Q1. What are the two latest user-defined exception constraints in Python 3.X?

In Python 3.X, the two latest user-defined exception constraints are:

Implicit BaseException Inheritance: User-defined exceptions must be derived from the BaseException class or its subclasses. This helps ensure that all exceptions are part of a common hierarchy.

String Argument Deprecation: Providing a string argument to raise an exception is deprecated. Instead, exceptions should be raised with instances of the exception class. For example, instead of raise "CustomError", you should use raise CustomError().

Q2. How are class-based exceptions that have been raised matched to handlers?

Class-based exceptions that have been raised are matched to handlers based on their class hierarchy. When an exception is raised, Python searches for an appropriate except block that matches the exception class or one of its base classes. If a match is found, the code in the corresponding except block is executed.

Q3. Describe two methods for attaching context information to exception artefacts.

Two methods for attaching context information to exception artifacts are:

* Custom Exception Attributes: You can define custom attributes within your exception classes to store additional context information. These attributes can be accessed when handling the exception to provide more context about the error.
* Logging: Logging libraries like the logging module can be used to record contextual information when exceptions occur. This information can help in diagnosing issues and understanding the circumstances leading to the exception.

Q4. Describe two methods for specifying the text of an exception object's error message.

Two methods for specifying the text of an exception object's error message are:

* Using the Constructor: You can pass an error message as an argument when raising an exception using its constructor. For example:

raise CustomError("An error occurred.")

* Overriding \_\_str\_\_ Method: You can define the \_\_str\_\_ method in your exception class to customize the error message. When the exception is raised, the \_\_str\_\_ method will be called to generate the error message.

Q5. Why do you no longer use string-based exceptions?

String-based exceptions were deprecated and are no longer used because they lacked proper structure and made it difficult to categorize and handle exceptions consistently. Using class-based exceptions provides a clear hierarchy, better organization, and allows you to attach additional information to exceptions. Class-based exceptions also allow for more advanced exception handling and better integration with the rest of the Python programming language.