



IIT Madras

ONLINE DEGREE

Programming in Python
Professor Sudarshan Iyengar
Department of Computer Science and Engineering
Indian Institute of Technology, Ropar
Inheritance and Method Overriding

(Refer Slide Time: 00:16)

Inheritance and Method Overriding

Inheritance and Method Overriding



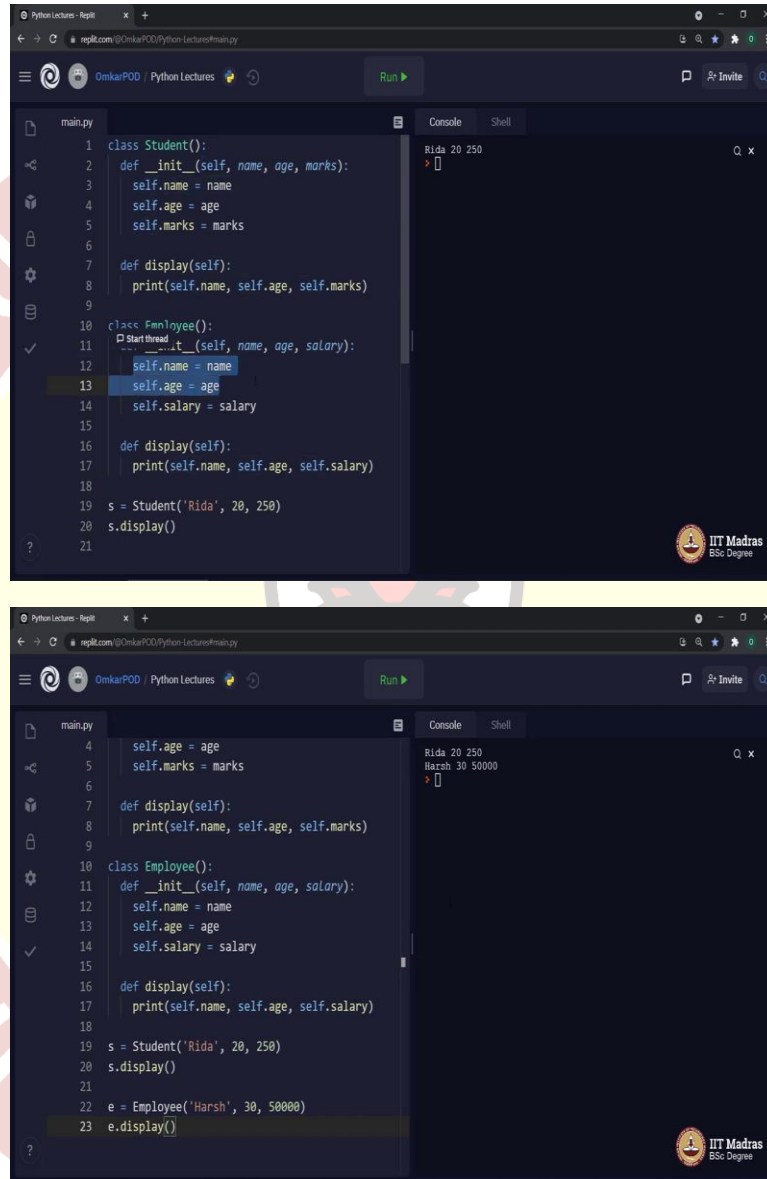
Hello Python students. In this lecture, we will continue our discussion on object oriented programming concepts. Consider a real life scenario where a child acquires properties of parents by birth. For example, your father likes to paint and he is very good at it. And due to that, you are also a good painter without even taking any formal training. Similarly, consider your mother has brown hairs. And once again, due to that, you also have brown hairs.

And as I have explained in first lecture, OOP allows us to translate real life objects into Python code. Therefore, there has to be some mechanism through which we can translate this relation between you and your parents into Python code. And with respect to OOP concepts the name given to this mechanism is inheritance. Now, because of this inheritance, you know painting and you have brown hairs. But this does not stop you from learning, let us say, swimming or you can color your hairs red.

In first case, you are adding one more quality to your existing one, whereas in second case, you are replacing your existing property with something else. Once again, you should have some way to handle both these situations as well and the solution is method overriding. I assume you must

have understood the basic idea behind these OOP concepts. So, now let us switch to Python editor and start coding using these concepts.

(Refer Slide Time: 02:19)



The image displays two screenshots of a Python IDE, likely Replit, showing the implementation of two classes: Student and Employee. The IDE interface includes a file explorer on the left, a code editor in the center, and a console on the right. The background features a large, semi-transparent watermark of the Indian Institute of Technology Madras logo.

Top Screenshot: The code editor shows the initial implementation of the Student class. The Employee class is partially visible but not yet fully defined. The console shows the output of the Student class instantiation.

```
1 class Student():
2     def __init__(self, name, age, marks):
3         self.name = name
4         self.age = age
5         self.marks = marks
6
7     def display(self):
8         print(self.name, self.age, self.marks)
9
10 class Employee():
11     def __init__(self, name, age, salary):
12         self.name = name
13         self.age = age
14         self.salary = salary
15
16     def display(self):
17         print(self.name, self.age, self.salary)
18
19 s = Student('Rida', 20, 250)
20 s.display()
21
```

Bottom Screenshot: The code editor shows the completion of the Employee class. The console now displays the output of both the Student and Employee class instantiations.

```
4     self.age = age
5     self.marks = marks
6
7     def display(self):
8         print(self.name, self.age, self.marks)
9
10 class Employee():
11     def __init__(self, name, age, salary):
12         self.name = name
13         self.age = age
14         self.salary = salary
15
16     def display(self):
17         print(self.name, self.age, self.salary)
18
19 s = Student('Rida', 20, 250)
20 s.display()
21
22 e = Employee('Harsh', 30, 50000)
23 e.display()
```

The console output for the top screenshot is:

```
Rida 20 250
```

The console output for the bottom screenshot is:

```
Rida 20 250
Harsh 30 50000
```

```
Python Lectures - Replit
replit.com/@OmkarPOD/Python-Lectures/main.py

main.py
1 class Person:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
5
6     def display(self):
7         print(self.name, self.age)
8
9 class Student(Person):
10     def __init__(self, name, age, marks):
11         self.marks = marks
12
13     def display(self):
14         print(self.marks)
15
16 class Employee(Person):
17     def __init__(self, name, age, salary):
18         self.salary = salary
19
20     def display(self):
21         print(self.salary)
22
```

Console

```
Rishi 20 250
Harsh 30 50000
>
```

IIT Madras
BSc Degree

```
Python Lectures - Replit
replit.com/@OmkarPOD/Python-Lectures/main.py

main.py
1 class Person:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
5
6     def display(self):
7         print(self.name, self.age)
8
9 class Student(Person):
10     def __init__(self, name, age, marks):
11         super().__init__(name, age)
12         self.marks = marks
13
14     def display(self):
15         print(self.marks)
16
17 class Employee(Person):
18     def __init__(self, name, age, salary):
19         super().__init__(name, age)
20         self.salary = salary
21
22     def display(self):

```

Console

```
250
50000
>
```

IIT Madras
BSc Degree

```
Python Lectures - Replit
replit.com/@OmkarPOD/Python-Lectures#main.py

main.py
11 super().__init__(name, age)
12 self.marks = marks
13
14 def display(self):
15     super().display()
16     print(self.marks)
17
18 class Employee(Person):
19     def __init__(self, name, age, salary):
20         super().__init__(name, age)
21         self.salary = salary
22
23     def display(self):
24         super().display()
25         print(self.salary)
26
27 s = Student('Rida', 20, 250)
28 s.display()
29
30 e = Employee('Harsh', 30, 50000)
31 e.display()
```

Console

```
Rida 20
250
Harsh 30
50000
```

IIT Madras
BSc Degree

```
Python Lectures - Replit
replit.com/@OmkarPOD/Python-Lectures#main.py

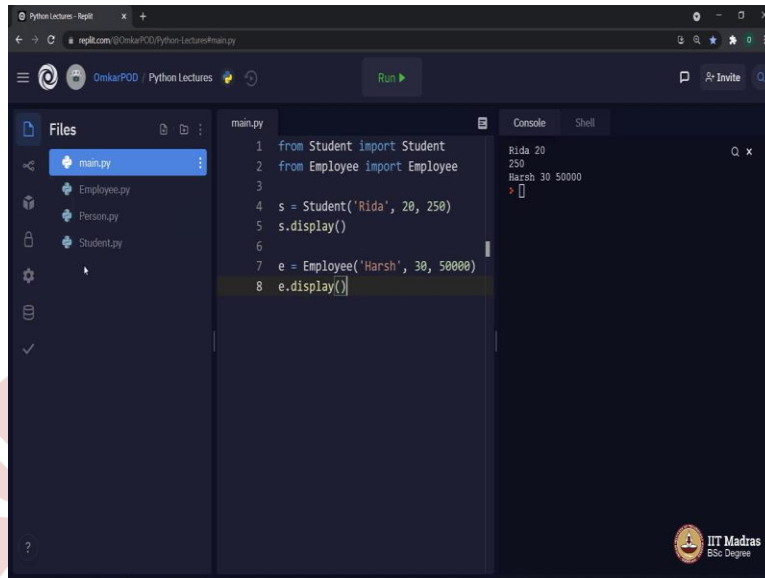
Files
main.py
Employee.py
Person.py
Student.py

main.py
1 s = Student('Rida', 20, 250)
2 s.display()
3
4 e = Employee('Harsh', 30, 50000)
5 e.display()
```

Console

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    s = Student('Rida', 20, 250)
NameError: name 'Student' is not defined
```

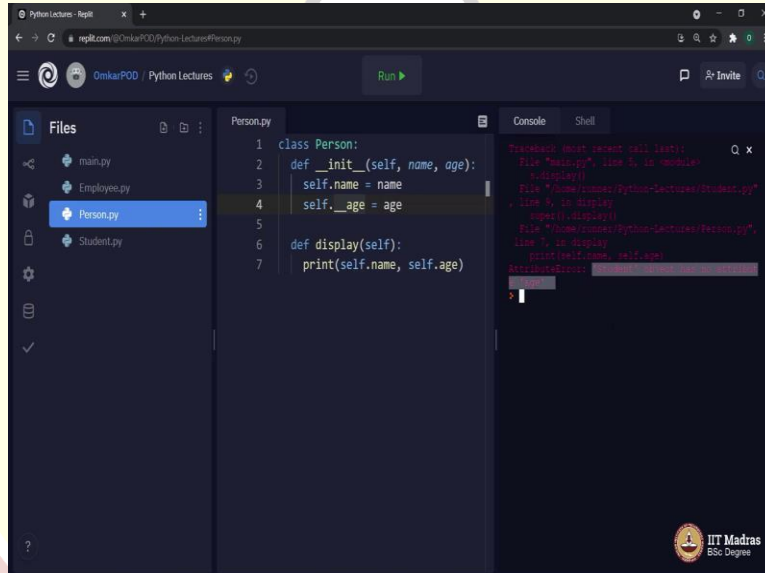
IIT Madras
BSc Degree



```
1 from Student import Student
2 from Employee import Employee
3
4 s = Student('Rida', 20, 250)
5 s.display()
6
7 e = Employee('Harsh', 30, 50000)
8 e.display()
```

Console output:

```
Rida 20
250
Harsh 30 50000
```



```
1 class Person:
2     def __init__(self, name, age):
3         self.name = name
4         self.__age = age
5
6     def display(self):
7         print(self.name, self.age)
```

Console output:

```
Traceback (most recent call last):
  File "main.py", line 4, in module
    s.display()
  File "/home/omkar/python-lectures/student.py", line 4, in display
    s.display()
  File "/home/omkar/python-lectures/person.py", line 7, in display
    print(self.name, self.age)
AttributeError: 'Student' object has no attribute 'age'
```

Let us look at this particular class Student. In this class, we have three attributes name, age and marks. And we have one function or as we know we call it method called display. We created one object of this class and call that particular method display. As expected, we got the output Rida 20, 250. This is something which we have studied in our previous lecture.

Now, after this, what if I want to create one more class called employee and this is my employee class. In this class, I have once again three attributes name, age, and instead of marks now I have salary and display method, I will create object of this class employee and I will call that method display. Let us execute. There is nothing new about this. We created two classes, created two

different objects of those two classes and use those two objects to call this display method from its respective class.

But if you observe, there is lot of repetition, lot of duplication with respect to these two classes student and employee. These two attributes name and age are common in both the classes, whereas only one parameter which is different which is marks in student and salary in employee. And if you remember, previously I have said we are programmers. We do not repeat the same thing twice. And keeping that in mind, it is not a good practice to repeat same thing again for a different class. Maybe it was okay because we have only two classes.

But consider a situation where instead of just student class I have many different classes which represent different students, let us say, school level students, high school level students, college level students, post graduation level students, PhD level students and so on. Similarly, instead of employee class, let us say I have different classes representing different types of employees.

For example, employees working in medical field, employees working in manufacturing field, employees working in educational field and so on. So, if we consider these students and employees like this then there can be hundreds of such classes. And in this case, it is not practically possible to repeat same thing 100 times.

On top of that, instead of just name and age, there are many other parameters, many other attributes which will be common for all these students as well as employees like gender, address, mobile number, email address and so on. So, once again, as I said, it is not good practice to repeat these things for a different class. To avoid this duplication, the concept of inheritance comes to rescue.

Earlier, we discussed one example where you inherited properties of your parents, painting and brown hairs. Similarly, you will see there are a lot of common qualities between you and your sibling. Why, because of same reason, because both of you, you and your sibling inherited those properties from your parent, which means those common qualities might be there in both of you, but they actually came from someone who is once again common to both of you, which is your parent. That same idea we will apply over here.

Now, student and employee, let us consider them as siblings, and the common qualities or the common attributes for these two are name and age. So, we will create one parent class for employee and student and in that parent class, we will put these common attributes. So, let us create one class called person and in that person class, we will initialize both these parameters name and age, whereas parameters like marks and salary will stay in those respective classes student and employee, because they are specific to that particular class.

Every person will not have marks or every person will not have salary. But every person will have name and age, every student will have marks, every employee will have salary. And in this sense, every student is a person, every employee is a person and this kind of a relationship is generally referred as is a relationship.

So, now, I hope this particular structure between these three classes is clear in your head. So, now, let us go ahead and modify this Python program in order to accommodate that new class called person and then we will see how to implement that inheritance with the combination of these three classes. Let us create that class person. In this lecture, you might have noticed that I have added this parenthesis after class name. Either you can have this parenthesis or you can create class without this parenthesis. Both are correct syntaxes in Python.

So, in this class person, we will copy these things. And now, we do not want these marks, self name, age. Name equal to name, self dot age is equal to age inside init. That is correct. Now, we do not want these two things over here, not required, as well as over here. Similarly, if you see, even the display method is pretty same except the third value which we are printing.

So, we can even make this a generic one inside person and let us remove marks from here and name and age from here. Now, we brought the common functionalities of student and employee class into person class and only the unique features of student and employee class are remaining in those two respective classes.

Somehow we have to tell computer that person is the parent class, whereas student and employee are two child classes of this particular parent class. And the way to do that is very simple. Inside these brackets, we should mention the name of parent class, student in bracket person, employee in bracket person. This means, student inherits the properties of person, employee is a child class or a subclass of this parent class or a superclass. Sometimes, parent

class is also referred as superclass, whereas child class is referred as subclass. And this is how we achieve inheritance.

Now, let us execute and see whether we are getting the expected output as it is currently there on the right hand side of the screen or is there something more we have to do. We got value 250 as in 250 marks and 50,000 as salary, which means these two lines, the line number 11 and line number 18 are working fine and then we are displaying marks and salary. Till that point it is working. But still these things are not getting executed. Even though we said student is a subclass of person, employee is a subclass of a person, still computer is not executing any of these lines.

This initialization of name and age is not happening as well as displaying these values is also not happening, which means somehow we have to tell computer whenever you assign these marks also assign name and age, whenever you print marks also print name and age. And the way to do that is using one command which we should execute over here, which means at this particular point, we should tell computer please go to line number two and execute this init. And the way to do that is using one special function and that special function is called super dot. Here we will call init method and as always we will pass these two parameters name and age to this particular init method.

First, computer will execute this line. It will take these three parameters, Rida, 20 as in age and 250 as in marks then it will come to this init method and as soon as it reaches to this line number 11 which says super dot init now computer will come to know it is supposed to execute init method from parent. And because of this, it will go over here and it will execute this init method and it will initialize these two variables name and age.

Once that is done, it will come back to this particular line number 12 and will execute this self dot marks is equal to marks. I have missed two underscores here. But anyways this is how it will work. Same thing we have to do in other class as well for employee. Let us execute and see whether is it working. It is working, but still we are not getting the expected output because of same reason. This display method is not yet getting executed. Just like this, somehow we have to tell computer you execute this display method as well.

And the way to do is over here, we should call once again, super dot display. Now, whenever we say s dot display, computer will execute this particular display method because it is there in student class and when it reaches to this line number 15 it will come to know that now I am supposed to execute this display method which is there in super, super means in super class, in parent class, which is class person in this case, which will execute this particular line number 7. Similarly, this line should be added over here.

Now, let us execute and see. And now as a result, we got our expected output as name, age, marks; name, age, salary. Now, if you relate this Python program with our previous example of painting and hair color, you will remember I talked about two different scenarios where along with painting you decided to learn swimming, along with name and age you decided to print marks. This is first scenario.

And in second scenario, instead of your brown hairs, you decided to replace those with red hairs. And that can be done by simply rewriting the whole display method by adding this name and age here itself and we will remove this. Now, we are not at all referring to display method which was there in original person class. So, this employee class inherited this function, inherited this method, but now we are replacing that inherited property with its own property.

So, consider this was that brown color and we replaced this brown color with red color. And this particular feature is called method overriding, which we introduced earlier. Let us execute and still we should get same output as it was earlier. So, so far we have understood; first, inheritance allows a child to inherit properties from parent or if there are common properties between siblings, we can bring them together in a common class called parent class and both siblings both child can use those properties as we have done over here.

This allows us to reuse the same code and also provides us a way to modularize our code as in to divide our code into small, small blocks, instead of writing everything inside a single class or inside a single function. That is why this is the good place and a good example to address your doubt which you had in first week of this Python course, which was related to this replit why every time replit executes only this file main dot py. Even though I create multiple files over here, replit is not allowing me to execute those files.

So, now, let us see how to solve that problem. Let us create one file, add let us say person dot py. And in this file, I will put this entire class. I have removed it from here and I will put it inside this new Python file called person dot py. Similarly, I will put this entire student class in one more file called student dot py. And then let us create one more file called employee dot py. And here we will put our class employee.

Now, what is remaining in main? In main, we have nothing but just object of those two classes and calling of those two methods. Let us try to execute this code with this new modularized approach. It says name student is not defined, because once again as you have observed earlier, replit is executing only main dot py. And because of that, it is not able to find this particular class student. Currently, Python does not even know what this student is or what this employ is. In this case, we have to guide this particular Python program what is student and where is it?

We have already defined what is student in this file. Now, we have to tell where is it. So, from student import student, with this we are telling Python, there is a file called student and inside that file, I have a class called student. So, please go ahead and import that for me so that I can use it like this. Similarly, from employee import employee. I think now it should work. What do you think? Let us try. Still there is an error. It says name person is not defined.

Now, the question is why? We imported both. Now, why it is not able to find person. Can you figure that out? If we go to this particular student class, it says person, if you go to this employee class, it says person, but at this particular file, still Python does not know what is person. Once again, here, we do not know what is person. So, once again, we have to say from person, import person, same thing we should copy in employee class as well.

Now, let us try to execute this main dot py. And we have our answers as they were earlier. I hope this will help you a lot with respect to all your Python programs here onwards. You can create as many dot py files as you want. But remember, every time you will have to import that particular class in this manner and then only you can create object of it and call methods from those classes.

Before moving to next concept, I have one question for you. Whenever we implement this inheritance, there is an issue of security. Now, you must be wondering, what does Python has to do with security. As we know, everything which is there in this person class is accessible inside

employee class. Similarly, everything is accessible inside student class as well, because we are inheriting that class person and this is the reason there is a problem of security.

What if this person class do not want to reveal something to anybody else, what if this person class has some secret information which it prefers to keep to itself. Even though it is inherited, it do not want anybody else to know about that secret information. So, there has to be some way through which we should control that particular information at this particular class level. And even though any other class inherits this person class, all those subclasses, all those child classes will not have access to that secret information. And ideally, Python should have some mechanism to log this information at this class level because that is how it works in real life.

You might inherit a lot of things from your parents but they may not tell you all their secrets. That might be a bad example, but this is how it is done in Python. For example, this person do not want to reveal its age, then we can add simply two underscores before the variable name. And this will solve our issue of security. It is that simple. Only two underscores will solve the problem. Let us execute.

Now, it says student object has no attribute age. Even though student class has inherited this person class, this particular attribute age is not accessible to student and that is how this person class made sure no one outside will have access to this secret which I have with me, which is age in this case.

If you remove these underscores, once again, it will be available to everyone and we will have the output. And these kind of variables which we try to keep secret by adding these underscore are referred as private members. This is a private member of this class person. And if we do not have these underscores, then it is a public member. Hence, it is accessible to everyone.

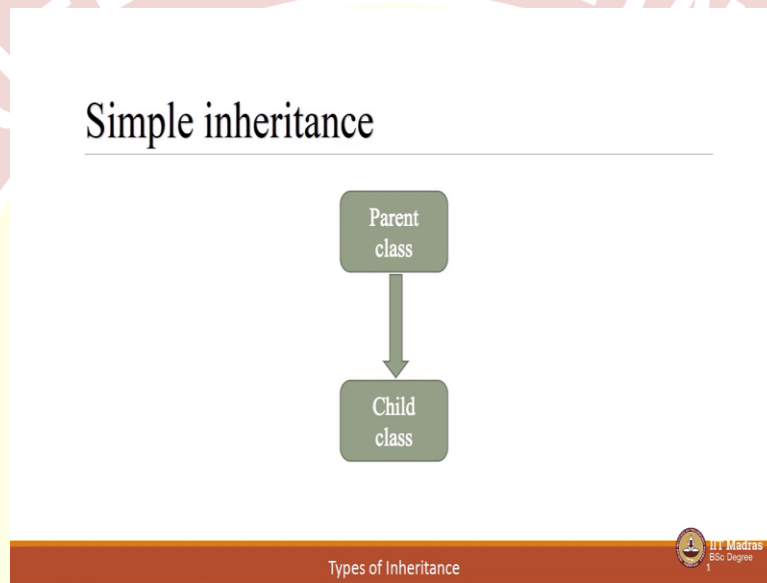
So, far we have understood what is private member, what is method overloading, what is method overriding, what is inheritance and so on. But the question is, is this inheritance a generic term or are there any sub-types of inheritance, because initially we talked about two super classes, your father and your mother, and you were subclass, a child.

Then in this case, we talked about one superclass which was person and it has two subclasses, employee and student. So, if you see there is small variation in both these inheritances. So, based

on number of super classes, subclasses and the relation between them, there are various types of inheritance.

Next question, how many such types? The answer is, there are total four, in fact, five different types of inheritance, which we will see briefly in next couple of minutes. And then it will be your responsibility to implement all those types using Python program.

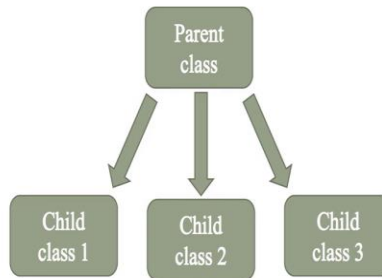
(Refer Slide Time: 29:05)



So, the first type of inheritance is called simple inheritance, where we have only one parent class and one child class.

(Refer Slide Time: 29:16)

Hierarchical inheritance



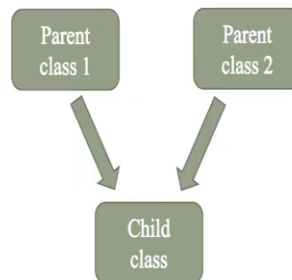
Types of Inheritance



Second type is called hierarchical inheritance. This is what we implemented using that person, student and employee example, where person was parent class, whereas student and employee were child class one and child class two. So, in order to generalize this in hierarchical inheritance, we have single parent class and multiple child classes.

(Refer Slide Time: 29:49)

Multiple inheritance



Types of Inheritance



Next type of inheritance, multiple inheritance. And this was that inheritance when we talked about painting and brown hair colors, where we have more than one parent class and single child class.

(Refer Slide Time: 30:06)

Multilevel inheritance



Types of Inheritance



Fourth type of inheritance is called multilevel inheritance, where the top level we have parent class, then there is one class which is referred as intermediate class, because that intermediate class behaves like a child class for the parent class, whereas that intermediate class behaves like a parent class for its child class. So, it is more like a grandparent, parent and child kind of a situation.

(Refer Slide Time: 30:40)

Hybrid inheritance

Types of Inheritance



And then there is last type which is hybrid inheritance. And as you can see, there is no standard diagram, which represents hybrid inheritance, because as the name suggests, hybrid, it is any

combination of those four inheritances which we discussed just now. You mix simple, multiple, hierarchical, multilevel in any way you want and that is hybrid inheritance. As I mentioned earlier, now, it is your responsibility to implement all these different types of inheritance using Python program. Thank you for watching this lecture. Happy learning.

