Introduction to

Object **O**riented **P**rogramming **S**ystem

Object Oriented Python

Like other general-purpose programming languages, Python is also an object-oriented language since its beginning.

It allows us to develop applications using an Object-Oriented approach. In Python, we can easily create and use classes and objects.

Object Oriented Python

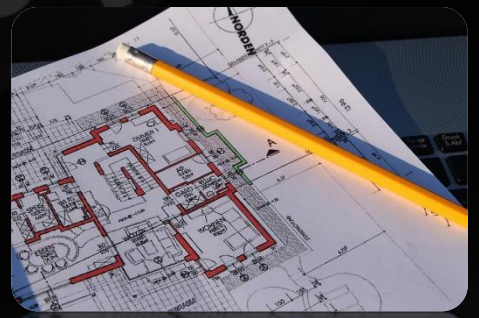
- Object
- Class
- Inheritance
- Methods
- Encapsulation
- Attributes
- Abstraction
- Polymorphism

Class

The class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods.

A class is a blueprint for the object.

We can write a class to represent attributes (Variables) and behaviour (Methods) of object



for customized exceptions but not for pre defined exceptions

Object

Physical existence of a class is nothing but object. We can create any number of objects for a class.

Objects are an instance of a class that can access the data.

How to Define a Class

SYNTAX

```
class className:
    ''' documentation string '''
    Constructor

    variables
```

- Instance
- static
- local

```
    methods
```

- Instance
- static
- class

To Access Doc String

Within the class doc string is always optional.

We can get doc string by using the following 2 ways.

```
print(classname.__doc__)
```

```
help(classname)
```

SYNTAX

ReferenceVariable = classname()

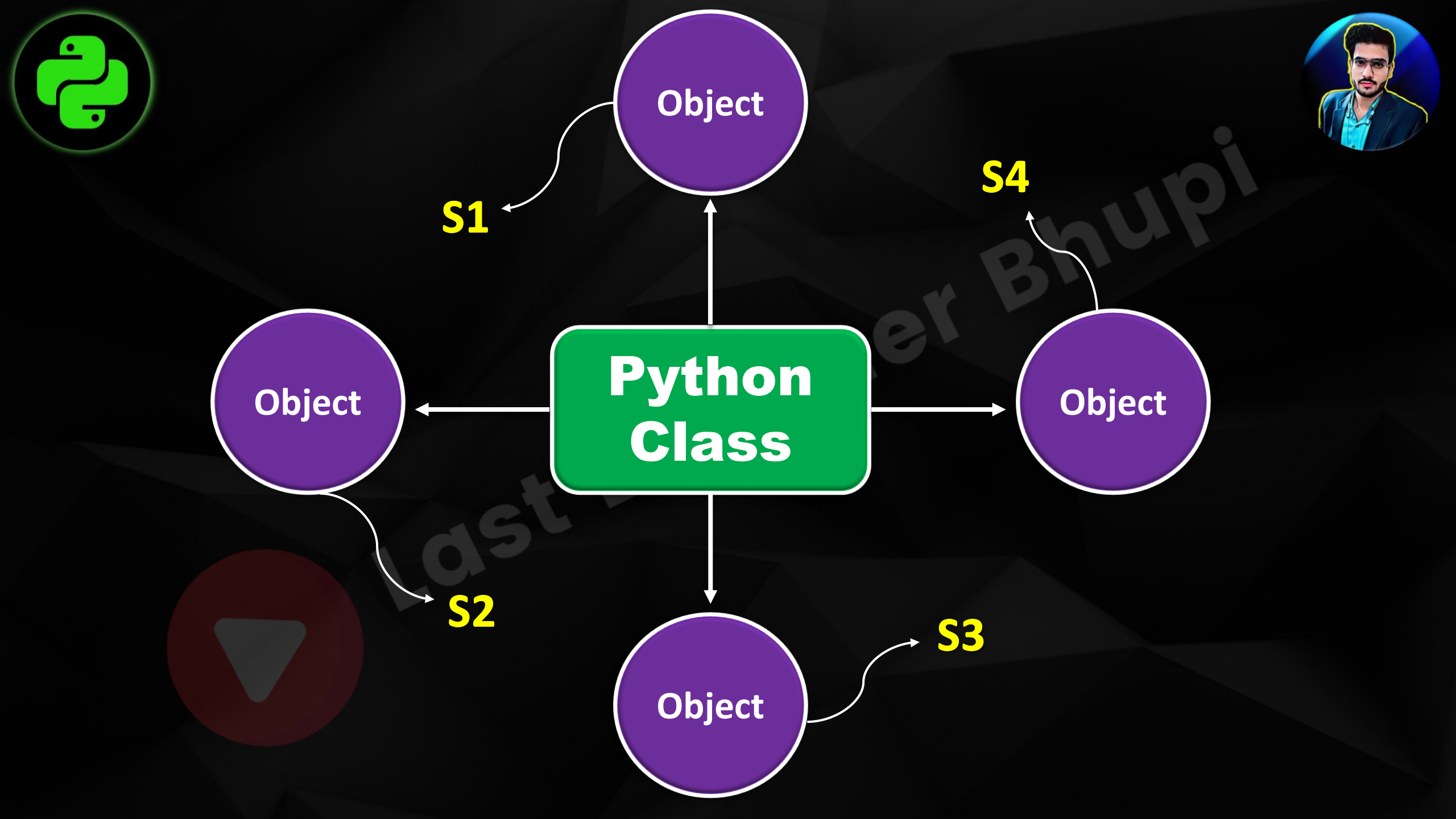
Example

Obj = Test()

Reference Variable

The variable which can be used to refer object is called reference variable.

By using reference variable, we can access properties and methods of object


Object
S1
S4
Object
Python
Class
Object
S2
Object
S3

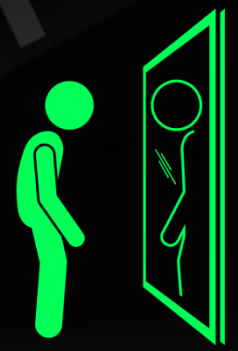
Activity Time

Write a Python program to create a Movie class and Creates an object to it. Call the method display() to display Movie details

SELF

self is the default variable which is always pointing to current object (like this keyword in Java)

By using self we can access instance variables and instance methods of object.



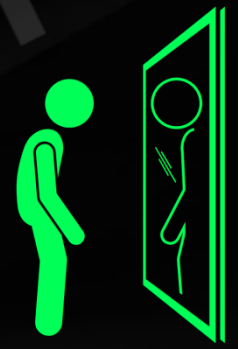
SELF

1. self should be first parameter inside constructor

```
def __init__(self):
```

2. self should be first parameter inside instance methods

```
def display(self):
```



Constructor

1. The name of the constructor should be __init__(self)
2. Constructor will be executed automatically at the time of object creation.
3. The main purpose of constructor is to declare and initialize instance variables.
4. Per object constructor will be exeucted only once.
5. Constructor can take atleast one argument(atleast self)
6. Constructor is optional and if we are not providing any constructor then python will provide default constructor.

SYNTAX

Constructor

```
def __init__(self,name,rollno,marks):
    self.name=name
    self.rollno=rollno
    self.marks=marks
```

Local Variables

Instance Variables

METHOD VS CONSTRUCTOR

Method	Constructor
Name of method can be any name	Constructor name should be always __init__
Method will be executed if we call that method	Constructor will be executed automatically at the time of object creation.
Per object, method can be called any number of times.	Per object, Constructor will be executed only once
Inside method we can write business logic	Inside Constructor we have to declare and initialize instance variables

Types of Variables

- Instance
- Static
- Local

Instance Variables

If the value of a variable is varied from object to object, then such type of variables are called instance variables.

For every object a separate copy of instance variables will be created.

Where we can Declare Instance Variables

1. Inside Constructor by using self variable
2. Inside Method by using self variable
3. Outside of the class by using object reference variable

1 Inside Constructor by using self variable

We can declare instance variables inside a constructor by using self keyword. Once we creates object, automatically these variables will be added to the object.

```
def __init__(self):
```

Inside Constructor by using self variable

```
class Movie:

    def __init__(self):
        self.name = "Laal Singh Chaddha"
        self.budget = "150 Cr"
        self.revenue = "100 Cr"

obj = Movie()

print(obj.__dict__)
```

2 Inside Method by using self variable

We can also declare instance variables inside instance method by using self variable. If any instance variable declared inside instance method, that instance variable will be added once we call that method.

PREDICT THE OUTPUT

```
class Test:

    def __init__(self):
        self.x = 100
        self.y = 200
    def m1(self):
        self.z = 300

obj = Test()
print(obj.__dict__)
```

A X = 100 , Y = 200 , Z =300

B X = 100 , Y = 200

C Name Error

D Syntax Error

CONCLUSION

IF your Answer is A

3 Outside of the class by using object reference variable

We can also add instance variables outside of a class to a particular object.

3 Outside of the class by using object reference variable

```
class Test:

    def __init__(self):
        self.a = 1
        self.b = 2
    def m1(self):
        self.c = 3
        self.d = 4

obj = Test()
obj.m1()
obj.e = 5
print(obj.__dict__)
```

How to access Instance variables

How to Delete Instance variables

1. Within a class we can delete instance variable

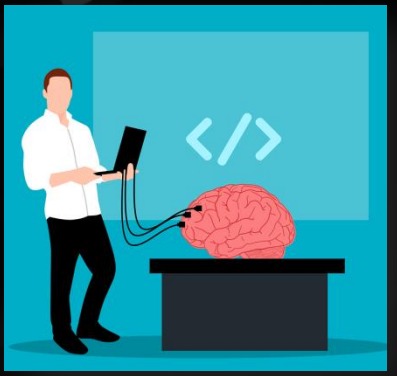
2. From outside of class we can delete instance variables

Note

If we change the values of instance variables of one object then those changes won't be reflected to the remaining objects, because for every object the separate copy of instance variables are available.

Static Variables

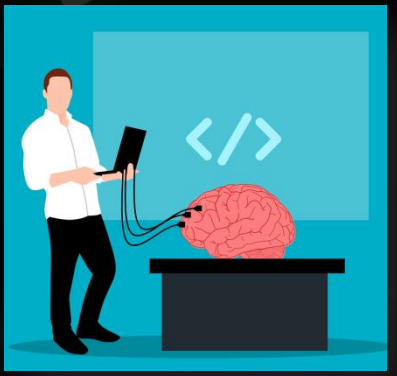
If the value of a variable is not varied from object to object, such type of variables we have to declare with in the class directly but outside of methods. Such type of variables are called Static variables.



Static Variables

For whole class only one copy of static variable will be created and shared by all objects of that class.

We can access static variables either by class name or by object reference. But recommended to use class name.



Where we can Declare Static Variables

1. We can declare within the class directly but from out side of any method
2. Inside constructor by using class name
3. Inside instance method by using class name
4. Inside classmethod by using either class name or cls variable
5. Inside static method by using class name

How to access Static Variables

1. inside constructor: by using either self or classname
2. inside instance method: by using either self or classname
3. inside class method: by using either cls variable or classname
4. Inside static method: by using cls variable
5. From outside of class: by using either object reference or classname

Modify Static Variables

Anywhere either with in the class or outside of class we can modify by using classname.

But inside class method, by using cls variable .

PREDICT THE OUTPUT


```
class Test:
    a = 5
    def m1(self):
        Test.b = 10

obj = Test()
obj.m1()
print(Test.__dict__)
```

A { a : 5 }

B {a : 5 , b : 10}

C { }

D Syntax Error

PREDICT THE OUTPUT


```
class Test:
    a = 5
    def m1(self):
        self.a = 5000

obj = Test()
print(obj.__dict__)
```

A { a : 5 }

B { }

C {a : 5 , a : 5000}

D Syntax Error

CONCLUSION

If we change the value of static variable by using either self or object reference variable, then the value of static variable won't be changed , just a new instance variable with that name will be added to that particular object.

Objects

ID = 1000	ID = 1001	ID = 1002
Name = Bhupi	Name = Manoj	Name = Deepali
Company = HCL	Company = HCL	Company = HCL
P1	P2	P3

Here Company Name is Same for all the Objects So it is better to declare

Company Name as Static Variable

Company = HCL

Static Variable

SHARED TO ALL

ID = 1000
Name = Bhupi
Company = HCL

ID = 1001
Name = Manoj
Company = HCL

ID = 1002
Name = Deepali
Company = HCL

P1

P2

P3

How to Delete Static variables

1. Within a class — `del classname.variable`
2. inside classmethod — `del cls.variablename`

Basis	Instance Variables	Static Variables
Means	If the value of a variable is varied from object to object is called instance variables	If the value of a variable is not varied from object to object is called static variables
Nature	These are object level variables	These are class level variables
Memory Consumption	For every object, a separate copy will be created.	A single copy will be created at class level and shared by all the objects of that class
Modification	If we try to modify instance variables then those changes won't be reflected to other.	If we try to modify static variables then those changes will be reflected to all objects

Note

By using object reference variable/self we can read static variables, but we cannot modify or delete.

If we are trying to modify, then a new instance variable will be created

If we are trying to delete then we will get error.

IMPORTANT

Local Variables

Sometimes to meet temporary requirements of programmer , we can declare variables inside a method directly.

Local variables will be created at the time of method execution and destroyed once method completes.

Local variables of a method cannot be accessed from outside of method.

Local Variables

```
class Test:
    def m1(self):
        a = 50
        print(a)
    def m2(self):
        print(a)

obj = Test()
obj.m1()
obj.m2()
```

Here a is Local Variable

NameError

Example 1

Local Variables

```
class Test:
    def __init__(self,ID,Name,Salary):
        self.id = ID
        self.name = Name
        self.sal = Salary
    def display(self):
        print(ID)
        print(Name)
        print(Salary)

obj = Test(100,"Alia Bhatt",25000)
obj.display()
```

Local Variables

NameError

Example 2

Types of Methods

Instance Methods

Static Methods

Class Methods

Instance Methods

Inside method implementation if we are using instance variables then such type of methods are called instance methods.

Inside instance method declaration , we have to pass self variable.

Instance Methods

By using self variable inside method we can able to access instance variables.

```
def m1(self):
    # body of m1
```

Setter Methods
And
Getter Methods

Setter Methods

Setter methods can be used to set values to the instance variables. setter methods

```
def setVariable(self,variable):
    self.variable=variable
```

Getter Methods

Getter methods can be used to get values of the instance variables.

```
def getVariable(self):
    return self.variable
```

Demo Program to Understand

Getter & Setter Methods

Class Methods

Inside method implementation if we are using only static variables , then such type of methods we should declare as class method.

We can declare class method explicitly by using @classmethod decorator.

For class method we should provide cls variable at the time of declaration

Demo Program to Understand

Class Methods

Basis	Instance Method	Class Method
Means	Inside Instance Methods, Atleast one instance variable must be present	if we are using only Static variables inside method are known as class method
Access	Inside Instance Method, We can access instance and static variables	Inside Class Method, We can only access static variables not instance
Decorator	Not Required	@classmethod decorator required
Argument	First Argument should be self	First Argument should be cls
Calling	We can call instance method by using object reference	We can call class method by using object reference or by using class but recommended to use class name

Static Methods

In general these methods are general utility methods.
Inside these methods we won't use any instance or class variables.
Here we won't provide self or cls arguments at the time of declaration.
We can declare static method explicitly by using @staticmethod decorator
We can access static methods by using classname or object reference

Demo Program to Understand

Static Methods

Instance Methods

Class Methods

Static Methods

Instance V/s Class V/s Static Methods

1. If we are using any instance variables inside method body then we should go for instance method. We should call by using object reference only

2. If we are using any static variables inside method body, we should declare such type of methods as class method. We can declare class method explicitly by using @classmethod decorator.

3. If we are not using any instance variable and any static variable inside method body, we should go for static methods explicitly by using @staticmethod

Complete Story of

Nested Class

(Inner Class)

Advantages of Nested Class

1. It Improves Modularity of the Application

2. It Improves Security of the Application

Nested Class

Without Existing Outer class object, there is no chance of existing inner class object.

Inner class object is strongly associated with outer class object

Complete Story of

Nested Method

(Inner Method)

Nested Method

We can declare a method inside another method, such type of methods are called Nested Methods

Inside a Method if any functionality repeatedly required , that functionality we can define as a separate method, and we can call that method any number of times based our requirement.

Advantages of Nested Method

1. Code Reusability

2. Improves Modularity

INTRODUCING

Garbage Collector In Python

Garbage Collector Comes

Garbage

Garbage Collection

In old languages like C++, programmer is responsible for both creation and destruction of objects.

Usually programmer taking very much care while creating object, but neglecting destruction of useless objects. Because of his neglectance, total memory can be filled with useless objects which creates memory problems and total application will be down with Out of memory error.

Garbage Collection

But in Python, We have some assistant which is always running in the background to destroy useless objects. This assistant is nothing but Garbage Collector.

Hence the main objective of Garbage Collector is to destroy useless objects.
If an object does not have any reference variable then that object eligible for Garbage Collection.

When can we say that an object is eligible for Garbage Collection

If the object doesn't contain any reference variable then only it is eligible for Garbage Collection

i.e. if the reference count is zero then only object is eligible for Garbage Collection

How to Enable / Disable Garbage Collector

How to Enable / Disable Garbage Collector

1. gc.isenabled() Returns True if GC enabled
2. gc.disable() To disable GC explicitly
3. gc.enable() To enable GC explicitly

Destructor

Destructor is a special method and the name should be __del__

Just before destroying an object Garbage Collector always calls destructor to perform clean up activities (Resource deallocation activities like close database connection etc).

Once destructor execution completed then Garbage Collector automatically destroys that object.

Note

The job of destructor is not to destroy object and it is just to **perform clean up activities.**

Example

Performing CleanUp Activities

(Destructor)

Demo Program to Understand

Destructor

3

Important Questions For Interview Room

1

What is the Difference Between

del t1

And

t1 = None

del t1

t1 Deleted and Corresponding Object eligible for GC.
We cannot use t1.

If we don't want object and corresponding reference variable then we have to use this approach.

t1 = None

t1 Now onwards pointing to None object.
The object eligible for GC
We can use t1 (but its value is None)

If we want reference variable but not object then we have to use this approach

2

How to Find Number of References of an Object

3

Constructor

V/s

Destructor

Basis	Constructor	Destructor
Name	The Name of the Constructor should be __init__()	The Name of the Destructor should be __del__()
Objective	The main objective of Constructor is to perform Initialization Activities of an object	The main objective of Destructor is to perform Cleanup Activities of an object
Execution	Just after creating an object , PVM will execute constructor automatically	Just before destroying an object , GC will execute Destructor automatically

Which of the following is TRUE

1. The Main purpose of Constructor is to create an object
2. The main purpose of Constructor is to perform initialization of an obj
3. The Main Purpose of Destructor is to destroy the object.
4. The Main Purpose of Destructor is to perform cleanup activities

Using Members of One Class To Another Class

Using Member of One class inside another class

1. By Composition (**Has-A** Relationship)
2. By Inheritance (**IS-A** Relationship)

1 By Composition (Has-A Relationship)

By using Class Name or by creating object we can access members of one class inside another class is nothing but composition (Has-A Relationship).

The main advantage of Has-A Relationship is Code Reusability

Demo Lab

Demo Program - Has a Relationship - 113

Main Pillars of OOP's

Inheritance

Abstraction

Polymorphism

Encapsulation

Inheritance

Parent

Child

INHERITANCE

(IS-A Relationship)

Whatever variables, methods and constructors available in the parent class by default available to the child classes and we are not required to rewrite.

Hence the main advantage of inheritance is Code Reusability and we can extend existing functionality with some more extra functionality.

How to Achieve Inheritance

```
class parent:
    def property(self):
        print("Property")
    def bank(self):
        print("Bank Balance")

class child(parent):
    def enjoy(self):
        print("I am Enjoying")

obj = child()
obj.property()
obj.bank()
```

Inheritance

Object Creation

Advantages

Inheritance

1. Reusability
2. Extendibility

Disadvantages

Inheritance

1. Memory Wastage (Slower Execution)
2. Tightly Coupled Classes

IS-A V/S Has-A

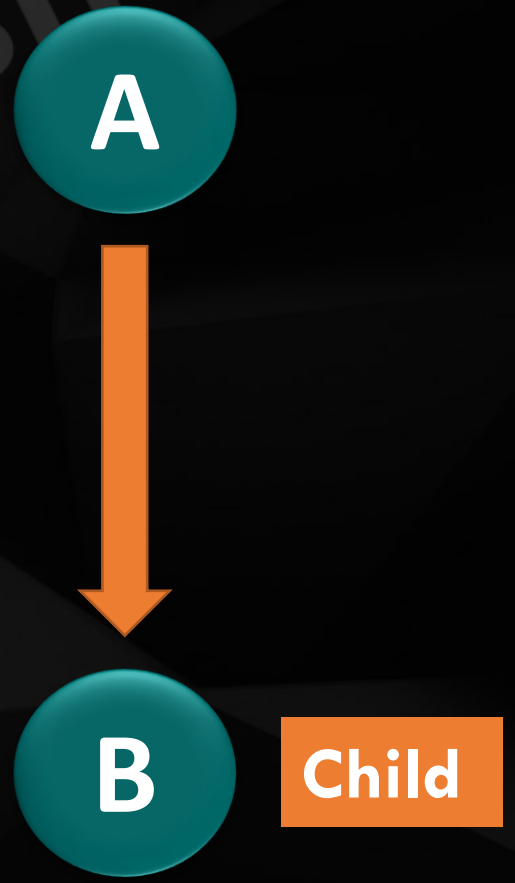
If we want to extend existing functionality with some more extra functionality then we should go for IS-A Relationship

If we don't want to extend and just we have to use existing functionality then we should go for HAS-A Relationship

Composition (Has-A Relationship)

To Access Members of class A
we have to create object of A
Class
obj = a()

Inheritance (IS-A Relationship)



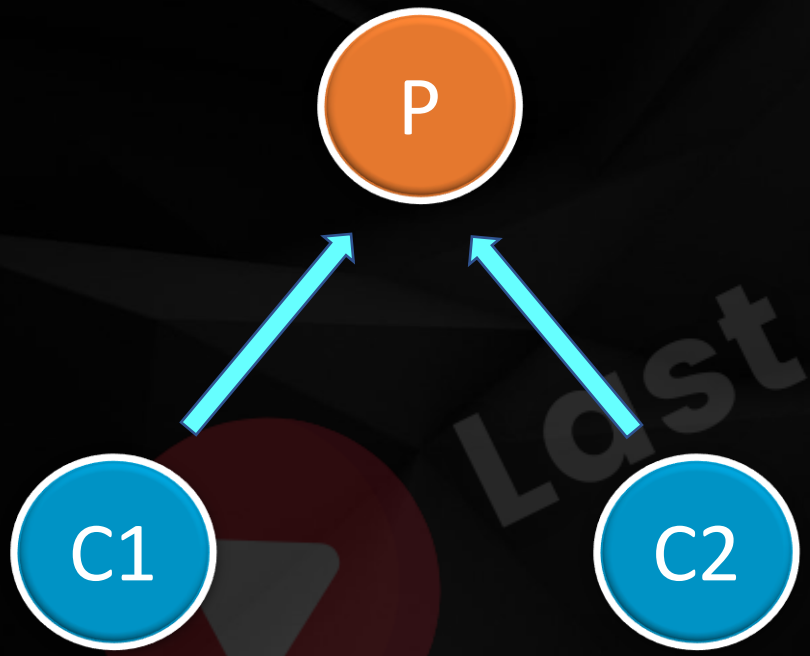
Types of Inheritance

Single Inheritance

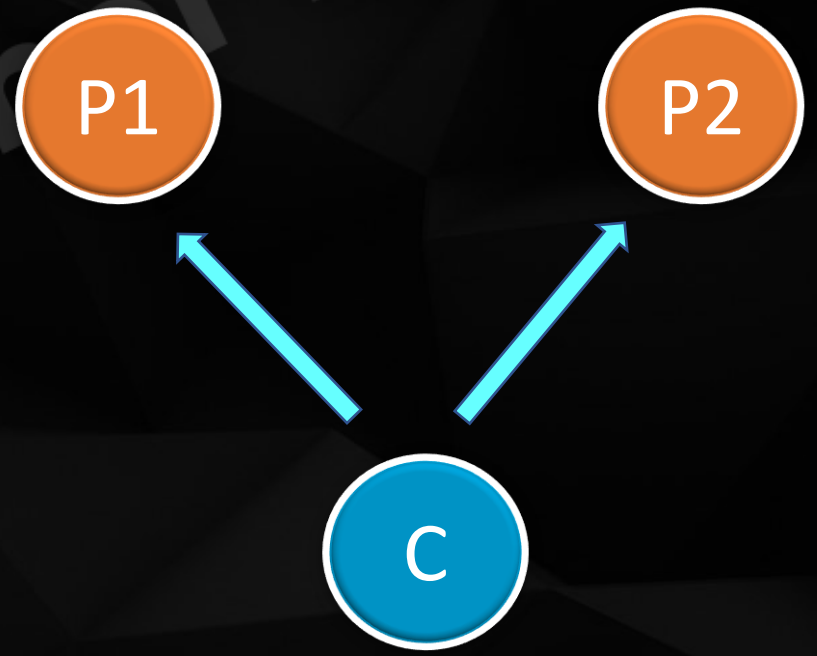
Multi Level Inheritance

Types of Inheritance

Hierarchical Inheritance

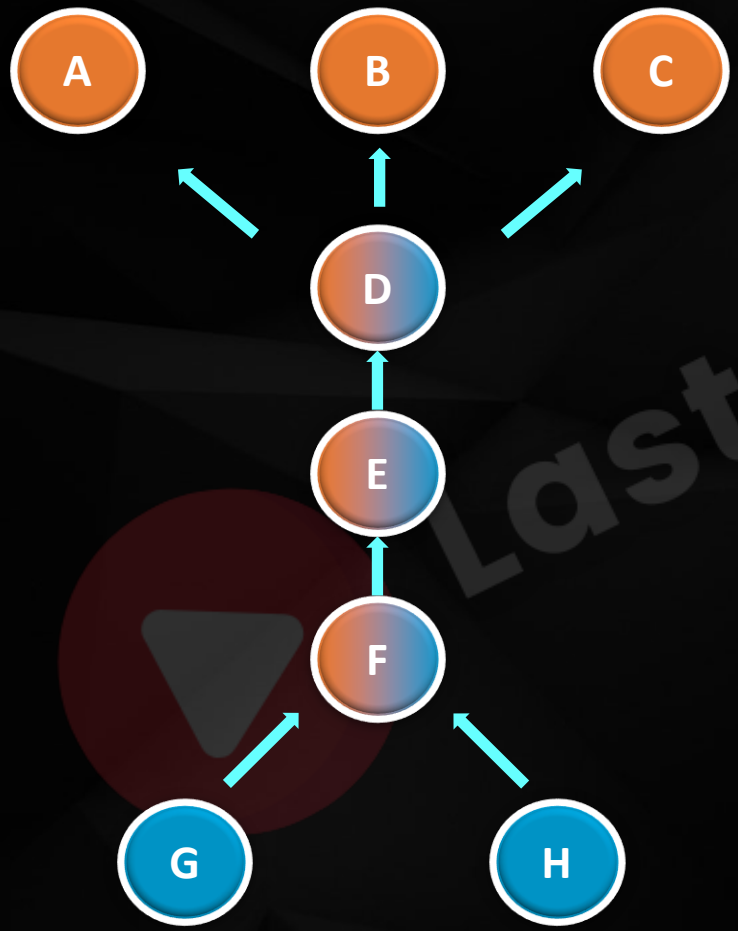


Multiple Inheritance

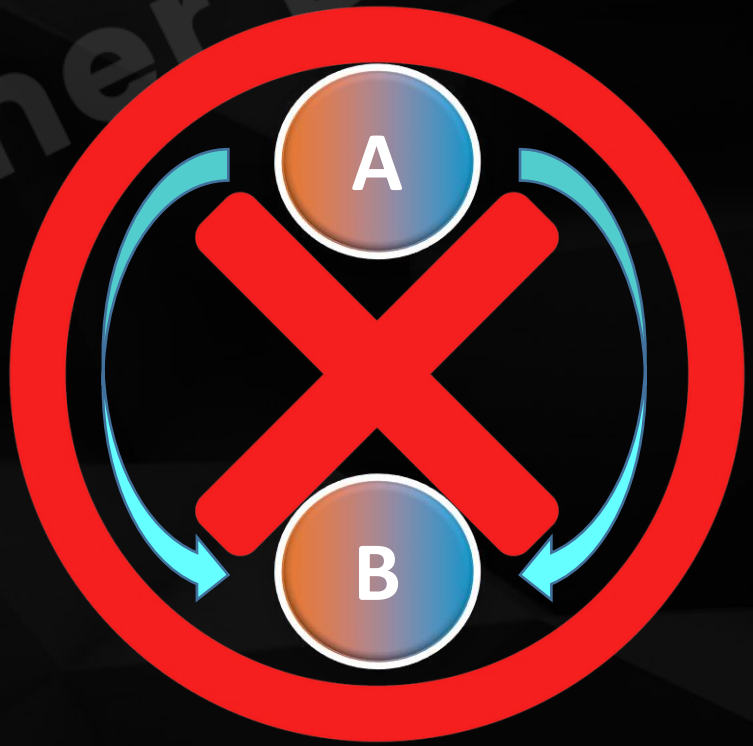


Types of Inheritance

Hybrid Inheritance

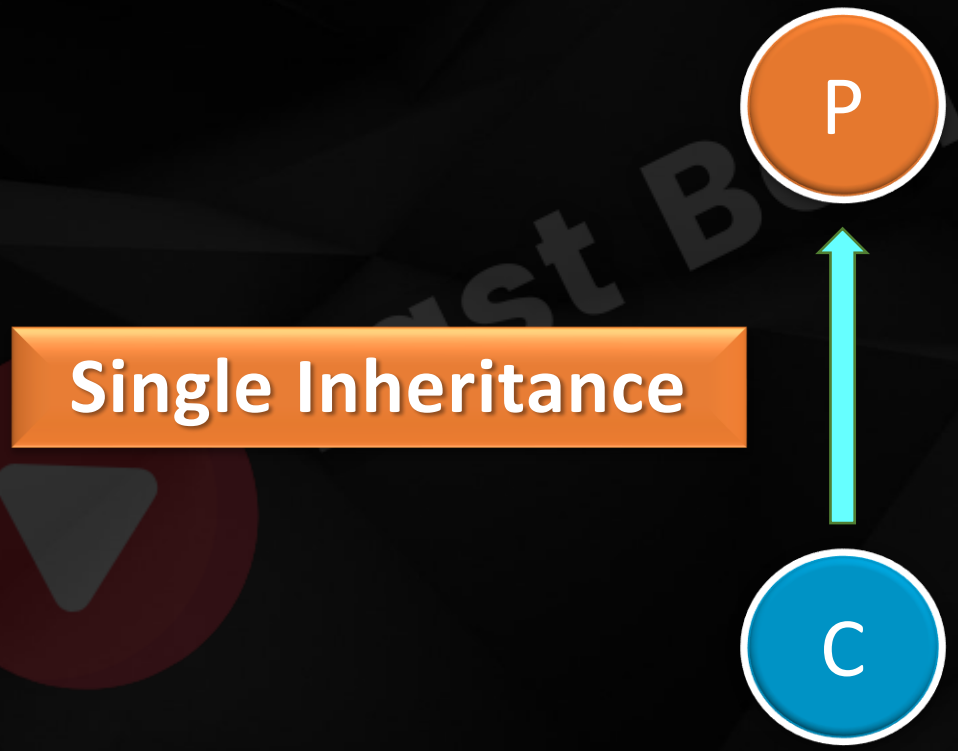


Cyclic Inheritance



Single Inheritance

The concept of inheriting the properties from one class to another class is known as single inheritance.



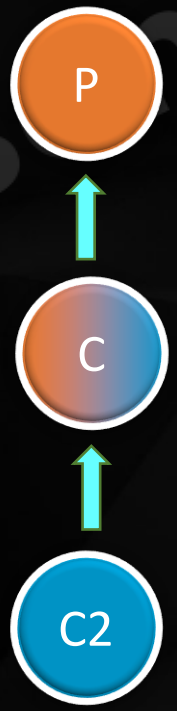
Single Inheritance

```
class P:
    def m1(self):
        print("Parent Class")

class C(P):
    def m2(self):
        print("Child Class")

obj = C()
obj.m1()
obj.m2()
```

Multi Level Inheritance

The concept of inheriting the properties from multiple classes to single class with the concept of one after another is known as multilevel inheritance

Multi Level Inheritance



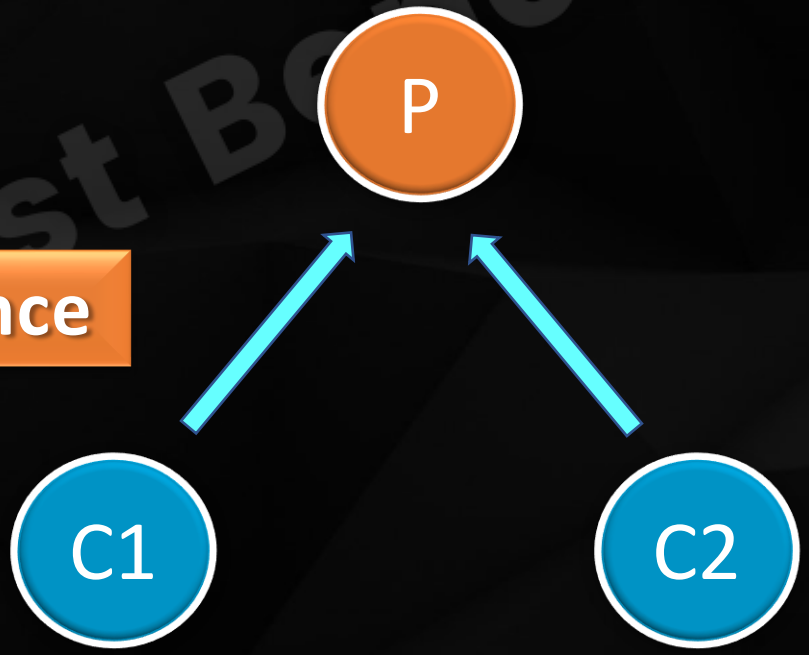
Multi Level Inheritance

```
class P:
    def m1(self):
        print("Parent Class")

class C(P):
    def m2(self):
        print("Child Class")

class C2(C):
    def m3(self):
        print("Sub Child Class")

obj = C2()
obj.m1()
obj.m2()
```

Hierarchical Inheritance

The concept of inheriting properties from one class into multiple classes which are present at same level is known as Hierarchical Inheritance

Hierarchical Inheritance

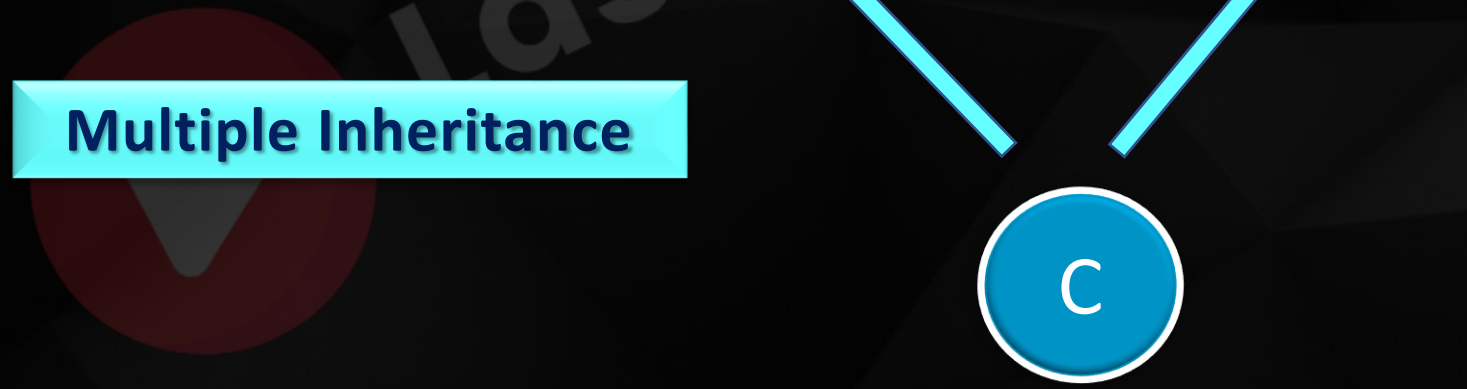


Hierarchical Inheritance

```python
class P:
    def m1(self):
        print("Parent Class")

class C1(P):
    def m2(self):
        print("Child1 Class")

class C2(P):
    def m3(self):
        print("Child2 Class")

obj = C1()
obj.m1()
obj.m2()
obj = C2()
obj.m1()
obj.m3()
```

Multiple Inheritance

The concept of inheriting the properties from multiple classes into a single class at a time, is known as Multiple inheritance.



Multiple Inheritance

```
class P1:
    def m1(self):
        print("Parent1 Class")

class P2:
    def m2(self):
        print("Parent2 Class")

class C(P1,P2):
    def m3(self):
        print("Child Class")

obj = C()
obj.m1()
obj.m2()
obj.m3()
```

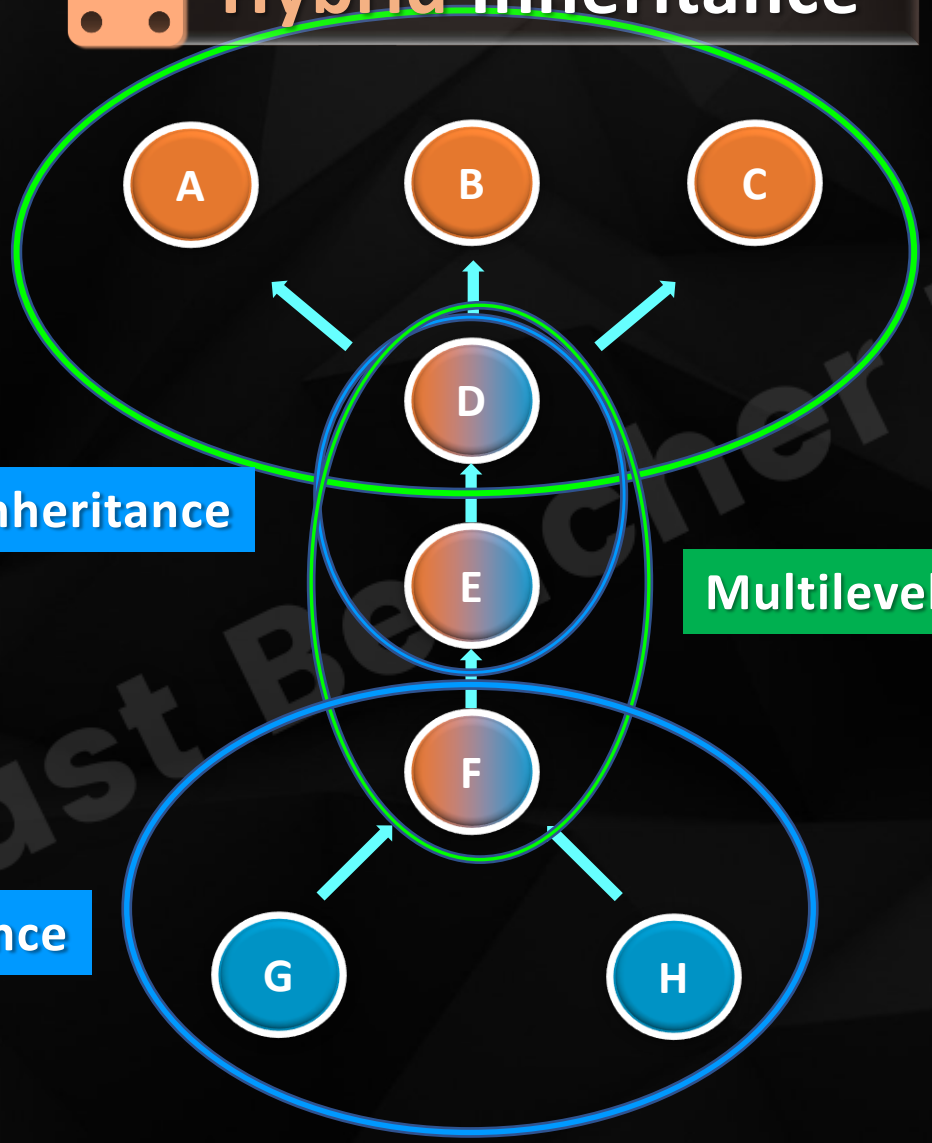
Note

If the same method is inherited from both parent classes , then Python will always consider the order of Parent classes in the declaration of the child class.

Hybrid Inheritance

Combination of Single, Multi level, multiple and Hierarchical inheritance is known as Hybrid Inheritance

Hybrid Inheritance



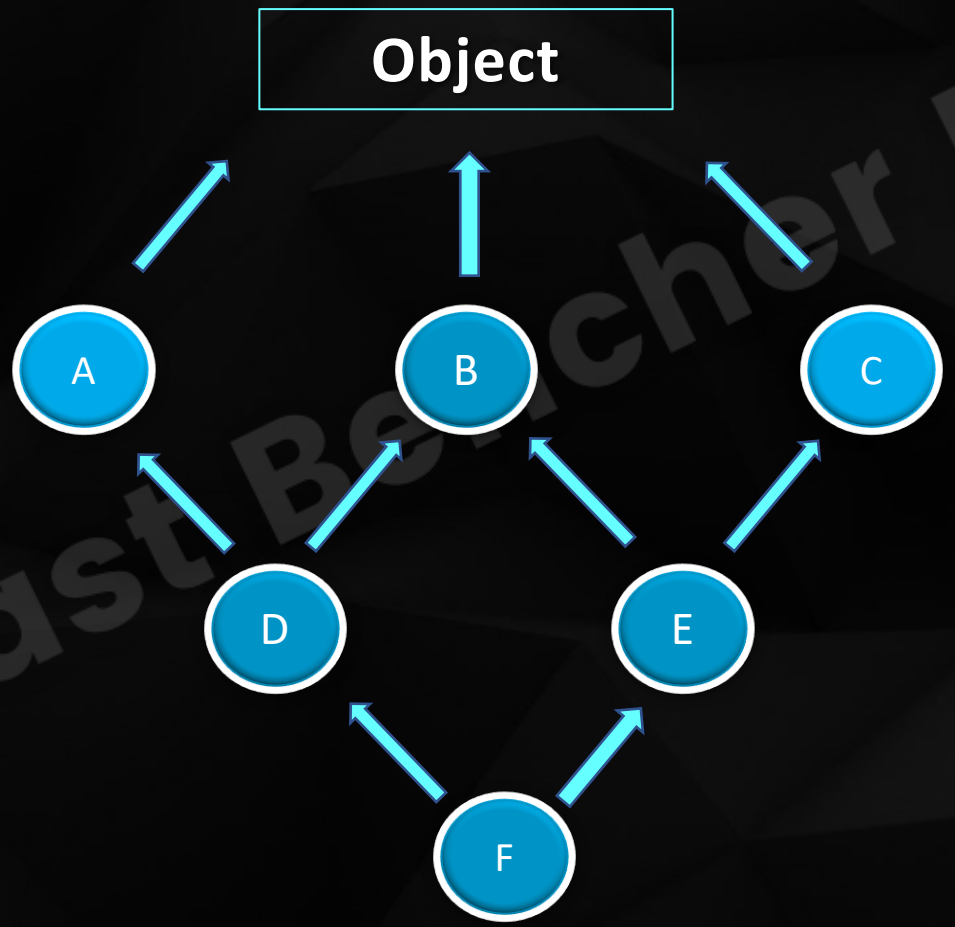
Multiple Inheritance

Single Inheritance

Multilevel Inheritance

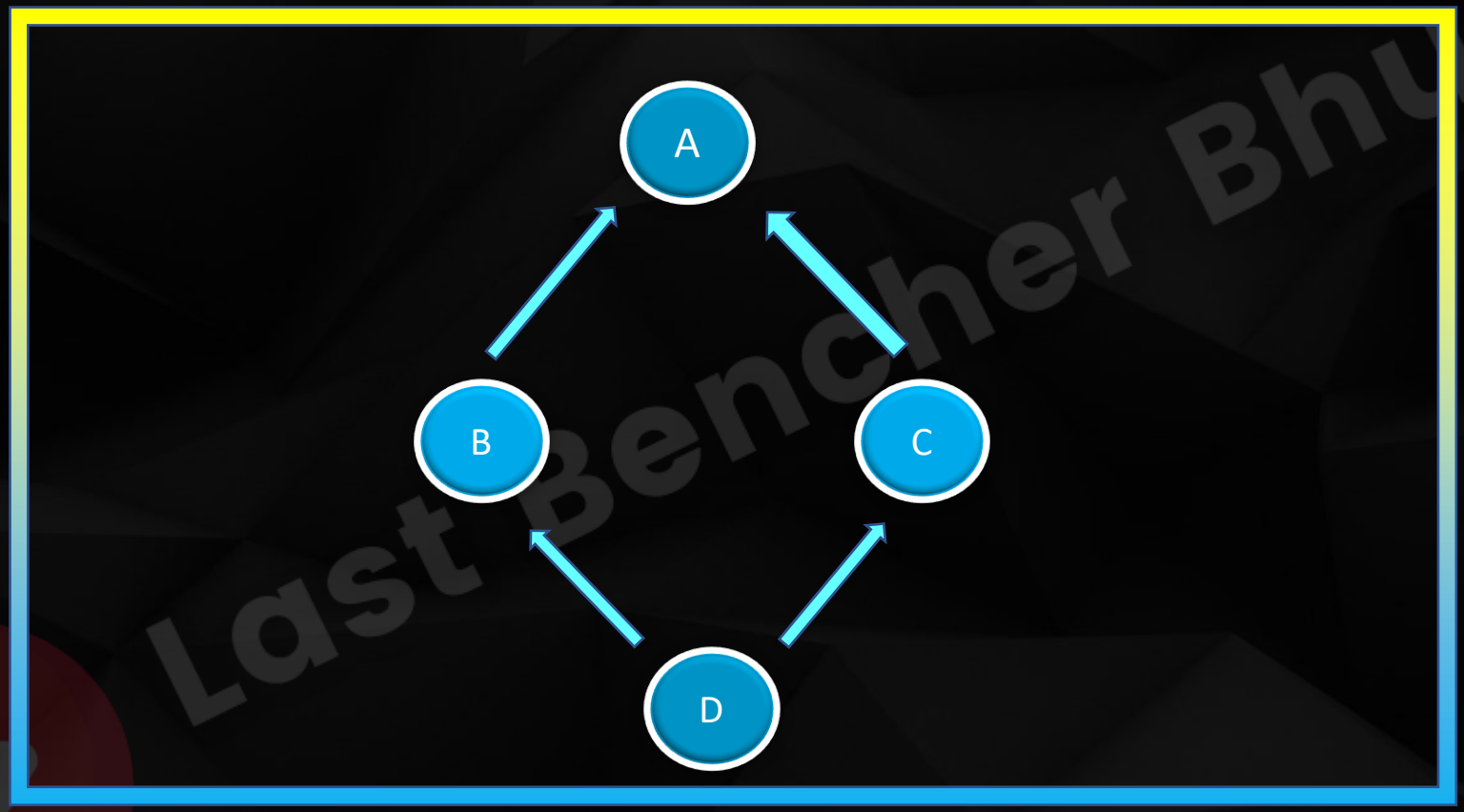
Hierarchical Inheritance

Hybrid Inheritance

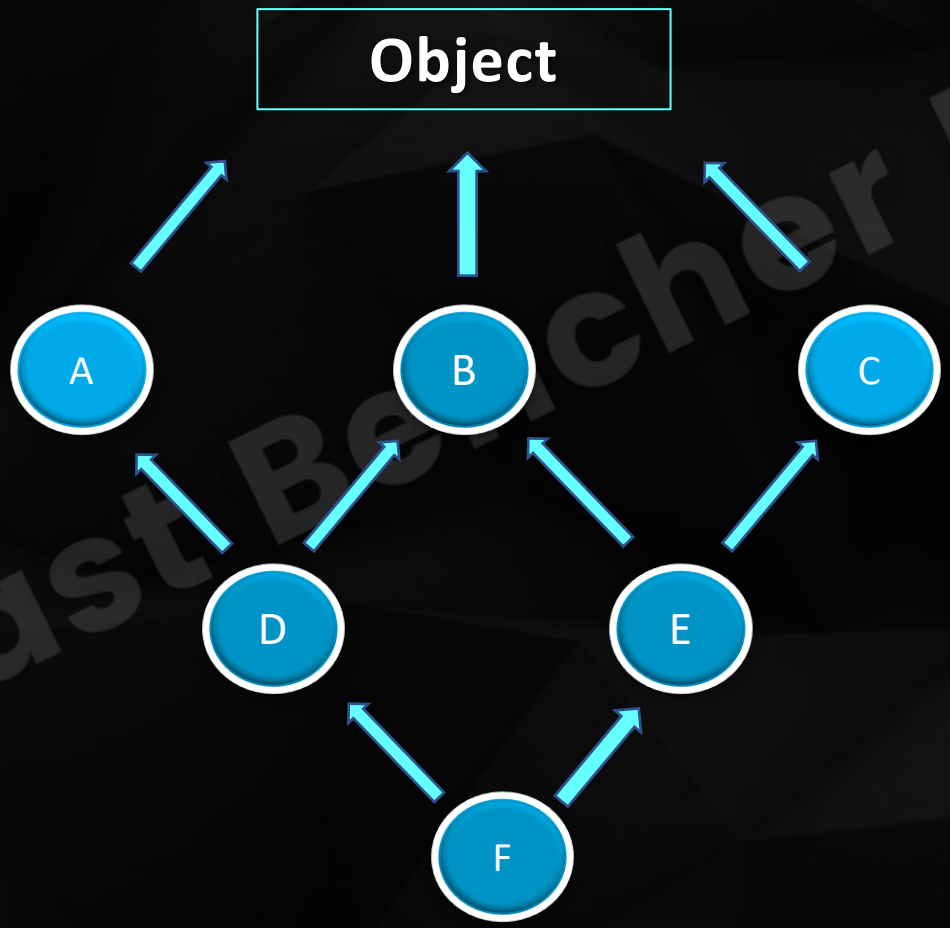


Method Resolution Order (MRO)

MRO



MRO



MRO

In Python, MRO stands for Method Resolution Order. It defines the order in which Python looks for methods and attributes in a hierarchy of classes during inheritance. MRO is particularly important when dealing with multiple inheritance because it ensures that the correct method or attribute is accessed from the appropriate class.

How to Call

Parent Class Constructor
From
Child Class Constructor

INTRODUCING

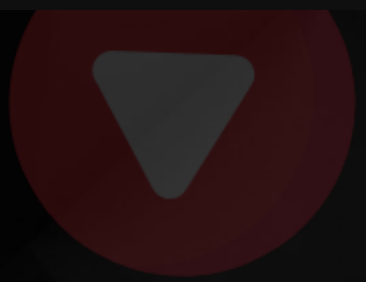
super()

super ( )

To understand about python super function, you have to know about Python Inheritance.
In Python Inheritance, the child class inherit from the parent class.
super() is a built-in method which is useful to call the parent class constructors , variables and methods from the child class

To Call the Parent Class Constructor

```python
class parent:
    def __init__(self):
        print("Parent Constructor Executed")

class child(parent):
    def __init__(self):
        print("Child Constructor Executed")

obj = child()
```



Hence to Call the Parent Class Constructor
We need super()

To Call the Parent Class Variable

```
class parent:
    a = 100

class child(parent):
    a = 200

    def display(self):
        print(self.a)          <-- 200
        print(super().a)       <-- 100

obj = child()
obj.display()
```

Hence, To Differentiate
Variables of Same name
We need super()

To Call the Parent Class Methods

```
class parent:
    def m1(self):
        print("Parent Class m1 Executed")

class child(parent):
    def m1(self):
        super().m1()
        print("Child Class m1 Executed")

obj = child()
obj.m1()
```

Hence to Differentiate
Methods of Same name
We need super()

super ( )

If Naming Conflict is there that means if parent class and child class having Constructors , Variables , Methods of same name then some mechanism must be required to differentiate which is nothing but

super()

Polymorphism

Poly means many

Morphs means forms

Polymorphism







Overloading

Overriding

Overloading

- Operator Overloading
- Method Overloading
- Constructor Overloading

Overriding

- Method Overriding
- Constructor Overriding

Operator Overloading

We can use the same operator for multiple purposes, which is nothing but operator overloading.

Python supports operator overloading.

INTRODUCING
Magic Methods

MAGIC METHOD

Python Magic methods are the methods starting and ending with double underscores '__'. They are defined by built-in classes in Python and commonly used for operator overloading.

Magic Methods

Operation	Symbol	Magic Method
Arithmetic Operators		
Addition	`+`	`__add__(self, other)`
Subtraction	`-`	`__sub__(self, other)`
Multiplication	`*`	`__mul__(self, other)`
True Division	`/`	`__truediv__(self, other)`
Floor Division	`//`	`__floordiv__(self, other)`
Modulus	`%`	`__mod__(self, other)`

Magic Methods

Comparison Operators		
Equal to	`==`	`__eq__(self, other)`
Not Equal to	`!=`	`__ne__(self, other)`
Less than	`<`	`__lt__(self, other)`
Less than or Equal	`<=`	`__le__(self, other)`
Greater than	`>`	`__gt__(self, other)`
Greater than or Equal	`>=`	`__ge__(self, other)`

Importance of __str__()

___str___()

1. Whenever we are trying to print any object reference , internally __str__() method will be called.

2. The Default implementation of this method returns the string in the following format : <__main__ Student object at 0x00..

3. To provide meaningful string representation for our object, we have to override __str__() method in our class

Customizing Operators In your Python Class

Method Overloading

If two methods having same name but different type of arguments then those methods are said to be overloaded methods.

But in Python Method overloading is not possible.
If we are trying to declare multiple methods with same name and different number of arguments then Python will always consider only last method.

Method Overloading

```
class Myclass:
    def m1(self):
        print('no-arg method')
    def m1(self,a):
        print('one-arg method')
    def m1(self,a,b):
        print('two-arg method')

t = Myclass()
#t.m1()
#t.m1(10)
t.m1(10,20)
```

Ignored

Considered

Why Python Won't Support Method Overloading

How We can Achieve Overloaded Method Requirements

In Python, method overloading is not possible;

if you really want access the same function with different features , you don't need to define the method again n again you do all it single definition of method

But HOW

How We can Achieve

Overloaded Method Requirements

1. By Providing Default Arguments
2. By Providing Variable Length Arguments

By Providing Default Arguments

```
class Myclass:
    def sum(self,a=None,b=None,c=None):
        if a!=None and b!= None and c!= None:
            print('The Sum of 3 Numbers:',a+b+c)
        elif a!=None and b!= None:
            print('The Sum of 2 Numbers:',a+b)
        else:
            print('Please provide 2 or 3 args')


t = Myclass()
t.sum(10,20)
t.sum(10,20,30)
t.sum(10)
```

By Providing Variable Length Arguments

```
class Myclass:
    def sum(self,*a):
        total=0
        for x in a:
            total=total+x
        print('The Sum:',total)

t = Myclass()
t.sum(10,20)
t.sum(10,20,30)
t.sum(10)
t.sum()
```

Constructor Overloading

If you understand Method Overloading well then Constructor overloading is nothing for you

Again Constructor overloading is not possible in Python.
If we define multiple constructors then the last constructor will be considered.

Constructor Overloading

```
class Myclass:
    def __init__(self):
        print('No-Arg Constructor')
    def __init__(self,a):
        print('One-Arg constructor')
    def __init__(self,a,b):
        print('Two-Arg constructor')

#t1=Myclass()
#t1=Myclass(10)
t1=Myclass(10,20)
```

Ignored

Considered

How We can Achieve

Overloaded Constructor Requirements

1. By Providing Default Arguments
2. By Providing Variable Length Arguments

Method Overriding

What ever members available in the parent class are by default available to the child class through inheritance.
If the child class not satisfied with parent class implementation then child class is allowed to redefine that method in the child class based on its requirement. This concept is called overriding.
Overriding concept applicable for both methods and constructors.

Constructor Overriding

Parent Class Constructor By default available to the child class. If child class not satisfied with the parent class constructor then child class is allowed to redefine the constructor based on its requirement

Demo Lab - 12

Understand Overriding Demo Program - 1

Basis	Method Overloading	Method Overriding
Definition	In the method overloading, methods must have the same name and different Parameters	Whereas in the method overriding, methods must have the same name and same Parameters
Also Known	It is also known as compile time polymorphism.	It is also known as run time polymorphism.
Inheritance	Inheritance not required	Inheritance is always required
Performed	Method overloading is performed between methods within the class.	Whereas method overriding is done between parent class and child class methods.

Public Member / Attributes

If any method or variable is public, then we can access that member from anywhere either within the class or from outside of the class

By Default every member present in python class is public

Private Member / Attributes

Private attributes can be accessed only within the class .i.e from outside of the class we cannot access.

We can declare a variable as private explicitly by prefexing with 2 underscore symbols.

How to access private variables from outside of the class:

Name Mangling in Python

We Cannot Access private members directly from outside of the class . But we can access indirectly as follows

Name Mangling will be happened for the private variables Hence every private variable name will be changed to new name

Name Mangling in Python

_ClassName__VariableName

SYNTAX

Protected Member / Attributes

Protected members we can access within the class from anywhere but from outside of the class we can access only in child classes.

We can declare members as protected explicitly by prefixing with one underscore symbol.

Public V/s Private V/s Protected

x = 10 → Public

__x = 10 → Private

_x = 10 → Protected

Abstraction

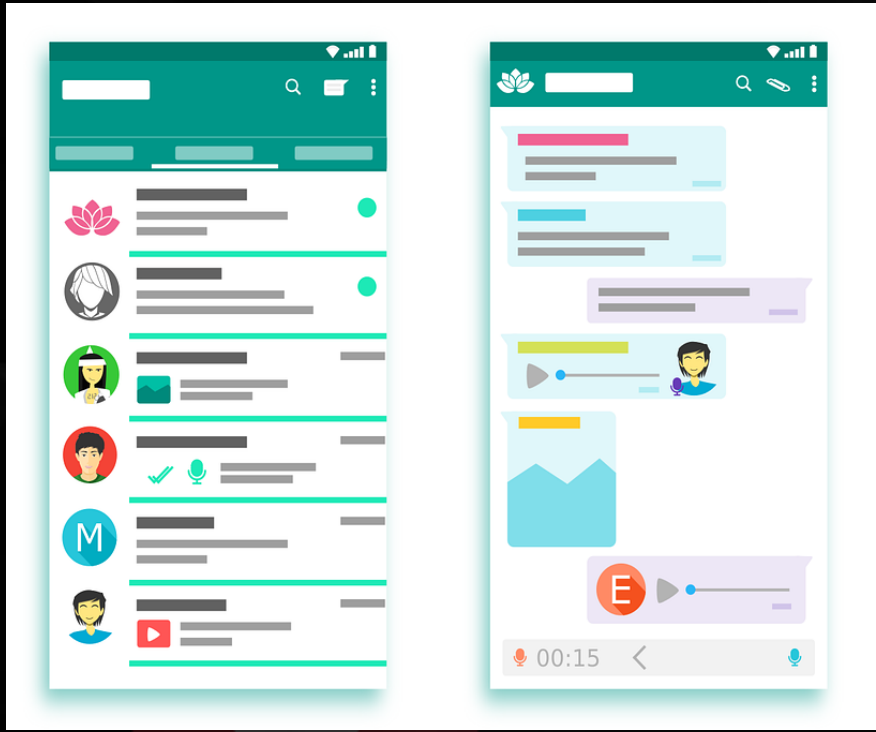
Abstraction

Abstraction in python is defined as a process of hiding unnecessary information from the user. This is one of the core concepts of object-oriented programming (OOPS) languages.

Abstraction

1. Abstraction as Hiding Complexity
2. Abstraction as Incompleteness

Abstraction in Real World



How to Achieve Abstraction

- Abstract Method
- Abstract Class
- Interface

Abstract Method

Sometimes we don't know about implementation , still we can declare a method. Such type of methods are called abstract methods .i.e abstract method has only declaration but not implementation.

In python we can declare abstract method by using @abstractmethod decorator as follows.

```
@abstractmethod
def m1(self): pass
```

Abstract Method

@abstractmethod decorator present in abc module. Hence compulsory we should import abc module , otherwise we will get error.

abc › abstract base class module

Child classes are responsible to provide implementation for parent class abstract methods.

Abstract Class

Some times implementation of a class is not complete , such type of partially implementation classes are called abstract classes.

Every abstract class in Python should be derived from ABC class which is present in abc module.

Note

If a class contains at least one abstract method and are extending ABC class then

instantiation is not possible

Case - 1

```
from abc import *
class Test:
    pass

t=Test()
```

A Possible

B Not Possible

In the above code we can create object for Test class bcoz it is concrete class and it does not contain any abstract method.

Case - 2

```
from abc import *
class Test(ABC):
    pass

t=Test()
```

A Possible

B Not Possible

In the above code we can create object, even it is derived from ABC class, bcoz it does not contain any abstract method.

Case - 3

```python
from abc import *
class Test(ABC):
    @abstractmethod
    def m1(self):
        pass

t=Test()
```

A Not Possible

B Possible

TypeError: Can't instantiate abstract class Test with abstract methods m1

Case - 4

```
from abc import *
class Test:
    @abstractmethod
    def m1(self):
        pass

t=Test()
```

A Possible

B Not Possible

We can create object even class contains abstract method bcoz we are not extending ABC class.

Note

If we are creating child class for abstract class, then for every abstract method of parent class compulsory we should provide implementation in the child class ,

Otherwise child class is also abstract and we cannot create object for child class

Class Can contain both abstract and non abstract methods ?

Yes

If a Class contains both abstract and non abstract methods then child is responsible to provide Implementation only for abstract methods

Interfaces

In general if an abstract class contains only abstract methods such type of abstract class is considered as interface.

There is no interface keyword in python which is there in java but we can implement interfaces concept through abstract class which we have already studied

Demo Lab - 12

Interface Demo Program - 1

Concrete Class

Abstract Class

Interface

Concrete class V/s Abstract class V/s Interfaces

1. If we don't know anything about implementation just we have requirement specification then we should go for interface.

2. If we are talking about implementation but not completely then we should go for abstract class. (partially implemented class)

3. If we are talking about implementation completely and ready to provide service then we should go for concrete class

Encapsulation

Encapsulation

The Process of binding / grouping / encapsulating data and corresponding behavior (methods) into a single unit is nothing but encapsulation.

OR

Wrapping the things together in a Single unit is known as Encapsulation

Data Hiding

Our Internal Data should not go out directly.

i.e. outside person should not access our internal data directly.

By declaring data members as private , we can implement Data Hiding

Encapsulation

A Python Class is also an Example of Encapsulation

Advantages

1. Security

2. Enhancement

SUMMARY

Inheritance

Abstraction

Polymorphism

Encapsulation