OOPS in Python (Object-Oriented Programming System)

OOP (Object-Oriented Programming) is a programming paradigm based on the concept of **"objects"**. Objects can contain **data (attributes/variables)** and **functions (methods)** to operate on the data.

Class and Object Example in Python

What is a Class?

A **class** is a blueprint for creating objects.

It defines **attributes (variables)** and **methods (functions)**.

What is an Object?

An **object** is an **instance of a class**.

Through the object, we can access the variables and methods of the class.

Constructor:

A constructor is a unique function that gets called automatically when an object is created of a class. (def __init__(self): constructor) create a instance of object automatically call (not mandatory to call).

The main purpose of a constructor is to initialize or assign values to the data members of that class.

Self:

Self(self) denotes current object (used)

Self-keyword → used to object refers

The self-keyword is used to refer to the instance of the class, allowing access to its variables and methods.

The self-keyword refers to the object itself."

Exercise:

1) Create a class called student

create a variable = name and register number using constructor.

Create a function called display which should display the name and register number of the student

2) Create a class called Fruit

Create a variable called color using init method

Create a object called apple "Pass the color variable as a parameter through object".

3) Create a class called teacher

Create a variable = name and register number using constructor

Create a function called display which should displav the name and register number of the teacher

Create t1 and t2 object and pass the name and reg no value through object.

4) Create a class called calculator

Create two variable a and b , Create a function called add,sub,mul,div all the functions should take 2 variables as a parameter, Pass the a and b value through object()

Class (Variables and Types):

Instance Variable

Class Variable

Instance Variable

Instance variables are *unique to each instance of a class*. They are defined within methods and are prefixed with the self-keyword. These variables store data that is specific to an instance, making them essential for object-oriented programming (OOP) principles like encapsulation.

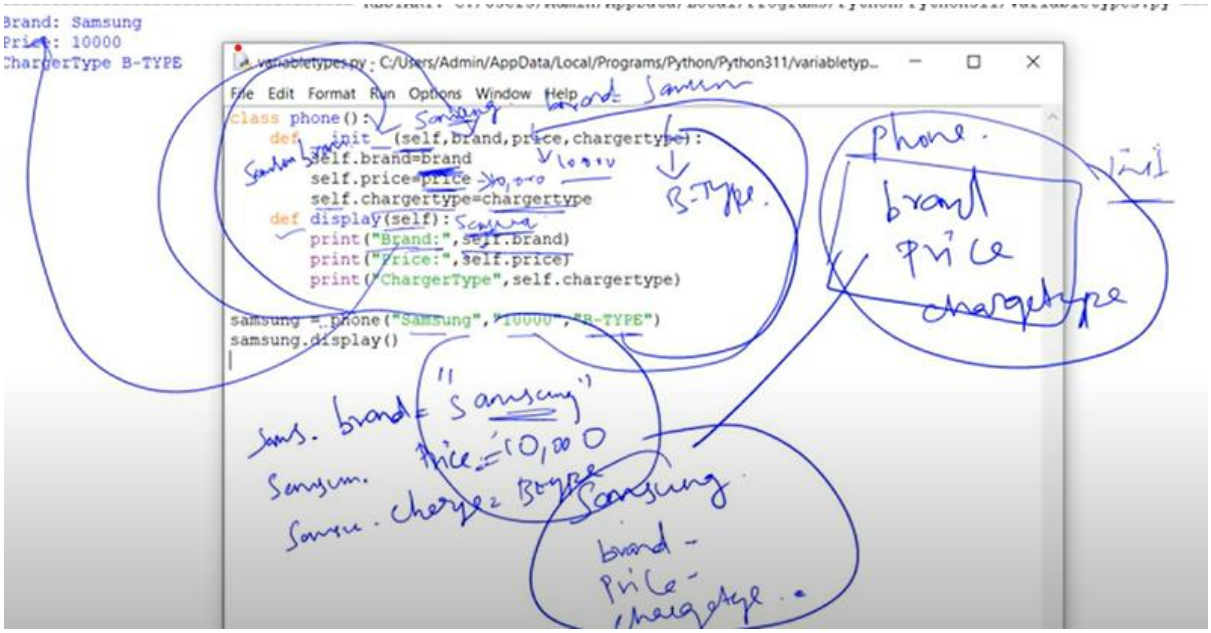
Self is object instance.

Self used like, self.brand,self.price, self.ram ➔ this is called instance variable

Eg.

```
class phone():
  def __init__(self,brand,price,ram):
```

```
        self.brand=brand
        self.price=price
        self.ram=ram
    def display(self):
        print("Brand: ",self.brand)
        print("Price: ",self.price)
        print("RAM: ",self.ram)
oppo=phone("Oppo","15000","16gb")
oppo.display()
```



Class Variable

In contrast, class variables are *shared among all instances of a class*. Class variables store data that is common to all instances, making them a powerful tool for managing shared state and settings.

Eg.

```
class phone():
    chargetype="C-TYPE"
    def __init__(self,brand,price):
```

```
        self.brand=brand
        self.price=price
    def display(self):
        print("\nBrand: ",self.brand)
        print("Price: ",self.price)
        print("ChargeType: ",self.chargetype)
phone.chargetype="B-TYPE "
oppo=phone("Oppo","15000")
oppo.display()
redmi=phone("Redmi","10000")
redmi.display()
vivo=phone("vivo","5000")
vivo.display()
```

Types of Methods in Python Class

In Python, methods inside a class are mainly of three types:

Instance Method

Definition:
These methods work with a specific object (instance) of the class.

How:
- First parameter is always 'self' (refers to the object calling the method).
- Can access and change both object's data and class data.

Use:
Used when you want to perform actions related to one particular object.

Example:

```
class Student:
    def show(self):
        print("This is an instance method")
```

Class Method

Definition:
These methods work with the class itself, not with objects.

How:
- First parameter is always 'cls' (refers to the class itself).
- Use '@classmethod' decorator.
- Can access and change class-level data, but not object data directly.

Use:
Used to create alternative constructors or modify class-level information.

Example:

```
class Student:
  @classmethod
  def info(cls):
    print("This is a class method")
```

Static Method

Definition:
These are just like normal functions inside a class. They don't use class data or object data.

How:
- No 'self' or 'cls' needed.
- Use '@staticmethod' decorator.

Use:
Used for utility/helper functions that relate to the class but don’t need any class or object information.

Example:

```
class Student:
  @staticmethod
  def greet():
    print("Welcome to Python!")
```

Inheritance in Python (Simple Definition)

Inheritance is a feature in Python where a child class can access the properties and methods of a parent class. It is mainly used for code reusability and helps in creating relationships between classes.

Types of Inheritance in Python (Short Definitions)

Single Inheritance:

A child class inherits from one parent class.

Multiple Inheritance:

A child class inherits from multiple parent classes.

Multilevel Inheritance:

Inheritance occurs through multiple levels (grandparent → parent → child).

Hierarchical Inheritance:

One parent class is inherited by multiple child classes.

Hybrid Inheritance:

A combination of two or more types of inheritance.

Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

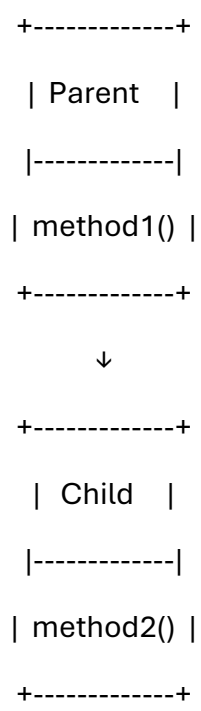
Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Single Inheritance:

Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.

Diagram



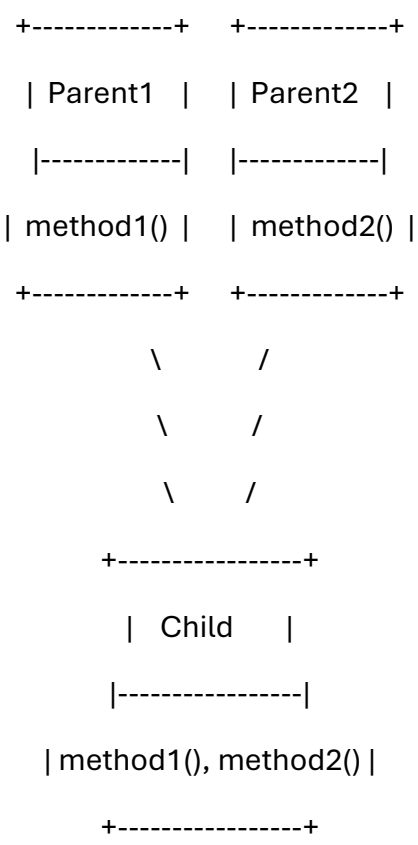
Eg.

```
#Single Inheritance
#base class
class dad():
   def phone(self):
      print("Dad's phone")
#derived class
class son(dad):
   def laptop(self):
      print("Son's Laptop")
ram=son()
#ram.laptop()
ram.phone()
```

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.

Diagram



Eg.

```
#Multiple Inheritance
#base class1
class dad():
   def phone(self):
      print("Dad's phone")
#base class2
class mom():
   def sweet(self):
      print("Mom's Sweet")
class son(dad,mom):
```

```
    def laptop(self):
        print("Son's Laptop")
ram=son()
#ram.laptop()
ram.phone()
ram.sweet()
```

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.

Diagram

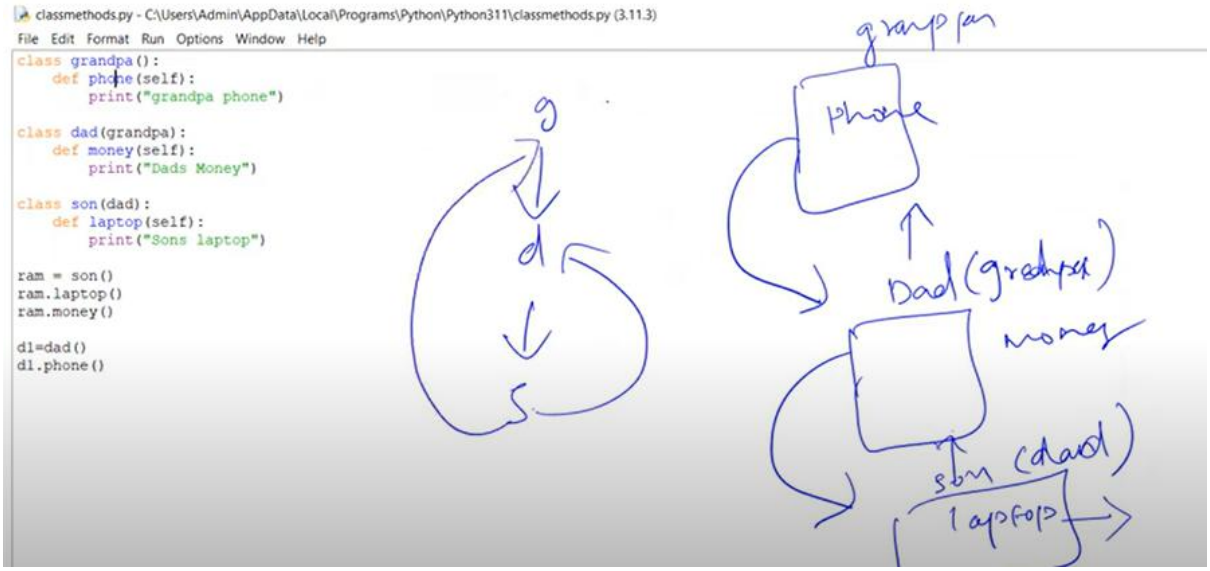
Grandparent(home) (Base Class)

↓

Parent (car) (Derived from grandparent)

↓

Child (bike) (Derived from Parent)



Hierarchical Inheritance

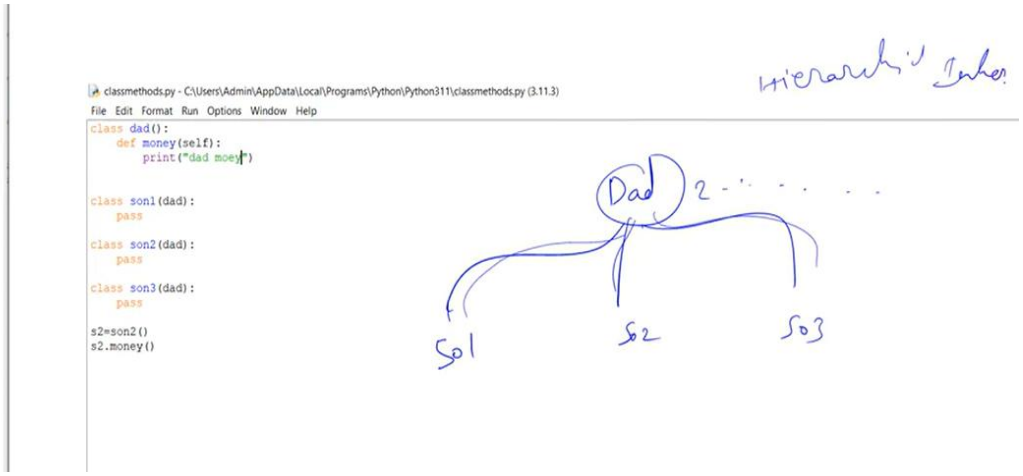
Hierarchical Inheritance is a type of inheritance in Python where **multiple child classes inherit from the same single parent class**.

Diagram

Parent

/ \ \

Child1 Child2 Child3



Hybrid Inheritance

Hybrid Inheritance is a combination of **two or more types of inheritance** (single, multiple, multilevel, hierarchical) in a single program.
It helps in building complex class relationships where a class might inherit features from multiple parent classes in different ways.

Diagram

Parent

/ \

Child1 Child2

\ /

Child3