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

Ans1

assert spam >= 0, "spam non-neg int"

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).

Ans2

assert eggs.lower() != bacon.lower(), "eggs and bacon should not be the same (case insensitive)"

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

Ans3

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()?

Ans4

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?

Ans5

import logging

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

logging.debug('debug message')

6. What are the five levels of logging?

Ans6

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

INFO: Confirmation that things are working as expected.

WARNING: An indication that something unexpected or potentially problematic has occurred, or indicative of

some problem in the near future (e.g., ‘disk space low’). The software is still working as expected.

ERROR: Due to a more serious problem, the software has not been able to perform some function.

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?

Ans7

logging.disable(logging.CRITICAL)

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

Ans8

Granular control over log messages: Logging provides a way to control the verbosity and severity of log messages. You can filter log messages based on their level, such as only logging messages with a level of WARNING or higher.

Flexible output destinations: With logging, you can direct log messages to various output destinations such as the console, a file, or a network stream. This makes it easier to collect and analyze log messages from different sources.

Easy integration with third-party libraries: Many third-party libraries support logging and can send their log messages to the same logging framework.

Improved performance: When using print(), every message is sent to stdout or stderr, which can be slow and impact the performance.

Dynamic log level changes: You can dynamically change the log level of your application without modifying the source code or restarting the application.

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

Ans9

|  |  |  |
| --- | --- | --- |
| Step over | Step in | Step out |
| The Step Over button executes the current line of code and then moves the debugger's current execution point to the next line of code. If the current line of code contains a function call, the entire function is executed without stepping into it, and the debugger's current execution point is moved to the next line of code after the function call returns. | The Step In button executes the current line of code and then moves the debugger's current execution point to the first line of code inside any function that is called on the current line of code. This allows to step inside a function and debug it line by line. | The Step Out button is used when debugging inside a function and want to move the debugger's current execution point outside of the current function. When click Step Out, the current function is executed to completion, and the debugger's current execution point is moved to the line of code immediately following the function call that led to the current execution point. |

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

Ans10

clicking Continue in the debugger will cause the debugger to continue executing the program until it reaches a breakpoint, an exception, or the end of the program.

A breakpoint: the debugger will stop at the first line of code after the breakpoint.

An exception: If an unhandled exception is raised during program execution, the debugger will stop at the line of code where the exception occurred.

The end of the program: If the program completes execution without encountering a breakpoint or an exception, the debugger will stop at the end of the program.

11. What is the concept of a breakpoint?

Ans11

Breakpoints are an essential tool for debugging software, allowing developers to gain insight into the program's state and behavior at specific points in time, and ultimately leading to more efficient and effective bug fixing.