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

Ans- of my knowledge cut off date of September 2021, there were no specific "latest" user-defined exception constraints in Python 3.x. However, I can provide information on two common types of user-defined exception constraints in Python:

1. Inheritance from the built-in ‘Exception’ class: When defining a user-defined exception in Python, it is common practice to inherit from the built-in Exception class.
2. Raising custom exceptions with custom error messages: When raising exceptions in Python, you can include a custom error message that provides more information about what went wrong. To do this with a custom exception, you can define an ‘\_\_init\_\_’ method that takes an error message as an argument and stores it as an instance variable.

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

Ans- When a class-based exception is raised in Python, the interpreter first looks for an exception handler in the current function. If no matching handler is found, the interpreter then looks for a matching handler in the calling function, and so on up the call stack until a handler is found or the exception reaches the top-level of the program. When searching for a matching handler, the interpreter compares the exception type (i.e., the class of the raised exception) to the types of exceptions specified in each except clause in the try/except block. If the exception type matches one of the specified types in an except clause, the code in the corresponding except block is executed.

If no matching handler is found for the exception, the interpreter will print an error message and terminate the program unless the exception is caught by an unhandled exception handler defined using the sys.excepthook() function.

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

Ans- Context information can be very useful when debugging and handling exceptions. Here are two methods for attaching context information to exception artifacts in Python:

1.Adding Context Information to the Exception Object:

One method for attaching context information to exception artifacts is to add attributes to the exception object itself. For example, if you catch an exception related to a file operation, you might add attributes to the exception object that provide additional information about the file, such as its name, location, and file type. This can help you diagnose the problem and provide more helpful error messages to users.

2.Using a Context Manager:

Another method for attaching context information to exception artifacts is to use a context manager. A context manager is a Python object that defines methods that are called when a block of code is entered and exited. The ‘with’ statement is used to create a context and execute the code within it.

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

Ans- There are several ways to specify the text of an exception object's error message, but here are two common methods:

1.Directly passing an error message string to the exception constructor:

When creating an exception object, you can pass a string argument to the constructor to specify the error message.

2.Formatting an error message using string interpolation or concatenation:

Another method is to use string interpolation or concatenation to format an error message with relevant information. This approach is often used when the error message needs to include dynamic data such as variable values or timestamps.

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

Ans- In the past, string-based exceptions were used in some programming languages to raise and handle exceptions by simply passing an error message as a string. However, this approach has several drawbacks, which is why it has largely been deprecated in modern programming languages.One of the main issues with string-based exceptions is that they lack structure and type information. This means that it can be difficult to determine the cause and nature of an exception, as well as how to handle it properly. In addition, string-based exceptions can be more error-prone because they rely on manual parsing and comparison of error messages, which can be time-consuming and prone to errors.

Modern programming languages typically use structured exception objects, which contain more information about the exception, including its type, stack trace, and additional data. This makes it easier to identify and handle exceptions in a consistent and reliable manner, as well as to provide more informative error messages to users or developers.