Prof R Madana Mohana

OBJECT ORIENTED PROGRAMMING (USING PYTHON)

INHERITANCE

Types of Inheritance

https://www.youtube.com/c/RASINENIMADANAMOHANA

OUTLINE

Types of Inheritance

1. Single Level Inheritance
2. Multiple Inheritance
3. Multi-level Inheritance
4. Hierarchical Inheritance
5. Multi-path or Hybrid Inheritance

Types of Inheritance

- **Python** supports different types or variants of Inheritance.

There are 5 types of Inheritance:

1. Single Level Inheritance
2. Multiple Inheritance
3. Multi-level Inheritance
4. Hierarchical Inheritance
5. Multi-path or Hybrid Inheritance

Single Level Inheritance

- The concept of inheriting properties from only one class into another class is known as a **single inheritance**.

Syntax :

```
class BaseClassname:
    Body of base class
class DerivedClassname(BaseClassname):
    Body of derived class
```

Single Level Inheritance

Example: Program to implement single inheritance

```
class A:
    def m1(self):
        print("m1 is method of Main or Base Class A..")
class B(A): #Class B is derived from the Parent or Base Class
A
    def m2(self):
        print("m2 is method of Derived or Sub or Child Class
B..")
b1 = B()
b1.m1()
b1.m2()
```

Single Level Inheritance

Example: Program to implement single inheritance

OUTPUT:

```
m1 is method of Main or Base Class A..
m2 is method of Derived or Sub or Child Class B..
```

Multiple Inheritance

- When a **derived class** inherits features from more than one **base class**, it is called **multiple inheritance**.
- The **derived class** has all the features of both the **base classes** and in addition to them can have additional new features.

Multiple Inheritance

- The syntax for **multiple inheritance** is similar to that of **single inheritance** and can be given as:

```
class Base1:
    statement block
class Base2:
    statement block
class Derived (Base1, Base2):
    statement block
```

Multiple Inheritance

Example: Program to implement multiple inheritance

```
class Base1: #First Base Class
    def __init__(self):
        super(Base1,self).__init__()
        print("Base1 Class..")
class Base2:#Second Base Class
    def __init__(self):
        super(Base2,self).__init__()
        print("Base2 Class..")
```

Multiple Inheritance

Example: Program to implement multiple inheritance

```
class Derived(Base1, Base2): #Derived class
derived from 2 base classes Base1 & Base2
    def __init__(self):
        super(Derived,self).__init__()
        print("Derived Class..")
Derived()
```

Multiple Inheritance

Example: Program to implement multiple inheritance

OUTPUT:

```
Base2 Class..
Base1 Class..
Derived Class..
```

Multiple Inheritance

- In the above program output order is based on **Method Resolution Order (MRO)**, it works on depth-first traversal.
- The set of rules used to find the ***linearization order*** is called **Method Resolution Order (MRO)**.
- The **MRO** ensures that a class appears before its parent classes.

Multiple Inheritance

- However, in case of **multiple inheritance**, the **MRO** is same as a tuple of base classes.
- We can check the **MRO** of a class by either the **__mro__** attribute or **mro()** method.
- While the **__mro__** attribute returns a **tuple**, the **mro()** method returns a **list**.

Multiple Inheritance

- In the above program the order of class hierarchy can be given as:

  Derived -> Base1 and **Derived -> Base2.**

- When we create an instance of the derived class, the following things happen.

Step-1: The **__init__()** method of Derived class is called.

Step-2: The **__init__()** method of Base1 class is invoked (according to **MRO**) from the **__init__()** method of Derived class.

Multiple Inheritance

Step-3: The **__init__()** method of Base2 class is invoked (according to **MRO**) from the **__init__()** method of Base1 class.

Step-4: From the **__init__()** method of Base2 class the **__init__()** method of Object is invoked which does nothing. Finally, Base2 class gets printed on the screen and the control is returned to the **__init__()** method of Base1 class.

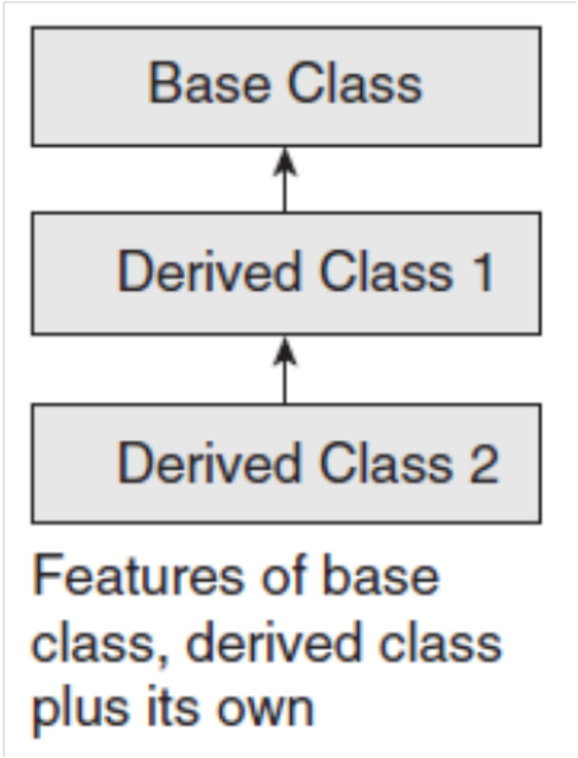
Step-5: Base1 class gets printed and the control is transferred back to the **__init__()** method of Derived class.

Step-6: Derived class gets printed on the screen and hence the result.

Multi-level Inheritance

- **Multi-Level inheritance** is possible in **python** like other object-oriented programming languages.
- The technique of deriving a class from an already derived class is called **multi - level inheritance**.
- In **multi - level inheritance**, number of levels can go up to any number based on the requirement.

Multi-level Inheritance



- In the above Fig., **Base Class** acts as the base for ***Derived Class1*** which in turn acts as a base for ***Derived Class2.***

Multi-level Inheritance

- The ***Derived Class1*** has the features of Base Class plus its own features.
- The ***Derived Class1*** is known as the intermediate class as this class provide a link for inheritance between the ***Base Class*** and the ***Derived Class2***.
- The chain of classes:

Base Class --> Derived Class1 --> Derived Class2 is known as the ***inheritance path***.

Multi-level Inheritance

The syntax of **multi-level inheritance** is given below:

```
class Base:
    body of the Base class
class Derived1(Base):
    body of the Derived1 class
class Derived2(Derived1):
    body of the Derived2 class
```

Multi-level Inheritance

- In **Multi-level inheritance** scenario, any specified attribute is first searched in the *current class* (***Derived Class2***). If it is not found there, then the ***Derived Class1*** is searched, if it is not found even there, then the ***Base Class*** is searched. If the attribute is still not found, then finally the object class is checked. This order is also called ***linearization*** of ***Derived Class2***.
- Correspondingly, the set of rules used to find this *linearization order* is called **Method Resolution Order(MRO).**

Multi-level Inheritance

Example: Program to implement multi-level inheritance

```
class Person: # Base class
    def name(self):
        faculty_name = input("Enter Eligible
Faculty Name for HOD position..")
        print("Name:", faculty_name)
```

Multi-level Inheritance

Example: Program to implement multi-level inheritance

```
class Teacher(Person):# Class Teacher derived
from the Base class Person
    def Qualification(self):
        print("Qualification of HOD is Ph.D
mandatory:")
```

Multi-level Inheritance

Example: Program to implement multi-level inheritance

```
class HOD(Teacher): # Class HOD is derived
from the Derived class Teacher
    def Experience(self):
        print("Experience of HOD is at least
15 years:")
```

Multi-level Inheritance

Example: Program to implement multi-level inheritance

```
print(Person.__mro__)
h = HOD()
h.name()
h.Qualification()
h.Experience()
```

Multi-level Inheritance

Example: Program to implement mult-level inheritance

OUTPUT:

```
Enter Eligible Faculty Name for HOD position..
Dr R MADANA MOHANA
Name: Dr R MADANA MOHANA
Qualification of HOD is Ph.D mandatory:
Experience of HOD is at least 15 years:
(<class '__main__.HOD'>, <class '__main__.Teacher'>, <class
'__main__.Person'>, <class 'object'>)<class 'object'>)
```

Multi-level Inheritance

Example: Program to implement mult-level inheritance

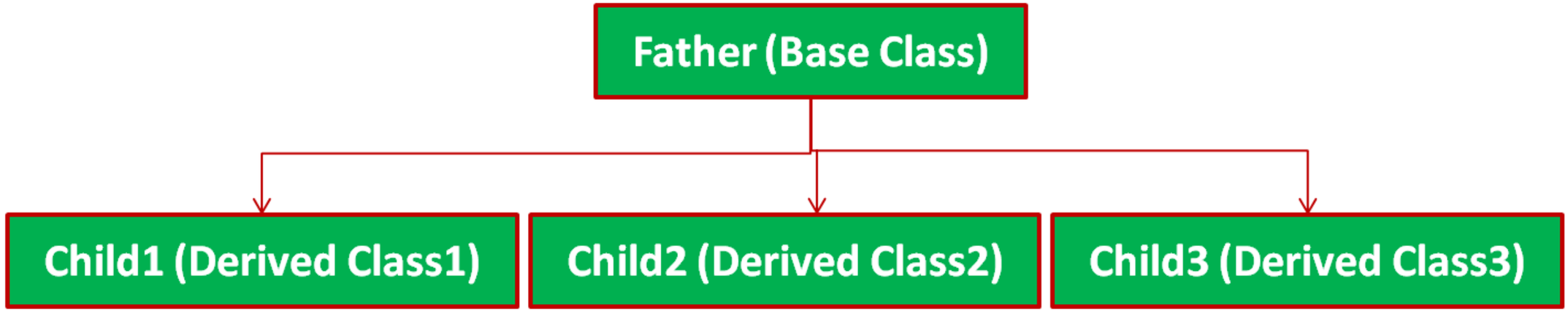
OUTPUT:

```
Enter Eligible Faculty Name for HOD position..
Dr R MADANA MOHANA
Name: Dr R MADANA MOHANA
Qualification of HOD is Ph.D mandatory:
Experience of HOD is at least 15 years:
(<class '__main__.HOD'>, <class '__main__.Teacher'>, <class
'__main__.Person'>, <class 'object'>)<class 'object'>)
```

Hierarchical Inheritance

- The concept of inheriting properties from one class into multiple classes is known as a **hierarchical inheritance**.
- When more than one derived class are created from a single base class, this type of inheritance is called **hierarchical inheritance.**
- It contains only one base class and multiple derived classes.

Hierarchical Inheritance



In the above Fig., **Father** is only one ***Base Class***. ***Child1***, ***Child2*** & ***Child3*** are three **sub** or ***derived classes*** derived from ***Base Class*** **Father**.

Hierarchical Inheritance

Syntax:

```
class A:
    body of base class
class B(A):
    body of derived-1 class
class C(A):
    body of derived-2 class
```

Hierarchical Inheritance

Ex: Program to implement hierarchical inheritance

```
class A: # Base class
    def m1(self):
        print("I am m1 of Base Class A!")
class B(A): # Derived Class1 from Base Class A
    def m2(self):
        print("I am m2 of Derived Class B!")
class C(A): # Derived Class2 from Base Class A
    def m3(self):
        print("I am m3 of Derived Class C!")
class D(A): # Derived Class3 from Base Class A
    def m4(self):
        print("I am m4 of Derived Class D!")
```

Hierarchical Inheritance

Ex: Program to implement hierarchical inheritance

```
b1 = B()
b1.m1() # derived from Base Class A
b1.m2() # Own method of Class B
c1 = C()
c1.m1() # derived from Base Class A
c1.m3() # Own method of Class C
d1 = D()
d1.m1() # derived from Base Class A
d1.m4() # Own method of Class D
```

Hierarchical Inheritance

Ex: Program to implement hierarchical inheritance

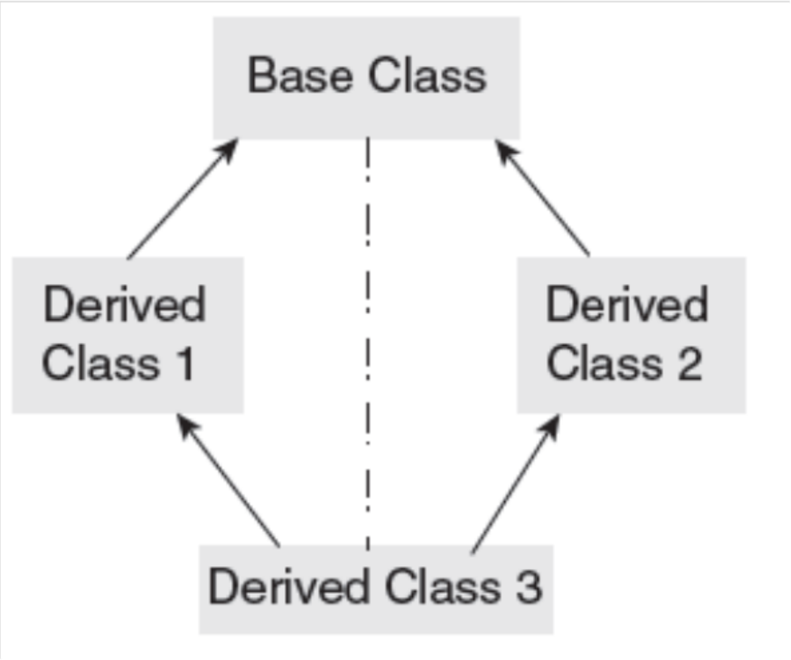
OUTPUT:

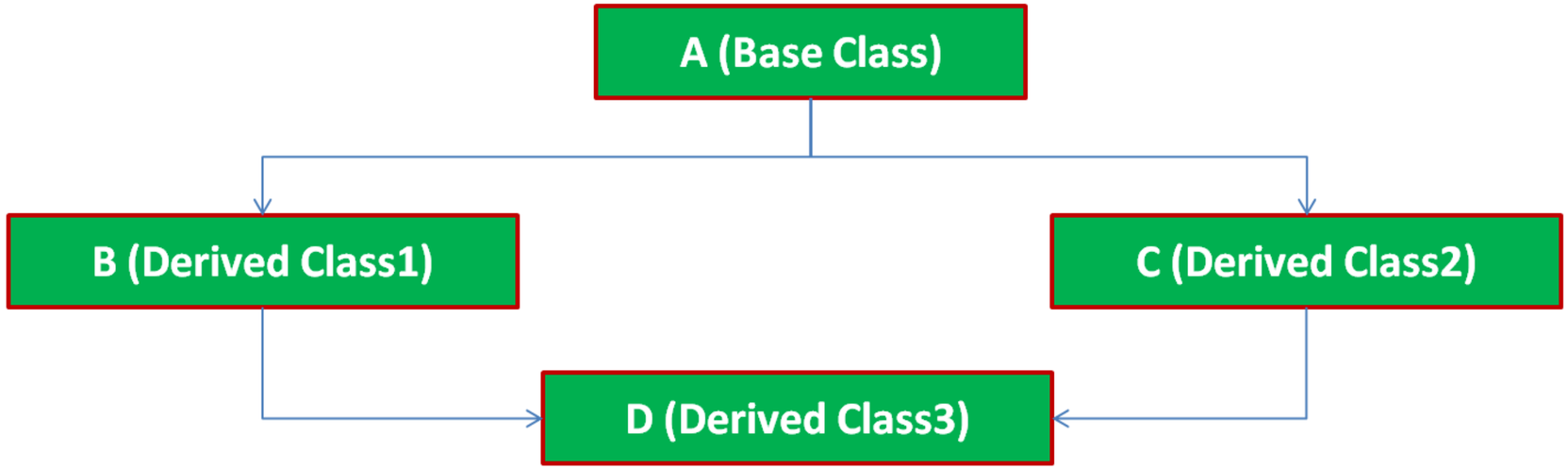
```
I am m1 of Base Class A!
I am m2 of Derived Class B!
I am m1 of Base Class A!
I am m3 of Derived Class C!
I am m1 of Base Class A!
I am m4 of Derived Class D!
```

Multi-path or Hybrid Inheritance

- Deriving a class from other derived classes that are in turn derived from the same base class is called **multi-path or Hybrid inheritance**.
- **Hybrid Inheritance** is that type in which we combine two or more types of inheritance.
- It is the combination of ***sinlge + Hierarchical + Multiple + Multilevel Inheritances*** are called **Hybrid or Multi-path Inheritance.**

Multi-path or Hybrid Inheritance

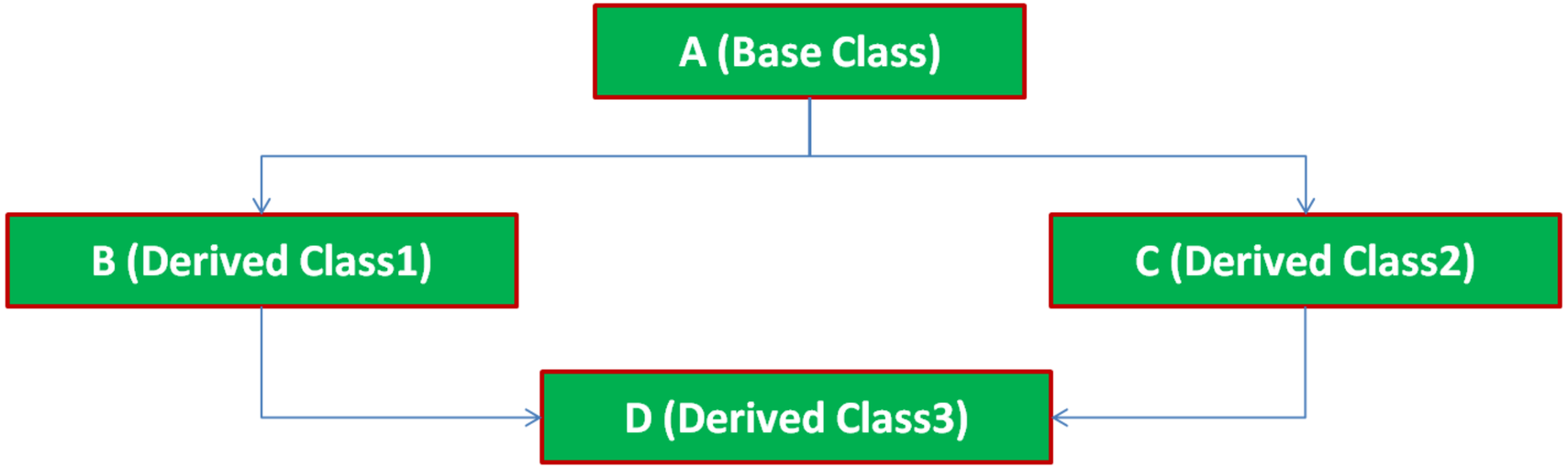




Multi-path or Hybrid Inheritance

- In the above Fig., the ***Derived Class3*** has two immediate Base classes **Derived Class1** and ***Derived Class2***. Both these base classes are themselves derived from the ***Base Class***, thereby forming a grand parent, parent, and child form of a relationship.
- The Derived Class inherits the features of of the B***ase Class*** (Grand Parents) via two separate paths. Therefore, the ***Base Class*** is also known as the ***Indirect Base Class***.

Multi-path or Hybrid Inheritance



- **A --> B, A --> C, B --> D & C --> D** are **Single Inheritances**
- **A --> B-->D & A --> C-->D** are **Multi-level Inheritances**
- **B, C --> D** is **Multiple Inheritance**
- **A-->B & A-->C** is **Hierarchical Inheritance**

Multi-path or Hybrid Inheritance

Syntax:

```
class A:
    body of class A
class B(A):
    body of class B
class C(A):
    body of class C:
class D(B, C):
   body of class D
```

Multi-path or Hybrid Inheritance

Ex. Program to demonstrate multi-path inheritance

```
class Student: # Base class
    def name(self):
        print("Name:")
class Academic_Performance(Student): # Derived
Class1 from Base Class Student
    def Academic_Score(self):
        print("Overall CGPA is 9.7 and above out
of 10")
```

Multi-path or Hybrid Inheritance

Ex. Program to demonstrate multi-path inheritance

```
class Written_Test(Student): # Derived Class2 from
Base Class Student
    def Written_test_Score(self):
        print("Written Test Score is 65% and
above")
```

Multi-path or Hybrid Inheritance

Ex. Program to demonstrate multi-path inheritance

```
class Result(Academic_Performance, Written_Test ): #
Derived Class3 from Derived Classes Academic_Performance
& Written_Test
    def Eligibility(self):
        print("---Minimum Eligibility to Apply for
Placements in Core Tier-1 Companies----")
        self.Academic_Score()
        self.Written_test_Score()
r1 = Result()
r1.Eligibility()
```

Multi-path or Hybrid Inheritance

Ex. Program to demonstrate multi-path inheritance

OUTPUT:

```
---Minimum Eligibility to Apply for
Placements in Core Tier-1 Companies----
Overall CGPA is 9.7 and above out of 10
Written Test Score is 65% and above
```