### **Q1. What is the purpose of Python's OOP?**

Python's Object-Oriented Programming (OOP) allows for structuring programs in a way that bundles data (attributes) and behavior (methods) into objects. The purpose is to facilitate code reusability, modularity, and organization by representing real-world entities as classes and objects. OOP helps in managing complexity by enabling inheritance, encapsulation, and polymorphism.

### **Q2. Where does an inheritance search look for an attribute?**

In an inheritance search, Python first looks for the attribute in the instance object, then in the class it was created from, and then recursively up the chain of superclasses (i.e., the class's parent classes). This search follows the method resolution order (MRO), which is usually a depth-first, left-to-right search through the inheritance hierarchy.

### **Q3. How do you distinguish between a class object and an instance object?**

A class object is the blueprint from which instances are created, while an instance object is a specific occurrence of a class. Class objects define the structure and behavior (attributes and methods) that the instance objects will have. Instance objects hold the actual data and can invoke the methods defined in the class. You can distinguish between them by checking their types or by observing that instance objects can access instance-specific data while class objects cannot.

### **Q4. What makes the first argument in a class’s method function special?**

The first argument in a class's method function is typically named self. It is special because it refers to the instance of the class upon which the method is being called. self allows access to the attributes and other methods of the instance within the class. It is automatically passed to instance methods, and it must be explicitly included in the method definition.

### **Q5. What is the purpose of the \_\_init\_\_ method?**

The \_\_init\_\_ method is a special method in Python, also known as a constructor. Its purpose is to initialize a new instance of a class. It is automatically called when a new instance is created and can be used to set initial values for the instance's attributes or perform any setup that is needed when the object is instantiated.

### **Q6. What is the process for creating a class instance?**

To create a class instance, you simply call the class using the class name followed by parentheses, possibly passing arguments if the class's \_\_init\_\_ method requires them. For example:

python

Copy code

class MyClass:

def \_\_init\_\_(self, value):

self.value = value

instance = MyClass(10)

In this example, instance is an instance of MyClass, initialized with a value of 10.

### **Q7. What is the process for creating a class?**

To create a class in Python, you use the class keyword followed by the class name and a colon. The body of the class contains method definitions and possibly class attributes. Here's a simple example:

python

Copy code

class MyClass:

class\_variable = 0 # class attribute

def \_\_init\_\_(self, value):

self.instance\_variable = value # instance attribute

def method(self):

print(self.instance\_variable)

### **Q8. How would you define the superclasses of a class?**

You define the superclasses of a class by listing them in parentheses after the class name in the class definition. This creates a subclass that inherits from the superclasses. For example:

python

Copy code

class SuperClass1:

pass

class SuperClass2:

pass

class SubClass(SuperClass1, SuperClass2):

pass

In this example, SubClass inherits from both SuperClass1 and SuperClass2, making them the superclasses of SubClass.