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

A1. In Python 3.x, the two latest user-defined exception constraints are:

1. User-defined exceptions must be derived from the built-in **Exception** class or one of its subclasses.
2. User-defined exceptions should provide a docstring that explains the error condition and how to recover from it (if possible).

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

A2. When a class-based exception is raised, Python searches for an exception handler for the exception in the following way:

1. It checks the current function's try statement's except clause for a matching exception handler. If found, it executes that handler.
2. If the current function doesn't have a try statement or if no matching handler is found in the current function's try statement, Python searches for an except clause in the caller's try statement. If found, it executes that handler.
3. This search proceeds up the call stack until either a matching except clause is found, or Python reaches the top of the call stack, at which point the program terminates and the exception is printed to the console.

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

A3. There are two methods for attaching context information to exception artifacts in Python:

1. Adding custom attributes to the exception object: We can add attributes to the exception object that contain additional context information related to the exception. For example, we can add the current time, the filename of the module that raised the exception, or any other relevant information that could help in diagnosing the problem.
2. Using the contextlib module: The contextlib module provides a context manager called "contextmanager" that can be used to add context information to exceptions. We can use this context manager to wrap a block of code and add a description of what the code is doing, which will be displayed along with the exception traceback if an exception is raised inside the block. This can be especially useful for debugging complex code that involves multiple function calls and modules.

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

A4. In Python, there are two methods for specifying the text of an exception object's error message:

1. The first method is to pass the error message as a string argument to the exception class. This will create an instance of the **Exception** class with the specified error message, and raise it.
2. The second method is to override the **\_\_str\_\_** method of the exception class. This method should return a string that represents the error message. When an instance of **Exception** class is raised, Python will call the **\_\_str\_\_** method to get the error message. The method returns the string "This is an error message," which is used as the error message.

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

A5. String-based exceptions have been deprecated and are no longer used in modern Python because they lack some of the features that class-based exceptions provide, such as the ability to pass arguments and to handle exceptions in a more structured way. Additionally, string-based exceptions do not allow for hierarchy and inheritance, making it difficult to create complex exception hierarchies. Class-based exceptions provide a more organized and structured way of handling exceptions and allow for more flexibility and customization in exception handling. As a result, it is recommended to use class-based exceptions instead of string-based exceptions in modern Python code.