Q1. What is the purpose of the try statement?

The purpose of the try statement in Python is to enclose a block of code that might raise exceptions. It allows you to handle exceptions and control the program's behavior in case an exception occurs during the execution of the code within the try block. It's used to prevent program crashes and to handle error conditions gracefully.

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

The two most popular variations of the try statement are:

try-except: This variation allows you to specify one or more except blocks following the try block. If an exception occurs within the try block, the corresponding except block is executed based on the type of the raised exception.

try-finally: This variation combines the try block with a finally block. The code in the finally block is guaranteed to be executed, whether an exception occurs or not. It's used for cleanup operations such as releasing resources.

Q3. What is the purpose of the raise statement?

The raise statement is used to raise exceptions in Python. It can be used to raise built-in exceptions or custom exceptions that you define. By raising exceptions, you can indicate that an error or exceptional situation has occurred in your code. The raise statement is often used in conjunction with try and except blocks to handle exceptions or propagate them to higher levels.

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

The assert statement is used for debugging purposes to check whether a given condition is True. If the condition is False, an AssertionError is raised. It's used to catch logical errors during development. The assert statement is similar to the if statement, but it is used specifically for checking and debugging conditions that should always be true in a correct program.

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

The with/as statement is used to simplify the management of resources like files, network connections, and locks. It provides a way to ensure that resources are properly acquired and released, even if exceptions occur. It is similar in purpose to a try-finally block but provides a more structured and concise way to manage resources, reducing the risk of resource leaks.

Example of using with statement with a file:

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

data = f.read()

# File is automatically closed after the block