Q1. Describe three applications for exception processing.

Q2. What happens if you don&#39;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.

### **Q1. Describe three applications for exception processing.**

1. **Error Handling in User Input**:
   * **Application**: When a program takes input from users, the input might not always be valid (e.g., entering text when a number is expected). Exception processing allows the program to handle such cases gracefully by catching and handling the errors without crashing.

**Example**:  
python  
Copy code  
try:

number = int(input("Enter a number: "))

except ValueError:

print("That's not a valid number!")

1. **File Operations**:
   * **Application**: When performing file operations such as reading or writing, files might be missing, or permissions might be inadequate. Exception processing allows the program to handle these issues, such as by prompting the user or creating the file if it doesn't exist.

**Example**:  
python  
Copy code  
try:

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

data = file.read()

except FileNotFoundError:

print("File not found. Please check the file name.")

1. **Network or Database Connections**:
   * **Application**: When working with network operations or database connections, exceptions can occur due to connectivity issues, timeouts, or authentication failures. Exception processing can help retry connections, log errors, or switch to a backup strategy.

**Example**:  
python  
Copy code  
import requests

try:

response = requests.get('https://example.com')

response.raise\_for\_status()

except requests.exceptions.HTTPError as err:

print(f"HTTP error occurred: {err}")

except requests.exceptions.ConnectionError:

print("Failed to connect to the server.")

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

If you don't handle an exception explicitly, the Python interpreter will terminate the program and print a traceback to the console, showing the sequence of function calls that led to the exception. The program will stop executing at the point where the unhandled exception occurred.

Example:

python

Copy code

number = int(input("Enter a number: "))

# If the user enters a non-integer value, the program will raise a ValueError and terminate.

**Result**:

* The program crashes.
* A traceback message is printed, indicating where the error occurred.

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

1. **Using try-except Blocks**:
   * You can catch and handle specific exceptions using try-except blocks. This allows the program to continue running after an exception has been handled.

Example:  
python  
Copy code  
try:

number = int(input("Enter a number: "))

except ValueError:

print("That's not a valid number. Please try again.")

number = 0

1. **Using try-except-else**:
   * The else block can be used to execute code only if no exceptions were raised in the try block.

Example:  
python  
Copy code  
try:

number = int(input("Enter a number: "))

except ValueError:

print("That's not a valid number.")

else:

print("No exceptions occurred, proceeding with:", number)

1. **Using try-except-finally**:
   * The finally block can be used to ensure that specific cleanup code is executed, whether an exception occurred or not.

Example:  
python  
Copy code  
try:

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

except FileNotFoundError:

print("File not found.")

finally:

if file:

file.close()

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

1. **Using the raise Statement**:
   * You can explicitly trigger an exception using the raise statement. This is useful when you want to enforce certain conditions in your code.

Example:  
python  
Copy code  
def divide(a, b):

if b == 0:

raise ValueError("Cannot divide by zero.")

return a / b

1. **Using Assertions**:
   * Assertions can be used to trigger exceptions when a condition is not met. The assert statement checks a condition and raises an AssertionError if the condition is False.

Example:  
python  
Copy code  
def square\_root(x):

assert x >= 0, "x must be non-negative"

return x \*\* 0.5

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

1. **Using the finally Block**:
   * The finally block in a try-except structure is used to define cleanup code that will run no matter what happens in the try block, even if an exception is raised.

Example:  
python  
Copy code  
try:

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

except FileNotFoundError:

print("File not found.")

finally:

print("This will run no matter what.")

if file:

file.close()

1. **Using Context Managers (with Statement)**:
   * Context managers automatically manage resources by defining setup and teardown actions. The with statement ensures that resources are properly cleaned up when the block is exited, even if an exception occurs.

Example:  
python  
Copy code  
with open('file.txt', 'r') as file:

data = file.read()

# The file is automatically closed when the block is exited, regardless of exceptions.

These mechanisms ensure that important cleanup actions, like closing files or releasing network resources, are executed no matter what, preventing resource leaks and maintaining program stability.