Q1. Describe three applications for exception processing.

1. Error Handling: Exceptions are commonly used to handle errors and exceptional conditions that may occur during the execution of a program. For example, when working with file operations, exceptions can be raised if the file does not exist, cannot be opened, or encounters an error during reading or writing. By catching and handling these exceptions, you can gracefully handle errors and provide appropriate error messages or alternative actions to the user.
2. Input Validation: Exceptions can be used for input validation to ensure that user input meets certain criteria or constraints. For instance, when accepting user input for a numerical value, an exception can be raised if the input is not a valid number. By catching and handling these exceptions, you can prompt the user to provide valid input or display error messages indicating the specific validation failure.
3. Resource Management: Exceptions can be useful for managing resources such as files, network connections, or database connections. In situations where resources need to be properly allocated and released, exceptions can be used to handle exceptional cases where resource allocation or release fails. By catching and handling these exceptions, you can ensure that resources are properly managed and released, even in the event of errors or unexpected situations.

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

It will propagate up the call stack until it reaches the top-level of your program or an exception handler is encountered. When an unhandled exception reaches the top-level, it typically results in the termination of the program and an error message or traceback is displayed.

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

When an exception occurs in your script, you have several options for recovering from it:

1. Handle the Exception: You can use a try-except block to catch and handle specific exceptions. By enclosing the code that might raise an exception within a try block and providing corresponding except blocks, you can define the actions to be taken when a particular exception occurs. This allows you to gracefully recover from the exception and continue with the execution of the script.
2. Propagate the Exception: If you are not able to handle the exception locally, you can choose to propagate it up the call stack. This means that you don't catch the exception in your current code block, but instead allow it to be passed to the calling code. The exception will continue to propagate until it is caught and handled by an appropriate try-except block higher up in the call stack.
3. Retry the Operation: In some cases, it may be appropriate to retry the operation that raised the exception. For example, if you encounter a network connection error, you can implement a retry mechanism that attempts the operation again after a certain delay or up to a certain number of times. This can be useful for handling transient errors and improving the robustness of your script.
4. Provide Default Values or Alternative Logic: If an exception occurs while retrieving a value or performing a specific operation, you can provide default values or alternative logic to handle the exceptional case. This can involve returning a default value, substituting with a placeholder value, or executing alternative code paths to continue the execution of the script.
5. Log the Exception: Logging the exception can be helpful for debugging and troubleshooting purposes. You can use a logging library to log the details of the exception, including the type, message, stack trace, and any relevant contextual information. This allows you to review the logs later and analyze the cause of the exception.

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

In Python, you can trigger exceptions in your script using various methods. Here are two common methods for triggering exceptions:

1. Raise an Exception Explicitly: You can raise an exception explicitly using the raise statement. This allows you to indicate that a specific error condition has occurred and needs to be handled. You can raise built-in exceptions like ValueError, TypeError, or RuntimeError, or you can create custom exceptions by subclassing the built-in Exception class. Here's an example of raising an exception explicitly:

def divide(x, y):

if y == 0:

raise ValueError("Cannot divide by zero")

return x / y

try:

result = divide(10, 0)

except ValueError as e:

print(e)

1. Trigger an Exception by Invoking Built-in Functions or Methods: Certain built-in functions or methods in Python can raise exceptions under specific conditions. By invoking these functions or methods with arguments that trigger the exceptional behavior, you can indirectly trigger an exception. For example:

file\_path = "nonexistent\_file.txt"

try:

with open(file\_path, "r") as file:

contents = file.read()

except FileNotFoundError:

print(f"File not found: {file\_path}")

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

In Python, you can specify actions to be executed at termination time, regardless of whether or not an exception exists, using the following two methods:

1. Finally Block: You can use a finally block to specify code that should be executed regardless of whether an exception is raised or not. The code inside the finally block will always be executed, ensuring that necessary cleanup or termination actions are performed. Here's an example:

file = None

try:

file = open("file.txt", "r")

# Perform some operations

finally:

if file:

file.close()

1. Context Managers (with Statement): Another method to specify actions at termination time is by using context managers with the with statement. Context managers provide a convenient way to manage resources and ensure their proper cleanup, even in the presence of exceptions. The with statement automatically sets up the context, executes the code within it, and then cleans up the context afterward. Here's an example:

with open("file.txt", "r") as file:

# Perform some operations