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

Ans: Python 3.x does provide the ability to create user-defined exceptions using the raise statement with a custom exception class.

To define a custom exception class, you can create a new class that inherits from the built-in Exception class. For example, the following code creates a custom exception class called MyException:

class MyException(Exception):

pass

You can then raise an instance of the MyException class by using the raise statement:

raise MyException("Something went wrong.")

This will raise an instance of the MyException class with the message "Something went wrong." as the argument to the constructor.

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

Ans: In Python, when an exception is raised, it is propagated up through the call stack until it is caught by an appropriate exception handler. The process of finding an appropriate handler for an exception involves searching for the closest matching handler in the call stack.

When a class-based exception is raised, Python searches the call stack for a handler that matches the exception's class or one of its superclasses. If a matching handler is found, it is executed to handle the exception. If no matching handler is found, the exception is propagated up the call stack until it reaches the top-level of the program, at which point the program terminates with a traceback.

**Ex:**

class MyException(Exception):

pass

try:

raise MyException("Something went wrong.")

except MyException:

print("Handling MyException")

except Exception:

print("Handling Exception")

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

Ans: Exception artifacts, such as error messages or stack traces, often provide important clues about the cause and location of an error in software. Attaching context information to these artifacts can help developers and users better understand the error and quickly diagnose and fix the problem. Here are two methods for attaching context information to exception artifacts:

**a. Logging:** Logging is a common technique used to attach context information to exception artifacts. Developers can include additional information in log messages that are generated when an exception occurs. For example, the log message could include information about the state of the application or the user's input at the time the error occurred. These log messages can then be stored in a central location, allowing developers to quickly search and analyze them to identify patterns and diagnose errors.

**b. Custom exception classes:** Another method for attaching context information to exception artifacts is to create custom exception classes that include additional information about the error. For example, a custom exception class for a database connection error could include information about the database server and the query that caused the error. When an error occurs, the custom exception class can be thrown with this additional information, providing developers with more context about the error. By creating custom exception classes for specific types of errors, developers can provide more meaningful and targeted error messages that can help users quickly identify and fix problems.

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

Ans: When an exception is thrown in a program, the error message associated with the exception can be a critical piece of information for developers to understand the cause of the error. Here are two methods for specifying the text of an exception object's error message:

**a. Hard-coded messages:** One common method for specifying the text of an exception object's error message is to hard-code the message directly into the code.

**b. Message templates:** Another approach is to use message templates that can be customized at runtime with specific values.

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

Ans:

a. It makes it harder to handle exceptions programmatically. When you raise a string-based exception, the calling code has to parse the string to determine what kind of exception was raised. This can be error-prone and make it harder to write reliable error-handling code.

b. It can make debugging harder. If you raise a string-based exception, it may be harder to determine the root cause of the exception because the exception message may not contain all the necessary information.

c. It can be less efficient. When you raise a string-based exception, the string message has to be allocated and stored in memory, which can be slower and take up more memory than raising a typed exception.

For these reasons, many modern programming languages discourage or prohibit the use of string-based exceptions in favor of typed exceptions, where the type of the exception indicates the kind of error that occurred. This can make it easier to handle exceptions programmatically, improve debugging, and make code more efficient.