**Python Assignment 11**

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

assert spam>=0,’Asserton Error’

**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).**

assert eggs’.lower()!=bacon.lower(),’ AssertionError Eggs and bacon can not same’.

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

assert False,’ AssertionError Eggs and bacon can not same’.

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

To call **logging.debug()** in your Python code, you must have these two lines at the beginning of your code:

import logging  
logging.basicConfig(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?**

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?**

The five levels of logging, in increasing order of severity, are:

1. **DEBUG**: Detailed information for debugging purposes
2. **INFO**: General information about the program's operation
3. **WARNING**: An indication that something unexpected or undesirable happened, but the program can still continue running
4. **ERROR**: An indication that something has gone wrong, and the program cannot perform a specific function
5. **CRITICAL**: An indication that a critical error has occurred, and the program may not be able to continue running

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

logging.disable(logging.CRITICAL)

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

Using logging messages is generally better than using **print()** to display the same message for several reasons:

* **Control over output**: With logging, you can control where messages are sent (console, file, email, etc.), the format of the messages, and the severity of the messages that are output. This makes it easier to filter messages, and to ensure that only the most important messages are logged.
* **Granular control over logging levels**: Logging allows for granular control over which types of messages are displayed, from **DEBUG** messages for detailed information, to **CRITICAL** messages for severe errors. This is more flexible than simply using **print()** statements, which are either displayed or not.
* **Performance**: While **print()** statements are generally fast, they can still slow down your code if they are executed frequently or in performance-critical sections of your code. Logging statements can be disabled or set to a higher severity level if performance becomes a concern.
* **Maintainability**: Using logging messages can make your code more maintainable by providing a clear record of what the code is doing and when, making it easier to debug and troubleshoot issues that arise. Logging statements can also provide useful information for future development and testing.

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

The Step Over, Step In, and Step Out buttons are debugging tools in an IDE that allow you to control the flow of your code while it is running in debug mode. The differences between these buttons are:

* Step Over: This button allows you to execute the current line of code and then move on to the next line. If the current line of code contains a function call, the function is executed, but the debugger does not step into the function.
* Step In: This button allows you to execute the current line of code and then move into any function calls on that line. If the current line of code does not contain a function call, Step In behaves like Step Over.
* Step Out: This button allows you to finish executing the current function and return to the line that called it. If the current line of code is not inside a function, Step Out has no effect.

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

After clicking the Continue button, the debugger will stop either when it reaches the next breakpoint (if any), encounters an unhandled exception, or reaches the end of the program.

**11. What is the concept of a breakpoint?**

In the context of debugging, a breakpoint is a point in the code where the debugger will pause the program's execution, allowing the developer to inspect the current state of the program, step through the code line-by-line, modify variables and observe their effect, and diagnose issues. A breakpoint can be set by the developer at any point in the code where they want to pause the program's execution. Once the program hits a breakpoint, the developer can use the debugger's features to investigate the code and determine what is happening