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

Ans-Assert statement to throw an AssertionError if the variable spam is a negative integer:

assert spam >= 0, "Variable spam should be a non-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).**

Ans-Assert statement to trigger an AssertionError if eggs and bacon contain case-insensitive matching strings:

assert eggs.lower() != bacon.lower(), 'eggs and bacon strings must not be the same (case-insensitive)'

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

Ans-Assert statement that throws an AssertionError every time:

assert False, 'This assertion always fails'

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

Ans-The two lines that must be present to call logging.debug():

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?**

Ans-Two lines to send logging messages to a file named programLog.txt:

import logging

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

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

Ans-The five levels of logging (in increasing order of severity) are:

* DEBUG
* INFO
* WARNING
* ERROR
* CRITICAL

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

Ans-Line of code 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?**

Ans-Advantages of using logging messages over print():

* Logging messages can be easily turned on or off without modifying the code, allowing for more flexible debugging.
* Logging messages can be filtered based on severity levels, allowing you to control the amount of information displayed.
* Logging messages can be directed to different outputs (e.g., console, file) without modifying the code, making it easier to manage the logging output.

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

Ans-Differences between Step Over, Step In, and Step Out buttons in the debugger:

* Step Over: Executes the current line of code and moves to the next line in the same function, without stepping into any function calls within the current line.
* Step In: If the current line contains a function call, it will step into that function and move to the first line of the called function.
* Step Out: Executes the remaining lines of the current function and returns to the line of code that called the current function.

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

Ans-After clicking Continue, the debugger will stop when it reaches the next breakpoint or when the program execution is completed.

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

Ans-A breakpoint is a designated point in the code where the debugger pauses program execution to allow the developer to examine the program's state, variables, and perform step-by-step debugging. It is used to identify and diagnose issues in the code by allowing the developer to interactively analyse the program's behaviour at specific points.