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

* assert (spam > 0, ‘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).

* assert (eggs.lower() != bacon.lower(), ‘variables eggs and bacon are the same’)
* assert (eggs.upper() != bacon.upper(), ‘variables eggs and bacon are the same’)

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

* assert(False, ‘This assertion always triggers’)

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, 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?

* import logging
* logging.basicConfig(filename='programLog.txt', level=logging.DEBUG, format='%(asctime)s %(levelname)s %(message)s')
* logging.debug(f’ this is my debug log)

6. What are the five levels of logging?

* DEBUG, INFO, ERROR, WARNING, CRITICAL

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?

* print() is good to check the result in the console but doesn’t tell you where and where error occurred. In production settings having track of when, where and why some mistake occurred helps resolving the issue, logging messages facilitate timestamp, error/warning message for easy spot of errors and fixing issues.

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

* step in: if there is a function call, it goes inside the function, so you can see how function executes line by line until it returns and you go back to the next line after function call
* step over: if there is a function call,it executes like a black box and returns the result, you can not see how function was executed
* step out: if you have STEPPED IN a function, and you want to skip seeing how the rest of the function executes, you just Step out and the function returns. Then, go to the next line after the function call.

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

* start running until next breakpoint

11. What is the concept of a breakpoint?

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