



TOPSTechnologies

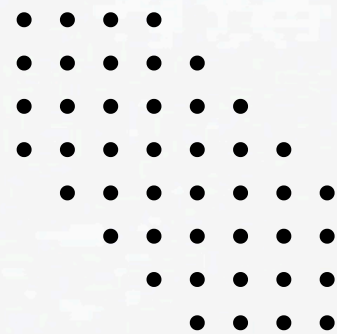
# Method Overloading and Overriding

**Presented for :**

TOPs Technologies

**Presented by :**

Sumit B Yadav



### *Que 1*

Method Overloading allows multiple methods to have the same name but with different parameters in the same class. This is useful when you want a method to perform similar operations but with different inputs.

#### Rules for Method Overloading:

- Methods must have the same name but different parameters (different type, number, or sequence).
- Return type can be the same or different, but it does not determine overloading.
- Overloading occurs within the same class.
- Access modifiers (public, private, protected) can be different.
- Method signatures must be unique (based on parameter list).
- Overloading is resolved at compile-time (also called compile-time polymorphism or static binding).

example:

```
class MathOperations:  
    def add(self, a, b, c=0): # Default value for c  
        return a + b + c
```

# Usage

```
obj = MathOperations()  
print(obj.add(5, 10))    # Calls method with two arguments  
print(obj.add(5, 10, 15)) # Calls method with three arguments
```

### Que. 2

Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its parent class. The overridden method in the child class must have the same name and parameters as in the parent class.

#### Rules for Method Overriding

- The method name and parameters must be the same in both parent and child classes.
- The child class method overrides the parent class method when called using a child object.
- Supports runtime polymorphism (method resolution happens at runtime).
- The `super()` function can be used to call the parent class method inside the child class.

Basic syntax:

```
class Animal: def sound(self):  
    print("Animals make sounds")
```

```
class Dog(Animal):  
    def sound(self): # Overriding the parent class method  
        print("Dog barks")
```