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

**Answer: java**

**assert spam >= 0, 'Error: spam is a negative integer'**

1. **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).**

**Answer: lua**

**assert eggs.lower() != bacon.lower(), 'Error: eggs and bacon are the same, even if their cases are different'**

1. **Create an assert statement that throws an AssertionError every time.**

**Answer: python**

**assert False, 'This assertion always triggers an AssertionError'**

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

**Answer: The following two lines must be present in the software to call logging.debug():**

**python**

**import logging**

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

1. **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?**

**Answer: The following two lines must be added to the program to send logging messages to a file named programLog.txt:**

**python**

**import logging**

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

1. **What are the five levels of logging?**

**Answer: The five levels of logging, in increasing order of severity, are:**

**DEBUG**

**INFO**

**WARNING**

**ERROR**

**CRITICAL**

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

**Answer: scss**

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

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

**Answer: Using logging messages instead of print() is better because logging messages can be easily turned on or off depending on the level of logging desired. Additionally, logging messages can be directed to various outputs, such as a file or the console, making it easier to manage and analyze large amounts of data.**

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

**Answer: Step Over: Executes the current line of code and then stops at the next line of code. If the current line contains a function call, the entire function is executed before stopping at the next line of code.**

**Step In: If the current line of code contains a function call, Step In will move the debugger to the first line of the function and stop there. If the current line does not contain a function call, Step In behaves the same as Step Over.**

**Step Out: When inside a function call, Step Out will continue execution until the function returns and stop at the first line after the function call.**

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

**Answer: The debugger will stop when it encounters another breakpoint or when the program finishes executing.**

1. **What is the concept of a breakpoint?**

**Answer: A breakpoint is a point in the program's code where the debugger stops execution to allow the programmer to examine the state of the program. Breakpoints can be set on specific lines of code or conditions, and can help the programmer pinpoint errors or unexpected behavior in the program.**