Inheritance in Python

Python Inheritance

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

Python Inheritance

- **Parent class ?**
- **Child class ?**

Python Inheritance

- **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.

Inheritance

Why?

- Inheritance **provides code reusability to the program** because we can use an existing class to create a new class instead of creating it from scratch.
- **Inheritance allows us to inherit attributes and methods from the base/parent class.**

Inheritance

What?

- In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class.
- **A child class can also provide its specific implementation to the functions of the parent class.**
- This is useful as we can create sub-classes and get all of the functionality from our parent class. **Then we can overwrite and add new functionalities without affecting the parent class.**

Python Inheritance

Create a Parent Class

- The existing class is a base class (or parent class or super class).
- Any class can be a parent class, so the syntax is the same as creating any other class:

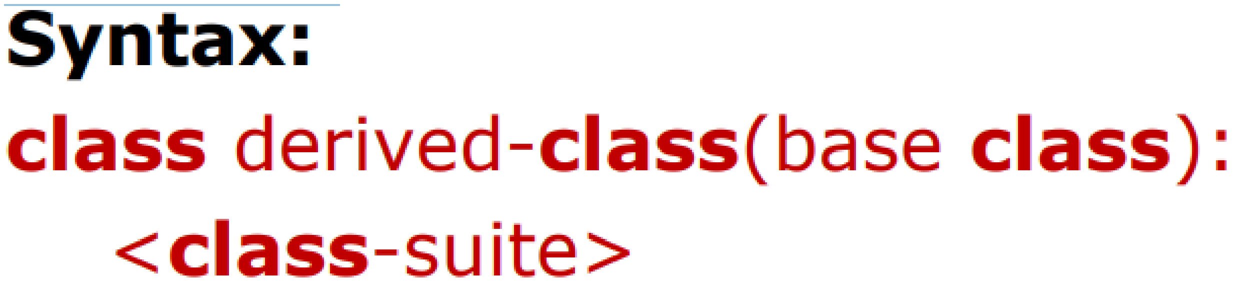
Create a Child Class

- The newly formed class is a derived class (or child class or sub-class).

- To create a class that inherits the functionality from another class, **send the parent class as a parameter when creating the child class:**

Syntax:

```
class child_class_name(Parentclass_name):
        properties
```



Syntax:

```
class derived-class(base class):
    <class-suite>
```

Example

- Create a class named Person, with firstname and lastname properties, and a printname method:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(“First name :”,self.firstname, “Last name: “,
self.lastname)

x = Person("John", "Doe")
x.printname()
```

Create a Child Class

Example

- Create a class named Student, which will inherit the properties and methods from the Person class:

```
class Student(Person):
  pass
x = Student("Mike", "Olsen")
x.printname()
```

- **Note:** Use the pass keyword when you do not want to add any other properties or methods to the class.
- Now the Student class has the same properties and methods as the Person class.

Add the __init__() Function

- So far we have created a child class that inherits the properties and methods from its parent.

- **We want to add the __init__() function to the child class (instead of the pass keyword).**

Add the __init__() Function

- Add the __init__() function to the Student class:

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

- **When you add the __init__() function, the child class will no longer inherit the parent's __init__() function.**

- **Note: The child's __init__() function overrides the inheritance of the parent's __init__() function.**

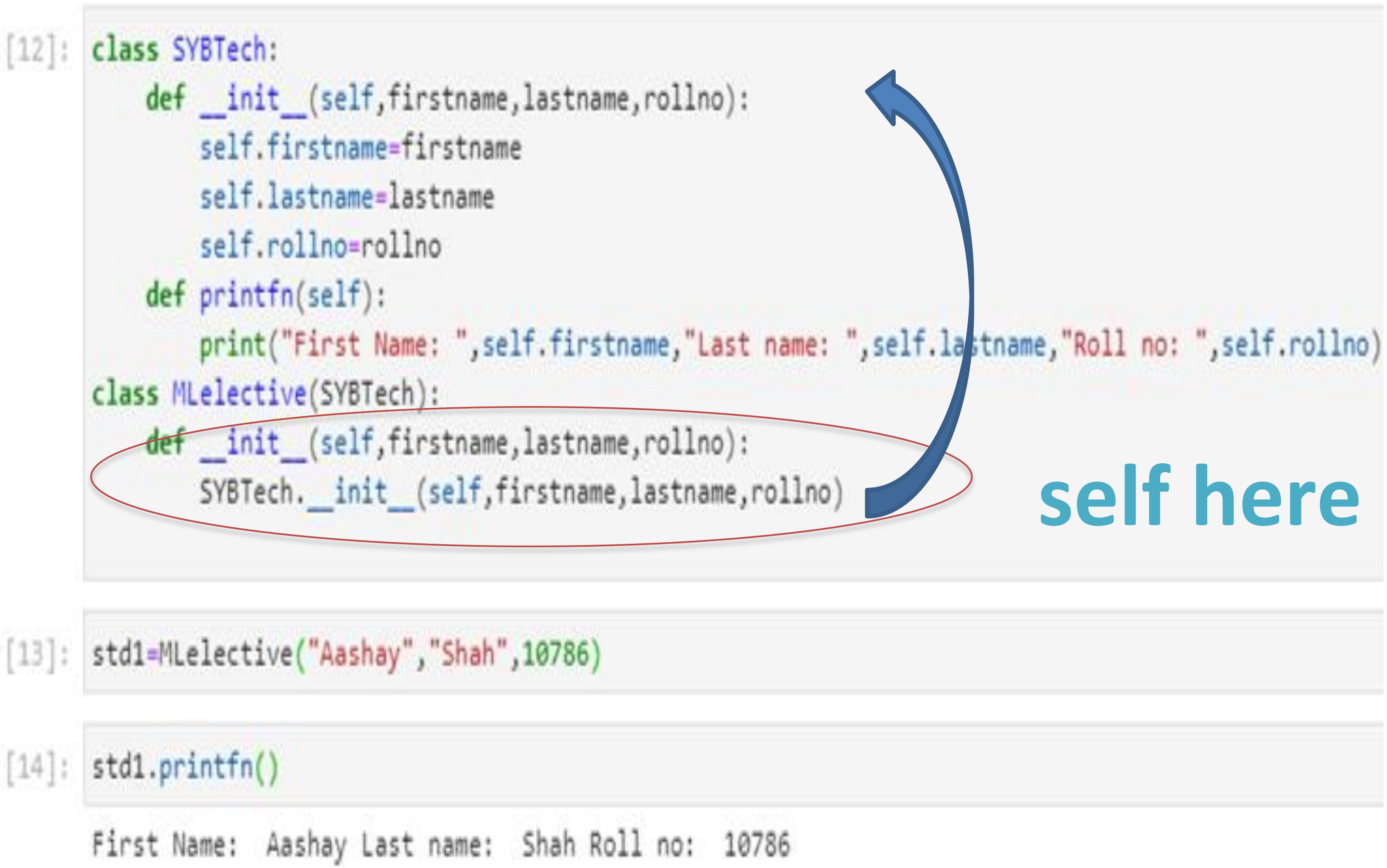
Add the __init__() Function

- **To keep the inheritance of the parent's __init__() function, add a call to the parent's __init__() function:**

Example

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

Add the __init__() Function



```
[12]: class SYBTech:
          def __init__(self,firstname,lastname,rollno):
              self.firstname=firstname
              self.lastname=lastname
              self.rollno=rollno
          def printfn(self):
              print("First Name: ",self.firstname,"Last name: ",self.lastname,"Roll no: ",self.rollno)
      class MLelective(SYBTech):
          def __init__(self,firstname,lastname,rollno):
              SYBTech.__init__(self,firstname,lastname,rollno)
```

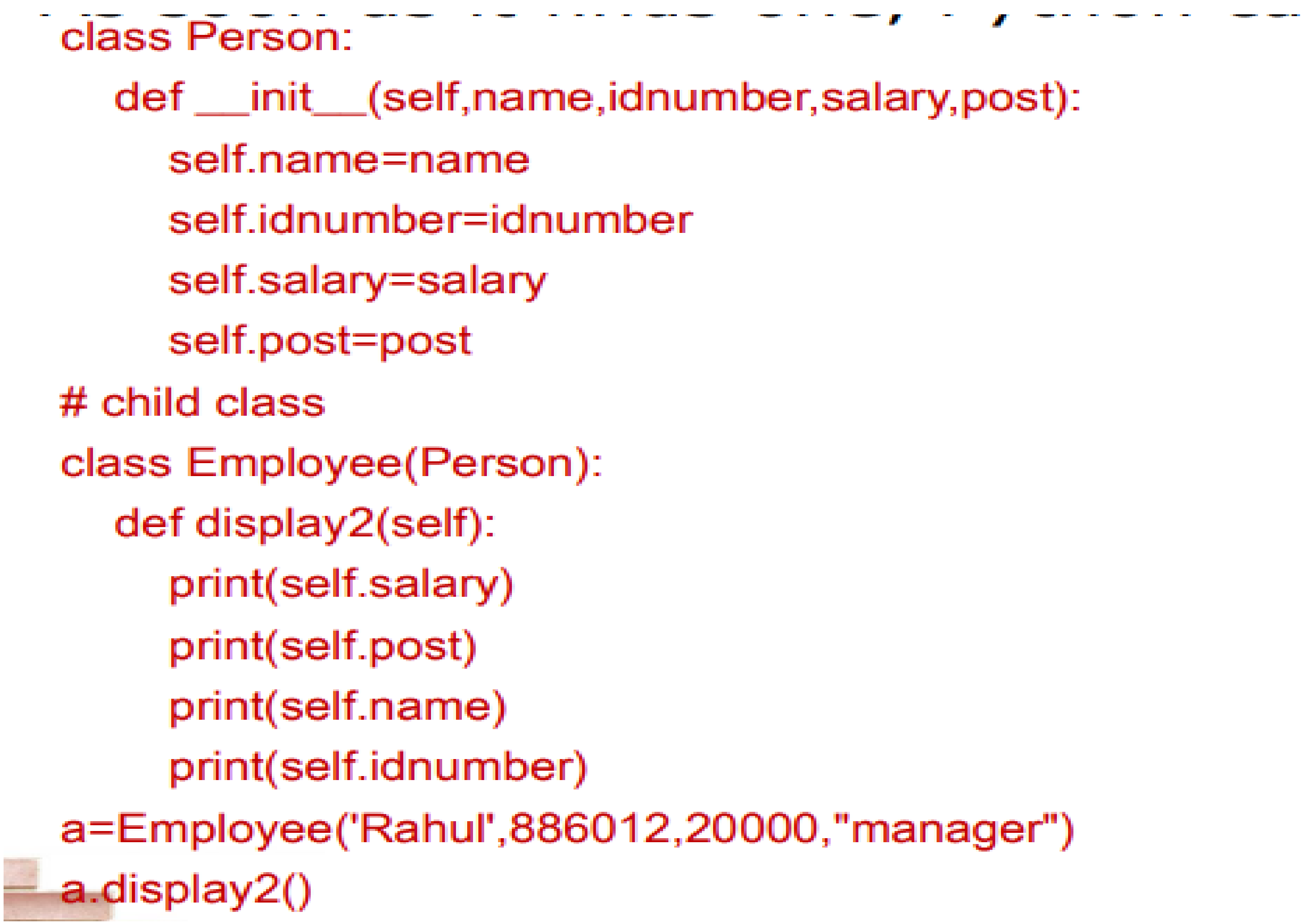
```
[13]: std1=MLelective("Aashay","Shah",10786)
```

```
[14]: std1.printfn()

      First Name:  Aashay Last name:  Shah Roll no:  10786
```

self here

Can Constructor be inherited?

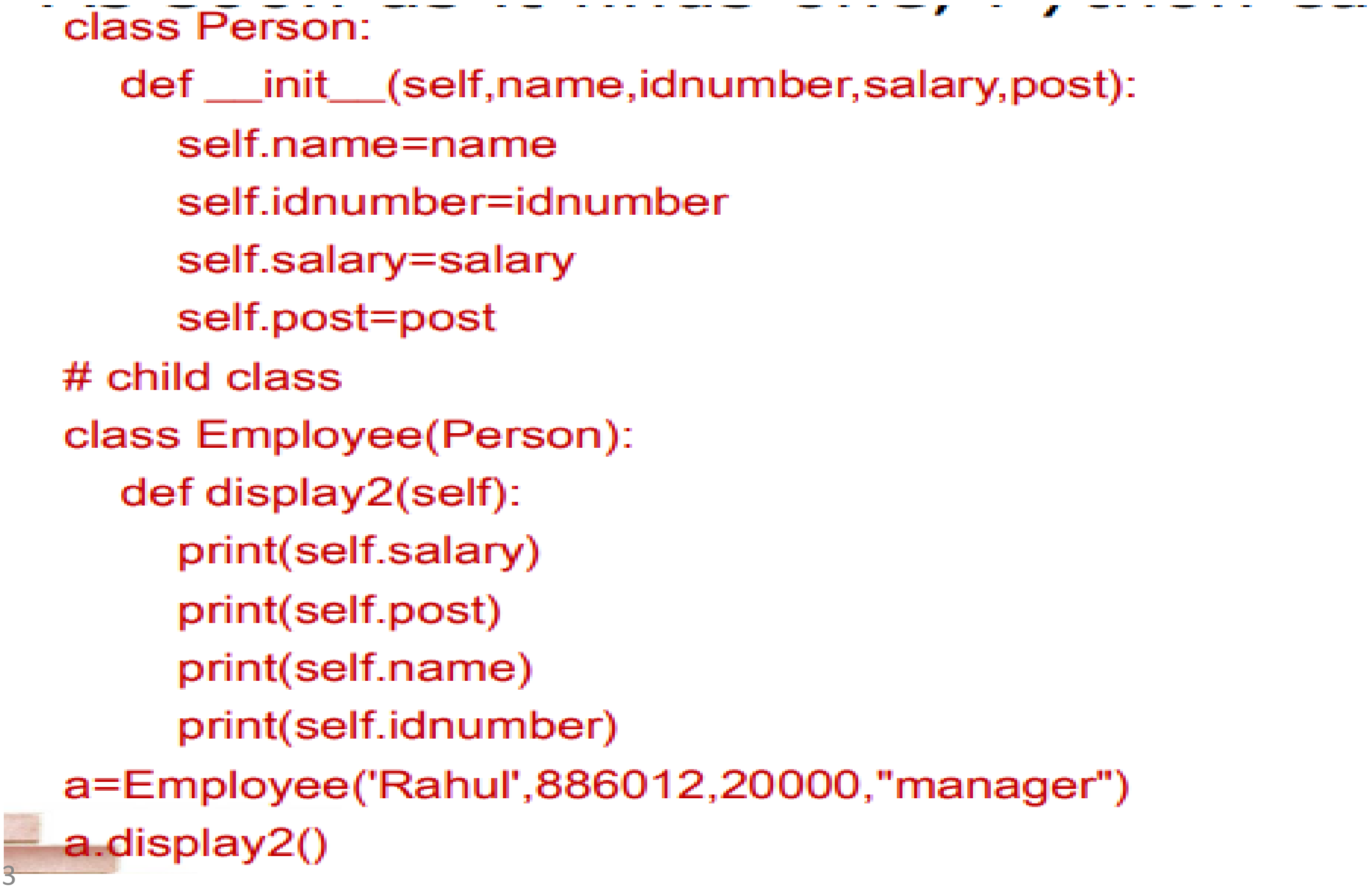
- **init__ is like any other method; it can be inherited**



```
class Person:
    def __init__(self,name,idnumber,salary,post):
        self.name=name
        self.idnumber=idnumber
        self.salary=salary
        self.post=post
# child class
class Employee(Person):
    def display2(self):
        print(self.salary)
        print(self.post)
        print(self.name)
        print(self.idnumber)
a=Employee('Rahul',886012,20000,"manager")
a.display2()
```

Can Constructor be inherited?

- **If a class does not have a __init__ constructor, Python will check its parent class to see if it can find one.**
- **As soon as it finds one, Python calls it and stops looking**



```
class Person:
    def __init__(self,name,idnumber,salary,post):
        self.name=name
        self.idnumber=idnumber
        self.salary=salary
        self.post=post
# child class
class Employee(Person):
    def display2(self):
        print(self.salary)
        print(self.post)
        print(self.name)
        print(self.idnumber)
a=Employee('Rahul',886012,20000,"manager")
a.display2()
```

super() Function

- **Python also has a super() function that will make the child class inherit all the methods and properties from its parent**

- **super() builtin returns a proxy object that allows you to refer parent class by 'super'.**

super() Function

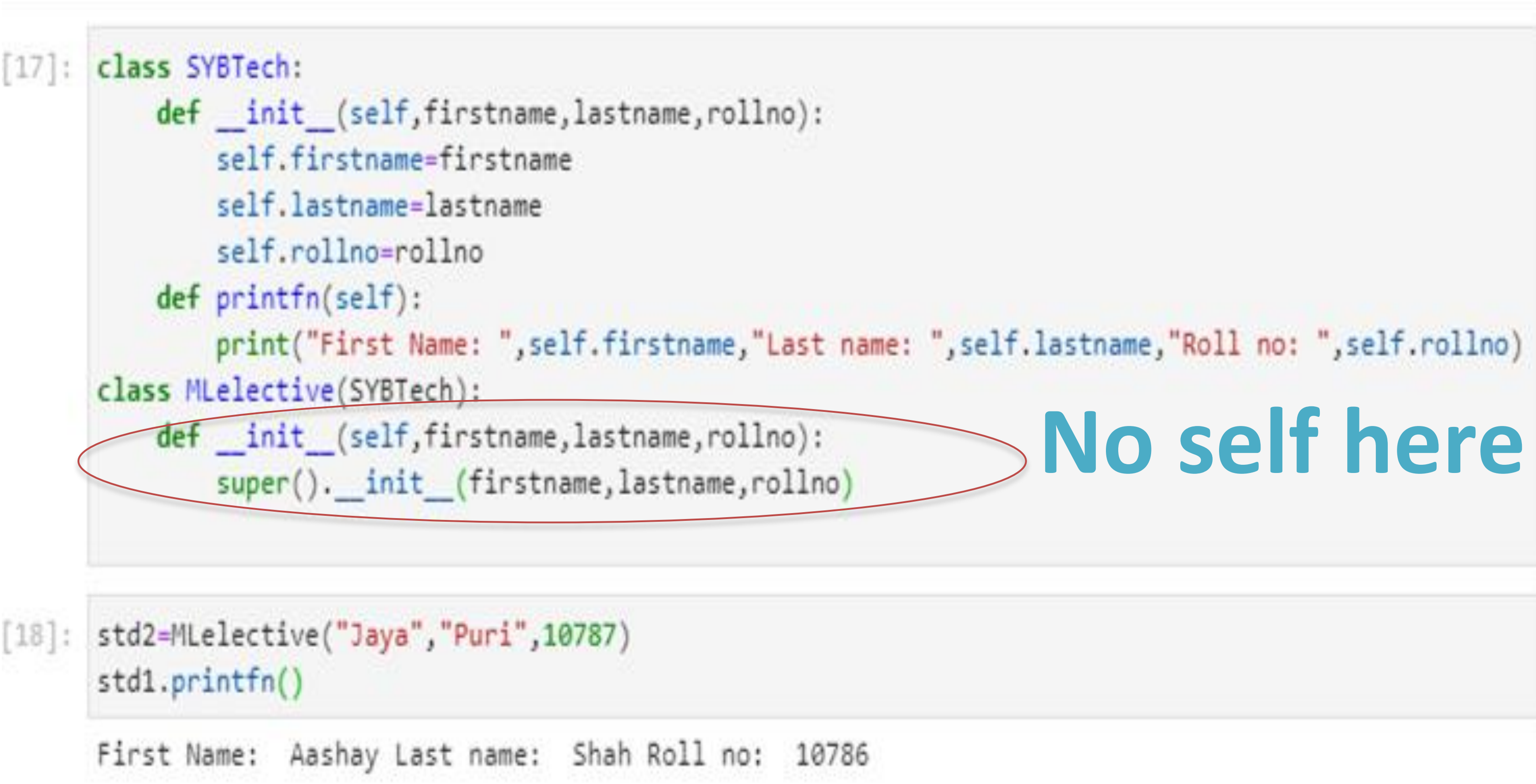
Example

```
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
```

- By using the super() function,
  - **Child automatically inherits the methods and properties from its parent.**

super() Function

- By using the super() function,
- **No need to use the name of the parent element**
- **Self parameter also does not need to be used with super()**



```
[17]: class SYBTech:
          def __init__(self,firstname,lastname,rollno):
              self.firstname=firstname
              self.lastname=lastname
              self.rollno=rollno
          def printfn(self):
              print("First Name: ",self.firstname,"Last name: ",self.lastname,"Roll no: ",self.rollno)
      class MLelective(SYBTech):
          def __init__(self,firstname,lastname,rollno):
              super().__init__(firstname,lastname,rollno)
```

No self here

```
[18]: std2=MLelective("Jaya","Puri",10787)
      std1.printfn()

      First Name:  Aashay Last name:  Shah Roll no:  10786
```

- If you need to call the parent constructor separately from the child constructor, you can define a separate function in the child class that calls the parent constructor, like this:

```
class Parent:
    def __init__(self, arg1):
        self.arg1 = arg1

class Child(Parent):
    def __init__(self, arg1, arg2):
        self.arg2 = arg2  # initialize child attribute

    def call_parent_constructor(self, arg1):
        super().__init__(arg1)  # call parent constructor

c = Child(1, 2)
c.call_parent_constructor(3)
print(c.arg1)  # prints 3
print(c.arg2)  # prints 2
```

Adding Properties & Methods to the Child Class

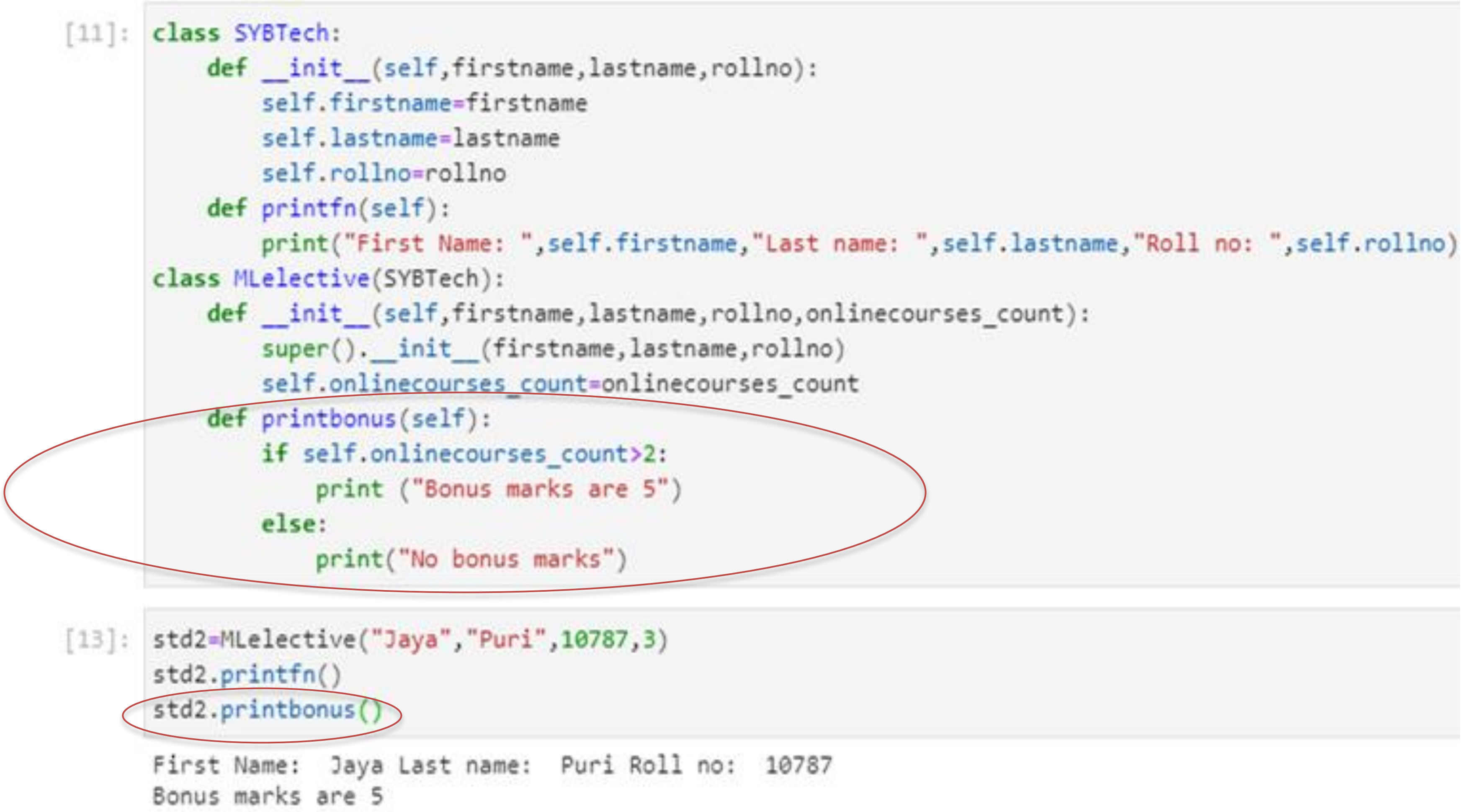
Add Properties

- **One new property :count of online courses done ,added**

```
class Parent:
   def __init__(self, firstname,lastname,rollno):
      self.firstname = firstname
      self.lastname = lastname
      self.rollno = rollno

class Child(Parent):
   def __init__(self, firstname,lastname,rollno,marks):
      super(). __init__(firstname,lastname,rollno)
      self.marks = marks

   def printf(self):
      print("Firstname: ",self.firstname,"lastname: ",self.lastname,"rollno: 
",self.rollno,"marks: ",self.marks)



c = Child("jaya","prakash", 1, 25)
c.printf()
```

Add Methods

- Method printbonus() added to child class



```
[11]: class SYBTech:
          def __init__(self,firstname,lastname,rollno):
              self.firstname=firstname
              self.lastname=lastname
              self.rollno=rollno
          def printfn(self):
              print("First Name: ",self.firstname,"Last name: ",self.lastname,"Roll no: ",self.rollno)
      class MLelective(SYBTech):
          def __init__(self,firstname,lastname,rollno,onlinecourses_count):
              super().__init__(firstname,lastname,rollno)
              self.onlinecourses_count=onlinecourses_count
          def printbonus(self):
              if self.onlinecourses_count>2:
                  print ("Bonus marks are 5")
              else:
                  print("No bonus marks")
```

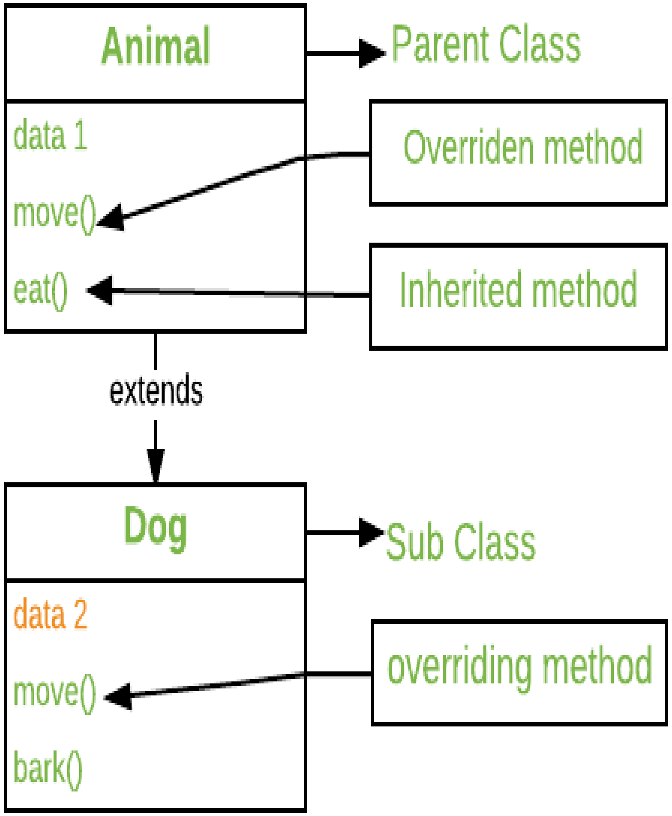
```
[13]: std2=MLelective("Jaya","Puri",10787,3)
      std2.printfn()
      std2.printbonus()
```

```
      First Name:  Jaya Last name:  Puri Roll no:  10787
      Bonus marks are 5
```

Method Resolution Order

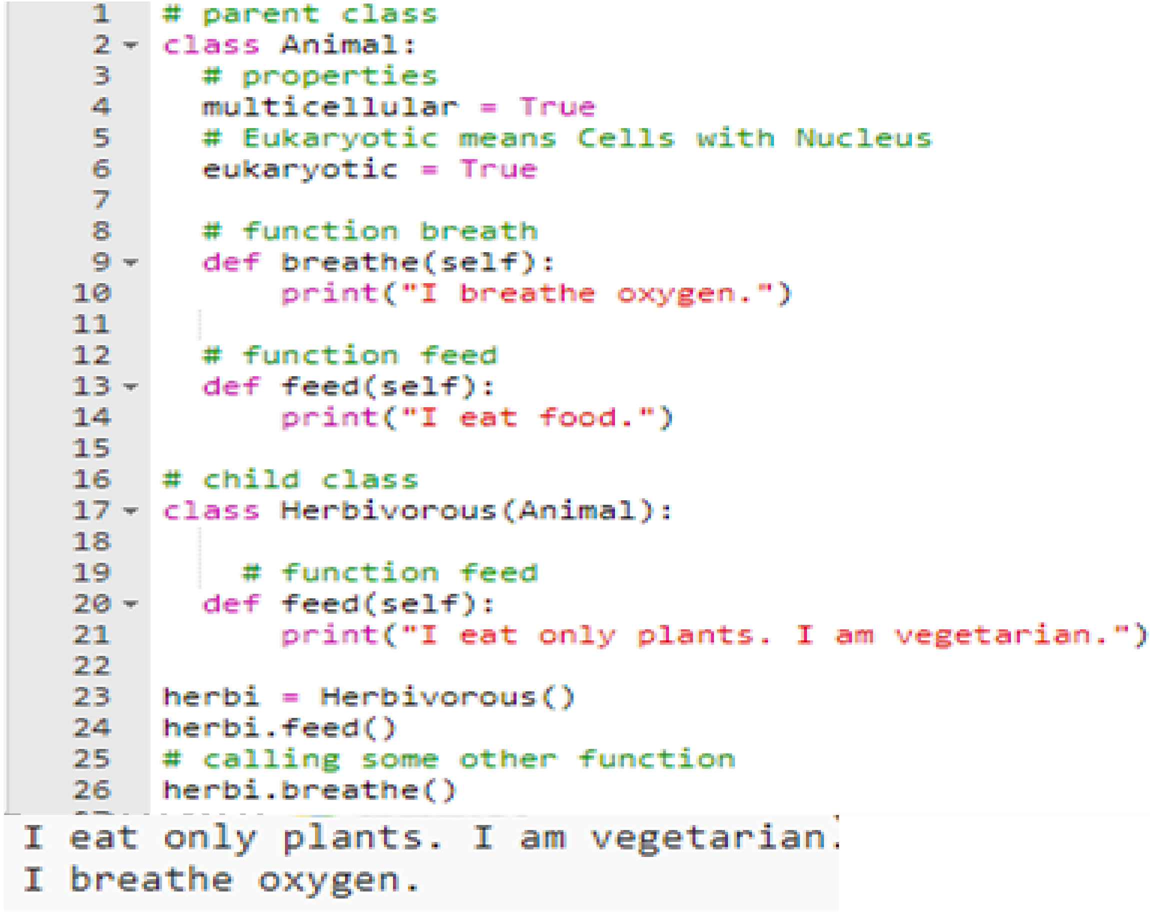
Method Overriding

- If you add a method in the child class with the same name as a function in the parent class, the inheritance of the parent method will be overridden.



Method Overriding

- Child class Herbivorous which extends the class Animal
- Feed() function in child overrides the parent function



```python
# parent class
class Animal:
  # properties
  multicellular = True
  # Eukaryotic means Cells with Nucleus
  eukaryotic = True

  # function breath
  def breathe(self):
      print("I breathe oxygen.")

  # function feed
  def feed(self):
      print("I eat food.")

# child class
class Herbivorous(Animal):

    # function feed
  def feed(self):
      print("I eat only plants. I am vegetarian.")

herbi = Herbivorous()
herbi.feed()
# calling some other function
herbi.breathe()
```

```
I eat only plants. I am vegetarian.
I breathe oxygen.
```

Parent and child class having same function name

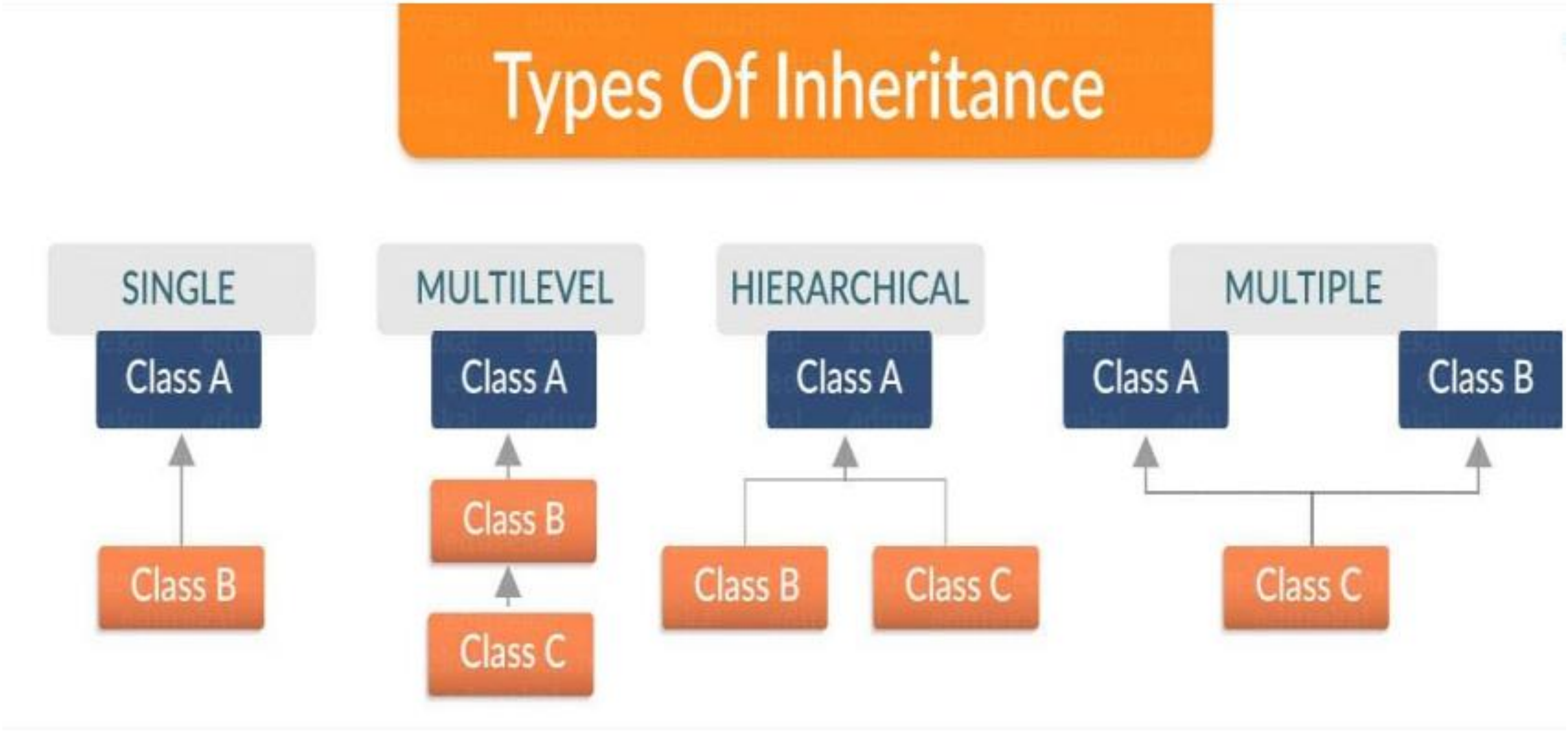
```
class Parent:
    def my_function(self):
        print("This is the parent's version of my_function.")

class Child(Parent):
    def my_function(self):
        print("This is the child's version of my_function.")
        super().my_function()  # Call parent version of the function

c = Child()
c.my_function()
```

Parent and child class having same function name

```
class Parent:
    def my_function(self):
        print("This is the parent's version of my_function.")

class Child(Parent):
    def my_function(self):
        print("This is the child's version of my_function.")

c = Child()
c.my_function()  # Calls the child's version of the function
super(Child, c).my_function()  # Calls the parent's version of the function
```

Types of Inheritance

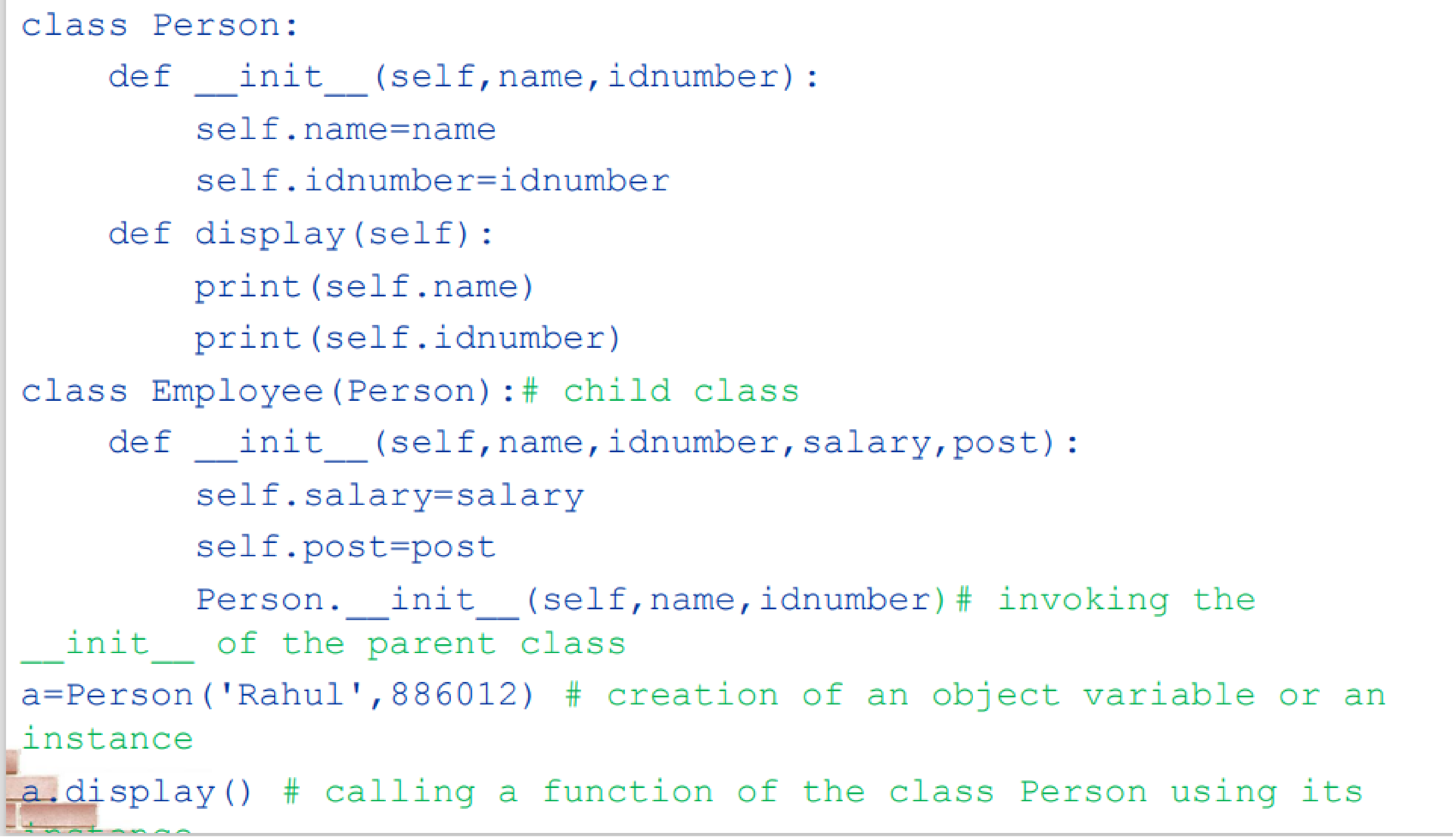
- There are basically 4 types of inheritances



SINGLE INHERITANCE

- When a child class inherits from only one parent class, it is called as single inheritance.

SINGLE INHERITANCE

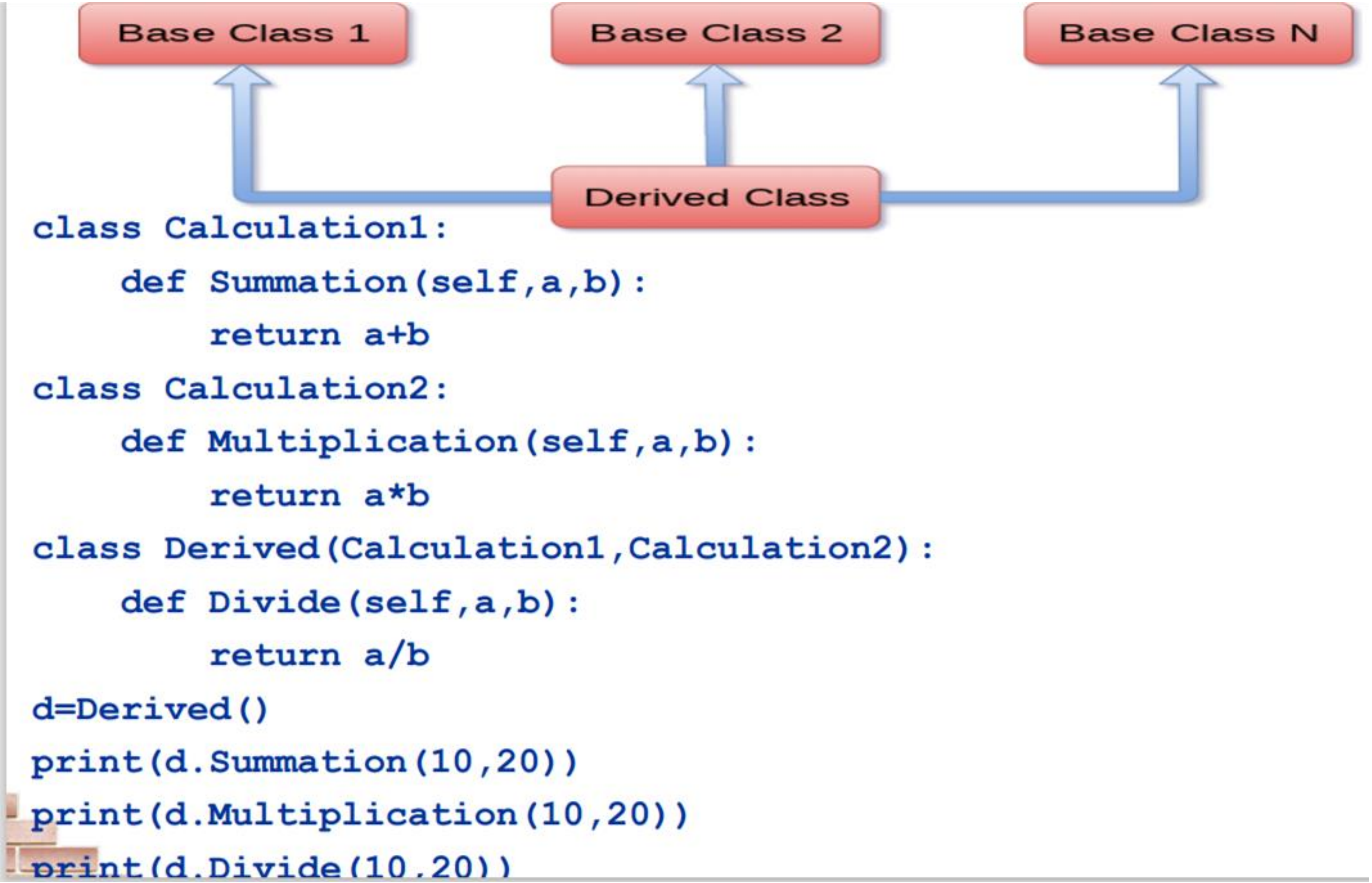


```
class Person:
    def __init__(self,name,idnumber):
        self.name=name
        self.idnumber=idnumber
    def display(self):
        print(self.name)
        print(self.idnumber)
class Employee(Person):# child class
    def __init__(self,name,idnumber,salary,post):
        self.salary=salary
        self.post=post
        Person.__init__(self,name,idnumber)# invoking the
__init__ of the parent class
a=Person('Rahul',886012) # creation of an object variable or an
instance
a.display() # calling a function of the class Person using its
instance
```

Multiple INHERITANCE

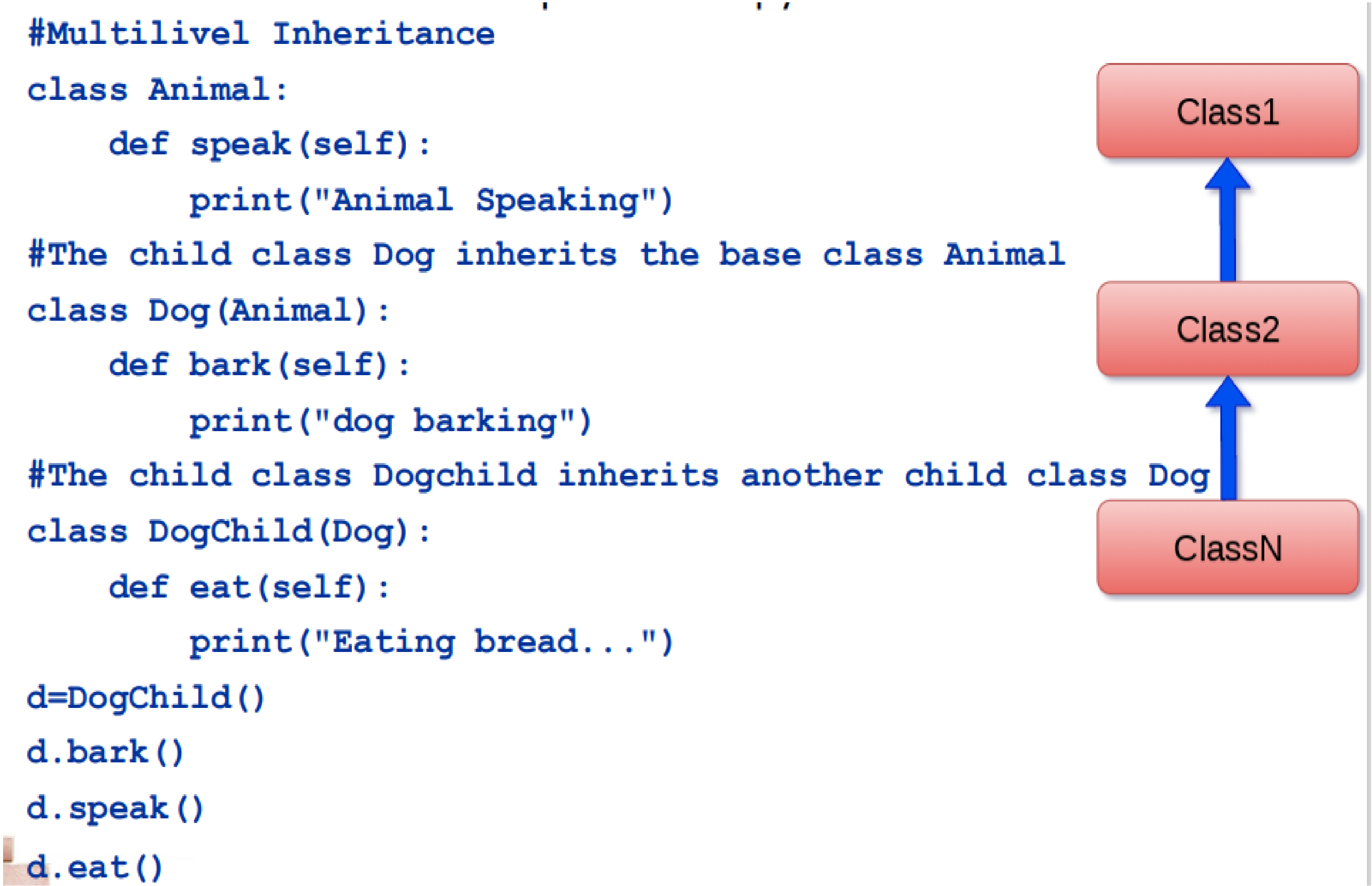
- Python provides us the flexibility to inherit multiple base classes in the child class which is known as multiple inheritance

Multiple INHERITANCE



```
class Calculation1:
    def Summation(self,a,b):
        return a+b
class Calculation2:
    def Multiplication(self,a,b):
        return a*b
class Derived(Calculation1,Calculation2):
    def Divide(self,a,b):
        return a/b
d=Derived()
print(d.Summation(10,20))
print(d.Multiplication(10,20))
print(d.Divide(10,20))
```

Multilevel Inheritance



```
#Multilivel Inheritance
class Animal:
    def speak(self):
        print("Animal Speaking")
#The child class Dog inherits the base class Animal
class Dog(Animal):
    def bark(self):
        print("dog barking")
#The child class Dogchild inherits another child class Dog
class DogChild(Dog):
    def eat(self):
        print("Eating bread...")
d=DogChild()
d.bark()
d.speak()
d.eat()
```

Other Inheritances

- **Hierarchical inheritance :**

More than one derived classes are created from a single base.

- **Hybrid inheritance:**

This form combines more than one form of inheritance. Basically, it is a blend of more than one type of inheritance