The final argument, traceback, is also optional (and rarely used in practice), and if present, is the traceback object used for the exception.

Example

An exception can be a string, a class or an object. Most of the exceptions that the Python core raises are classes, with an argument that is an instance of the class. Defining new exceptions is quite easy and can be done as follows:

```
def functionName( level ):

   if level < 1:

      raise "Invalid level!", level

      # The code below to this would not be executed

      # if we raise the exception
```

Note: In order to catch an exception, an "except" clause must refer to the same exception thrown either class object or simple string. For example, to capture above exception, we must write the except clause as follows:

```
try:

   Business Logic here...

except "Invalid level!":

   Exception handling here...

else:

   Rest of the code here...
```

User-Defined Exceptions

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

Here is an example related to *RuntimeError*. Here, a class is created that is subclassed from *RuntimeError*. This is useful when you need to display more specific information when an exception is caught.

In the try block, the user-defined exception is raised and caught in *the except* block. The variable e is used to create an instance of the class *Networkerror*.

```
class Networkerror(RuntimeError):
```