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

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

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

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

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

Answer:

Q1. The two latest user-defined exception constraints in Python 3.x are:

* Exception chaining: This allows exceptions to include information about the original exception that caused them, making it easier to diagnose and fix errors.
* Exception suppression: This allows exceptions to be suppressed and not reported, which can be useful in certain situations where a recoverable error occurs.

Q2. Class-based exceptions that have been raised are matched to handlers based on their inheritance hierarchy. Python searches for a handler for the exception type by looking for the first except block with a matching exception class or one of its superclasses. If no match is found, the exception is propagated up the call stack until it is caught or the program terminates.

Q3. Two methods for attaching context information to exception artifacts are:

* Using exception attributes: Python exceptions have several built-in attributes, including **args**, **\_\_cause\_\_**, and **\_\_context\_\_**, that can be used to attach additional information to the exception object.
* Using exception chaining: When one exception causes another, the original exception can be attached to the new exception using the **\_\_cause\_\_** or **\_\_context\_\_** attributes, allowing for more detailed error reporting.

Q4. Two methods for specifying the text of an exception object's error message are:

* Using the **raise** statement with a string argument: When raising an exception, you can provide a string argument that specifies the error message.
* Defining a custom exception class with a **\_\_str\_\_** method: By defining a custom exception class and implementing a **\_\_str\_\_** method, you can customize the error message that is displayed when the exception is raised.

Q5. String-based exceptions are no longer used because they are not as flexible or extensible as class-based exceptions. String-based exceptions did not allow for the full range of functionality that is available with class-based exceptions, such as exception inheritance and chaining, custom exception types, and the ability to attach additional context information to exceptions. Class-based exceptions also provide a more structured and organized way to handle errors in Python code.