Python Exception Handling — Questions & Answers (Set 6)

Q1. Describe three applications for exception processing.

1) Error handling in user input (e.g., catching ValueError when parsing).
2) File/network/resource operations (IOError/OSError, socket errors).
3) Program flow control for retries/fallbacks and graceful degradation.

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

An uncaught exception halts execution, prints a traceback (type, message, stack),
and terminates the program (or returns to the REPL prompt in interactive mode).

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

- Catch and handle it with try/except (set defaults, prompt again, reconnect).
- Retry/fallback logic; exponential backoff; circuit breakers.
- Log, then reraise with 'raise' to propagate.
- Graceful termination with cleanup instead of crashing.

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

1) 'raise' to explicitly signal an error: raise ValueError('Invalid input').
2) 'assert' to enforce invariants in development: assert x > 0, 'x must be positive'.

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

1) 'finally' clause: always runs after try/except, ideal for cleanup.
2) 'with' statement (context managers): ensures __exit__ runs for cleanup (e.g., files, locks, DB connections).