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

Ans- The purpose of Python's Object-Oriented Programming (OOP) is to organize code into reusable and self-contained objects, which are instances of classes. OOP helps in modeling real-world entities and their interactions, making code more modular, maintainable, and easier to understand. It promotes concepts like encapsulation, inheritance, and polymorphism, allowing for more efficient and organized code development.

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

Ans- Inheritance search for an attribute in Python looks first in the instance itself, and if not found, it searches in the class, and then in its base classes or parent classes in a depth-first order. This order is often referred to as the "Method Resolution Order" (MRO).

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

Ans- A class object is the blueprint for creating instances, while an instance object is a specific realization or instantiation of that class. In simpler terms, a class defines the structure and behavior of objects, while an instance is an actual object created from that class.

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

Ans-The first argument in a class's method function is typically named **self**, although you can use any name you like (though it's a convention to use **self**). This argument refers to the instance of the class on which the method is called. It allows you to access and manipulate the instance's attributes and methods. It's automatically passed when you call an instance's method, and it helps differentiate between instance-specific data and class-level data.

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

Ans- The \_\_init\_\_ method, also known as the constructor, is a special method in Python classes. Its purpose is to initialize the attributes of an instance when it is created from the class. It is called automatically when you create a new instance of a class and can be used to set initial values for the object's attributes.

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

Ans- Define a class with the class keyword.

Create an instance of the class by calling the class name as if it were a function, optionally passing any required initialization arguments.The \_\_init\_\_ method of the class is automatically called, initializing the instance's attributes. You can then access the instance's attributes and methods using dot notation.

class MyClass:

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

self.value = value

instance = MyClass(42) # Creating an instance

print(instance.value) # Accessing an attribute

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

Ans- To create a class in Python, you use the class keyword followed by the class name and a colon. You can define attributes and methods within the class to specify its behavior and structure. Here's a basic outline of the process:

Define the class using the class keyword.

Define attributes in the class to represent data.

Define methods to specify the behavior of instances of the class.

You can then create instances of the class as needed.

class MyClass:

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

self.value = value

def get\_value(self):

return self.value

instance = MyClass(42) # Creating an instance of MyClass

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

Ans- The superclasses of a class are also known as parent classes or base classes. They are the classes from which the current class inherits attributes and methods. In Python, you can specify superclasses by including them in parentheses after the class name when defining the class using the class keyword.