1. Create an assert statement that throws an AssertionError if the variable spam is a negative integer.

assert spam >= 0, "spam cannot be a negative integer"

2. Write an assert statement that triggers an AssertionError if the variables eggs and bacon contain strings that are the same as each other, even if their cases are different (that is, 'hello' and 'hello' are considered the same, and 'goodbye' and 'GOODbye' are also considered the same).

assert eggs.lower() != bacon.lower(), "eggs and bacon strings cannot be the same"

3. Create an assert statement that throws an AssertionError every time.

assert False, "This assertion always triggers an AssertionError"

4. What are the two lines that must be present in your software in order to call logging.debug()?

import logging

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

5. What are the two lines that your program must have in order to have logging.debug() send a logging message to a file named programLog.txt?

import logging

logging.basicConfig(filename='programLog.txt', level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

6. What are the five levels of logging?

DEBUG

This level is used for detailed information, typically useful for debugging purposes.

Debug messages are usually used to provide insight into the flow of the program, including variable values and function calls.

These messages are typically only logged during development and testing phases and are usually not enabled in production environments to avoid cluttering the log files.

INFO

This level is used to provide informational messages that highlight the progress or status of the program.

Info messages are typically used to convey general information about the execution of the program, such as startup/shutdown messages, configuration changes, or major milestones.

These messages are useful for administrators or users to understand what the program is doing without needing detailed debugging information.

WARNING

This level is used to indicate potential issues or situations that could lead to errors or unexpected behavior.

Warning messages are used to alert users or administrators about conditions that may require attention, such as deprecated features, low disk space, or failed connections to external services.

While warnings do not necessarily indicate an error, they suggest a potential problem that should be investigated to prevent more severe issues.

ERROR

This level is used to indicate errors that have occurred during the execution of the program.

Error messages signify problems that have occurred but are not critical enough to halt the program entirely. Examples include file not found errors, database connection failures, or input validation errors.

These messages help developers and administrators identify and troubleshoot issues that need to be resolved to ensure the proper functioning of the program.

CRITICAL

This level is used to indicate critical errors or failures that require immediate attention.

Critical messages signify severe problems that may lead to the termination of the program or significant data loss. Examples include unhandled exceptions, system resource exhaustion, or security breaches.

These messages are essential for alerting developers and administrators to urgent issues that need immediate investigation and resolution to prevent system failure or data loss.

7. What line of code would you add to your software to disable all logging messages?

```
logging.disable(logging.CRITICAL + 1)
```

8. Why is using logging messages better than using print() to display the same message?

Logging provides more flexibility in handling and filtering messages based on severity levels.

Logging messages can be redirected to different outputs (e.g., console, file) without changing the code.

Logging allows for easy configuration and customization, such as setting log levels and formatting.

9. What are the differences between the Step Over, Step In, and Step Out buttons in the debugger?

Step Over: Executes the current line of code and moves to the next line in the current function, skipping over function calls.

Step In: Moves the debugger into the function call, allowing you to step through each line of the called function.

Step Out: Executes the rest of the code in the current function and returns to the caller, stopping at the line immediately after the function call.

10. After you click Continue, when will the debugger stop ?

The debugger will stop when it encounters another breakpoint or when the program execution is completed.

11. What is the concept of a breakpoint?

A breakpoint is a designated point in the code where the debugger will pause execution, allowing you to inspect variables, step through code, and troubleshoot issues.