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

ANSWER.

As of Python 3.x, there are no specific constraints or limitations on user-defined exceptions. You can define custom exceptions in Python by creating new exception classes that inherit from the built-in `Exception` class or any of its subclasses.

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 of `except` clauses in `try`...`except` blocks. When an exception is raised, Python searches for a matching `except` clause to handle the exception, considering both the type of the raised exception and its superclass hierarchy.

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

ANSWER.

Attaching context information to exception artifacts in Python can be crucial for providing additional details about the error, aiding in debugging, and improving the user experience. Here are two common methods for attaching context information to exception artifacts:

1. Exception Arguments:

- Many built-in exception classes in Python allow you to pass additional information as arguments when raising the exception. This additional information can provide context about the error and help users understand the cause of the exception.

- Syntax:

```python

raise ExceptionType("Error message", context\_info)

```

2. Using `with\_traceback()` Method:

- When re-raising exceptions or creating new exceptions based on existing ones, you can use the `with\_traceback()` method to attach the original traceback information to the new exception. This preserves the original stack trace, allowing users to trace the error back to its source.

- Syntax:

```python

raise NewException("Error message").with\_traceback(original\_exception.\_\_traceback\_\_)

```

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

ANSWER.

In Python, you can specify the text of an exception object's error message using various methods. Here are two common methods for specifying the text of an exception object's error message:

1. Passing Error Message as Argument:

- When raising an exception, you can provide an error message as an argument to the exception class constructor. This error message will be stored as the `args` attribute of the exception object and can be accessed using the `str()` function or implicitly converted to a string when the exception is printed or logged.

- Syntax:

```python

raise ExceptionType("Error message")

```

2. Formatting Error Message with f-strings or String Formatting:

- You can dynamically construct error messages by formatting strings using f-strings (Python 3.6+) or string formatting methods (`str.format()` or `%` formatting). This allows you to include dynamic information such as variable values or context information in the error message.

- Syntax:

```python

error\_message = f"Error occurred: {variable}"

raise ExceptionType(error\_message)

```

.

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

ANSWER.

Using string-based exceptions in Python is discouraged because it leads to several drawbacks and limitations compared to using exception classes. Here are some reasons why string-based exceptions are no longer recommended:

1. Limited Information: String-based exceptions provide limited information about the error, typically only including the error message as a string. This lack of structured information makes it challenging to determine the type or cause of the error, hindering debugging and error handling.

2. Lack of Differentiation: String-based exceptions do not allow for differentiation between different types of errors. Without distinct exception types, it becomes challenging to implement specific error handling logic for different error scenarios.

3. Reduced Flexibility: String-based exceptions lack the flexibility and extensibility provided by exception classes. With exception classes, you can define custom exception hierarchies, add additional attributes or methods to exceptions, and implement specialized error handling logic.

4. Limited Compatibility: String-based exceptions are not compatible with Python's built-in exception handling mechanisms, such as `try`...`except` blocks and the `raise` statement. Using string-based exceptions may lead to inconsistencies in error handling and make it harder to integrate with existing code or libraries.

5. Readability and Maintainability: Exception classes offer better readability and maintainability compared to string-based exceptions. By using exception classes, you provide a clear and structured representation of errors, making it easier for developers to understand and reason about the code.