

In Python, **Object-Oriented Programming (OOP)** allows developers to structure their code in a more efficient, scalable, and maintainable way by using classes and objects. These objects can have attributes (data) and methods (functions) that represent both the state and behaviour of an entity. Python OOP provides a powerful approach to managing complex systems and applications.

## Table of Content

- [Basic OOPS Interview Questions](#)
- [Intermediate Python OOPs Interview Questions](#)
- [Advanced Python Interview OOPs Questions](#)

In this article, we will explore common **Object-Oriented Programming(OOP) interview questions** related to Python. Let's explore some key OOP interview questions that will help you master Python's OOP features.

## Basic OOPS Interview Questions

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

Object-Oriented Programming (OOP) is a fundamental paradigm in Python which is designed to model real-world entities through classes and objects. It offers features like **encapsulation, inheritance, polymorphism, and abstraction** enabling developers to write reusable, modular, and maintainable code.

### 2. What are the key features of OOP?

The key features of oop are:

- **Encapsulation**: Bundles data and methods within a class, hiding internal details from outside access.
- **Abstraction**: Hides complex implementation details and exposes only necessary functionality.
- **Inheritance**: Allows a new class to inherit properties and methods from an existing class.
- **Polymorphism**: Enables a single interface to represent different behaviors.

### 3. What is a class and an object in Python?

- **Class**: A blueprint for creating objects, defining their properties and methods.

- **Object:** An instance of a class.

### Example:

```
1 class Dog:
2
3     def __init__(self, name, breed):
4
5         self.name = name
6
7         self.breed = breed
```

## 4. What is the difference between a class and an instance?

- Class: Defines the structure and behavior (attributes and methods).
- Instance: A concrete occurrence of a class with actual data

## 5. What is the `__init__` method in Python?

The `__init__` method is a constructor used to initialize an object's attributes when it is created and it is also known as constructor. The `__init__` method is part of Python's object-oriented programming (OOP) mechanism and is used to set up the initial state of the object when it is instantiated.

### Example:

```
1 class Person:
```



2

```
def __init__(self, name):
```

## 6. What is `self` in Python classes?

`self` is a reference to the current instance of the class. It is used to access attributes and methods of the class. When you define a method inside a class, the first parameter of the method is always `self`, which allows the method to refer to the object (or instance) on which the method was called.

## 7. What is the difference between instance variables and class variables?

- **Instance variables:** Instance variables unique to each object.
- **Class variables:** Shared among all objects of a class.

### Example:



1

```
class Demo:
```



2

```
    class_var = "shared" # class variable
```

3

## 8. What is inheritance in Python?

Inheritance allows a class to inherit attributes and methods from another class. Inheritance enables the child class to inherit the properties (attributes) and behaviors (methods) of the parent class, and it can also define its own additional attributes and methods or override existing ones.

### Example:



```
1  # Define the Parent class (base class)
2
3  class Parent:
```

## 9. What is method overloading in Python?

**Method overloading** in Python refers to the ability to define multiple methods with the same name but with different **parameters** (different numbers or types of arguments).

**Example:**

```
1  def greet(name="Guest"):
2
```

## 10. What is method overriding in Python?

**Method overriding** allows a subclass to provide a specific implementation of a method that is already defined in its superclass. This means that the subclass can "override" the behavior of the method inherited from the parent class, providing a different version that is more suitable for the subclass.

**Example:**

```
1  class Parent:
2
3      def show(self):
```

## Intermediate Python OOPS Interview Questions

### 11. What is polymorphism in Python?

Polymorphism allows objects to be treated as instances of their parent class, enabling different implementations for the same interface. It enables a single function, method, or operator to work with different types of objects in a way that is determined at runtime, allowing code to be more flexible and reusable.

**Example:**

```
1  class Bird:
2
3      # Method to make a sound for the Bird class
4
5      def speak(self):
6
7          print("Chirp")
```

### 12. What is encapsulation, and how does Python achieve it?

Encapsulation is one of the fundamental principles of OOP. It allows the internal representation of an object to be hidden from the outside world and exposes only what is necessary through a controlled interface.

**Example:**

### 13. WI

`super().`

**Examp**

```
class Example:
```

```
    def __init__(self):
```

structor.



```
class A:
```

```
    def method(self):
```

```
        print("Method in class A")
```

```
class B(A):
```

### 14. What are abstract classes in Python?

Abstract class is a class that serves as a blueprint for other classes. It defines a structure that derived classes must follow but does not provide full implementation, abstract classes cannot be instantiated directly which means they are meant to be subclassed.

**Example:**



```
from abc import ABC, abstractmethod
```

```
class Animal(ABC):
```

## 15. What is the difference between `is` and `==`?

'is' checks if two objects are the same instance(identity) and '==' checks if the two objects have the same value(equality).

## 16. What is multiple inheritance in Python?

Multiple inheritance allows a subclass to inherit features from multiple parent classes, making it more versatile and capable of combining behaviors from different sources.

```
1 class ChildClass(ParentClass1, ParentClass2, ...):
```

## 17. What is the diamond problem in multiple inheritance? How does Python handle it?

The **Diamond Problem** in multiple inheritance is a classic issue that arises when a class inherits from two classes that both inherit from a common ancestor. The problem occurs because, in such a scenario, it's unclear which version of the inherited method should be invoked when the method is called on the subclass.

## 18. What are class methods in Python?

Class methods are defined with the @classmethod decorator and take 'cls' (class reference) as their first argument.

```
1 class Demo:
2     @classmethod
```

## 19. What is the difference between `staticmethod` and `classmethod`?

In Python, both `@staticmethod` and `@classmethod` are used to define methods that are not bound to the instance of the class (i.e., they can be called on the class itself).

Aspect	<code>staticmethod</code>	<code>classmethod</code>
Binding	Not bound to either the instance or the class.	Bound to the class, not the instance.
First Argument	Does not take <code>self</code> or <code>cls</code> as the first argument.	Takes <code>cls</code> as the first argument (refers to the class).
Access to Class/Instance Variables	Cannot access or modify instance or class variables.	Can access and modify class variables (but not instance variables).
Call	Can be called on the class or an instance.	Can be called on the class or an instance.

## 20. What are magic methods in Python?

Magic methods (dunder methods) start and end with double underscores providing operator overloading and custom behaviors.

**Examples:** `__str__`, `__len__` and `__init__` so on...

## Advanced Python Interview OOPs Questions

### 21. How does Python handle garbage collection?



Python uses automatic garbage collection to manage memory by identifying and removing unused objects and It employs reference counting and a cyclic garbage collector.

## 22. What is metaclass in Python?

A metaclass is a class of a class that defines how classes behave and classes are instances of metaclasses. Essentially, metaclasses define the "rules" for building classes, much like a class defines the rules for creating objects.

**Example:**

```
1 class Meta(type):  
2     # This is an empty metaclass that inherits from 'type'  
3     pass
```

## 23. How is data hiding implemented in Python?

Data hiding is implemented using private attributes (\_\_attribute) and even though it is not completely hidden, it is still difficult to access without name mangling.

## 24.What is the purpose of \_\_slots\_\_ in Python classes?

\_\_slots\_\_ attribute limits the attributes that can be added to an instance improving memory usage.

**Example:**

```
1 class Example:
```

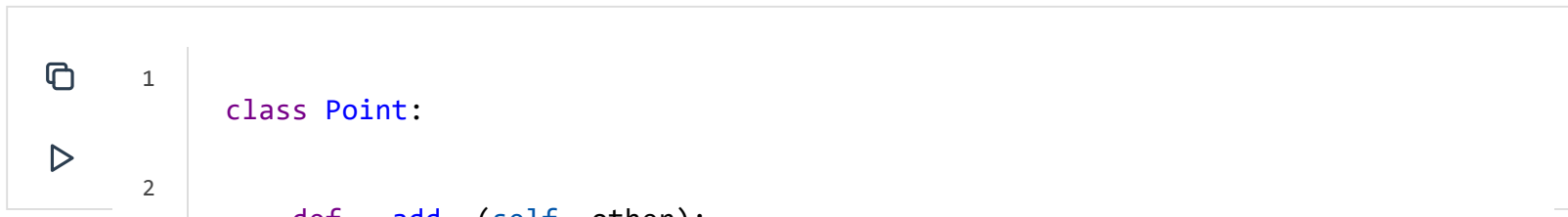
## 25. What is the Global Interpreter Lock (GIL) and its impact on Python OOP?

The GIL ensures only one thread executes Python bytecode at a time affecting multithreaded programs. Its purpose is to prevent multiple native threads from executing Python bytecode at the same time, ensuring thread safety in Python's memory management system. .

## 26.How do you achieve operator overloading in Python?

Operator overloading is achieved using magic methods like `__add__` and `__eq__`.

**Example:**



```
1 class Point:
2     def add (self, other):
```

## 26. What is the difference between shallow copy and deep copy in Python?

In Python, shallow copy and deep copy are used to create copies of objects. They differ in how they handle nested objects, such as lists within lists or objects containing other objects. A **shallow copy** creates a new object but does not copy the nested objects inside it and A deep copy creates a new object and recursively copies all objects within it, including nested objects.

## 27. What is the difference between composition and inheritance?

Composition and inheritance are two fundamental object-oriented programming (OOP) techniques for creating relationships between classes. The main difference between inheritance and composition is that:

- **Inheritance:** "Is-a" relationship (e.g., a Car is-a Vehicle).
- **Composition:** "Has-a" relationship (e.g., a Car has-a Engine).

## 28. How do you implement design patterns like Singleton in Python?

Singleton ensures only one instance of a class exists. its design pattern ensures that a class has only one instance and provides a global point of access to it.

#### Example:

```
1 class Singleton:
2     _instance = None
3
4     def __new__(cls, *args, **kwargs):
```

## 29. Explain Python's MRO with an example.

Python's `MRO`, determines the order in which classes are searched when calling a method or accessing an attribute. It is crucial in object-oriented programming, especially when dealing with multiple inheritance.

#### Example:

```
1 class A:
2     # Class A is a base class with no methods or attributes
3
4     pass
5
6 class B(A):
```

6

# Class B inherits from A

7

### 30. How would you identify the MRO of a class programmatically?

We can identify the Method Resolution Order (MRO) of a class programmatically in Python using the `mro()` Method or by using the `__mro__` attribute.

- **Using the `mro()` method:** This method returns a list of classes in the order they are checked for method resolution.
- **Using the `__mro__` attribute:** This attribute directly provides the MRO as a tuple of classes.