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

try:

spam = int(input("Enter an integer value for spam: "))

except ValueError:

print("Invalid input; please enter an integer.")

raise

# Assertion to ensure 'spam' is not negative

assert spam >= 0, "spam must not be a negative integer"

1. 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).

eggs = "goodbye"

bacon = "GOODbye"

# Assert statement to check if eggs and bacon are the same regardless of case

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

1. Create an assert statement that throws an AssertionError every time.
   1. assert False, "This assertion always fails"
2. What are the two lines that must be present in your software in order to call logging.debug()?
   1. import logging
   2. logging.basicConfig(level=logging.DEBUG)
3. 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?
   1. import logging
   2. logging.basicConfig(filename='programLog.txt', level=logging.DEBUG, filemode='w', format='%(asctime)s - %(levelname)s - %(message)s')
4. What are the five levels of logging?
   * 1. DEBUG: This is the lowest level, used for small details. Typically, these messages are of interest only when diagnosing problems.
        1. Method: logging.debug()
        2. Numeric value: 10
     2. INFO: This level is used for routine information, such as confirmation that things are working as expected.
        1. Method: logging.info()
        2. Numeric value: 20
     3. WARNING: This level indicates that something unexpected happened, or indicative of some problem in the near future (e.g., 'disk space low'). The software is still working as expected.
        1. Method: logging.warning()
        2. Numeric value: 30
     4. ERROR: This level indicates a more serious problem, due to which the software has not been able to perform some function.
        1. Method: logging.error()
        2. Numeric value: 40
     5. CRITICAL: This is the highest level, indicating a serious error, indicating that the program itself may be unable to continue running.
        1. Method: logging.critical()
        2. Numeric value: 50
5. What line of code would you add to your software to disable all logging messages?
   1. logging.disable(logging.CRITICAL + 1)
6. Why is using logging messages better than using print() to display the same message?
7. Severity Levels: Logging allows you to specify the severity of messages (debug, info, warning, error, critical), which helps in filtering and handling outputs based on the importance or urgency.
8. Flexibility: You can direct logging output to different destinations, like files, streams, or external systems, without changing the logging calls in your code. print() only outputs to the console.
9. Configuration: Logging behavior can be configured globally in an application, including setting different output levels and formats for different parts of the application. This configuration can also be changed without modifying the code.
10. Performance: Logging can be disabled with minimal overhead, and it's possible to automatically disable lower priority messages in production environments.
11. Integration: The logging module works seamlessly across modules and libraries, making it easier to maintain consistent logging practices across a complex application.
12. What are the differences between the Step Over, Step In, and Step Out buttons in the debugger?
    1. Step In: This command in a debugger makes the debugger enter into functions or methods. If the line of code contains a function call, Step In will pause execution at the first line of that function.
    2. Step Over: This command also advances the execution of code like Step In, but it treats function calls as a single step. This means it will execute the function in its entirety but pause once it returns to the function from where it was called.
    3. Step Out: This command is used when you are already inside a function and want to continue executing the rest of this function automatically, pausing only once the execution returns to the calling function.

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

* When you click "Continue" (often labeled as "Resume" in some debuggers) in a debugger, the debugger will continue executing the program without pausing until it encounters the next breakpoint. If there are no further breakpoints set, the debugger will continue until the program terminates or an exception is encountered that isn't otherwise handled.

1. What is the concept of a breakpoint?
   1. A breakpoint is a tool used in debugging that allows a developer to temporarily halt the execution of a program at a specific point. This is useful for examining the state of the program, including the values of variables, the program’s execution path, and the call stack at a certain moment in time. Breakpoints can be set on specific lines of code. When the execution reaches a line with a breakpoint, the debugger pauses, allowing you to inspect and manipulate the internal state of the application to diagnose issues and understand behavio