Q1. Describe three applications for exception processing.

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

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

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

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

Solutions:

Q1. Three applications for exception processing are:

1. Error handling: Exception processing is commonly used for handling errors in software applications. When an unexpected or exceptional situation occurs during the execution of a program, an exception is raised. By using exception handling techniques, the program can gracefully handle the error, log the error details, and potentially recover from the error without crashing or terminating the program abruptly.

2. Input validation: Exception processing can be used for validating user input or data integrity. When receiving user input or reading data from external sources, such as files or databases, it's important to validate the input to ensure it meets certain criteria or constraints. If the input violates the expected format or range, an exception can be raised to handle the invalid input appropriately.

3. Resource management: Exception processing is also valuable for managing system resources, such as files, network connections, or database connections. If an exception occurs during the use of a resource, it allows the program to release the resource properly and prevent resource leaks. Exception handling can ensure that resources are properly cleaned up, regardless of whether an exception occurs.

Q2. If you don't do something extra to treat an exception, it will result in an unhandled exception. When an exception is not handled, the program execution will be halted, and an error message or stack trace will be displayed to indicate the location and nature of the exception. This can lead to program termination or crashes, leaving the application in an unpredictable state.

Q3. When recovering from an exception in a script, you have several options:

1. Try-catch block: You can use a try-catch block to catch and handle specific exceptions. The code within the try block is monitored for exceptions, and if an exception occurs, it is caught by the corresponding catch block. In the catch block, you can include code to handle the exception, such as logging the error, displaying a user-friendly message, or taking corrective actions. After executing the catch block, the program can continue running.

2. Finally block: The finally block is used in conjunction with a try-catch block. The code within the finally block is executed regardless of whether an exception occurred or not. It is typically used for cleanup operations, such as releasing resources or closing connections. Even if an exception is thrown and caught, the code within the finally block will still execute.

Q4. Two methods for triggering exceptions in your script are:

1. Explicitly raising exceptions: You can raise an exception explicitly at any point in your script using the `raise` keyword. By specifying the type of exception to be raised, along with an optional error message, you can trigger an exception intentionally. For example, you can raise a ValueError if a function receives invalid arguments or a FileNotFoundError if a required file is not found.

2. Built-in exceptions: Python provides a set of built-in exceptions that can be raised in specific circumstances. For instance, the `IndexError` is raised when an invalid index is used to access a list or tuple, and the `TypeError` is raised when an operation is performed on incompatible data types. By performing operations that can potentially lead to these exceptions, you can trigger them in your script.

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

1. Using the `finally` block: As mentioned earlier, the `finally` block is executed regardless of whether an exception occurred or not. You can include the desired actions within the finally block, and they will be executed before the program terminates. This ensures that the specified actions are performed even if an exception was raised and caught.

2. Registering an exception handler with `atexit`: The `atexit` module in Python provides a way to register functions to be called when the program is about to exit. By using the `atexit.register()` function, you can register a function that will be executed at termination time. This method allows you to specify custom actions to be performed before the program exits, regardless of any exceptions that may have occurred.