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

The purpose of Python's Object-Oriented Programming (OOP) is to organize code into reusable and modular components called classes. OOP allows for the creation of objects that encapsulate data and behavior, promoting code reusability, modularity, and abstraction. It enables developers to model real-world entities, relationships, and interactions in a more natural and intuitive way.

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

Inheritance search in Python looks for an attribute first in the instance itself, then in its class, followed by its superclass (if any), and so on up the inheritance chain until it reaches the top-level object class.

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

A class object is a blueprint for creating instances, while an instance object is a specific realization of a class. In other words, a class object defines the attributes and methods shared by all instances of that class, whereas an instance object contains specific values for those attributes and can invoke the methods defined in the class.

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

The first argument in a class's method function (usually named 'self' by convention) is special because it represents the instance on which the method is called. When a method is called on an instance, Python automatically passes the instance itself as the first argument to the method, allowing the method to access and manipulate the instance's attributes and state.

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

The purpose of the \_\_init\_\_ method (also known as the constructor) in Python classes is to initialize new instances of the class. It is called automatically when a new instance is created, allowing you to set up the initial state of the object by assigning values to its attributes.

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

The process for creating a class instance involves calling the class name followed by parentheses containing any necessary arguments to initialize the instance. For example:

class MyClass:

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

self.x = x

# Creating an instance of MyClass

my\_instance = MyClass(10)

Q7. What is the process for creating a class?

The process for creating a class in Python involves using the class keyword followed by the class name and a colon. Inside the class block, you define the attributes and methods of the class. For example:

class MyClass:

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

self.x = x

def my\_method(self):

return self.x

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

The superclasses of a class are the classes from which it directly or indirectly inherits. In Python, you can define superclasses by specifying them inside parentheses after the class name in the class definition. For example:

class SubClass(SuperClass):

# Class definition