## UNIT 5

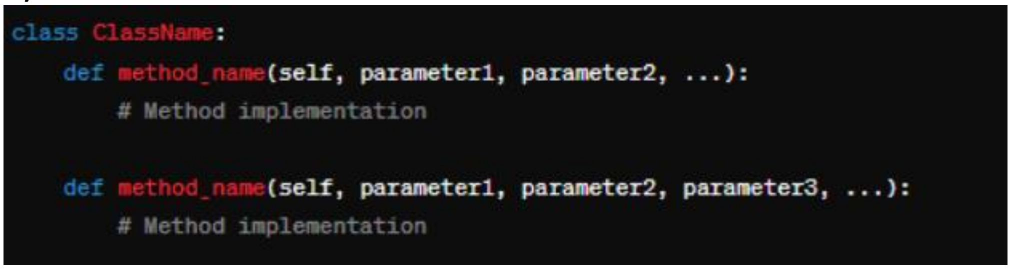
**1.Explain the concept of class & object with syntax and example.**

Class: A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together.
Object: An object is an instance of a class that has some attributes and behavior. Objects can be used to access the attributes of the class.

**2. Explain the concept of method overloading with syntax and example.**

Method overloading is the ability to define the method with the same name but with a different number of arguments and data types.
With this ability one method can perform different tasks, depending on the number of arguments or the types of the arguments given.
Method overloading is a concept in which a method in a class performs operations according to the parameters passed to it.
Python does not support method overloading, that is, it is not possible to define more than one method with the same name in a class in Python.
This is because method arguments in python do not have a type. A method accepting one argument can be called with an integer value, a string or a double as shown in next example.

Syntax:



```
class ClassName:
    def method_name(self, parameter1, parameter2, ...):
        # Method implementation

    def method_name(self, parameter1, parameter2, parameter3, ...):
        # Method implementation
```

Example: With a method to perform different operations using method overloading.

```
class operation:
def add(self,a,b):
return a+b
op1=operation()
print("Addition of integer numbers=",op1.add(10,20))
# To add two floting point numbers
print("Addition of integer numbers=",op1.add(11.12,12.13))
# To add two strings
print("Addition of integer numbers=",op1.add("Hello","Python"))
```

Output:

```
Addition of integer numbers= 30
Addition of integer numbers= 23.25
Addition of integer numbers= HelloPython
```

**3. Explain the concept of method overriding with syntax and example.**

Method overriding is a concept in object-oriented programming where a subclass provides a different implementation of a method that is already defined in its superclass. This allows the subclass to modify or extend the behavior of the inherited method.
In Python, method overriding is achieved by defining a method with the same name in the subclass as the one in the superclass. The syntax for method overriding in Python is as follows:

Syntax:
```
class Superclass:
    def method(self):
        # Superclass method implementation

class Subclass(Superclass):
    def method(self):
        # Subclass method implementation
```

Here's an example to illustrate method overriding in Python:

```
class Animal:
    def sound(self):
        print("Animal makes a sound.")

class Dog(Animal):
    def sound(self):
        print("Dog barks.")

class Cat(Animal):
    def sound(self):
        print("Cat meows.")

animal = Animal()
dog = Dog()
cat = Cat()

animal.sound()  # Output: "Animal makes a sound."
dog.sound()     # Output: "Dog barks."
cat.sound()     # Output: "Cat meows."
```

In this example, we have a superclass called `Animal` with a method called `sound()`. The `Dog` and `Cat` subclasses inherit from the `Animal` class and override the `sound()` method with their own implementations.

When we create objects of the `Animal`, `Dog`, and `Cat` classes and call the `sound()` method on each object, the appropriate implementation based on the object's class is executed.

**4. Differentiate between method overloading and method overriding in detail with example.**

**5. What is inheritance? List its types with example.**

One of the core concepts in object-oriented programming (OOP) languages is inheritance. It is a mechanism that allows you to create a hierarchy of classes that share a set of properties and methods by deriving a class from another class. Inheritance is the capability of one class to derive or inherit the properties from another class.
It represents real-world relationships well.
It provides the 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.

```
Class BaseClass:
   {Body}
Class DerivedClass(BaseClass):
   {Body}
```

Types :

**Single Inheritance:** A subclass inherits from only one superclass.

**Multiple Inheritance:** A subclass inherits from more than one superclass.

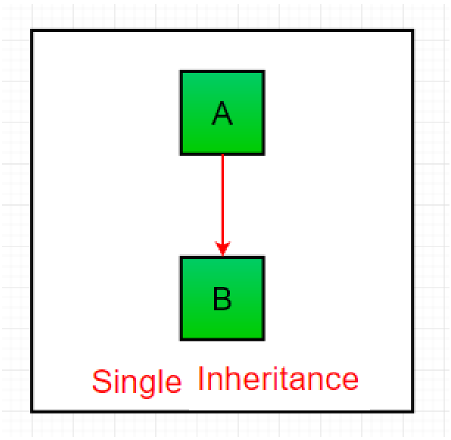
**Multilevel Inheritance:** A subclass inherits from another subclass, forming a hierarchy of classes.

**Hierarchical Inheritance:** Multiple subclasses inherit from the same superclass.

**Hybrid Inheritance:** Combination of two or more types of inheritance.

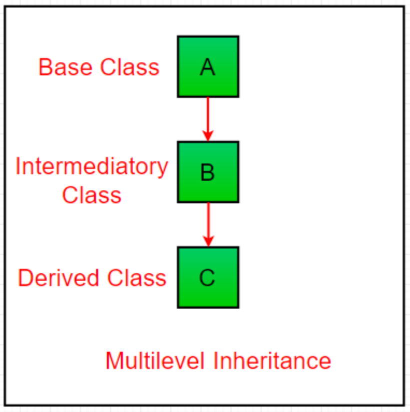
**6. Explain Single inheritance with example and syntax.**

Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.



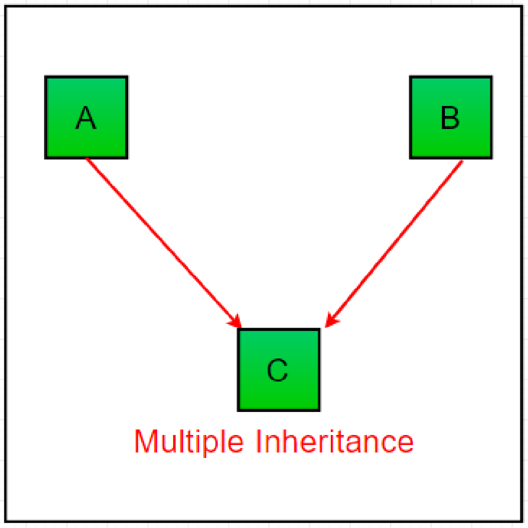
**7. Explain multi level inheritance with example and syntax.**

In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class. This is similar to a relationship representing a child and a grandfather.



**9. Explain multiple inheritance with example and syntax.**

When a class can be derived from more than one base class this type of inheritance is called multiple inheritances. In multiple inheritances, all the features of the base classes are inherited into the derived class.



**10. Explain use of namespace in python**
**11. Illustrate class inheritance in Python with an example**
**13. Illustrate the use of method overriding? Explain with example**