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

Python detects all the critical errors that occur during Compile-time and Runtime. It stops the program's execution if the error occurs and raises an exception. Some commonly raised Exceptions are ArithmeticError, AttributeError, ImportError

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

In general, an exception is any unusual condition. Exception usually indicates errors but sometimes they intentionally puts in the program, in cases like terminating a procedure early or recovering from a resource shortage. There are number of built-in exceptions, which indicate conditions like reading past the end of a file, or dividing by zero. We can define our own exceptions called custom exception.

Exception handling enables you handle errors gracefully and do something meaningful about it. Exception handling has two components: “throwing” and ‘catching’.

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

As contents of an Argument can vary depending upon different types of Exceptions in Python, Variables can be supplied to the Exceptions to capture the essence of the encountered errors. Same error can occur of different causes, Arguments helps us identify the specific cause for an error using the except clause.

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

First, the try clause (the statement(s) between the try and except keywords) is executed.

If no exception occurs, the except clause is skipped and execution of the try statement is finished.

If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then, if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try/except block.

If an exception occurs which does not match the exception named in the except clause, it is passed on to outer try statements; if no handler is found, it is an unhandled exception and execution stops with a message as shown above.

BaseException is the common base class of all exceptions. One of its subclasses, Exception, is the base class of all the non-fatal exceptions. Exceptions which are not subclasses of Exception are not typically handled, because they are used to indicate that the program should terminate. They include SystemExit which is raised by sys.exit() and KeyboardInterrupt which is raised when a user wishes to interrupt the program.

Exception can be used as a wildcard that catches (almost) everything. However, it is good practice to be as specific as possible with the types of exceptions that we intend to handle, and to allow any unexpected exceptions to propagate on.

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

User-defined Python exceptions can be either strings or Python classes. Since classes have many nice properties when used as exceptions, it is desirable to migrate to a situation where classes are used exclusively. Prior to Python 1.5 alpha 4, Python's standard exceptions (IOError, TypeError, etc.) were defined as strings. Changing these to classes posed some particularly nasty backward compatibility problems.

In Python versions 1.5 and later, the standard exceptions are Python classes, and a few new standard exceptions have been added. The obsolete AccessError exception has been deleted. Because it is possible (although unlikely) that this change broke existing code, the Python interpreter can be invoked the command line option -X to disable this feature, and use string exceptions like before. This option is a temporary measure - eventually the string-based standard exceptions will be removed from the language altogether. It hasn't been decided whether user-defined string exceptions will be allowed in Python 2.0.