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

**Ans:** In 3.X, exceptions must be defined by classes (that is, a class instance object is raised and caught). In addition, exception classes must be derived from the built-in class **BaseException**; most programs inherit from its **Exception** subclass, to support catchall handlers for normal kinds of exceptions.

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

**Ans:** Class-based exceptions match by superclass relationships: naming a superclass in an exception handler will catch instances of that class, as well as instances of any of its subclasses lower in the class tree. Because of this, you can think of superclasses as general exception categories and subclasses as more specific types of exceptions within those categories.

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

**Ans:** We can attach context information to class-based exceptions by filling out instance attributes in the instance object raised, usually in a custom class constructor. For simpler needs, built-in exception superclasses provide a constructor that stores its arguments on the instance automatically (as a tuple in the attribute **args**). In exception handlers, you list a variable to be assigned to the raised instance, then go through this name to access attached state information and call any methods defined in the class.

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

**Ans:** The error message text in class-based exceptions can be specified with a custom **__str__** operator overloading method. For simpler needs, built-in exception superclasses automatically display anything you pass to the class constructor. Operations like **print** and **str** automatically fetch the display string of an exception object when it is printed either explicitly or as part of an error message.

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

**Ans:** Because they have been removed as of both Python 2.6 and 3.0. There are arguably good reasons for this: string-based exceptions did not support categories, state information, or behavior inheritance in the way class-based exceptions do. In practice, this made string-based exceptions easier to use at first when programs were small, but more complex to use as programs grew larger.