Q1. Describe three applications for exception processing.

Ans: Here are three common applications for exception processing:

1. Error Handling: Exception processing is primarily used for error handling. When a program encounters an unexpected situation, such as division by zero, file not found, or invalid input, it can raise an exception. By catching and handling these exceptions, the program can gracefully recover from the error, log relevant information for debugging, and provide helpful error messages to users, thus improving the overall robustness and user experience of the application.
2. Resource Management: Another important application of exception processing is in resource management. When working with external resources, such as files, network connections, or database connections, it's crucial to handle exceptions to ensure that these resources are properly managed, released, or closed, even in the event of errors. Exception handling can help ensure that resources are freed correctly and prevent memory leaks or other resource-related issues.
3. Control Flow: Exception processing can also be used to control the flow of a program. By raising and catching exceptions at strategic points in the code, developers can alter the program's execution path based on specific conditions or errors. This can be particularly useful in situations where certain operations are contingent upon the successful completion of other operations. By using exceptions, developers can implement complex control flow logic that facilitates more robust and flexible program behavior.

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

Ans: If you don't handle an exception explicitly or provide any form of exception handling in your code, the program will terminate abruptly when an exception occurs. This can lead to an unhandled exception error, often accompanied by an error message that might not be informative to the end user. Additionally, the program might not clean up resources properly, potentially leading to memory leaks or other issues.

When an unhandled exception occurs, the program's execution is interrupted, and the exception traceback, which includes information about the type of exception, the line number where it occurred, and the call stack, is typically displayed. This can be problematic, especially in production environments or user-facing applications, as it can result in a poor user experience and leave the user confused about what went wrong.

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

Ans: In brief, the options for recovering from an exception in your script include:

1. Implementing try and except blocks to handle specific exceptions gracefully.
2. Setting up a robust logging mechanism to capture essential information about the exception for debugging purposes.
3. Implementing a retry mechanism for transient errors to increase the likelihood of successful execution.
4. Designing a strategy for graceful degradation to allow the script to continue running with reduced functionality in the presence of non-critical exceptions.
5. Displaying user-friendly error messages to guide users in understanding the nature of the exception and how to resolve it, enhancing the overall user experience.

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

Ans: Raise an Exception Manually: You can manually raise an exception using the raise statement. This allows you to generate and specify custom exceptions based on specific conditions within your script. You can raise built-in exceptions like ValueError, TypeError, or even create custom exceptions that inherit from the Exception class.

Call a Function That Raises an Exception: Calling a function that is designed to raise exceptions based on certain conditions is another way to trigger exceptions in your script. Many built-in functions or third-party libraries have error-handling mechanisms that trigger exceptions when specific criteria are not met.

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

Ans: In Python, you can use the try and finally block to ensure that certain actions are executed at termination time, regardless of whether or not an exception occurs. The finally block is used to define code that must be executed, ensuring cleanup or finalization tasks, even if an exception is raised.