--QUESTIONS--

1. Create an assert statement that throws an AssertionError if the variable spam is 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).
3. Create an assert statement that throws an AssertionError every time.
4. What are the two lines that must be present in your software in order to call 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?
6. What are the five levels of logging?
7. What line of code would you add to your software to disable all logging messages?
8.Why is using logging messages better than using print() to display the same message?
9. What are the differences between the Step Over, Step In, and Step Out buttons in the debugger?
10.After you click Continue, when will the debugger stop ?
11. What is the concept of a breakpoint?

--ANSWERS--

1- assert(spam < 0, 'The spam variable is less than 0.')

2- assert(eggs.lower() != bacon.lower(), 'The eggs and bacon variables are the same!') or assert(eggs.upper() != bacon.upper(), 'The eggs and bacon variables are the same!')

3- assert(False, 'This assertion always triggers.')

4- To be able to call logging.debug(), you must have these two lines at the start of your program: import logging logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)s - %(message)s')

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

6- DEBUG, INFO, WARNING, ERROR, and CRITICAL

7- logging.disable(logging.CRITICAL)

8- You can disable logging messages without removing the logging function calls. You can selectively disable lower-level logging messages. You can create logging messages. Logging messages provides a timestamp.

9- The Step button will move the debugger into a function call. The Over button will quickly execute the function call without stepping into it. The Out button will quickly execute the rest of the code until it steps out of the function it currently is in.

10- click Go, the debugger will stop when it has reached the end of the program or a line with a breakpoint.

11- breakpoint is a setting on a line of code that causes the debugger to pause when the program execution reaches the line