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

The purpose of Object-Oriented Programming (OOP) in Python is to structure code in a way that models real-world entities as objects. It allows for the creation of reusable, modular, and maintainable code by grouping data (attributes) and behavior (methods) together into objects. OOP promotes concepts like encapsulation, inheritance, and polymorphism, which enhance code organization, reusability, and flexibility.

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

An inheritance search for an attribute starts in the instance itself, and if not found, it searches the class of the instance, followed by its superclasses in the order specified by the method resolution order (MRO). The MRO is determined using the C3 linearization algorithm and is accessible through the \_\_mro\_\_ attribute or the built-in super() function.

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

A class object is the blueprint or template for creating instances. It defines the structure and behavior of objects but does not hold actual data. An instance object, on the other hand, is a specific occurrence of a class, containing actual data in its attributes and able to invoke methods defined in the class.

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

The first argument in a class's method function (typically named self) refers to the instance object itself. When a method is called on an instance, Python automatically passes the instance as the first argument to the method. This allows the method to access and manipulate the instance's attributes and behavior.

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

The \_\_init\_\_ method is a special method (constructor) in Python classes that is automatically called when an instance of the class is created. It is used to initialize the attributes of the instance and perform any setup that is required for the instance to function properly.

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

To create a class instance, we call the class name followed by parentheses. For example, if we have a class named Person, we create an instance like this: person\_instance = Person(). This invokes the \_\_init\_\_ method to initialize the instance attributes.

Q7. What is the process for creating a class?

To create a class in Python, we use the class keyword followed by the class name (usually in CamelCase) and a colon. Inside the class block, we define attributes and methods that describe the behavior and properties of the class.

Example:

python

Copy code

class Car:

def \_\_init\_\_(self, make, model):

self.make = make

self.model = model

Q8. How would we define the superclasses of a class?

The superclasses of a class are the classes from which the current class inherits. In other words, they are the classes that are higher up in the class hierarchy and contribute attributes and methods to the current class. We can see the superclasses of a class using the \_\_bases\_\_ attribute or the super() function.

Example:

python

Copy code

class ChildClass(ParentClass):

...

In this example, ParentClass is a superclass of ChildClass.