Here are some common Object-Oriented Programming (OOPs) interview questions and answers, organized by key concepts.

Fundamental OOP Concepts

1. What is Object-Oriented Programming (OOP)?

Object-Oriented Programming is a programming paradigm based on the concept of "objects," which can contain data (in the form of fields, often known as attributes or properties) and code (in the form of procedures, often known as methods). The primary goal of OOP is to organize complex programs by bundling related properties and behaviors into individual objects.

2. What's the difference between a class and an object?

This is a foundational OOP question. 🏗️

- A **Class** is a **blueprint** or template for creating objects. It defines a set of attributes and methods that the created objects will have. For example, a Car class would define properties like color and model.
- An **Object** is an **instance** of a class. It's the real entity created from the class blueprint. For example, my_red_toyota could be an object of the Car class with the color "red" and model "Toyota".

Python

```python
# This is the blueprint (Class)
class Car:
    def __init__(self, color, model):
        self.color = color
        self.model = model

# These are the actual instances (Objects)
car1 = Car("Blue", "Honda")
car2 = Car("Red", "Toyota")
```

The Four Pillars of OOP

3. What are the four main principles of OOP?

The four fundamental principles of Object-Oriented Programming are:

1. **Encapsulation**
2. **Abstraction**
3. **Inheritance**
4. **Polymorphism**

4. What is Encapsulation?

Encapsulation is the bundling of data (attributes) and the methods that operate on that data into a single unit, or "class." It also involves restricting direct access to some of an object's components, which is a key part of data hiding. 💊

Think of a capsule pill. The plastic casing (the class) encloses the medicine (the data). You don't interact with the medicine directly; you use the capsule as intended. In programming, this prevents accidental modification of data. This is often achieved using **private** or **protected** access modifiers.

Python

```python
class Account:
    def __init__(self, initial_balance):
        self.__balance = initial_balance # Private attribute

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount

    def get_balance(self):
        return self.__balance

# The __balance can only be modified via the deposit method.
```

5. What is Abstraction?

Abstraction means hiding complex implementation details and showing only the essential features of the object. It helps manage complexity by focusing on *what* an object does instead of *how* it does it. 🚗

Think about driving a car. You only need to know how to use the steering wheel, pedals, and gear stick (the essential interface). You don't need to know the complex mechanics of the engine or transmission (the hidden implementation) to drive it.

---

6. What is Inheritance?

Inheritance is a mechanism where a new class (child/subclass) derives attributes and methods from an existing class (parent/superclass). It promotes code reusability ("don't repeat yourself") and establishes a relationship between classes. 👨‍👩‍👧

For example, Car, Truck, and Motorcycle classes can all inherit common properties like speed and color from a parent Vehicle class.

There are several types of inheritance:

- **Single Inheritance:** A class inherits from only one parent class.
- **Multiple Inheritance:** A class inherits from more than one parent class.
- **Multi-level Inheritance:** A class inherits from a parent class, which in turn inherits from another parent class (e.g., Class C -> Class B -> Class A).
- **Hierarchical Inheritance:** Multiple classes inherit from a single parent class (e.g., Car and Truck both inherit from Vehicle).

Python

```python
# Parent class
class Animal:
    def speak(self):
        print("Animal speaks")

# Child class inheriting from Animal
class Dog(Animal):
    def bark(self):
        print("Dog barks")

d = Dog()
d.speak() # Inherited from Animal class
d.bark()
```

7. What is Polymorphism?

Polymorphism, which means "many forms," is the ability of an object to take on many forms. In practice, it means that a single function or method name can work differently depending on the object it's called on. 🎭

The two main types are:

- **Method Overriding (Runtime Polymorphism):** A subclass provides a specific implementation of a method that is already provided by its parent class. The call to the method is resolved at runtime.
- **Method Overloading (Compile-time Polymorphism):** Defining multiple methods with the same name but with different parameters (either in number or type). *Note: Python doesn't support traditional method overloading like C++ or Java, but it can be simulated.*

Example of Method Overriding:

Python

```python
class Bird:
    def fly(self):
        print("Most birds can fly")

class Penguin(Bird):
    # Overriding the parent's fly method
    def fly(self):
        print("Penguins can't fly, they swim")

p = Penguin()
p.fly() # Calls the Penguin's version of fly()
```