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

```
spam = -9

assert spam > 0, "is negative"
```

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).

```
stng = "I like and Bacon so much"

assert "eggs" in stng.lower() and "bacon" in stng.lower(), "not present"
```

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

```
assert False, “Assertion error"
```

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')
```

6. What are the five levels of logging?

Information

Warning

Error

Critical

Debug

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

```
Import logging

logging.disable(logging.CRITICAL)
```

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

Once we're done debugging, we'll end up spending a lot of time removing print() calls from our code for each log message. We might even accidentally remove some print() calls that were being used for non log messages. The nice thing about log messages is that you're free to fill your program with as many as you like, and you can always disable them later by adding a single

logging.disable(logging.CRITICAL) call. Unlike print(), the logging module makes it easy to switch between showing and hiding log messages.

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

STEP will cause the debugger to execute the next line of code and then pause again. If the next line of code is a function call, the debugger will "step into" that function and jump to the first line of code of that function.

OVER will execute the next line of code, similar to the STEP button. However, if the next line of code is a function call, the OVER button will "step over" the code in the function.

OUT will cause the debugger to execute lines of code at full speed until it returns from the current function.

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

It will stop at next break point or till it finish run or next exception stage

11. What is the concept of a breakpoint?

It will stop running at that point so that developer can check the values or status of execution for debugging issue.