**Inheritance Questions in Python**

1. Single Inheritance:

Create a base class Person with a method display_name(). Inherit it in a class Student and call the method.

```
class Person:
    def display_name(self):
        print("Name: RAKSHA G A")
class Student(Person):
    def student_info(self):
        print("This is a student.")
s = Student()
s.display_name()
s.student_info()
```

**Output :**

```
Name: RAKSHA G A
This is a student.
```

2. Multilevel Inheritance:

Design 3 classes: Animal → Mammal → Dog, where each class has its own method and Dog inherits all behaviors.

```
class Animal:
    def sound(self):
        print("Animals make sounds")
class Mammal(Animal):
    def walk(self):
        print("Mammals walk on land")
class Dog(Mammal):
    def bark(self):
        print("Dogs bark")
```

```
d = Dog()
d.sound()
d.walk()
d.bark()
```

**Output :**

```
Animals make sounds
Mammals walk on land
Dogs bark
```

3. Multiple Inheritance: Create two classes Flyable and Swimmable, each with a method. Derive a class Duck from both and call both methods.

```
class Flyable:
    def fly(self):
        print("Duck Fly in the sky.")
class Swimmable:
    def swim(self):
        print("Duck Swim in the water.")
class Duck(Flyable, Swimmable):
    def display(self):
        print("The Duck")
d = Duck()
d.display()
d.fly()
d.swim()
```

**Output :**

```
The Duck
Duck Fly in the sky.
Duck Swim in the water.
```

4. Hierarchical Inheritance:

Define a parent class Vehicle, and create two child classes Car and Bike. Show how each inherits from Vehicle.

```
class Vehicle:
    def start(self):
        print("Vehicle engine starts.")
class Car(Vehicle):
    def drive(self):
        print("Car is ready for driving.")
class Bike(Vehicle):
    def ride(self):
        print("Bike is ready for riding.")
c = Car()
b = Bike()
c.start()
c.drive()
b.start()
b.ride()
```

**Output :**

```
Vehicle engine starts.
Car is ready for driving.
Vehicle engine starts.
Bike is ready for riding.
```

5. Use super() in a derived class to call a parent class's method. What happens if both classes have the same method name?

When both the parent class and child class have a method with the same name, and we use super() in the child class to call that method, Python will first call the parent class’s code, and then execute the child class's code after it.

```
class A:
    def show(self):
        print("Hello from parent class")
class B(A):
    def show(self):
        super().show()
        print("Hello from child class")
b = B()
b.show()
```

**Output :**

```
Hello from parent class
Hello from child class
```

6. What is Method Resolution Order (MRO) in multiple inheritance? Demonstrate using a diamond problem structure.

Method Resolution Order (MRO) in Python defines the order in which base classes are searched when executing a method or attribute lookup. This becomes especially important in multiple inheritance, where a class inherits from more than one parent class.

```
class A:
    def show(self):
        print("class A")
class B(A):
    def show(self):
        print("class B")
class C(A):
    def show(self):
        print("class C")
class D(B,C):
```

```
    pass
obj = D()
obj.show() # d-->b--->c--->a--->object
obj1 = C()
obj1.show()
print(D.__mro__)
print(C.__mro__)
```

**Output :**

```
class B
class C
(<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>)
(<class '__main__.C'>, <class '__main__.A'>, <class 'object'>)
```

7. Define a constructor in the base class. In the derived class, call it using super().__init__() and add new attributes.

```
class Person:
    def __init__(self, name):
        self.name = name
class Student(Person):
    def __init__(self, name, roll):
        super().__init__(name)
        self.roll = roll
s = Student("Raksha", 24)
print(s.name, s.roll)
```

**Output :**

```
Raksha 24
```

8. Can you override a method in Python? Write a base class Shape with a method area() and override it in Circle.

Yes, we can override. If a child class has the same method name, it replaces the base one.

```
class Shape:
    def area(self):
        return "Area not defined"
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius ** 2
circle = Circle(10)
print("Area of circle is ",circle.area())
```

**Output :**

```
Area of circle is  314.0
```

**Polymorphism Questions in Python**

9. Method Overriding:

Write a base class Animal with method speak(). Create subclasses Dog, Cat that override speak().

```
class Animal:
    def speak(self):
        print("Animal sound")
class Dog(Animal):
    def speak(self):
        print("Dog barks")
class Cat(Animal):
    def speak(self):
        print("Cat meows")
```

```
d = Dog()
c = Cat()
d.speak()
c.speak()
```

**Output :**

```
Dog barks
Cat meows
```

10. Polymorphic Behavior:

Create a list of objects of Dog, Cat, Cow, each inheriting from Animal. Iterate and call speak() method.

```
class Animal:
    def speak(self):
        print("Animal sound")
class Dog(Animal):
    def speak(self):
        print("Dogs Bark")
class Cat(Animal):
    def speak(self):
        print("Cats Meow")
class Cow(Animal):
    def speak(self):
        print("Cows Moo")
ani = [Dog(), Cat(), Cow()]
for a in ani:
    a.speak()
```

**Output :**

Dogs Bark

Cats Meow

Cows Moo

11. Simulated Method Overloading:

Python doesn't support method overloading directly. Show how you can use default or *args to mimic it.

```
# using *args
class Calculator:
    def add(self, *args):
        total = sum(args)
        print(f"Sum is: {total}")
c = Calculator()
c.add(10, 20)
c.add(5, 15, 25, 35)
```

**Output :**

Sum is: 30

Sum is: 80

```
# using default
class Greet:
    def hello(self, name):
        print(f"Hello, {name}!")
g = Greet()
g.hello("Alia")
g.hello("Raksha")
```

**Output :**

Hello, Alia!

Hello, Raksha!

12. Write a class Calculator with a method add() that supports 2 and 3 arguments using default parameters or *args.

```
class Calculator:
    def add(self, *args):
        total = sum(args)
        print("Sum:", total)
calc = Calculator()
calc.add(5, 10)
calc.add(1, 2, 3)
```

**Output :**

Sum: 15

Sum: 6

13. Can you override the __str__() method in Python? Create a class Book that returns a custom string when printed.

```
class Book:
    def __init__(self, title):
        self.title = title
    def __str__(self):
        return f"Book Title: {self.title}"
b = Book("Python")
print(b)
```

**Output :**

Book Title: Python

14. Demonstrate polymorphism using duck typing. Write a function start_engine(vehicle) that takes any object with a method start().

```
class Car:
    def start(self):
        print("Car engine started")
class Bike:
    def start(self):
        print("Bike engine started")
def start_engine(vehicle):
    vehicle.start()
start_engine(Car())
start_engine(Bike())
```

**Output :**

```
Car engine started
Bike engine started
```

15. How does polymorphism help in writing more generic functions in Python? Provide a small real-world code snippet.

```
class CreditCard:
    def pay(self, amount):
        print(f"Paid {amount} using Credit Card.")
class UPI:
    def pay(self, amount):
        print(f"Paid {amount} using UPI.")
def process_payment(method, amount):
    method.pay(amount)
c = CreditCard()
upi = UPI()
```

process_payment(c, 500)

process_payment(upi, 200)

**Output :**

Paid 500 using Credit Card.

Paid 200 using UPI.