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

```python

assert spam >= 0, "spam should not be 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.

```python

assert eggs.lower() != bacon.lower(), "eggs and bacon should not be the same (case-insensitive)"

```

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

```python

assert False, "This assertion always triggers an AssertionError"

```

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

```python

import logging

logging.basicConfig(level=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`?

```python

import logging

logging.basicConfig(filename='programLog.txt', level=logging.DEBUG)

```

### 6. What are the five levels of logging?

The five levels of logging are:

1. DEBUG

2. INFO

3. WARNING

4. ERROR

5. CRITICAL

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

```python

logging.disable(logging.CRITICAL)

```

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

Using logging messages is better than using `print()` because:

- \*\*Granular control\*\*: Logging allows setting different severity levels (DEBUG, INFO, WARNING, ERROR, CRITICAL) which helps in filtering messages.

- \*\*Configurability\*\*: Logging can be configured to output to different destinations, such as files, consoles, or remote servers, and can be formatted consistently.

- \*\*Disabling\*\*: Logging can be easily enabled or disabled in different parts of the code without removing the logging statements.

- \*\*Thread safety\*\*: The logging module is thread-safe, making it suitable for multi-threaded applications.

- \*\*Management\*\*: It provides better control over how messages are handled and can be integrated with external monitoring tools.

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

- \*\*Step Over\*\*: Executes the current line of code and then pauses at the next line. If the current line contains a function call, it executes the entire function call and pauses at the next line in the current function.

- \*\*Step In\*\*: Executes the current line of code and if the line contains a function call, it pauses at the first line inside the called function.

- \*\*Step Out\*\*: Continues executing the remaining lines of the current function and pauses when the function returns to the caller.

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

The debugger will stop when:

- It encounters the next breakpoint.

- An exception is raised (if the debugger is configured to stop on exceptions).

- The program terminates.

### 11. What is the concept of a breakpoint?

A breakpoint is a designated stopping point set in the code where the debugger will pause execution. This allows developers to inspect the current state of the program, including variable values, the call stack, and control flow, which helps in debugging and understanding the program’s behavior.