Q1. Describe three applications for exception processing.

Applications for exception processing are:

1. **File I/O and Data Processing:** When dealing with file input/output operations and data processing, exceptions can occur due to various reasons such as file not found, permission issues, corrupted data, or unexpected formats. Exception processing ensures that the program can gracefully handle these situations. For instance, when reading data from a file, if the file is not found or cannot be opened, the program can catch the relevant exception and display a user-friendly error message. Similarly, if the data being processed doesn't conform to the expected format, exception handling can prevent the program from crashing and allow it to continue processing other data or notify the user about the issue.
2. **Network Communication:** Applications that involve network communication, such as web services, client-server interactions, or internet-connected devices, often encounter exceptions due to network disruptions, server unavailability, timeouts, or invalid responses. Exception processing in these scenarios enables the program to handle connectivity issues gracefully. For instance, if a web application is making API requests to a remote server and the server is down, the application can catch the corresponding exception and provide users with a clear error message instead of freezing or crashing.
3. **User Input Validation:** User input is a common source of unexpected data that can lead to exceptions. Exception processing is essential to validate and sanitize user input to prevent potential security vulnerabilities or program crashes. For instance, if a user provides input that doesn't match the expected data type (e.g., entering letters in a field meant for numbers), the program can raise an exception and guide the user to correct the input. This helps maintain the integrity of the program's functionality and enhances the user experience by preventing unintended errors.

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

Not treating exceptions appropriately can lead to program instability, poor user experience, security vulnerabilities, and difficulties in maintaining and debugging your code. Exception handling is an essential part of writing robust and reliable software, as it helps your program gracefully handle unexpected situations and ensures that it can recover or fail gracefully rather than crashing outright.

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

When recovering from an exception in your script, you have several options to consider. The appropriate approach depends on the type of exception, the context of your code, and the desired behavior of your program. Here are some common options:

1. **Retry Mechanism:** If the exception is caused by a transient issue, such as a network timeout or temporary unavailability of a resource, you can implement a retry mechanism. This involves catching the exception, waiting for a brief period, and then attempting the operation again. You might limit the number of retries to prevent getting stuck in an endless loop.
2. **Fallback Values:** If an exception occurs while retrieving or processing data, you can provide fallback values or default data to ensure that your program can continue running. For example, if you're fetching weather data from an API and encounter an exception, you could provide a default "N/A" value for the temperature.
3. **Logging and Reporting:** Even if you cannot recover directly from an exception, you can catch it, log relevant information, and notify developers or system administrators. This allows you to investigate the cause of the exception and potentially implement a fix in the future.
4. **User-Friendly Messages:** When an exception is raised, you can catch it and display a user-friendly error message instead of exposing technical details. This enhances the user experience by providing clear instructions on what went wrong and how to proceed.
5. **Graceful Termination:** In some cases, it might not be possible or safe to recover from an exception. For example, if a critical system component is unavailable, your script might need to terminate gracefully and inform users about the issue.
6. **Partial Operation:** If an exception occurs during a complex operation, you might choose to continue with the parts of the operation that were successful and skip the problematic parts. This can be especially useful in batch processing scenarios.
7. **Resource Cleanup:** If an exception occurs during resource-intensive operations (like file I/O or database connections), you can catch the exception and ensure that any acquired resources are properly closed and released before the script exits.

8. **User Input Handling:** When handling user input, catching exceptions related to invalid data (e.g., wrong data type, out-of-range values) can allow you to prompt the user for correct input instead of crashing the program.
9. **Fallback Paths:** In complex workflows, you can design fallback paths or alternative methods to achieve the same goal if the primary approach encounters an exception.

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

In script, one can intentionally trigger exceptions using various methods to simulate scenarios where errors or unexpected conditions occur. Two of them are:

1. **Using the raise Statement:** The **raise** statement is used to explicitly raise exceptions in Python and many other programming languages. You can use this statement to create and raise exceptions of a specific type or customize error messages
2. **Using Built-in Functions or Method**s: Some built-in functions or methods can trigger exceptions under specific conditions. For instance:
    - int("abc") would raise a ValueError because the string "abc" cannot be converted to an integer.
    - open("nonexistent_file.txt") would raise a FileNotFoundError because the file doesn't exist.
    - Accessing an index that is out of bounds in a list, tuple, or other sequence would raise an IndexError.
    - Trying to access a key that doesn't exist in a dictionary would raise a KeyError.

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

Two methods for specifying actions to be executed at termination time, regardless of whether or not an exception exists are:

1. Using finally block
2. Using Context Managers (with Statements)