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

Ans:

spam = -5

assert spam >= 0, " 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:

eggs = "HEllo"

bacon = "hello"

assert eggs.lower() != bacon.lower(), 'AssertionError'

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

Ans: assert False, 'AssertionError'

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

Ans:

import logging

logging.basicConfig(filename='log.log', 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?

Ans:

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?

Ans:

- **DEBUG: Detailed information, typically useful for debugging.**
- **INFO: General information about the application's execution.**
- **WARNING: Indication of a potential issue or a non-critical problem.**

- **ERROR: Designates errors that caused the program to fail to perform a function.**
- **CRITICAL: Critical errors that may lead to a complete failure of the application.**

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: It is not possible to print all error messages of the code is in production environment. So if we use logging then its possible to go through all the logs from a saved file.

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

Ans: When you click the Step Over button, the debugger will execute the current line of code and move to the next line in the same function. If the current line contains a function call, the debugger will not step into the called function. Instead, it will execute the entire function and move to the next line after the function call.

The Step In button is used to step into a function call. When you click Step In, the debugger will move the execution to the first line of the function being called (if available) and allow you to step through each line of the called function.

The Step Out button is used to complete the execution of the current function and return to the caller. When you click Step Out, the debugger will continue executing the code until it reaches the end of the current function or encounters a return statement.

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

Ans: It will stop when the code has ended or an exception is raised or a breakpoint is reached.

11. What is the concept of a breakpoint?

Ans: A breakpoint is a designated point in your code where the debugger pauses the execution of the program.