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

Ans.

1. Explanation: Python's Object-Oriented Programming (OOP) is designed to help organize and structure code by modeling real-world entities as objects with attributes and methods.

2. Steps:

- Define a class using the `class` keyword.

- Create attributes (variables) and methods (functions) within the class.

- Instantiate objects from the class and access their attributes and methods.

3. Example:

class Dog:

def \_\_init\_\_(self, name, breed):

self.name = name

self.breed = breed

def bark(self):

return f"{self.name} says woof!"

my\_dog = Dog("Rex", "Labrador")

print(my\_dog.bark()) # Output: Rex says woof!

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

Ans :

1. Explanation: Inheritance search in Python looks for an attribute by traversing the class hierarchy in a specific order (method resolution order).

2. Steps:

- Check the instance's class.

- Traverse up the inheritance chain to parent classes in the order defined by the method resolution order (MRO).

- Continue searching until the attribute is found or the top of the hierarchy is reached.

3. Example:

class Animal:

def speak(self):

return "Some sound"

class Dog(Animal):

def bark(self):

return "Woof!"

class Labrador(Dog):

pass

my\_dog = Labrador()

print(my\_dog.speak()) # Output: Some sound

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

Ans :

1. Explanation: A class object defines a blueprint for creating instances, whereas an instance object is an individual object created from that blueprint.

2. Steps:

- Define a class to create a class object using the `class` keyword.

- Create an instance object by calling the class as if it were a function.

- Check the type of the object using the `type()` function.

3. Example:

class Dog:

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

self.name = name

my\_dog = Dog("Rex")

print(type(Dog)) # Output: <class 'type'>

print(type(my\_dog)) # Output: <class '\_\_main\_\_.Dog'>

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

Ans :

1. Explanation: The first argument in a class's method function is special because it refers to the instance of the class, typically named `self`.

2. Steps:

- Define a method inside a class.

- Use `self` as the first parameter to refer to the instance calling the method.

- Access or modify instance attributes and call other methods using `self`.

3. Example:

class Dog:

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

self.name = name

def bark(self):

return f"{self.name} says woof!"

my\_dog = Dog("Rex")

print(my\_dog.bark()) # Output: Rex says woof!

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

Ans :

1. Explanation: The `\_\_init\_\_` method in Python is used to initialize a newly created object's attributes.

2. Steps:

- Define the `\_\_init\_\_` method inside a class.

- Include `self` as the first parameter, followed by other parameters for initialization.

- Assign values to the object's attributes using `self`.

3. Example:

class Dog:

def \_\_init\_\_(self, name, breed):

self.name = name

self.breed = breed

my\_dog = Dog("Rex", "Labrador")

print(my\_dog.name) # Output: Rex

print(my\_dog.breed) # Output: Labrador

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

Ans :

1. Explanation: Creating a class instance involves calling the class as if it were a function, which triggers the `\_\_init\_\_` method to initialize the new object.

2. Steps:

- Define a class with an `\_\_init\_\_` method to initialize attributes.

- Call the class using its name and pass the required arguments.

- The `\_\_init\_\_` method initializes the instance with the provided values.

3. Example:

class Dog:

def \_\_init\_\_(self, name, breed):

self.name = name

self.breed = breed

my\_dog = Dog("Rex", "Labrador")

print(my\_dog.name) # Output: Rex

print(my\_dog.breed) # Output: Labrador

Q7. What is the process for creating a class?

Ans :

1. Explanation: Creating a class in Python involves defining a new type of object with its own attributes and methods using the `class` keyword.

2. Steps:

- Use the `class` keyword followed by the class name and a colon.

- Define an `\_\_init\_\_` method to initialize attributes (optional but common).

- Define other methods to perform actions related to the class.

3. Example:

class Dog:

def \_\_init\_\_(self, name, breed):

self.name = name

self.breed = breed

def bark(self):

return f"{self.name} says woof!"

# Class definition complete, now instances can be created.

my\_dog = Dog("Rex", "Labrador")

print(my\_dog.bark()) # Output: Rex says woof!

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

Ans :

1. Explanation: Superclasses of a class are defined by specifying them in parentheses after the class name.

2. Steps:

- Use the `class` keyword followed by the subclass name.

- In parentheses, list the superclasses that the subclass will inherit from.

- Define the subclass with its own attributes and methods, optionally overriding superclass methods.

3. Example:

class Animal:

def speak(self):

return "Some sound"

class Dog(Animal):

def speak(self):

return "Woof!"

# Dog is a subclass of Animal

my\_dog = Dog()

print(my\_dog.speak()) # Output: Woof!