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?

ANSWER

1. To throw an `AssertionError` if the variable `spam` is a negative integer, you can use the `assert` statement like this:

```python

assert spam >= 0, "spam cannot be a negative integer"

```

2. To trigger an `AssertionError` if `eggs` and `bacon` contain strings that are the same as each other (case-insensitive), you can use the `assert` statement with a case-insensitive comparison:

```python

assert eggs.lower() != bacon.lower(), "eggs and bacon cannot have the same value (case-insensitive)"

```

3. To throw an `AssertionError` every time, you can use a simple `assert` statement without a condition, like this:

```python

assert False, "This assertion always raises an error"

```

4. To use `logging.debug()`, you need to have the following two lines present in your software:

```python

import logging

logging.basicConfig(level=logging.DEBUG)

```

5. To have `logging.debug()` send a logging message to a file named `programLog.txt`, you can configure the `logging` module as follows:

```python

import logging

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

```

6. The five levels of logging in Python, in increasing order of severity, are:

- DEBUG

- INFO

- WARNING

- ERROR

- CRITICAL

7. To disable all logging messages, you can set the logging level to a level higher than the most severe level of logging. For example:

```python

import logging

logging.basicConfig(level=logging.CRITICAL + 1)

```

8. Using logging messages is better than using `print()` for several reasons:

- Logging can be configured to log messages to different outputs (e.g., console, files) at different levels.

- Logging messages can include timestamps, log levels, and other useful information.

- You can control the verbosity of logging using different log levels, making it easier to debug or troubleshoot issues.

- Logging can be globally configured, allowing you to enable or disable logging easily.

9. The differences between the Step Over, Step In, and Step Out buttons in the debugger are as follows:

- Step Over: Executes the current line of code and stops at the next line in the current function, even if that line contains a function call.

- Step In: If the current line contains a function call, it enters that function and stops at the first line of code inside the called function.

- Step Out: Continues execution until the current function returns, and then stops at the line where the function was called.

10. After clicking Continue in the debugger, it will continue executing the program without interruption until it reaches the next breakpoint, an exception is raised, or the program finishes.

11. The concept of a breakpoint is a designated point in your code where the debugger will temporarily pause the program's execution. It allows you to inspect variables, step through code, and troubleshoot issues in a controlled manner. Breakpoints are set at specific lines in your code, and when the program execution reaches a breakpoint, it pauses, allowing you to interact with the debugger.