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

Ans:-

assert spam >= 0, "spam cannot 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 (that is, 'hello' and 'hello' are considered the same, and 'goodbye' and 'GOODbye' are also considered the same).

Ans:-

assert eggs.lower() != bacon.lower(), "eggs and bacon cannot be the same string (ignoring case)"

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

Ans:-

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:-

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:-

import logging

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

6. What are the five levels of logging?

Ans:- DEBUG: Detailed information, typically of interest only when diagnosing problems.

INFO: General information about what's happening in the program.

WARNING: An indication that something unexpected or potentially problematic has happened, or indicative of some problem in the near future (e.g. 'disk space low').

ERROR: An indication that an error has occurred in the program, which may or may not prevent the program from continuing to run.

CRITICAL: A very severe error, indicating that the program is unable to continue running.

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

Ans:-

logging.disable(logging.CRITICAL)

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

Ans:-

Using logging messages is better than using print() because it allows you to control the severity level, output destination, formatting, and performance of your program's messages in a more flexible and powerful way.

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

**Ans:-**

**Step Over:** The Step Over button allows you to execute the current line of code and then move on to the next line, without stepping into any functions or methods that might be called on that line. If the current line contains a function or method call, the entire call is executed and the result is returned to the calling line, without stepping into the called function or method.

**Step In:** The Step In button allows you to step into the next function or method call on the current line of code, if there is one. This is useful when you want to debug a function or method in detail, by stepping through each line of code inside it. If there are no function or method calls on the current line, Step In behaves like Step Over and moves to the next line of code.

**Step Out:** The Step Out button allows you to step out of the current function or method, and return to the line of code that called it. This is useful when you have stepped into a function or method using Step In and want to return to the caller without stepping through all the remaining lines of code inside the function or method.

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

Ans:-

it will resume the normal execution of your program until it encounters either of the following events:

A breakpoint: If you have set any breakpoints in your code, the debugger will stop at the next breakpoint it encounters after you click "Continue". This allows you to pause the execution of your program at specific points and inspect the values of variables and other program state.

An exception: If your program raises an exception while running, the debugger will stop at the line of code where the exception occurred. This allows you to inspect the traceback and identify the source of the error.

11. What is the concept of a breakpoint?

Ans:-

In software development, a breakpoint is a debugging tool that allows you to pause the execution of your program at a specific line of code. When you set a breakpoint, the debugger will stop your program at that line and allow you to inspect the values of variables and other program state at that point in time.