Q1. Describe three applications for exception processing.

**Answer:** Exception processing in Python provides a way to handle and manage unexpected or exceptional situations that may occur during program execution. Here are three common applications of exception processing in Python:

**Error Handling:** Exceptions are commonly used for error handling to gracefully handle and recover from exceptional conditions or errors that may occur during program execution. For example, when working with file operations, exceptions like FileNotFoundError can be caught to handle the scenario where a file doesn't exist. This allows the program to handle the error condition and take appropriate actions, such as displaying an error message, logging the error, or attempting alternative approaches.

**Input Validation:** Exception processing can be used for input validation to handle invalid user inputs or data. When receiving input from users or external sources, it's essential to validate the data for correctness and integrity. If the input doesn't meet the required criteria or violates certain constraints, an exception can be raised to indicate the validation failure. By catching and handling these exceptions, you can provide meaningful feedback to the user, prompt for correct input, or take corrective measures.

**Resource Management:** Exceptions are instrumental in managing resources, such as files, network connections, or database connections. Exceptions allow you to handle situations where resources may fail to open, close, or operate correctly. By catching exceptions related to resource management, you can ensure that the resources are properly released or cleaned up, even if an error occurs. This can help prevent resource leaks and ensure efficient utilization of system resources.

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

**Answer:** If an exception occurs during the execution of a program and it is not explicitly handled or caught, the program will terminate abruptly, and an error message known as a traceback will be displayed. The traceback provides information about the exception that occurred, including the type of exception, the line of code where the exception occurred, and the sequence of function calls leading to the exception.

When an unhandled exception occurs, the program flow is disrupted, and any remaining code in the current function and subsequent functions on the call stack is not executed. This can result in incomplete or incorrect program behavior, leaving resources open and leading to potential data corruption or loss.

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

**Answer:** When an exception occurs in a script, you have several options for recovering from it and controlling the program flow. Here are some common techniques for exception recovery in Python:

1.**Catch and Handle the Exception:** You can use a try-except block to catch the exception and handle it gracefully.

2.**Catch Multiple Exceptions:** If you want to handle different types of exceptions differently, you can specify multiple except blocks, each targeting a specific exception type.

3.**Cleanup with finally:** The finally block is used to define code that will be executed regardless of whether an exception occurs or not. This block is typically used for cleanup operations, such as closing files or releasing resources, ensuring that necessary actions are taken even in the presence of exceptions.

4.**Raising Exceptions:** In some cases, you may need to raise exceptions yourself to indicate error conditions or exceptional scenarios. By using the raise statement, you can raise an exception of a specific type, along with an optional error message.

5.**Exception Propagation:** If you catch an exception and decide not to handle it, you can re-raise the exception using the raise statement without any arguments. This allows the exception to propagate up the call stack, potentially being caught by an outer try-except block or resulting in program termination if not handled.

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

**Answer:**

1.Using the raise Statement: The raise statement allows you to explicitly raise exceptions in your script. You can specify the type of exception to be raised and, optionally, provide an error message. This method is useful when you want to indicate an error condition or simulate an exceptional scenario.

2.Using built-in functions or methods: Python provides several built-in functions or methods that can raise exceptions under specific circumstances. By calling these functions or methods with specific inputs or in certain situations, you can trigger exceptions.

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

**Answer:**

1.Using the finally Block

2.Using the atexit Module