var=10

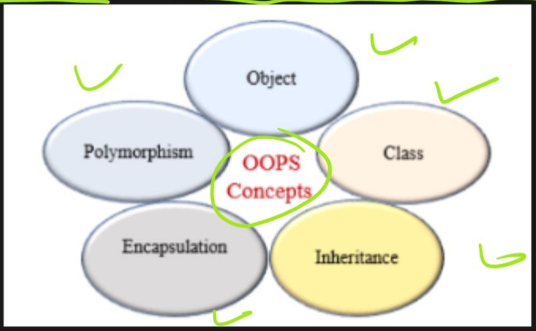
Day-6 14-08-2023

Object-Oriented Programming in Python

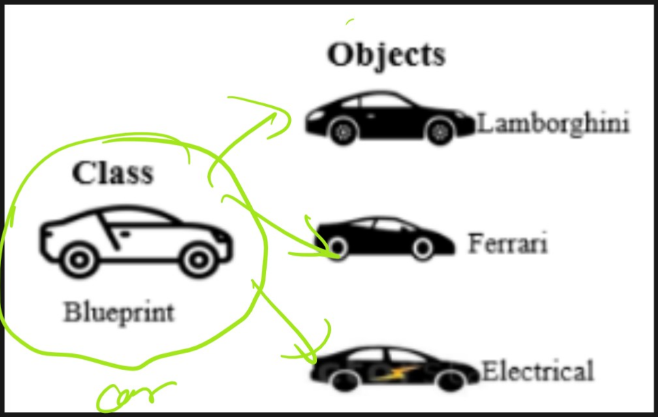
Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects". The object contains both data and code: Data in the form of properties (often known as attributes), and code, in the form of methods (actions object can perform).

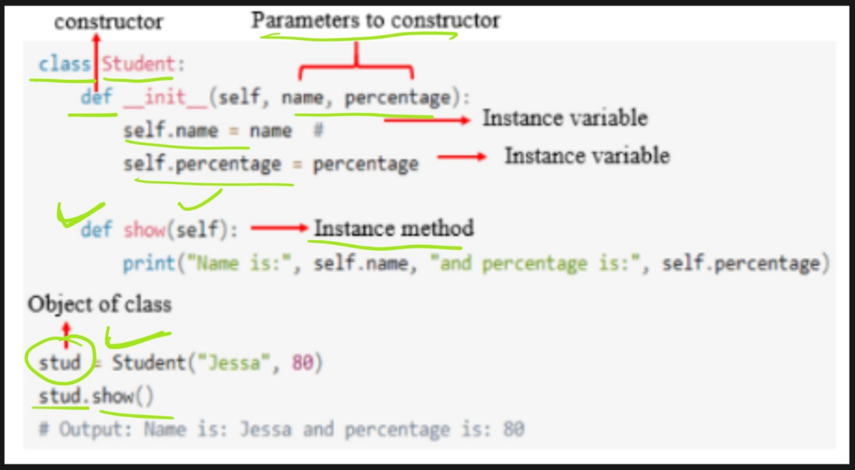
A class is a blueprint for the object. To create an object we require a model or plan or blueprint which is nothing but class.

Note: class can execute its self when Run/Execute the script.



A class contains the properties (attribute) and action (behavior) of the object. Properties represent variables, and the methods represent actions. Hence class includes both variables and methods.





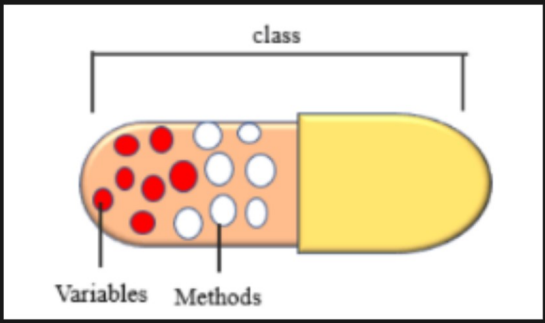

```python
class Student:
    def __init__(self, name, percentage):
        self.name = name  #
        self.percentage = percentage

    def show(self):
        print("Name is:", self.name, "and percentage is:", self.percentage)

stud = Student("Jessa", 80)
stud.show()
# Output: Name is: Jessa and percentage is: 80
```

Encapsulation in Python

In Python, encapsulation is a method of wrapping data and functions into a single entity. For example, A class encapsulates all the data (methods and variables). Encapsulation means the internal representation of an object is generally hidden from outside of the object's definition.



Polymorphism in Python

Polymorphism in OOP is the **ability of an object to take many forms**. In simple words, polymorphism allows us to perform the same action in many different ways.

Inheritance In Python

In an Object-oriented programming language, inheritance is an important aspect. In Python, inheritance is the process of inheriting the properties of the parent class into a child class.

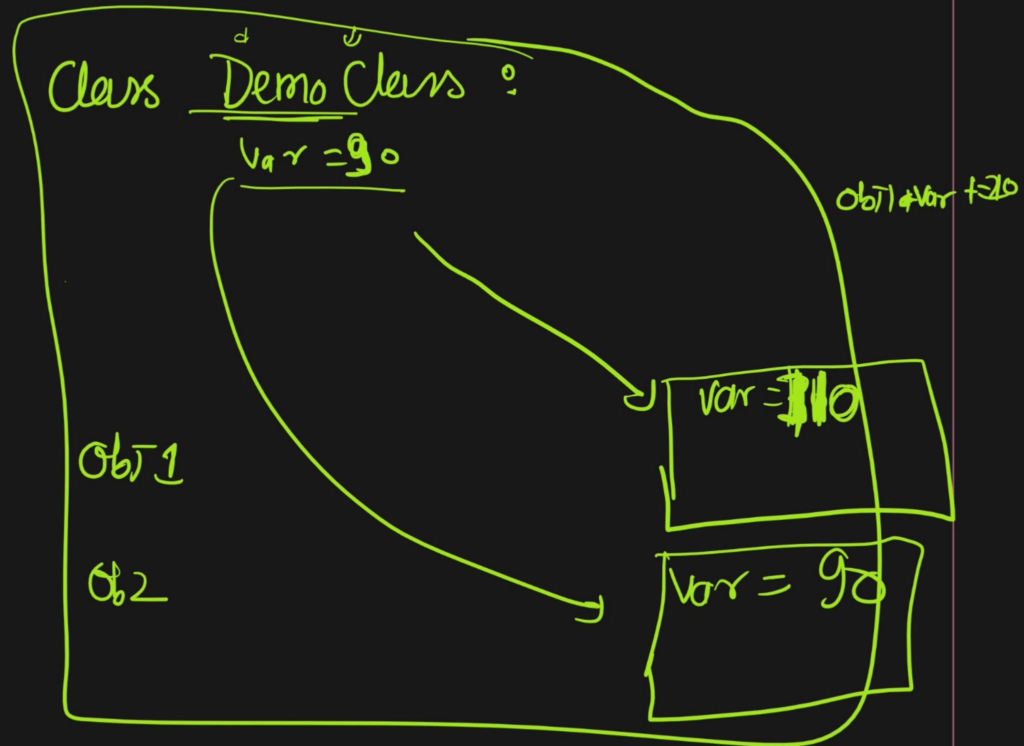
DisAdvantages of functions.

1. No Protection
2. Overriding — math 1m (SIS) / m in (SB)
3. Monitor the activity of any user

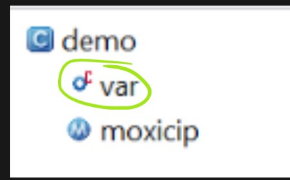
Class get execute without calling / creating an object.

Class Memory. (Class blueprint)

Object Memory.



attribute from the Class Memory.



⇒ demo.var

1. var
2. moxicip

attribute & methods from the Object Memory.

Inside Class	outside Class
self	give the reference of the object Name
self.var	→ obj1.var
self.moxicip()	→ Ob2.moxicip()

obj1 — var, moxicip

Obj2 — var, moxicip

Class memory

name = xyz → Class

Var = 78

def moxicip

Obj1

Obj2

Object class

Object memory

name = xyz

var = 78

def moxicip()

self.var

var = 78

class

i = 2

def __init__ ...

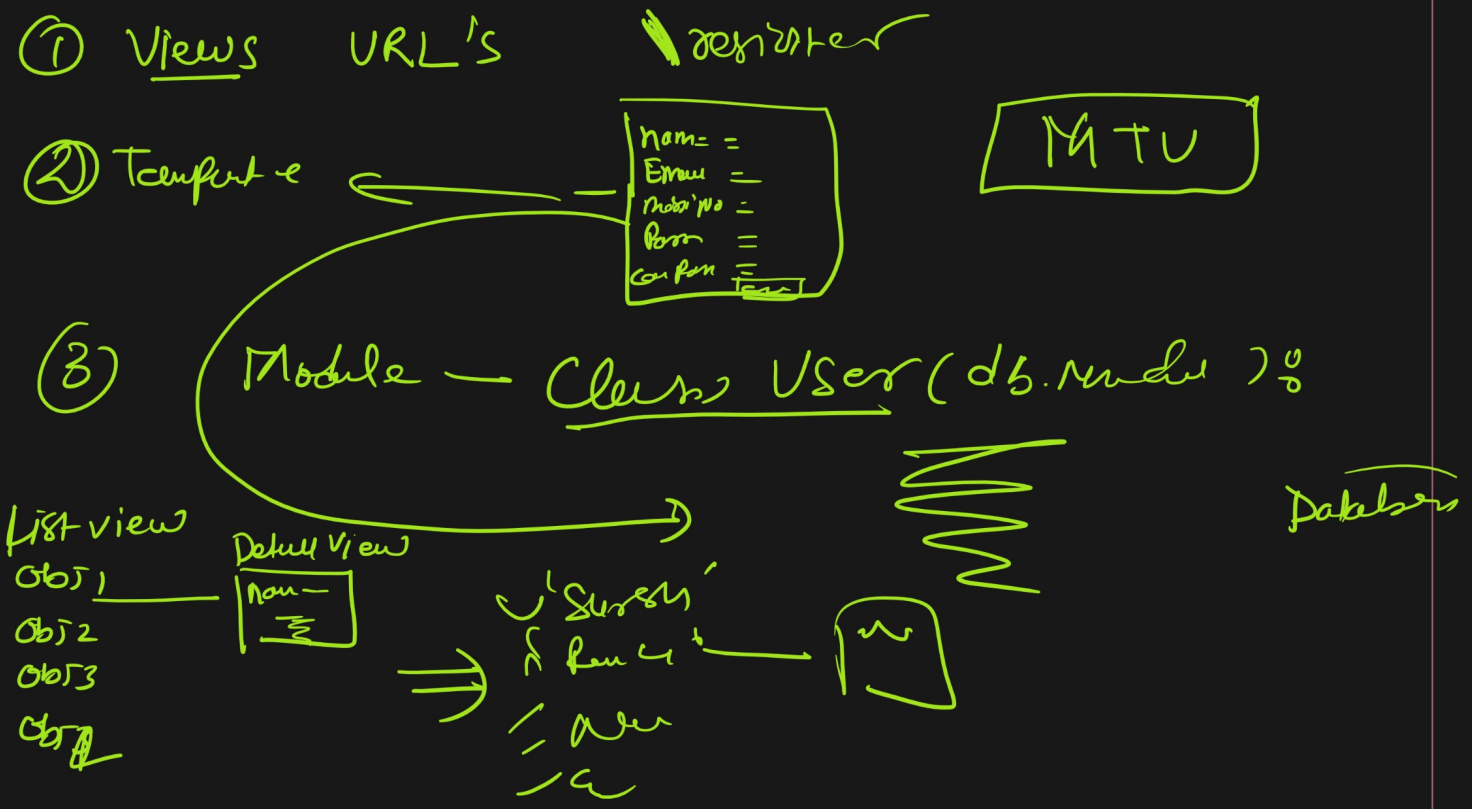
obj1 — i = 2

obj2 — i = 2

```
class details:
    i=0
    print("This is details class.")
    def __init__(self):
        details.i+=1
        print("this is init method.and object No.",self.i)

obj1=details()#1
obj2=details()#1
print(obj1.i)
print(obj2.i)
```

```
This is details class.
this is init method.and object No. 1
this is init method.and object No. 2
2
2
```



outside class.

* When we call something from class name.

demo.var

* When we call something from object name.
Use name of the object as refrence.

Obj1.var | Obj2.var

Obj1.myclip() | Obj2.myclip()

Inside Class

want a call something from class name.

Ex demo.var

↳ Need a refrence of class name

* want a call something from object name

→ self.var ← Instance variable

→ self.myclip() ← Instance Method

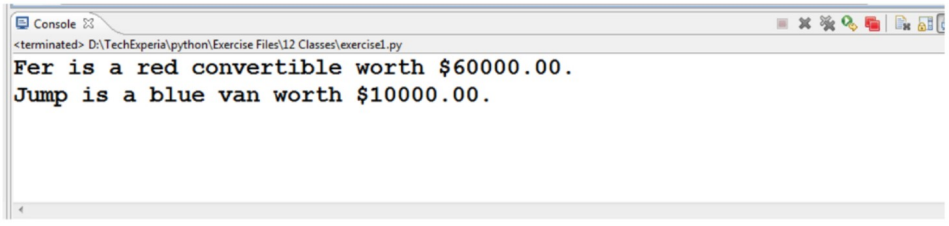
python powered™ verview of OOP Terminology

Solve the Prob in <=5 Lines

Exercise

We have a class defined for vehicles. Create two new vehicles called car1 and car2. Set car1 to be a red convertible worth $60,000.00 with a name of Fer, and car2 to be a blue van named Jump worth $10,000.00.

output



```python
class Person(object):
    """A simple class."""                          # docstring
    species = "Homo Sapiens"                       # class attribute

    def __init__(self, name):                      # special method
        """This is the initializer. It's a special
        method (see below).
        """
        self.name = name                           # instance attribute

    def __str__(self):                             # special method
        """This method is run when Python tries
        to cast the object to a string. Return
        this string when using print(), etc.
        """
        return self.name

    def rename(self, renamed):                     # regular method
        """Reassign and print the name attribute."""
        self.name = renamed
        print("Now my name is {}".format(self.name))
```

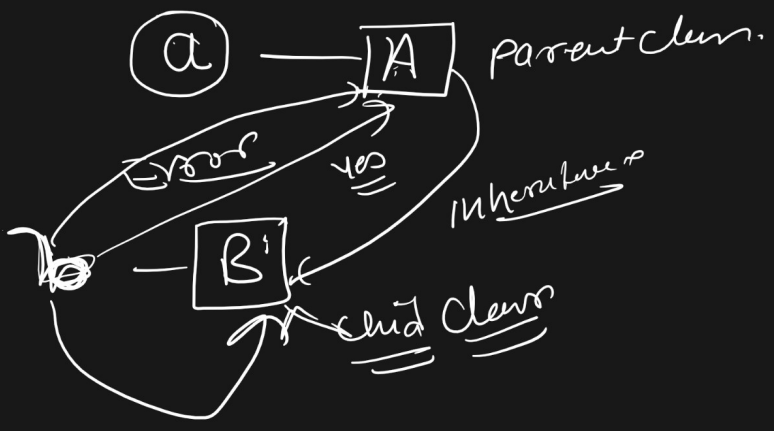
Python Inheritance

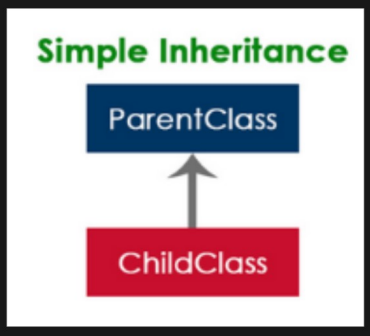
The inheritance is the process of acquiring the properties of one class to another class.

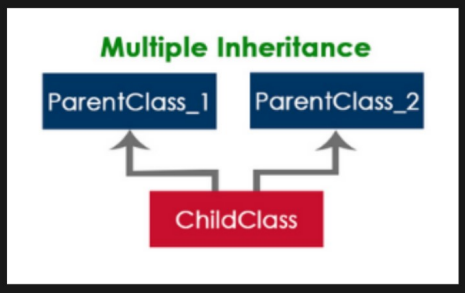
The Parent class is the class which provides features to another class. The parent class is also known as Base class or Superclass.

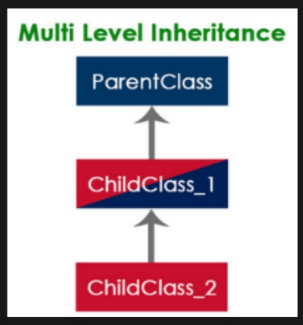
The Child class is the class which receives features from another class. The child class is also known as the Derived Class or Subclass.

In the inheritance, the child class acquires the features from its parent class. But the parent class never acquires the features from its child class.




Simple Inheritance
ParentClass
ChildClass


Multiple Inheritance
ParentClass_1
ParentClass_2
ChildClass


Multi Level Inheritance
ParentClass
ChildClass_1
ChildClass_2