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

Ans: 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).

Ans: assert eggs.lower() != bacon.lower(), "eggs and bacon should not be the same, ignoring case"

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

Ans: assert False, "This assertion always raises an error"

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

Ans:

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?

Ans:

import logging

# Configure the logging level and set up the file handler for programLog.txt

logging.basicConfig(filename='programLog.txt', level=logging.DEBUG)

6. What are the five levels of logging?

Ans:

1. DEBUG: Detailed information, typically used for debugging purposes.
2. INFO: Confirmation that things are working as expected.
3. WARNING: An indication that something unexpected happened, or an indication of a potential problem.
4. ERROR: A report of a serious issue or error that prevents the software from functioning as intended.
5. CRITICAL: A report of a critical error or a severe issue that may cause the program to terminate.

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

Ans: logging.disable(logging.CRITICAL)

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

Ans: using logging messages provides a structured and systematic approach to managing and handling application messages, making it a more scalable and effective way to handle logging in larger and more complex software projects.

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

Ans:

* **Step Over:**
  + When you click the Step Over button, the debugger executes the current line of code and moves to the next line. If the current line contains a function call, the entire function is executed, and the debugger moves to the next line after the function call.
  + Use Step Over to move to the next line of code without stepping into the details of function calls.
* **Step In:**
  + Clicking the Step In button allows you to delve deeper into the current line of code, particularly if the line contains a function call. The debugger enters the function call and stops at the first line of the called function, enabling you to examine the internal workings of the function.
* **Step Out:**
  + Step Out is used when you are debugging within a function, and you want to execute the rest of the code in that function without stepping through each line. It helps you quickly move out of the current function and return to the calling context.
  + The Step Out button is beneficial when you have stepped into a function using Step In and want to quickly return to the caller's context.

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

Ans:

1. A breakpoint is encountered: If the program execution reaches a line with a breakpoint set, the debugger will pause at that line, allowing you to inspect the program's state and variables at that particular point in the code.
2. An exception is raised: If an unhandled exception occurs during the program's execution, the debugger will pause at the line where the exception was raised, providing you with information about the exception and the context in which it occurred.
3. The program's execution is completed: If the program execution reaches its end without encountering any exceptions or breakpoints, the debugger will stop once the program has finished running, allowing you to review the final state of the program and its variables.

11. What is the concept of a breakpoint?

Ans: In the context of software development and debugging, a breakpoint is a designated point in the source code where the debugger will pause the program's execution during the debugging process. When the program reaches a line with a breakpoint set, the debugger stops the program, allowing the developer to inspect the program's state, variable values, and the context of the code at that particular point in the execution flow.