TOPSTechnologies

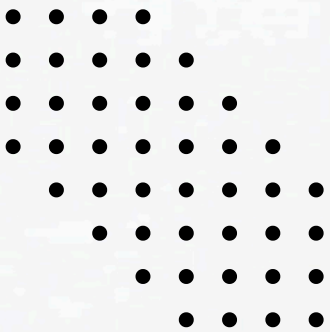
7. Inheritance

Presented for :

TOPs Technologies

Presented by :

Sumit B Yadav



Que 1

Inheritance is a fundamental concept in Object-Oriented Programming (OOP) that allows one class (child) to inherit the properties and behaviors (methods and attributes) of another class (parent).

1. Single Inheritance

- A single child class inherits from a single parent class.
- This is the simplest form of inheritance.

Basic Syntax:

```
class Parent:
def show(self):
print("This is the Parent class")

class Child(Parent):
def display(self):
print("This is the Child class")

c = Child()
c.show()  # Inherited method
c.display()
```

2. Multilevel Inheritance

- A child class inherits from a parent class, and another child class further inherits from it.
- Forms a chain of inheritance.

Basic Syntax:

```
class Grandparent:
def grandparent_method(self):
    print("Grandparent class")

class Parent(Grandparent):
    def parent_method(self):
        print("Parent class")

class Child(Parent):
    def child_method(self):
        print("Child class")

c = Child()
c.grandparent_method()  # Inherited from Grandparent
c.parent_method()       # Inherited from Parent
c.child_method()        # Defined in Child
```

3. Multiple Inheritance

- A child class inherits from more than one parent class.
- Python follows the Method Resolution Order (MRO) to decide which method to execute if there are conflicts.

basic syntax:

```
class Parent1:
def method1(self):
print("Method from Parent1")

class Parent2:
def method2(self):
print("Method from Parent2")

class Child(Parent1, Parent2):
def method3(self):
 print("Method from Child")

c = Child()
c.method1()  # From Parent1
c.method2()  # From Parent2
c.method3()  # Defined in Child
```

4. Hierarchical Inheritance

- Multiple child classes inherit from a single parent class.
- Each child class has access to the parent class's methods and attributes but does not inherit from each other.

basic syntax:

```
class Parent:
def common_method(self):
print("Common method from Parent")

class Child1(Parent):
def child1_method(self):
print("Child1 specific method")

class Child2(Parent):
def child2_method(self):
print("Child2 specific method")

c1 = Child1()
c2 = Child2()

c1.common_method()  # Inherited from Parent
c1.child1_method()

c2.common_method()  # Inherited from Parent
c2.child2_method()
```

5. Hybrid Inheritance

- A combination of two or more types of inheritance.
- Typically involves a mix of hierarchical, multiple, and multilevel inheritance.

basic syntax:

```
class Base:
def base_method(self):
 print("Base class method")

class Parent1(Base):
def parent1_method(self):
  print("Parent1 method")

class Parent2(Base):
def parent2_method(self):
  print("Parent2 method")

class Child(Parent1, Parent2):
def child_method(self):
  print("Child method")

c = Child()
c.base_method()   # From Base
c.parent1_method()  # From Parent1
c.parent2_method()  # From Parent2
c.child_method()  # From Child
```

In Python, the super() function allows you to access methods and properties of a parent (or superclass) class from within a child (or subclass) class. This is particularly useful when working with inheritance.

Using super() to Access Parent Class Properties
You can use super() to access attributes and methods from the parent class inside the child class
example:

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

def display(self):
print(f"Parent Name: {self.name}")

class Child(Parent):
def __init__(self, name, age):
super().__init__(name)  # Calls the Parent class's constructor
self.age = age

def display(self):
super().display()  # Calls the Parent class's display method
print(f"Child Age: {self.age}")

# Create an instance of Child
child = Child("Alice", 10)
child.display()
```