**Q1. Describe three applications for exception processing.**

**Ans:** Exception processing is useful for error handling, termination actions, and event notification. It can also simplify the handling of special cases and can be used to implement alternative control flows as a sort of structured “go to” operation. In general, exception processing also cuts down on the amount of error-checking code your program may require—because all errors filter up to handlers, you may not need to test the outcome of every operation.

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

**Ans:** Any uncaught exception eventually filters up to the default exception handler Python provides at the top of your program. This handler prints the familiar error message and shuts down your program.

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

**Ans:** If we don’t want the default message and shutdown, we can code try/except statements to catch and recover from exceptions that are raised within its nested code block. Once an exception is caught, the exception is terminated and the program continues after the try.

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

**Ans:** The raise and assert statements can be used to trigger an exception, exactly as if it had been raised by Python itself. In principle, we can also raise an exception by making a programming mistake.

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

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