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

**Ans:** User-defined exceptions are never raised by the server; they are raised explicitly by a RAISE statement. ... You can define exceptions in functions, procedures, packages, or anonymous blocks. While you cannot declare the same exception twice in the same block, you can declare the same exception in two different blocks.

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

**Ans:** The exceptions that can be handled are class-based and can be handled between TRY and ENDTRY. The associated exception classes are predefined in the system and begin with the prefix CX\_SY\_, such as CX\_SY\_ZERODIVIDE. In the ABAP keyword documentation, the exception classes whose exceptions may occur when a corresponding ABAP statement is executed are listed for each keyword.

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

**Ans:** GetHibernateTemplate().update( obj ) This works if and only if an object doesn't already exist in the hibernate session. Exceptions are thrown stating an object with the given identifier already exists in the session when I need it later.

GetHibernateTemplate().merge( obj ) This works if and only if an object exists in the hibernate session. Exceptions are thrown when I need the object to be in a session later if I use this.

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

**Ans:** Errors happen all the time in the software world. It might be an invalid user input or an external system that is not responding, or it is a simple programming error. In all these situations, the errors occur at runtime and the application needs to handle them. Otherwise, it crashes and cannot process further requests. Java provides a powerful mechanism which allows you to handle the exceptional event where it occurred or in one of the higher methods in the call stack.

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

**Ans:** Exceptions come in different types, and the type is printed as part of the message: the types in the example are [ZeroDivisionError](https://docs.python.org/3/library/exceptions.html" \l "ZeroDivisionError" \o "ZeroDivisionError), [NameError](https://docs.python.org/3/library/exceptions.html" \l "NameError" \o "NameError) and [TypeError](https://docs.python.org/3/library/exceptions.html" \l "TypeError" \o "TypeError). The string printed as the exception type is the name of the built-in exception that occurred. This is true for all built-in exceptions but need not be true for user-defined exceptions (although it is a useful convention). Standard exception names are built-in identifiers (not reserved keywords).The rest of the line provides detail based on the type of exception and what caused it.

The preceding part of the error message shows the context where the exception occurred, in the form of a stack traceback. In general, it contains a stack traceback listing source lines; however, it will not display lines read from standard input.