Question 1.

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

Answer 1:

```
spam = -10
assert spam > 0, 'The spam variable is is a negative integer.'
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-1-ebb011da199a> in <module>
      1 spam = -10
----> 2 assert spam > 0, 'The spam variable is is a negative integer.'

AssertionError: The spam variable is is a negative integer.
```

Question 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).

Answer 2:

```
eggs = 'hello'
bacon = 'Hello'

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

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-13-53586d290ac7> in <module>
      2 bacon = 'Hello'
      3
----> 4 assert eggs.lower() != bacon.lower(), 'The eggs and bacon variables are the same!'
      5 assert eggs.upper() != bacon.upper(), 'The eggs and bacon variables are the same!'

AssertionError: The eggs and bacon variables are the same!
```

```
eggs = 'goodbye'
bacon = 'GOODbye'

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

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-4-f665b36b2d80> in <module>
      2 bacon = 'GOODbye'
      3
----> 4 assert eggs.lower() != bacon.lower(), 'The eggs and bacon variables are the same!'
      5 assert eggs.upper() != bacon.upper(), 'The eggs and bacon variables are the same!'

AssertionError: The eggs and bacon variables are the same!
```

Question 3.

Create an assert statement that throws an AssertionError every time.

Answer 3:

assert False, 'This assertion always triggers.'

Question 4.

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

Answer 4:

```
import logging
logging.basicConfig(filename = 'abc.txt',level = logging.DEBUG, format = '%(asctime)s - %(message)s - %(levelname)s')
logging.debug('Start')
logging.debug('End')
```

Question 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?

Answer 5:

To be able to send logging messages to a file named programLog.txt with logging.debug(), we must have these two lines at the start of your program:

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

Question 6.

What are the five levels of logging?

Answer 6:

LEVELS OF LOGGING:

1. Debug
2. INFO
3. WARNINGS
4. ERROR
5. CRITICAL

Question 7.

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

Answer 7:

logging.disable(logging.CRITICAL)

Question 8.

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

Answer 8:

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

Question 9.

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

Answer 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.

Question 10.

After you click Continue, when will the debugger stop ?

Answer 10:

After we click Continue, the debugger will stop when it has reached the end of the program or a line with a breakpoint.

Question 11.

What is the concept of a breakpoint?

Answer 11:

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