Assignment 6

Q1. Describe three applications for exception processing.

Answer:

1. Error handling: Python raises exceptions whenever it detects errors in programs at runtime. You can catch and respond to the errors in your code, or ignore the exceptions that are raised.
2. Event notification : Exceptions can also be used to signal valid conditions without you having to pass result flags around a program or test them explicitly. For instance, a search routine might raise an exception on failure, rather than returning an integer result code (and hoping that the code will never be a valid result).
3. Special-case handling : Sometimes a condition may occur so rarely that it’s hard to justify convoluting your code to handle it. You can often eliminate special-case code by handling unusual cases in exception handlers in higher levels of your program.

Q2. What happens if you don’t do something extra to treat an exception?

Answer: Any uncaught exception eventually filters up to the *default exception handler* Python provides at the top of the program and simply prints the *standard error message* and immediately terminates the program. This course of action may make sense for simple scripts; However, working on complex projects such errors could be fatal, and the best one can do when they occur is inspect the standard error message.

NOTE: The standard error includes the exception that was raised, along with a stack trace — a list of all the lines and functions that were active when the exception occurred.

Q3. What are your options for recovering from an exception in your script?

Answer: If you don't want the default message and shutdown, you can code try/except statements to catch and recover from exceptions that are raised. Once an exception is caught, the exception is terminated and your program continues.

Q4. Describe two methods for triggering exceptions in your script.

Answer: **raise** and **assert** are two methods that can be used to trigger manual exceptions in your script.

1. **raise :** triggers an exception if the condition provided is true.
2. **assert** : mostly intended for trapping user-defined constraints, not for catching genuine programming errors.

Q5. Identify two methods for specifying actions to be executed at termination time, regardless of whether or not an exception exists.

Answer:

1. The **try/finally** statement is used to ensure actions are run after a block of code exits, regardless of whether it raises an exception or not.
2. The **with/as** statement is used to ensure termination actions are run, but only when processing object types that support it.

NOTE: The with/as statement is designed to be an alternative to a common try/finally usage idiom; like that statement, it is intended for specifying termination-time or "cleanup" activities that must run regardless of whether an exception occurs in a processing step.