 1                            # Class Variables
    y = 5.5                          # Class Variables
    z = "First Class"                # Class Variables

    def method1(self):               # First Method of a class
        print("This is method1")
        var2 = "Data Science"        # Local Variable
        self.city_name = "Mumbai"    # Instance Variable1
        self.state     = "Maharashtra" # Instance Variable2

    def method2(self):               # Method 2
        print("This is method2")
        var3 = "Machine Learning"    # Local Vartiable
        self.var4 = "NLP"            # Instance variable
        self.method1()
        print("Variable 1 from method1",self.city_name)
        print("Variable 2 from method2",self.x)
```

In [ ]:
```
1
```

class variables

:- it can be only access inside of a class, it can't converted into globle variable or can not be access globally.

Instance Variable

:- It can be access anywhere inside of a class, An instance variable is a variable which is declared in a class but outside of constructors, methods, or blocks. Instance variables are created when an object is instantiated (**init**), and are accessible to all the constructors, methods, or blocks in the class. Access modifiers can be given to the instance variable

```
- class ClassName():
        def __init__(self,attr_1, attr_2, attr_3):
                self.attr_1 = attr_1
                self.attr_2 = attr_2
                self.attr_3 = attr_3
```

What if the variables are local

If the variables are local variable then those variables will have limited scope in a class and we wil not be able to use them inside the whole class or another methods. to use them globaly inside a class then we have specify ("self.") keyword before local variables

init Method

```
1 __init__ This method gets execute automatically whenever we create an object
```

```
using self.variable name in each method for specifically will be so hectic. in order to overcome this situation we can
define __init__ method and will call every attributes inside of a __init__ method
```

- All the attributes of the class we define it in **init** method

Class variable

- class variable are those where we define it after creating a class and it is global variable for all methods of a class

- Class variable:-
  - this will reflects the changes to all instances that we had created
  - if we want to change the all variables of mathods of a whole class then use class variables
  - this will reflect the changes to the whole class methods
  - e.g - ClassName.variable
- instance variable:-
  - This will reflect the changes to specific instances that we had created
  - if we want to change the specific method variables then use instance variables
  - this will reflect the changes in a specific class methods
  - e.g - self.variable

- Some points on Python class:
  - Classes are created by keyword class.
  - Attributes are the variables that belong to a class.
  - Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute

In [ ]:
```
1
```

There are four fundamentals concept of Oop's

- Inheritance
- Encapsulation
- Polymorphism
- Abstraction

1. Inheritance

- It allows to define a class that inherits all the methods and properties from another class.
- parent class - it is a class that is being inherited from
- child class - it is a class that is inherits from another class
  - Parent class - Base class (Superclass)
  - child class - Derived class (subclass)

There are four types of inheritance

- 1. Single Inheritance
  - When a derived class inherits only from base class. it is known as single inheritance
- 2. Multiple Inheritance
  - When a class can be derived from more than one base class this type of inheritance is called multiple inheritances. In multiple inheritances, all the features of the base classes are inherited into the derived class.
- 3. Multilevel Inheritance
  - In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class. This is similar to a relationship representing a child and a grandfather.
- 4. Hirarchical Inheritance
  - When more than one derived class are created from a single base this type of inheritance is called hierarchical inheritance.

- 5. Hybrid Inheritance
  - Inheritance consisting of multiple types of inheritance is called hybrid inheritance.

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

2. MultiLevel Inheritance

```
class GrandFather():
    pass

class Father(GrandFather):
    pass

class Child(Father): # Multilevel Inheritance
    pass
```

In [ ]:
```
1
```

super().**init**()

- super().**init**():
  - This method can only work in single inheritance and multilevel inheritance.
  - super method refers to the parent or super class
  - when is comes to Hierachical or multiple inheritance etc.. except single inheritace

- super method won't able to distinguish between two super class attributes and variables while initializing attrinbutes into the derived class

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

3. Encapsulation

Encapsulation is one of the most fundamental concepts in object-oriented programming (OOP). This is the concept of wrapping data and methods that work with data in one unit. This prevents data modification accidentally by limiting access to variables and methods. An object's method can change a variable's value to prevent accidental changes. These variables are called private variables.

- Encapsulation used to restrict the access of methods and variables from outside the class.
- Protecting accidental change of data (Variables and Methods)

- syntax :-
  - to identify or to create an encapsulation method we assing
  - __variable_name or __Method_name()

example - class Base1():

```
def __init__(self, a, b):
    self.__a = a
    self.__b = b

def __addition(self):
    self.add = self.__a + self.__b
    print(f"Addition of {self.__a} and {self.__b} is {self.add}")

def __Multiplication(self):
    multi = self.__a * self.__b
    print(f"Multiplication of {self.__a} and {self.__b} is {self.multi}")
```

- in this way we can not access a single variable or method from the class

How can we call or access the variables and methods from the Encapsulation class

Name Mangling

- To access or modify the private variables and methods from outside the class
- Syntax :-
  - Obj._ClassName__VariableName

- We can access the "Encapsulation class" variables and methods by
  - Obj._ClassName__VariableName

In [ ]:
```
1
```

4. Abstraction

- Blueprint of other class/project
  - We can not create an object of the abstract class
  - Abstract class is only used for the declaration of the methods
  - In the abtract class we are not supposed to implement these methods

```
- @abstractmetho decorator --> used to define abstract method
```

  - The abstract method is a child class of the ABC (ABC is Abstract Base Class Module)
  - It is used to hide the internal functionality of the method

- An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class.
- A class which contains one or more abstract methods is called an abstract class.
- An abstract method is a method that has a declaration but does not have an implementation.
- While we are designing large functional units we use an abstract class.
- When we want to provide a common interface for different implementations of a component, we use an abstract class.

Why use Abstract Base Classes :

- By defining an abstract base class, you can define a common Application Program Interface(API) for a set of subclasses.
- This capability is especially useful in situations where a third-party is going to provide implementations, such as with plugins, but can also help you when working in a large team or with a large code-base where keeping all classes in your mind is difficult or not possible.

In [ ]:
```
1
```