**Q1. What is the purpose of OOP in Python?**

Answer: The Object Oriented Programming (OOP) technique in Python seeks to organize programs such that properties and behaviours are confined into individual objects, it makes code reusable, modular, and more structured, helps in modelling real world entities and their interactions so that code becomes more intuitive as well as easy to manage. By using classes and objects, encapsulation of data together with functions that work on them is achieved by python OOP; inheritance which allows sharing and extending functionalities through classes; polymorphism where methods can have different behaviors depending on object type.

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

Answer: In Python, when searching for attribute in an inheritance way it looks first in the instance object, then in the class that instance was created from, finally all classes up the inheritance chain following MRO (Method Resolution Order). This ensures a search up to the base classes so that most specific class is searched ahead of general ones. Thus derived classes are able to override or extend functionality from base classes.

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

Answer: The first argument in a class's method function is typically named self in Python. It refers to the instance of the class through which the method is being called. This allows the method to access and modify the instance's attributes and other methods. The self parameter is automatically passed when a method is called on an instance, making it possible to distinguish between instance-specific data and class-specific data.

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

Answer: The \_\_init\_\_ method in Python is a special method that acts as a constructor for the class. Its purpose is to initialize the instance's attributes when the instance is created. It allows for setting the initial state of the object, ensuring that the object is in a valid state right after its creation. The \_\_init\_\_ method is automatically invoked when a new instance of the class is instantiated.

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

Answer: Creating a class instance in Python involves calling the class itself, which triggers the \_\_init\_\_ method to initialize the new object. Here is a step-by-step process:

1. Define a class using the class keyword.
2. Optionally define an \_\_init\_\_ method to initialize the instance's attributes.

Create an instance by calling the class as if it were a function, passing any required arguments for the \_\_init\_\_ method.

1. For example:

python

Copy code

class MyClass:

 def \_\_init\_\_(self, value):

 self.value = value

instance = MyClass(10)

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

Answer: Creating a class in Python involves the following steps:

1. Use the class keyword followed by the class name and a colon.
2. Define any methods and attributes within the class.

Optionally include an \_\_init\_\_ method for initialization.

1. For example:

class MyClass:

      def \_\_init\_\_(self, value):

      se**lf.value = value**

      def display(self):

          print(self.value)

This defines a new class MyClass with an initializer and a method.

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

Answer: Superclasses of a class, also known as parent or base classes, are defined in the class definition line by including them in parentheses after the class name. This establishes inheritance, allowing the new class (subclass) to inherit attributes and methods from the superclasses. Multiple inheritance is also supported, allowing a class to inherit from multiple superclasses.

Example:

**class parent:**

**pass**