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

Two latest user-defined exception constraints in Python 3.X are:

1. **cause**: This attribute is used to link an exception to another exception that caused it.
2. **context**: This attribute is used to link an exception to another exception that is the context in which it was raised.

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

When a class-based exception is raised, Python looks for the first except block that matches the raised exception. If the exception class matches the class specified in the except block, then the code in that except block is executed. If no matching except block is found, then the exception is propagated up to the calling function or the default exception handler.

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

Two methods for attaching context information to exception artifacts are:

1. Using the contextlib module: The contextlib module provides a ContextDecorator class that can be used to attach context information to exceptions. This can be done by defining a class that inherits from ContextDecorator and overriding the **enter** and **exit** methods to set and clear the context information.
2. Using the exception's **init** method: The **init** method of an exception can be overridden to accept additional arguments that can be used to attach context information to the exception object. These arguments can then be stored as instance variables and accessed later when handling the exception.

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

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

1. Passing a message to the exception's constructor: Many built-in Python exceptions accept a message argument that is used to specify the text of the error message. This can be done by passing a string as an argument when creating an instance of the exception.
2. Overriding the exception's **str** method: The **str** method of an exception can be overridden to return a custom string that contains the error message. This can be done by defining a custom **str** method in a subclass of the exception.

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

String-based exceptions were used in older versions of Python, but they have been deprecated in Python 2.X and removed in Python 3.X. The main reason for this is that string-based exceptions are not as powerful as class-based exceptions. Class-based exceptions allow for more specific error handling and can provide more information about the cause of the error. They also allow for more fine-grained control over exception handling and better integration with other parts of the Python language. Additionally, class-based exceptions are more consistent with the object-oriented nature of Python, which makes code easier to understand and maintain.