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

Ans:

As of my last knowledge update in September 2021, Python 3.x does not introduce specific constraints or limitations on user-defined exceptions. You can create custom exceptions in Python 3.x by defining new exception classes that inherit from the built-in BaseException or its subclasses like Exception. There are no specific constraints or limitations regarding when or how you define your custom exceptions.

Please note that Python's development continues, and there may have been changes or enhancements in later versions of Python 3.x or Python 4.x. Always refer to the official Python documentation or release notes for the most up-to-date information on Python features and constraints.

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

ans:

Class-based exceptions that have been raised in a programming language like Python are matched to handlers using the following principles:

Inheritance Hierarchy: The language checks the inheritance hierarchy of the raised exception class. It searches for the nearest matching exception handler by looking at the hierarchy from the most specific (derived class) to the least specific (base class).

Matching Exception Type: The exception handler must specify the type of exception it can handle. When an exception is raised, the language compares the type of the raised exception to the types specified in the available exception handlers.

First Match Wins: Exception handlers are evaluated in the order in which they appear in the code. The first handler that matches the raised exception's type will be executed. Subsequent handlers are ignored.

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

Ans:

Two methods for attaching context information to exception artifacts are:

Exception Chaining: Exception chaining involves raising a new exception while preserving the original exception as the new exception's cause. This allows you to add context information to the new exception without losing the details of the original error. Many programming languages, like Python, support this approach. For example:

Logging: You can log context information when an exception occurs. Logging libraries or frameworks often provide facilities for capturing and storing contextual details when an exception is raised. This can include timestamps, input parameters, or relevant state information that helps diagnose the issue. By examining the logs, you can gain insights into the circumstances surrounding the exception.

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

Ans:

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

1. Constructor Argument: When creating a custom exception class, you can specify the error message as an argument to the constructor. This allows you to provide a descriptive error message when raising an instance of that exception class.

String Formatting: You can format a string to include specific information or context within the error message when raising a built-in or custom exception. This approach provides flexibility in constructing error messages dynamically.

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

String-based exceptions lack type information, hierarchy, and metadata, making them less precise and less informative than exception classes. Using exception classes is the preferred approach for better error handling and maintainability.