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

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

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

**Q4. Describe two methods for specifying the text of an exception object&#39;s error message.**

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

**SOLUTIONS**

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

* ***\_\_cause\_\_ which refers to the exception that caused the current exception to be raised***
* ***\_\_context\_\_ which refers to the context in which the exception occurred***

***2. Class-based exceptions that have been raised are matched to handlers based on the inheritance hierarchy of the exception classes. The interpreter searches for the closest matching handler by traversing up the hierarchy until it finds a handler that can handle the raised exception.***

***3. Two methods for attaching context information to exception artefacts are:***

* ***Using the raise statement with the from clause, which attaches the current exception as a cause of another exception.***
* ***Adding attributes to the exception object, such as exception.my\_context = "some context information".***

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

* ***Passing a string message as an argument when raising the exception, e.g. raise Exception("An error occurred").***
* ***Defining a \_\_str\_\_ method in the exception class, which returns the desired error message string when the exception is printed.***

***5. String-based exceptions are no longer used because they can lead to errors being caught unintentionally, as any string can be raised as an exception. Using class-based exceptions provides more control and specificity over the types of errors that can be raised and caught.***