Lab # 09

Inheritance

Object:

Get familiar with the concept of Inheritance in Python.

Theory:

Inheritance

One of the key concepts of object-oriented programming is inheritance, which allows new classes to inherit features or attributes from the existing classes. This mechanism creates a hierarchy of classes that share similar properties, methods, and features as new classes are derived from the old classes. In simpler terms, we can say that inheritance is the capability of a class to derive properties of another class. The new class is known as the child class, and the existing class is called the parent class.
The inherited features and properties are defined data structures and functions we can perform with methods. Inheritance promotes code reusability, which is one of the best industrial coding practices that makes the codebase modular. The new class, or derived class, copies all the functions of the older class, or base class, without rewriting the syntax in the new class.

Syntax

```
class Parent:
    # attributes and methods

class Child(Parent):
    # additional attributes and methods
```

Explanation:

Parent Class:

- This is the base class from which other classes inherit.
- It contains attributes and methods that the child class can reuse.

Child Class:

- This is the derived class that inherits from the parent class.
- The syntax for inheritance is class Child(Parent).
- The child class automatically gets all attributes and methods of the parent class unless overridden.

Example: basic concepts of inheritance

```
class Parent:
    def greet(self):
        print("Hello from Parent")

class Child(Parent):
    def speak(self):
        print("Child is speaking")

c = Child()
c.greet()  # Inherited from Parent
c.speak()  # Defined in Child
```

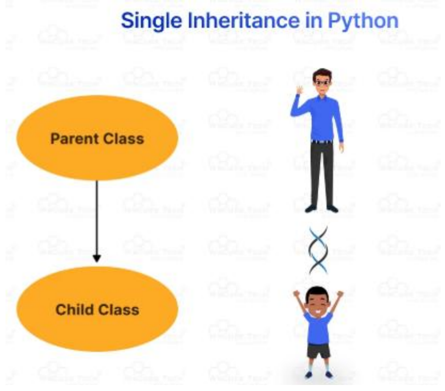
Types of Inheritance

Python offers various types of inheritance based on the number of child and parent classes involved in the inheritance.

- Single Inheritance in Python
- Multiple Inheritances in Python
- Multilevel Inheritance in Python
- Hierarchical Inheritance in Python
- Hybrid Inheritance in Python

Single Inheritance in Python

Single inheritance is the simplest type of inheritance where a single child class inherits from one parent class. It is also called simple inheritance due to its candid nature. You can create an instance of ConcreteClass and call the implemented methods.

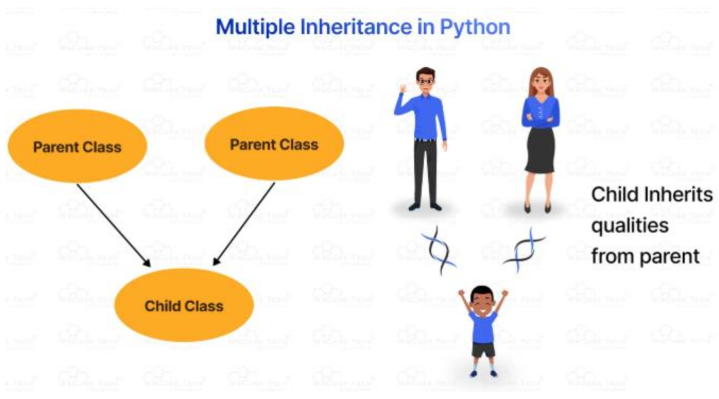


Example 1: Single level Inheritance

```
# Base class
class Animal:
    def speak(self):
        return "Animal makes a sound"

# Derived class
class Dog(Animal):
    def speak(self):
        return "Woof!"

# Create an instance of the derived class
my_dog = Dog()

# Call the method from the derived class
print(my_dog.speak())  # Output: Woof!
```

Multiple Inheritance in Python

In multiple inheritance, a single subclass inherits from multiple superclasses. So, the child class can access two or more parent classes it has inherited attributes and methods from. However, if two parent classes share the same ‘named’ methods, the child class performs only the method of the first parent class in order of reference.


Multiple Inheritance in Python
Parent Class
Parent Class
Child Class
Child Inherits qualities from parent

Example 2: Multiple level Inheritance

```
# Base class 1
class Person:
    def __init__(self, name):
        self.name = name
    def greet(self):
        return f"Hello, my name is {self.name}."
# Base class 2
class Employee:
    def __init__(self, employee_id):
        self.employee_id = employee_id
    def get_id(self):
        return f"My employee ID is {self.employee_id}."
# Derived class inheriting from both Person and Employee
class Manager(Person, Employee):
    def __init__(self, name, employee_id, department):
        Person.__init__(self, name)  # Initialize Person part
        Employee.__init__(self, employee_id)  # Initialize Employee part
        self.department = department
    def show_details(self):
        return f"{self.greet()} {self.get_id()} I manage the {self.department} department."

# Create an instance of the derived class
manager = Manager("Alice", "E12345", "Sales")

# Call methods from base classes and derived class
print(manager.greet())        # Output: Hello, my name is Alice.
print(manager.get_id())       # Output: My employee ID is E12345.
print(manager.show_details())  # Output: Hello, my name is Alice. My employee ID is E12345. I manage the Sales department.
```

Explanation:

Base Classes (Person, Employee):

- Person class has a name attribute and a method greet().
- Employee class has an employee_id attribute and a method get_id().

Derived Class (Manager):

- Inherits from both Person and Employee.
- The __init__ method initializes attributes from both base classes and adds its own attribute department.
- The show_details() method combines functionality from both base classes and its own method.

Object Creation and Method Calls:

- An instance of Manager is created with name, employee_id, and department.

- Calls to greet(), get_id(), and show_details() demonstrate the combined behavior from multiple base classes.

Tasks:

1. Single Inheritance
    a. Create a class Animal with a method eat() that prints "Animal eats".
    b. Create a class Dog that inherits from Animal and has a method bark() that prints "Dog barks".
    c. Create a Dog object and call both eat() and bark().
2. Method Overriding
    a. In the `Dog` class, override the `eat()` method to print "Dog eats dog food".
    b. Create a `Dog` object and call `eat()`.
3. Calling Parent Methods
    a. Modify the eat() method in Dog to:
        i. First call super().eat()
        ii. Then print "Dog eats dog food"
    b. Run and observe the combined behavior.