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

Answer: The purpose of Python's OOP is to provide a way to organize and structure code in a way that is reusable, maintainable, and scalable. OOP is a programming paradigm that focuses on creating objects, which are self-contained entities that contain both data and behavior.

Objects can interact with each other by sending and receiving messages. This allows programmers to create modular code that can be easily reused and extended.

Python's OOP features include:

Classes: Classes are used to define new types of objects. A class can contain both data attributes and methods.

Objects: Objects are instances of classes. They contain the data and behavior defined in the class.

Inheritance: Inheritance allows programmers to create new classes that inherit the properties of existing classes. This allows programmers to reuse code and create hierarchies of related classes.

Polymorphism: Polymorphism allows programmers to write code that can be used with different types of objects. This makes code more flexible and reusable.

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

Answer: When an attribute is not found in the current class, Python will search for it in the class's parent classes, and so on, until it reaches the top of the class hierarchy. This is called **attribute inheritance**.

The order in which Python searches for inherited attributes is called the **method resolution order (MRO)**. The MRO is a list of classes, starting with the current class and ending with the root class (object). Python searches for attributes in the order of the MRO, starting with the first class in the list.

Example: class Animal:

pass

class Dog(Animal):

pass

class Cat(Animal):

pass

class Poodle(Dog):

pass

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

Answer:

A class object is a blueprint for creating objects. It defines the attributes and methods that objects of that class will have. An instance object is a specific instance of a class. It has the attributes and methods defined in the class, but it may also have its own unique values for those attributes.

Here is an example:

Python

class Animal:

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

self.name = name

self.age = age

def speak(self):

pass

dog = Animal("Fido", 5)

cat = Animal("Garfield", 3)

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

Answer:

The first argument in a class's method function is special because it is automatically passed a reference to the class object. This is called the self argument.

The self argument is used by class methods to access and modify the class's attributes and methods. It is also used to pass references to the class object to other methods.

For example, the following class method uses the self argument to access and modify the class's name attribute:

Python

class Person:

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

self.name = name

def greet(self):

print(f"Hello, my name is {self.name}")

person = Person("Alice")

person.greet()

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

Answer:

The \_\_init\_\_ method is a special method in Python that is called when an object is created. It is used to initialize the object's attributes.

The \_\_init\_\_ method takes a single argument, which is a reference to the object itself. This is typically referred to as the self parameter.

Here is an example of a simple \_\_init\_\_ method:

Python

class Animal:

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

self.name = name

self.age = age

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

Answer:

To create a class instance in Python, we use the following steps:

a) Define the class. This involves using the class keyword to define the name of the class and its attributes and methods.

b) Create a new instance of the class. This involves using the class name followed by parentheses.

c) Pass any required arguments to the class constructor. The class constructor is a special method that is called when a new instance of the class is created.

d) Assign the new instance of the class to a variable.

Here is an example:

Python

class Animal:

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

self.name = name

self.age = age

dog = Animal("Fido", 5)

Q7. What is the process for creating a class?

Answer:

To create a class in Python, you use the following steps:

Use the class keyword to define the name of the class.

Define the attributes of the class. Attributes are variables that belong to the class.

Define the methods of the class. Methods are functions that belong to the class.

Indent the class body. This means that all of the code inside the class definition should be indented by one level.

End the class definition with a colon.

Here is an example of a simple class definition:

Python

class Animal:

name: str

age: int

def speak(self):

pass

Q8. How would you define the super classes of a class?

Answer:

To define the super classes of a class in Python, you use the super() function. The super() function returns a proxy object that allows you to access the methods of the parent class.

To call a method of the parent class, you can use the following syntax:

Python

super().method\_name()