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

**Ans:-**  assert spam >= 0, 'spam should be a non-negative integer'

This ‘assert’ statement checks that ‘spam’ is greater than or equal to zero, and if it is not, it raises an ‘**assertionerror’** with the message “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 eggs.lower() != bacon.lower(), 'eggs and bacon should have different string values'

This ‘asert’ statement checks that the lowercase version of the strings stored in ‘eggs’ and ‘bacon’ are not equal to each other. If they are equal, it raises an **‘Assertionerror’** with the message “eggs and bacon should have different string values.

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

**Ans:-**  assert False, 'This assertion always fails'

This ‘assert’ statement simply checks that the expression ‘False’ is true, which ofcourse it is not. When this assertion is triggered, it will raise an **‘Assertionerror’** with the message “This assertion always fails”.

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

**Ans:-** To use “logging.debug()” method in python, you must include the following two lines at the beginning of your program:-

import logging

logging.basicConfig(level=logging.DEBUG)

The first line imports the logging module, while the second line configures the logging system. After writing these two lines, you can now use “logging.debug()” to write the debug level log message.

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:-** To use the “logging.debug()” method in python and write log messages to a file named “programLog.txt”, you must include the following two lines at the beginning of your program.

import logging

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

The first line imports the “logging” module, while the second line configures the logging system. After writing these two lines in place, you can now use “logging.debug()” to write debug level log messages, and they will be written to the file named “programLog.txt”.

6. What are the five levels of logging?

**Ans:-** There are five standard levels of logging available in python:-

1. DEBUG :- Detailed information, typically of interest only when diagnosing the problem.
2. INFO :- General information about the progress of the application.
3. WARNING :- An indication that something unexpected happened, or indicative of the problem in the near future. The application is still running as expected.
4. ERROR :- Due to a more serious problem, The application has not been able to perform some function.
5. CRITICAL :- A very serious error, indicating that the program itself may be unable to continue running.

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

**Ans:-**  To disable all logging messages in python, you can add the following line of code at the beginning of your program,

logging.disable(logging.CRITICAL)

This line sets the logging level to “critical”, which is the highest level of severity. As a result, all log messages with a level of ‘critical’, ‘error’, ‘warning’, ‘info’, and ‘debug’ will be disabled, and will not be output by the logging system.

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

**Ans:-** Using logging messages is generally better than using ‘print()’ to display the same message in python for several reasons:-

1. Flexibility :- With logging, you can easily change the way the messages are handled, such as where they are sent, the format they are in, and there severity level without modifying the code.
2. Granularity ;- Logging allows you to add different levels of granularity to messages, such as debug, info, warning, error and critical. With these different levels, yu can filter the message based on their severity level and only display the ones that are important to you.
3. Efficiency :- Logging is generally more efficient than using ‘print()’. When we use ‘print()’, the messages are printed directly to the console, which can slow down your program. With logging, you can configure the messages, to be written to a file or sent over the network, which can improve performance.
4. Scalability :- When you have a large project with many files, it can be difficult to keep track of all the ‘print()’ statements. Logging provides a centralized wat to manage all the messages in your project, which makes it easier to debug issues and analyze your program’s behaviour.
5. Best practices :- Using logging is considered as the best practice in software development. It follows the principal of separation of concerns by seprating the logging logic from the business logic, which makes your code modular and easy to maintain.

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

**Ans:-** The step over, step in and step out buttons in a python debugger are used for code execution control during debugging. Here are the differences between these buttons;-

1. **Step Over** :- This button executes the current line of code and moves to the next line in the file, without entering any function calls on the current line. If the current line contains a function call, the entire function will be executed, but the debugger will not stop at each line of the function. The execution will return to the caller after the function has completed.
2. **Step in** :- This button executes the current line of code and enters any function call in the current line. The debugger will stop at the first line of the called function, allowing you to step through the function line by line.
3. **Step Out** :- This button continues the execution of the current function until it returns to the caller. The debugger will stop at the first line after the function call in the caller.

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

**Ans;-** Once the debugger is resumed by clicking on the continue button, it will continue executing the program until either it reaches the end of the program, or it encounters another breakpoint(if one is set).

11. What is the concept of a breakpoint?

**Ans:-**  In programming, A breakpoint is a marker that is set within the code of a program to pause the execution of the program at a specific point during runtime.

When the program encounters a breakpoint during execution, it will pause, and the developer can then use the debugger to examine the state of the program at that point, such as the value of variables, the call stack, and any other relevant information.