Q1. Describe three applications for exception processing.
Exception processing in Python is used for various purposes, including:
1. Error Handling: Exceptions provide a mechanism to handle errors and exceptional situations in a program. By catching and handling exceptions, you can gracefully recover from errors, log information, display appropriate error messages, and prevent program crashes.
2. Resource Cleanup: Exceptions can be used to ensure proper cleanup of resources, such as closing files, releasing network connections, or freeing up system resources. By using exception handling in combination with the `finally` block, you can guarantee that cleanup operations are executed even if an exception occurs.
3. Control Flow: Exceptions can be used for control flow operations, such as breaking out of nested loops or signaling specific conditions. By raising custom exceptions, you can control the flow of your program and handle specific situations or errors in a structured manner.

Q2. What happens if you don't do something extra to treat an exception?
If an exception is not explicitly handled or caught in our code, it will propagate up the call stack until it reaches the default exception handler, which typically results in the termination of the program and an error message being displayed. This can lead to unexpected program behavior and crashes.

Q3. What are your options for recovering from an exception in your script?
When an exception occurs in your script, we have several options for recovering from it:
1. Catch and Handle the Exception: We can use a try-except block to catch specific exceptions and perform alternative actions or error recovery procedures. By handling exceptions, we can prevent the program from crashing and take appropriate measures to continue execution.
2. Retry Operations: In some cases, it may be appropriate to retry a failed operation after an exception occurs. We can implement retry logic within a loop, catching exceptions and attempting the operation again until it succeeds or a maximum number of retries is reached.
3. Graceful Termination: If the exception is severe or cannot be recovered, we can perform cleanup operations and gracefully terminate the program. This ensures that any necessary cleanup tasks are executed before exiting.

Q4. Describe two methods for triggering exceptions in your script.
In Python, we can trigger exceptions using the following methods:
1. Raise Statement: we can explicitly raise an exception using the `raise` statement. This allows you to create and raise custom exceptions or raise built-in exceptions with specific arguments. For example, `raise ValueError("Invalid input")` raises a `ValueError` exception with a custom error message.
2. Built-in Functions: Some built-in functions can trigger exceptions based on specific conditions. For instance, the `int()` function raises a `ValueError` if the provided argument cannot be converted to an integer. Similarly, the `open()` function raises a `FileNotFoundError` if the specified file cannot be found.

Q5. Identify two methods for specifying actions to be executed at termination time, regardless of whether or not an exception exists.
In Python, we can specify actions to be executed at termination time, regardless of whether or not an exception occurs, using the following methods:
1. The `finally` Block: The `finally` block is used in conjunction with a `try-except` block to define a set of statements that are guaranteed to be executed, regardless of whether an exception is raised or caught. This allows us to ensure that certain cleanup operations or finalization tasks are performed, such as closing files or releasing resources, before the program exits.
2. The `atexit` Module: The `atexit` module provides a way to register functions to be called when the Python interpreter is about to exit. We can use the `atexit.register()` function to register one or more functions that will be automatically called when the program terminates normally. This is useful for performing cleanup tasks or saving state

before the program exits.