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

Answer:

Object-oriented programming (OOP) is based on the concept of “objects,” which can contain data and code: data in the form of instance variables (often known as attributes or properties), and code, in the form method. I.e., Using OOP, we encapsulate related properties and behaviors into individual objects

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

Answer:

In Python, when an attribute is accessed on an instance, Python first looks for the attribute on the instance itself. If it doesn’t find it there, it looks for it on the class. If it doesn’t find it there either, it looks for it on the superclasses of the class.

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

Answer:

A class object is an object that represents a class definition. An instance object is an object that represents an instance of a class

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

Answer:

The first argument in a class’s method function is always the instance object that the method was called on

Q5. What is the purpose of the init method?

Answer:

The init method is called constructors. 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?

Answer:

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?

Answer:

To create a class in Python, you use the keyword “class” followed by the name of your class and a colon.

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

Answer:

In Python, you can define superclasses by passing them as arguments to your new class definition.

Q9. What is the relationship between classes and modules?

Answer:

In Python, classes are defined in modules just like functions are defined in modules.

Q10. How do you make instances and classes?

Answer:

You make instances by calling classes using their names and passing in whatever arguments their init methods accept.

Q11. Where and how should be class attributes created?

Answer:

Class attributes should be created inside the class definition block but outside any methods.

Q12. Where and how are instance attributes created?

Answer:

Instance attributes are created inside methods using self.attribute_name = value syntax.

Q13. What does the term "self" in a Python class mean?

Answer:

“self” refers to the instance of a class that is being operated on by a method.

Q14. How does a Python class handle operator overloading?

Answer:

In Python, operator overloading is handled by defining special methods on your classes that correspond to each operator you want to overload.

Q15. When do you consider allowing operator overloading of your classes?

Answer:

You should consider allowing operator overloading when it makes sense for your objects to be used with operators like + or *.

Q16. What is the most popular form of operator overloading?

Answer:

The most popular form of operator overloading in Python is probably arithmetic operator overloading (+,-,* etc.).

Q17. What are the two most important concepts to grasp in order to comprehend Python OOP code?

Answer:

The two most important concepts to grasp in order to comprehend Python OOP code are classes and objects.

Q18. Describe three applications for exception processing.

Answer:

Three applications for exception processing include error handling, debugging, and testing.

Q19. What happens if you don't do something extra to treat an exception?

Answer:

If you don’t do something extra to treat an exception, your program will terminate with an error message when an exception occurs.

Q20. What are your options for recovering from an exception in your script?

Answer:

Your options for recovering from an exception in your script include using try/except blocks to catch exceptions and handle them gracefully, raising exceptions yourself using the raise statement, and using the finally clause to specify code that should be executed regardless of whether or not an exception is raised.

Q21. Describe two methods for triggering exceptions in your script.

Answer:

Two methods for triggering exceptions in your script include raising exceptions yourself using the raise statement and calling functions that are known to raise exceptions.

Q22. Identify two methods for specifying actions to be executed at termination time, regardless of whether or not an exception exists.

Answer:

Two methods for specifying actions to be executed at termination time, regardless of whether or not an exception exists include using the finally clause and using the atexit module.

Q23. What is the purpose of the try statement?

Answer:

The purpose of the try statement is to specify a block of code that may raise an exception and to specify how that exception should be handled if it is raised.

Q24. What are the two most popular try statement variations?

Answer:

The two most popular try statement variations are try/except and try/finally.

Q25. What is the purpose of the raise statement?

Answer:

The purpose of the raise statement is to raise an exception manually.

Q26. What does the assert statement do, and what other statement is it like?

Answer:

The assert statement checks whether a condition is true and raises an AssertionError if it isn’t. It’s similar to if statements that raise exceptions when a condition isn’t met.

Q27. What is the purpose of the with/as argument, and what other statement is it like?

Answer:

The with/as argument is used to create a context in which a resource is managed automatically by Python (e.g., closing a file when you're done with it). It's similar to try/finally statements that ensure resources are cleaned up even if an exception occurs.

Q28. What are *args, **kwargs?

Answer:

*args and **kwargs are special syntax in Python that allow you to pass a variable number of arguments to a function.

Q29. How can I pass optional or keyword parameters from one function to another?

Answer:

You can pass optional or keyword parameters from one function to another by using *args and **kwargs syntax.

Q30. What are Lambda Functions?

Answer:

Lambda functions are small anonymous functions that can be defined on-the-fly in Python code.

Q31. Explain Inheritance in Python with an example?

Answer:

Inheritance is a way of creating new classes from existing classes by inheriting their attributes and methods. For example, you might have a class called Animal that has attributes like name and age, as well as methods like eat() and sleep(). You could then create a new class called Dog that inherits from Animal and adds its own attributes like breed and bark().

Q32. Suppose class C inherits from classes A and B as class C(A,B).Classes A and B both have their own versions of method func(). If we call func() from an object of class C, which version gets invoked?

Answer:

If we call func() from an object of class C, Python will look for func() first in class C itself, then in class A (since A comes before B in the inheritance list), then in class B if it doesn’t find it in A.

Q33. Which methods/functions do we use to determine the type of instance and inheritance?

Answer:

We use type() function to determine the type of instance and bases attribute to determine inheritance.

Q34.Explain the use of the 'nonlocal' keyword in Python.

Answer:

The nonlocal keyword allows you to modify variables that are defined outside of your current scope (e.g., inside a nested function).

Q35. What is the global keyword?
Answer:
The global keyword allows you to modify variables that are defined outside of your current scope (e.g., inside a function).