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

assert spam >= 0, "spam should not 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.**

assert eggs.lower() != bacon.lower(), "eggs and bacon should not be the same (case-insensitive)"

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

assert False, "This assertion 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)

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

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

1. DEBUG - Detailed information for debugging.
2. INFO - General information about the program's execution.
3. WARNING - Indicates a potential issue that doesn't stop the program.
4. ERROR - Indicates a more serious issue that prevents a function from running.
5. CRITICAL - Indicates a severe error that may prevent the program from continuing.

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

logging.disable(logging.CRITICAL)

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

* **Logging provides levels of severity**: You can categorize messages (e.g., debug, info, warning, error, critical) and filter them based on importance.
* **Flexibility**: Logging can be configured to output to different destinations (e.g., console, files, external services).
* **Persistence**: Logging messages can be saved to files for later analysis.
* **Performance**: Logging can be disabled or enabled selectively without modifying the code.
* **Standardization**: Logging follows a consistent format, making it easier to parse and analyze.

**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 same function. If the line contains a function call, it executes the entire function without stepping into it.
* **Step In**: Moves into the function being called on the current line, allowing you to debug the function's code line by line.
* **Step Out**: Executes the remaining lines of the current function and returns to the line where the function was called.

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

The debugger will stop when:

* A breakpoint is encountered.
* An exception occurs (if the debugger is configured to break on exceptions).
* The program finishes execution.