Q1. What is the purpose of the try statement?

Q2. What are the two most popular try statement variations?

Q3. What is the purpose of the raise statement?

Q4. What does the assert statement do, and what other statement is it like?

Q5. What is the purpose of the with/as argument, and what other statement is it like?

### **Q1. What is the purpose of the try statement?**

The **purpose of the try statement** is to define a block of code in which exceptions (runtime errors) might occur and to specify one or more blocks of code (except, else, and finally) that will handle those exceptions. The try statement allows a program to continue running or perform specific actions if an error occurs, rather than terminating abruptly.

Example:

python

Copy code

try:

x = 10 / 0

except ZeroDivisionError:

print("Cannot divide by zero.")

### **Q2. What are the two most popular try statement variations?**

1. **try-except**:
   * This is the most common form, where the try block contains code that might raise an exception, and the except block contains code to handle the exception.

Example:  
python  
Copy code  
try:

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

except ValueError:

print("Invalid input! Please enter a number.")

1. **try-except-finally**:
   * In this variation, the finally block is added to define cleanup code that will run regardless of whether an exception was raised or not. The finally block is typically used for resource cleanup, such as closing files or network connections.

Example:  
python  
Copy code  
try:

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

# Some operations with the file

except FileNotFoundError:

print("File not found.")

finally:

print("This block always executes.")

if file:

file.close()

### **Q3. What is the purpose of the raise statement?**

The **purpose of the raise statement** is to trigger an exception intentionally in your code. This can be used to signal an error condition or to enforce certain conditions that must be met in the program. The raise statement can be used to raise built-in exceptions or custom exceptions.

Example:

python

Copy code

def divide(a, b):

if b == 0:

raise ValueError("Cannot divide by zero.")

return a / b

try:

result = divide(10, 0)

except ValueError as e:

print(e)

### **Q4. What does the assert statement do, and what other statement is it like?**

The **assert statement** is used to test a condition in your code. If the condition evaluates to True, the program continues running. If the condition evaluates to False, an AssertionError is raised, optionally with a specified error message. The assert statement is primarily used for debugging purposes to check for conditions that should never happen.

Example:

python

Copy code

x = -1

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

# This will raise an AssertionError with the message "x must be non-negative"

The assert statement is somewhat similar to the raise statement in that it can trigger an exception, but it is typically used for simpler condition checks and is often removed from production code.

### **Q5. What is the purpose of the with/as statement, and what other statement is it like?**

The **purpose of the with/as statement** is to simplify resource management by ensuring that resources are properly acquired and released. It is most commonly used with context managers, such as opening files, where the with statement handles the setup and teardown of resources automatically, even in the presence of exceptions.

Example:

python

Copy code

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

data = file.read()

# The file is automatically closed when the block is exited

The with/as statement is conceptually similar to the **try/finally** structure. The with statement handles resource management in a more concise and readable manner than explicitly using try/finally blocks.

Example equivalent to the above:

python

Copy code

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

try:

data = file.read()

finally:

file.close()