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: a) Exceptions must inherit from the BaseException class or one of its subclasses. b) Exceptions should preferably include the \_\_cause\_\_ and \_\_context\_\_ attributes for improved exception chaining and context management.

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

Class-based exceptions are matched to handlers based on their inheritance hierarchy. When an exception is raised, Python searches for handlers in the order of inheritance. The first handler that matches the exception class (or its parent class) is executed.

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

Two methods for attaching context information to exception artifacts are: a) Using the \_\_context\_\_ attribute to chain exceptions, providing a more informative context about the original exception. b) Utilizing the \_\_traceback\_\_ attribute to store the traceback information at the point where the exception was created.

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: a) Defining a custom \_\_str\_\_ method in the exception class to return a user-friendly error message when the exception is converted to a string. b) Passing an error message string as an argument when raising the exception, like raise MyException("An error occurred").

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

String-based exceptions were deprecated and are no longer used because they lacked structured information and prevented proper traceback and chaining of exceptions. Class-based exceptions offer better organization, improved traceback information, and support for more advanced exception handling features.