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

A. assert spam >= 0, 'spam should be a non-negative integer'

This code will check if **spam** is greater than or equal to zero. If it's not, an **AssertionError** will be raised with the message 'spam should be a non-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).

A. assert eggs.lower() != bacon.lower(), "eggs and bacon cannot have the same value, ignoring case."

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

A. 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()?

A.

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?

A.

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?

A. The five levels of logging in Python, from lowest to highest severity, are:

1. DEBUG: Detailed information, typically of interest only when diagnosing problems.
2. INFO: Confirmation that things are working as expected.
3. WARNING: An indication that something unexpected or potentially problematic has occurred, or indicative of some problem in the near future.
4. ERROR: Due to a more serious problem, the software has not been able to perform some function.
5. CRITICAL: A very serious error, indicating that the program itself may be unable to continue running.

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

A. logging.disable(logging.CRITICAL)

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

A. Using logging messages is better than using print() to display the same message because of the following reasons:

1. Control: With logging, we have more control over what messages get displayed and where they get displayed. We can choose which messages get logged and at what level. We can also choose where the messages are logged to, such as a file or a database.
2. Flexibility: Logging provides more flexibility than print statements because we can choose the level of detail that is appropriate for each message. This allows us to have more granular control over what gets displayed, especially when debugging.
3. Filter and search: Logging messages can be filtered and searched, making it easier to find specific messages when debugging. This is especially useful when working with large applications that generate a lot of output.
4. Security: Logging messages can be used to provide a security audit trail, which can be important in certain applications.
5. Performance: Finally, logging can be more performant than print statements because it is designed to be efficient and optimized for large volumes of data.

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

A. The Step Over, Step In, and Step Out buttons are used in a debugger to control program execution during debugging. Here are the differences between them:

1. Step Over: When we click the Step Over button, the debugger executes the current line of code and stops on the next line. If the current line contains a function call, the entire function is executed, but the debugger does not stop on any of the lines in the function. This means that the function is "stepped over" and execution continues on the next line of code after the function call.
2. Step In: When we click the Step In button, the debugger executes the current line of code and stops on the next line. If the current line contains a function call, the debugger steps into the function and stops on the first line of code in the function. This means that the function is "stepped into" and execution continues within the function.
3. Step Out: When we click the Step Out button, the debugger executes the remaining lines of code in the current function and stops on the line of code that called the function. This means that the function is "stepped out" and execution continues in the calling function.

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

A. After clicking the Continue button in the debugger, the debugger will stop at the next breakpoint or when the program terminates, whichever comes first. If there are no more breakpoints and the program continues to run until completion, the debugger will stop and the debugging session will end.

11. What is the concept of a breakpoint?

A. A breakpoint is a point in the source code of a program where the execution of the program is paused, allowing the developer to inspect the state of the program at that point. Breakpoints are often used in debugging to help developers understand how their code is behaving by allowing them to step through the program one line at a time, examining the state of the program at each step. When a breakpoint is encountered during the execution of the program, the debugger stops the program and allows the developer to examine the values of variables, evaluate expressions, and step through the code. Breakpoints can be set in most modern integrated development environments (IDEs) and text editors with debugging capabilities.