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

**Answer:**  
Object-Oriented Programming (OOP) in Python provides a way to structure code using objects and classes. The main purposes of OOP are:

* **Encapsulation** – Bundling data and methods together to prevent direct access to the internal data.
* **Inheritance** – Allowing new classes to reuse and extend existing class functionality.
* **Polymorphism** – Enabling different classes to be used interchangeably by having the same method names but different implementations.
* **Code Reusability** – Avoiding code duplication by defining reusable classes.

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

**Answer:**  
When searching for an attribute in an instance, Python follows the **Method Resolution Order (MRO)**:

1. The **instance** itself.
2. The **class** of the instance.
3. Any **superclasses** in the order they are defined (following **depth-first, left-to-right** in multiple inheritance).
4. The **object class** (if no match is found).

This process ensures Python searches for the attribute in a logical order up the inheritance chain.

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

**Answer:**

* A **class object** is the blueprint (template) from which instances are created. It defines attributes and methods that instances can use.
* An **instance object** is a specific realization of a class with its own unique data.

**Example:**

class Car:

pass

c1 = Car() # Instance of the Car class

* Car → Class object
* c1 → Instance object

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

**Answer:**  
The first argument of a class method is typically called self, and it refers to the instance of the class. It allows methods to access and modify instance attributes.

**Example:**

class Car:

def display(self):

print("This is a car")

c1 = Car()

c1.display() # 'self' refers to c1 in this case

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

**Answer:**  
The \_\_init\_\_ method is the **constructor** in Python. It is automatically called when an instance is created and is used to initialize instance attributes.

**Example:**

class Car:

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

self.brand = brand # Initializing the instance attribute

c1 = Car("Toyota")

print(c1.brand) # Output: Toyota

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

**Answer:**  
To create a class instance, you need to:

1. **Define a class** using the class keyword.
2. **Call the class** as if it were a function.
3. The \_\_init\_\_ method (if defined) will initialize the instance.

**Example:**

class Car:

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

self.brand = brand

c1 = Car("Tesla") # Creating an instance of the Car class

print(c1.brand) # Output: Tesla

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

**Answer:**  
To create a class in Python:

1. Use the class keyword.
2. Define attributes and methods inside the class.

**Example:**

class Car:

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

self.brand = brand

def display(self):

print("Car brand:", self.brand)

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

**Answer:**  
Superclasses (or parent classes) are the classes from which a class inherits. They are defined inside parentheses while creating a subclass.

**Example:**

class Vehicle:

pass

class Car(Vehicle): # Car is a subclass of Vehicle

pass

Here, Vehicle is the **superclass** of Car, and Car is a **subclass** of Vehicle.