Inheritance

Introduction

- In inheritance, one class is derived from another class, and the properties of the parent class can be used by the derived class.
- Inheritance is important because it allows the programmers to create new classes, which specify a new implementation while maintaining the same behaviours of the old class.
- Inheritance is the process of deriving a class from another existing class.
- Inheritance facilitates the reusability of the code and extends the original properties of the class without modifying it.

Motivations for using inheritance

- 1. code reuse. Because a child class can inherit behaviour from a parent class, the code does not need to be rewritten for the child.
- 2. Concept reuse: This occurs when a child's class overrides behaviour defined by the parent.

In deriving a new class from a class, two classes are defined as parent class and child class.

Parent class: A parent class is a class from which a class is derived. The parent class is also called super class.

Child Class: A child class is a class that is derived from another class and is also called derived class. It is also called sub class.

Inheritance in Python

- In python, there are no special keywords while deriving a class from a base class.
- For deriving the child class from a parent class, the name of the parent class should be given as a parameter while defining the child class.
- The syntax for inheriting a class from a class is as follows:

Inheritance and Types of Attributes

- There are two attributes, namely private and public.
- The public attributes are accessible across all the derived classes.
- The private attributed are defined and using the “__” double underscore operator.
- Types of Inheritance

1. Single Inheritance
2. Multiple Inheritance
3. Multi-level inheritance
4. Hybrid Inheritance

Single Inheritance

- When a derived class inherits its properties from one base class, it is called single inheritance.
- The Derived class possesses the characteristics of the base class and also its properties. The syntax for the single inheritance is as follows:

Python Syntax

```
class BaseClass:
        #Action Block for Base Class

class DerivedClass (BaseClass):
        #Action Block for Derived Class
```

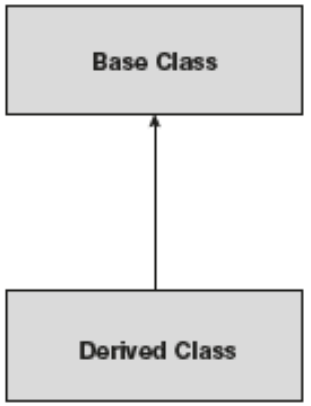


Illustration for single inheritance by deriving class Apple from the base class Eatable_fruits

Let's create the base class *Eatable_Fruits* consisting of a class attribute, *no_of_fruit_varieties,* with a value of 2000.

```
>>> #Defining the class Eatable_Fruits
>>> class Eatable_Fruits:
...         no_of_fruit_varities = 2000
...
>>>
```

A Subclass Apple is derived from the class *Eatable_Fruits*. Now, the class *Eatable_Fruits* is called the parent class, and the class *Apple* is called as child class. A class attribute *colour* is defined inside the class *Apple* as a set.

```
>>> class Apple(Eatable_Fruits):
... colour = {"Green", "Yellow", "White", "Striped", "Red"}
...
>>>
```

Multiple Inheritance

- Multiple inheritance refers to the type of inheritance where the derived class possesses the characteristics of one or more base classes.
- The derived class has all the features of base classes along with the new featured that are defined in the derived class. The Syntax as follows:

Python Syntax

```
class BaseClass1:
        #Action Block for Base Class 1

class BaseClass2:
        #Action Block for Base Class 2
...
Class BaseClassN:
        #Action Block for Base Class n

class DerivedClass (BaseClass1, BaseClass2,
... , BaseClassN):
        #Action Block for Derived Class
```

Illustration for multiple inheritance by creating a class TF representing Teaching faculty and another class student and then derive a class called TA from both the class TF and Student

Creating a class TF, as shown below, with a constructor that prints "Base class Teaching Faculty" also invokes the constructor from its superclass. It could be noticed that although the class *TF* is not a derived class, the super() function is used. This invokes the constructor of the class of the same order, which is class *Students*.

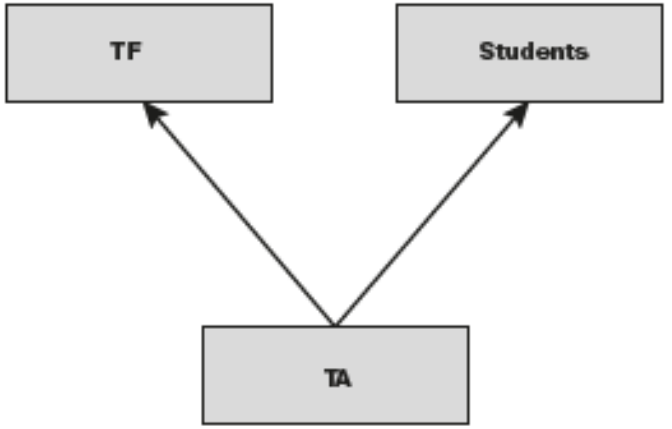
```
>>> class TF:
...   def __init__(self):
...     print("Base class Teaching Faculty")
...     super().__init__()
...
>>>
```

Now another class, *Students,* is defined where the constructor is defined, which prints "Base Class Student". The code of the class *Students* is shown below.

```
>>> class Students:
...   def __init__(Self):
...     print("Base Class Student")
...
>>>
```

The class *TA* is derived from the class *TF* and the class *Students*. Here, the constructor prints "TA Derived Class from TF and Students", and then the constructor of the parent class is invoked.

```
>>> class TA (TF, Students):
...         def __init__(self):
...                 print("TA Derived Class from TF and Students")
...                 super().__init__()
...
...
>>>
```



Multi-Level Inheritance

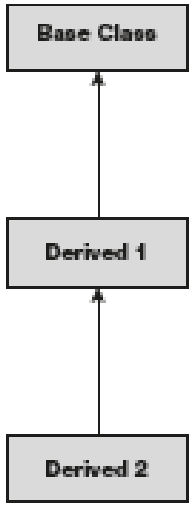
- Multi-level inheritance is the type of inheritance in which is derived from another derived class. The syntax for multi-level inheritance as follows:

Python Syntax

```
class BaseClass1:
        #Action Block for Base Class 1

class Derived1(BaseClass):
        #Action Block for Derived1

class Derived2 (Derived1):
        #Action Block for Derived2
```



Illustrating the multi-level inheritance by deriving a class Students from the class user and another class ClassActivities from the class Students.

Let us consider the above school management system, students are a class derived from person, and considering ClassActivities is another class that holds the details about the activities that are performed in Class Sessions. The following code illustrates how to implement this as a multi-level inheritance.

The class User is defined below.

```
>>> class User:
...     def __init__(self, fname, lname, aadhaar):
...         print("Constructor of Person")
...         self.first_name = fname
...         self.last_name = lname
...         self.aadhaar = aadhaar
...
>>>
```

The derived class Students from the base class Person is defined as follows.

```
>>> class Students(Person):
...     def __init__(self, fname, lname, aadhaar, csection):
...         print("Constructor of Students")
...         super().__init__(fname, lname, aadhaar)
...         self.csection = csection
...
>>>
```

The class ClassActivities is derived from the derived class students in which the class activities are stored. The constructor for the class *ClassActivities* is defined in which the "activities" is stored in the attribute *activities*. The definition of *Class Activities* is shown below.

```
>>> class ClassActivities (Students):
...     def __init__(self, fname, lname, aadhaar, csection, activities):
...         print("Constructor of Class Activities ")
...         super().__init__(fname, lname, aadhaar, csection)
...         self.activities = activities
...
>>>
```

The object is created for the class *ClassActivities* with the parameters shown below and accessed as follows.

```
>>> object1 = ClassActivities("Selva", "Ravi", "12345678", "X-B", "EX5.6")
The constructor of Class Activities
Constructor of Students
Constructor of Person
>>> print(object1.first_name + " has the activities of " + object1.activities)
'Selva has the activities of EX5.6
```

In the above example, the class "ClassActivities" is derived from the class "Students", which is already a derived class from the class "Person".

Hybrid Inheritance

- Deriving a class from the derived classes which are derived from a same base class is called hybrid inheritance. The syntax for the hybrid inheritance as follows:

Python Syntax

```
class BaseClass:
        #Action Block for BaseClass
class DerivedClass1(BaseClass):
        #Action Block for Derived Class1
class DerivedClass2(BaseClass):
        #Action Block for Derived Class 2
class DerivedClass3(DerivedClass1,
DerivedClass2):
        #Action Block for Derived Class 3
```

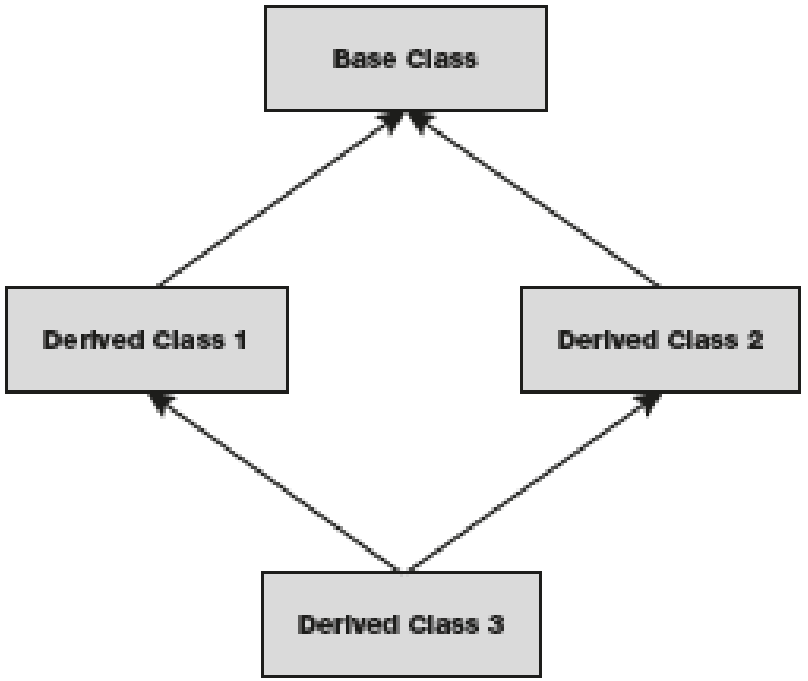


Illustration for hybrid inheritance by creating classes University, Course< Branch and Student using hybrid inheritance

The class University is defined as follows: the instance variable *univ* is initialized with the value *"Anna University"*. The *method* is also defined to display the name of the university.

```
# Creating a Base class named University
class University:
  def __init__(self, uni_name ):
    self.name_of_university = uni_name
  def display(self):
    print("%s"%(self.name_of_university))
```

Now, the derived class course from the parent class University is defined as follows that stores the course name and also the method function to print the course name.

```
# Derived Class Course
class Course(University):
  def __init__(self, course_name):
    self.course_name = course_name
  def display(self):
    print("%s"%(self.course_name))
```

Similar to the class *Course*, the class B*ranch* is also defined as the derived class from the class *University,* as shown below.

```
# Derived Class Branch
class Branch(University):
  def __init__(self, branch_name, uni_name):
    self.branch_name = branch_name
    University.__init__(self, uni_name)
  def display(self):
    print("%s"%(self.branch_name))
    University.display(self)
```

The class *Student* is derived from the class *Course* and the class *Branch,* where the constructor calls the constructor of the parent classes. The constructor of the indirect base class is called in the constructor of the class Course and so, in turn, the constructor of all the classes is invoked. The definition of the class *Student* is as shown below.

```
# Derived class for Student
class Student(Course, Branch):
  def __init__(self, name, roll_no, course_name, branch_name, uni_name):
    self.name = name
    self.roll_no = roll_no
    Branch.__init__(self, branch_name, uni_name)
    Course.__init__(self, course_name)
  def display(self):
    print("%s"%self.name)
    Course.display(self)
    Branch.display(self)
```

An object *student1* is created for the child class *Student,* and the display function in the class *Student* is invoked as follows.

```
student1 = Student("Hari", 2019209028, "Python Programming", "CSE", "Anna University")
print()
hari.display()
```

The above code is stored as Listing3.py and executed in the command prompt. The following shows the output of the execution of the file.

```
C:\Users\usr> python Listing3.py
Hari
Python Programming
CSE
Anna University
```