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 , name):

Print(“Name of the person” , name)

Obj = Person()

Obj.display\_name

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

class Animal:

def speak(self):

print("Animal Properties")

class Mammla(Animal):

def speak(self):

super().speak0()

print("Mammla properties")

class Dog(Animal):

def speak(self):

super().speak()

print("Dog barks")

objjj = Dog()

objjj.speak()

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("Flying")

class Swimmable:

def swim(self):

print("Swimming")

class Duck(Flyable, Swimmable):

pass

d = Duck()

d.fly()

d.swim()

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 feature(self):

print("Vehicle features")

class Car(Vehicle):

def car(self):

print("Car features")

class Bike(Vehicle):

def bike(self):

print("Bike features")

c = Vehicle()

c.feature()

he = Bike()

he.bike()

ge = Car()

ge.car()

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

class Camera:

def features(self):

print("Camera features")

class GPS(Camera):

def features(self):

super().features()

print("Location cracked")

class Smart\_phone(GPS):

def features(self):

super().features()

print("Smart phone features")

ohj = Smart\_phone()

ohj.features()

If both classes have a method with the same name, super() calls the parent class version.

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

MRO determines the order in which base classes are searched.

class A:

def show(self):

print("A")

class B(A):

def show(self):

print("B")

class C(A):

def show(self):

print("C")

class D(B, C):

pass

d = D()

d.show()

print(D.mro())

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, grade):

super().\_\_init\_\_(name)

self.grade = grade

s = Student("Gia", "A")

print(s.name, s.grade)

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

YES .

Class Shape:

Def shape(self):

Print(“The shape is Circle”)

Def area(self , r):

Print(“area of the circle”,3.14\*r\*r)

Obj = Shape()

Obj.area(5)

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 speaks")

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()

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):

pass

class Dog(Animal):

def speak(self):

print("Dog barks")

class Cat(Animal):

def speak(self):

print("Cat meows")

class Cow(Animal):

def speak(self):

print("Cow moos")

animals = [Dog(), Cat(), Cow()]

for animal in animals:

animal.speak()

11. Simulated Method Overloading:  
Python doesn’t support method overloading directly. Show how you can use default or \*args to mimic it.

class Demo:

def greet(self, \*args):

if len(args) == 0:

print("Hello!")

elif len(args) == 1:

print(f"Hello, {args[0]}!")

else:

print("Too many arguments")

d = Demo()

d.greet()

d.greet("John")

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):

return sum(args)

c = Calculator()

print(c.add(2, 3))

print(c.add(2, 3, 4)

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, author):

self.title = title

self.author = author

def \_\_str\_\_(self):

return f"{self.title} by {self.author}"

b = Book("1984", "George Orwell")

print(b)

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())

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

class Circle:

def draw(self):

print("Drawing a Circle")

class Square:

def draw(self):

print("Drawing a Square")

def render\_shape(shape):

shape.draw()

shapes = [Circle(), Square()]

for s in shapes:

render\_shape(s)