1. Create an assert statement that throws an AssertionError if the variable spam is a negative integer. **assert spam >= 0, "Spam is negative"**

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 have the same string value"**

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

**assert False, "This assert statement always fails"**

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)**

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 %(message)s')**

6. What are the five levels of logging?

**Debug: Detailed information used for debugging purposes.**

**Info: General information about the application's normal operations.**

**Warning: Indication of an unusual or unexpected event, but the application can continue to run.**

**Error: Indication of a significant problem that prevents the application from running as expected.**

**Critical: The highest level, indicating a catastrophic failure that requires immediate attention.**

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

**import logging**

**logging.disable(logging.CRITICAL)**

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

**Using logging messages instead of print() provides better separation of concerns, flexibility, persistence, scalability, and performance. Logging allows for flexible configuration of log levels and destinations, persistence of log messages in files, and improved performance through asynchronous writes to a buffer.**

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 without entering any function calls.**

**Step In executes the current line of code and enters any function calls made on that line.**

**Step Out continues execution of the code until the current function call returns and moves back to the calling function.**

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

**The debugger stops when it reaches a breakpoint set by the user or an exception is thrown in the code. Breakpoints are markers to pause the execution, and exceptions are unexpected events that cause the program to halt. The debugger provides a way to inspect the state of the program and debug any issues.**

11. What is the concept of a breakpoint?

**A breakpoint is a marker in code used during debugging to pause program execution, allowing inspection and modification of the program's state.**