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

Classes, which act as a blueprint for producing objects, may be created using Python's OOP capabilities. Instances of a class can be produced at runtime, and a class describes the characteristics and methods of an object.

Using OOP in Python has several advantages, including:

1. Encapsulation: OOP enables you to combine data and behaviour into a single unit, which makes managing and maintaining code simpler.

2. Inheritance: Python's OOP allows inheritance, allowing you to build new classes based on existing ones and inherit their characteristics and functions.

3. Polymorphism: As long as the objects in an OOP programme implement the same interface, you may build code that can interact with objects of various classes.

4. Modularity: OOP makes it simpler to divide large systems into smaller, easier-to-manage parts.

Generally speaking, Python's OOP characteristics enable the creation of more structured, reusable, and extendable code.

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

Tree searches are just tree grabs of attributes. The word "inheritance" is used because items at lower levels of a tree take on qualities from objects at higher levels of the tree. The items connected into a tree as the search moves up the tree are, in a sense, the union of all the characteristics defined in all of their tree parents.

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

|  |  |  |
| --- | --- | --- |
| class object | | instance object |
| Creation: | A class is created using the "class" keyword | while an instance is created using the class name followed by parentheses |
| Attributes: | A class object has attributes that describe the class itself, such as its name or its methods. | An instance object has attributes that describe its specific state, which may differ from other instances of the same class. |
| Methods: | A class object can have class methods, which act on the class itself | instance object can have instance methods, which act on the instance itself |

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

Self represents the instance of the class. By using the “self” keyword we can access the attributes and methods of the class in python. It binds the attributes with the given arguments. The reason you need to use self. is because Python does not use the @ syntax to refer to instance attributes. Python decided to do methods in a way that makes the instance to which the method belongs be passed automatically, but not received automatically: the first parameter of methods is the instance

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

The task of **init** method is to initialize(assign values) to the data members of the class when an object of class is created. It contains collection of statements that are executed at time of Object creation. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

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

To create instances of a class, you call the class using class name and pass in whatever arguments its **init** method accepts

Q7. What is the process for creating a class?

The class statement creates a new class definition. The name of the class immediately follows the keyword class followed by a colon

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

A superclass is the class from which many subclasses can be created. The subclasses inherit the characteristics of a superclass. The superclass is also known as the parent class or base class.