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

- Exceptions must inherit from the built-in Exception class or one of its subclasses

- Exceptions should provide a meaningful error message in the form of a string or other object that can be converted to a string.

Q2. When an exception is raised in Python, the interpreter looks for a handler that matches the exception type and hierarchy. Class-based exceptions are matched by the interpreter by checking if the exception class or one of its base classes matches the raised exception.

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

- The exception's \_\_context\_\_ attribute, which can be used to store another exception that was raised during the handling of the current exception.

- The exception's \_\_cause\_\_ attribute, which can be used to store an exception that caused the current exception to be raised.

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

- Passing the error message as an argument to the exception class's constructor when creating the exception object.

- Overriding the exception class's \_\_str\_\_ method to return a custom error message.

Q5. String-based exceptions have been deprecated and should no longer be used because they don't provide enough information about the exception, which makes it harder to diagnose and fix errors. Instead, user-defined exceptions should inherit from the built-in Exception class or one of its subclasses and provide a meaningful error message. This helps developers understand the cause of the exception and take appropriate action to handle it.