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

Answer : Inheriting from the Exception class:

It is recommended to inherit your custom exceptions from the built-in Exception class or any of its subclasses. This allows your custom exception to benefit from the existing exception hierarchy and facilitates consistent exception handling.

Providing descriptive error messages:

It is good practice to provide informative and meaningful error messages with your custom exceptions. The error message should clearly describe the exceptional condition or provide relevant context to aid in debugging.

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

Answer : In Python, class-based exceptions that have been raised are matched to handlers based on the inheritance hierarchy of the exception classes and the order in which the except blocks are defined.

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

Answer : 1. Adding Custom Attributes to Exception Objects:

You can attach context information to exception objects by adding custom attributes to them. When defining a custom exception class or handling an exception, you can include additional attributes that provide relevant context to the exception.

2. Using Exception Chaining:

Exception chaining allows you to attach additional exception information to an existing exception. It is useful when you catch an exception and want to provide more context or additional details about the original exception.

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

Answer : 1. Passing the Message as an Argument to the Exception Class:

When defining a custom exception class or raising an exception, you can pass the error message as an argument to the exception class constructor. This allows you to specify a custom error message that provides relevant information about the exception.

2. Formatting Error Messages using f-strings or String Formatting:

You can use f-strings (formatted string literals) or string formatting techniques to construct dynamic error messages for exceptions. This allows you to include variables, values, or contextual information within the error message.

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

Answer :

Lack of Exception Hierarchy: String-based exceptions do not provide a clear hierarchy or structure. In Python, exception classes are used to define a hierarchy of exceptions, allowing for more specific exception handling. String-based exceptions do not provide this hierarchy, making it difficult to differentiate and handle exceptions based on their type or severity.

Limited Error Information: String-based exceptions often only provide a simple error message without any additional information or attributes. Exceptions in Python typically include additional attributes or methods that provide useful information about the exception, such as stack traces or error codes. String-based exceptions lack this additional information, making it more challenging to diagnose and handle errors effectively.

Inability to Catch Specific Exceptions: With string-based exceptions, it is challenging to catch specific exceptions or specific types of errors. In Python, the except statement allows you to catch specific exception classes or types. However, with string-based exceptions, you would need to rely on parsing error messages or using string comparison techniques, which can be error-prone and less precise.

Code Maintainability and Readability: Using string-based exceptions can make your code less maintainable and harder to understand. Exception classes provide a clear and structured way to define and handle exceptions, improving code readability and maintainability. With string-based exceptions, it becomes harder to understand the intentions and expectations of the code, making it more difficult for others (including yourself) to work with or modify the code in the future.