Course Title: Programming Language II

Course Code: CSE 111

Semester: Summer 2020

Topic: Object-Oriented Programming (Inheritance)

Table of Contents

1. Inheritance:

Inheritance is a golden rule and powerful feature in OOP (Object Oriented Programming). It allows to define **a Derived Class** (Child Class) which takes all f**unctionalities** (attributes and methods) of the **Base class** (Parent Class). We can add more features in the child class according to our preference.

The best mechanism to reuse a code while building a software or system is inheritance. Otherwise, there will be a number of duplicate codes which can add more complexity.

The benefits of inheritance are:

- It represents real-world relationships well.
- It provides reusability of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

1.1. Parent Class:

Parent class is the class being inherited from, also called base class.
Any class can be a parent class, so the syntax is the same as creating any other class
Example:

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

  def printname(self):
    print(self.firstname, self.lastname)

x = Person("X", "Y")
x.printname()
```

Output:

```
X Y
```

1.2. Child Class:

Child class is the class that inherits from another class, also called derived class.
To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

Example:

Here, a class named Student, which inherit the properties and methods from the Person class.

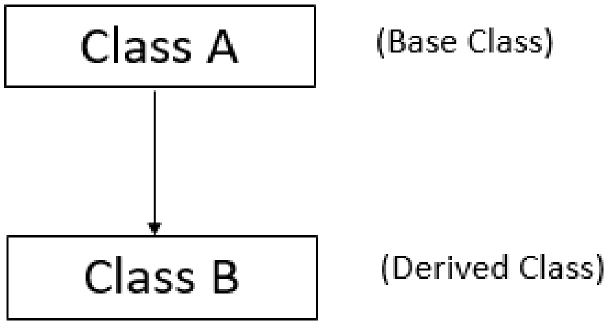
```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)


class Student(Person):
  pass

x = Person("X", "Y")
x.printname()

x = Student("A", "B")
x.printname()
```

Output:

```
X Y

A B
```

2. <u>Single Inheritance:</u>

When the child class inherits the properties or functionalities of a single parent class.



Example:

The ch1 object is an instance of the Child class, but it has an access to data attributes and methods described by both Parent and Child classes.

```python
# Parent class created
class Parent:
    parentname = ""
    childname = ""

    def show_parent(self):
        print(self.parentname)


# Child class created inherits Parent class
class Child(Parent):
    def show_child(self):
        print(self.childname)


ch1 = Child()  # Object of Child class
ch1.parentname = "Mark"   # Access Parent class attributes
ch1.childname = "John"
ch1.show_parent()    # Access Parent class method
ch1.show_child()     # Access Child class method
```

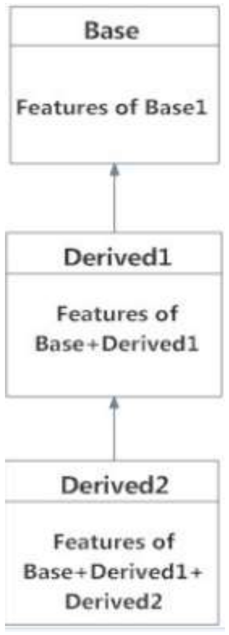
Output:

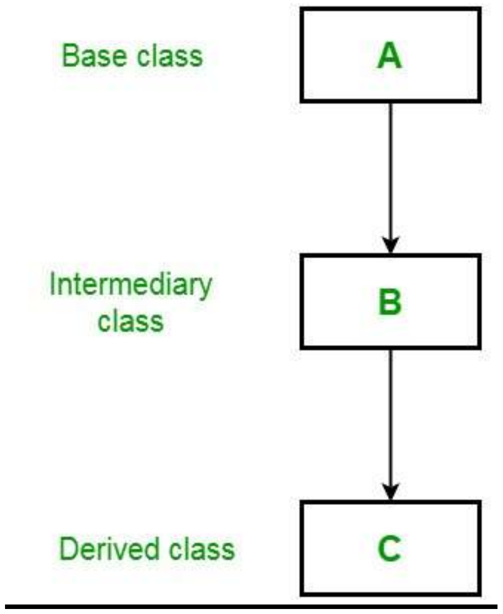
```
Mark

John
```

3. <u>**Multilevel Inheritance:**</u>

Multi-level inheritance is archived when a derived class inherits another derived class.

There is no limit on the number of levels up to which, the multi-level inheritance is archived in python. In this type of inheritance, a class can inherit from a child class or derived class.


Base
Features of Base1
Derived1
Features of Base+Derived1
Derived2
Features of Base+Derived1+ Derived2



Example:

Here, **Son** class inherited from **Father** and **Mother** classes which derived from **Family** class.

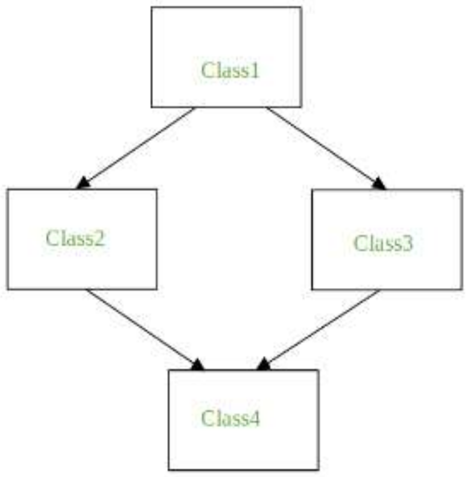
```
class Family:

    def show_family(self):
        print("This is our family:")

# Father class inherited from Family
class Father(Family):
    fathername = ""

    def show_father(self):
        print(self.fathername)

# Mother class inherited from Family
class Mother(Family):
    mothername = ""

    def show_mother(self):
        print(self.mothername)

# Son class inherited from Father and Mother classes
class Son(Father, Mother):
    def show_parent(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)

s1 = Son()  # Object of Son class
s1.fathername = "Mark"
s1.mothername = "Sonia"
s1.show_family()
s1.show_parent()
```

Output:

```
This is our family:

Father : Mark

Mother : Sonia
```

4. <u>Method Resolution Order (MRO):</u>

Method Resolution Order(MRO) it denotes the way a programming language resolves a method or attribute. Python supports classes inheriting from other classes. The class being inherited is called the Parent or Superclass, while the class that inherits is called the Child or Subclass. In python, method resolution order defines the order in which the base classes are searched when executing a method. First, the method or attribute is searched within a class and then it follows the order we specified while inheriting. This order is also called Linearization of a class and set of rules are called MRO(Method Resolution Order). While inheriting from another class, the interpreter needs a way to resolve the methods that are being called via an instance.



In the following example, to find the output, python will travel all the parents of Class4 first (from left to right) then to the upper level parent classes (if any).

```
class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    pass

class Class3(Class1):
    def m(self):
        print("In Class3")

class Class4(Class2, Class3):
    pass

obj = Class4()
obj.m()
```

Output:

```
In Class3
```

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

Python's built-in super() function provides a shortcut for calling base classes, because it automatically follows Method Resolution Order. If the child class inherits more than one class and Super() is used inside its constructor then the constructor of the leftmost parent class will be invoked.

```
class Animal:

  def __init__(self, Animal):
    print(Animal, 'is an animal.');

class Mammal(Animal):
  def __init__(self, mammalName):
    print(mammalName, 'is a warm-
blooded animal.')
    super().__init__(mammalName)

class NonWingedMammal(Mammal):
  def __init__(self, NonWingedMammal):
    print(NonWingedMammal, "can't fly.")
    super().__init__(NonWingedMammal)

class NonMarineMammal(Mammal):
  def __init__(self, NonMarineMammal):
    print(NonMarineMammal, "can't swim.")
    super().__init__(NonMarineMammal)

class Dog(NonMarineMammal, NonWingedMammal):
  def __init__(self):
    print('Dog has 4 legs.');
    super().__init__('Dog')

d = Dog()
print('')
bat = NonMarineMammal('Bat')
```

Output:

```
Dog has 4 legs.
Dog can't swim.
Dog can't fly.
Dog is a warm-blooded animal.
Dog is an animal.


Bat can't swim.
Bat is a warm-blooded animal.
Bat is an animal.
```

Here, a method in the derived calls is always called before the method of the base class.

In example, Dog class is called before NonMarineMammal or NoneWingedMammal. These two classes are called before Mammal, which is called before Animal, and Animal class is called before the object.

If there are multiple parents like Dog(NonMarineMammal, NonWingedMammal), methods of NonMarineMammal is invoked first because it appears first.