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

Ans. The purpose of Python's Object-Oriented Programming (OOP) is to provide a structured and modular approach to programming that emphasizes the creation of reusable code. OOP allows developers to create objects which are instances of classes that can be used to represent real-world entities or concepts in a program. These objects can contain data in the form of attributes, and behaviors in the form of methods, which can be used to interact with the object and other parts of the program. OOP in Python provides a powerful way to organize and structure code making it easier to write understand and maintain.

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

Ans. In Python's inheritance hierarchy, when an attribute is accessed on an object, Python looks for the attribute in a specific order known as the method resolution order (MRO). The MRO is determined by the class hierarchy and is typically computed using the C3 linearization algorithm.

If the attribute is still not found after searching all the classes in the MRO, Python will raise an Attribute Error.

If a subclass overrides a method or attribute defined in a superclass, the subclass's implementation will be used instead of the superclass's implementation. This is known as method overriding, and it allows subclasses to customize or extend the behavior of their parent classes.

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

Ans. In Python, a class is a blueprint or template for creating objects, while an instance is a specific object created from a class. To distinguish between a class object and an instance object, can consider the following:

Definition: A class is defined using the class keyword and contains methods and attributes that define the behavior and properties of objects that will be created from it. An instance is created by calling the class as if it were a function, which returns a new object of that class.

Usage: A class object is used to create new instance objects, and to access class-level methods and attributes. An instance object is used to access instance-level methods and attributes.

Memory Allocation: A class object is stored in memory once, while an instance object is created and stored in memory each time a new instance is created from the class.

Identifier: can use the type () function to determine if an object is a class or instance. If it returns <class 'type'>, the object is a class. If it returns the name of the class, the object is an instance.

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

Ans. In Python, the first argument in a class's method function is typically named self by convention, and it refers to the instance of the class on which the method is being called. This argument is special because it allows the method to access and manipulate the attributes and methods of the instance object.

When a method is called on an instance object, Python automatically passes the instance object as the first argument to the method. This allows the method to refer to the instance object's attributes and methods using the self keyword.

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

Ans. The \_\_init\_\_ method is a special method in Python classes that is called when an instance of the class is created. It is used to initialize the attributes of the instance object, and it is often referred to as the class constructor. The \_\_init\_\_ method takes the instance object self as its first argument, followed by any additional arguments needed to initialize the instance's attributes. The \_\_init\_\_ method can also perform additional operations on the instance object, such as setting default values for attributes, or calling other methods to set up the instance's state.

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

Ans. In Python, creating a class instance involves the following steps:

Define the class: First, you need to define the class that you want to create an instance of. This involves using the class keyword followed by the name of the class and its definition, which includes any methods and attributes that the class will have.

Instantiate the class: To create an instance of the class, you need to call the class as if it were a function. This will create a new instance of the class and return it to you.

Initialize the instance: Once you have the instance object, you can initialize its attributes using the \_\_init\_\_ method, which is called automatically when the instance is created. The \_\_init\_\_ method takes the instance object self as its first argument, followed by any additional arguments needed to initialize the instance's attributes.

Use the instance: Once you have initialized the instance's attributes, you can use its methods and attributes as needed.

Q7. What is the process for creating a class?

Ans. To create a class in Python, you need to follow these steps:

Use the class keyword: To define a class, you need to use the class keyword followed by the name of the class.

Define the class methods and attributes: Once you have defined the class, you can add methods and attributes to it. Methods are functions that belong to the class and can be called on instances of the class. Attributes are variables that belong to the class or its instances.

Instantiate the class: To use the class, you need to create an instance of it. This is done by calling the class as if it were a function. This will create a new instance of the class and return it to you.

Use the instance: Once you have an instance of the class, you can use its methods and attributes as needed.

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

Ans. In Python, the superclasses of a class can be defined using inheritance. When you define a new class, you can specify one or more parent classes from which the new class inherits. These parent classes are also known as superclasses. To define a superclass in Python, you simply specify the superclass as an argument to the class definition using parentheses.