ASSIGNMENT - 11(PYTHON)

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

You can use the assert statement to check if a condition is true. If the condition evaluates to False, it raises an AssertionError. Here's how you can create an assert statement to check if the variable spam is a negative integer:

```
# Assuming spam is the variable to be checked
assert spam >= 0, "spam should be a non-negative integer"
```

This statement will raise an AssertionError with the message "spam should be a non-negative integer" if spam is a negative integer. If spam is non-negative, the assert statement will not raise an error, and the program will continue executing

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).

```
# Assuming eggs and bacon are the variables to be checked
assert eggs.lower() != bacon.lower(), "eggs and bacon should not have the same value (case insensitive)"
```

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

```
assert False, "This assert statement always triggers an AssertionError"
```

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

```
import logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')
```

The first line imports the logging module, and the second line configures the logging system to output messages with a severity level of DEBUG or higher. It also specifies the format of the log messages. After these lines, you can call logging.debug() to log debug messages.

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')
```

The first line imports the logging module, and the second line configures the logging system to output messages with a severity level of DEBUG or higher to the file "programLog.txt". It also specifies the format of the log messages. After these lines, you can call logging.debug() to log debug messages, and they will be appended to the "programLog.txt" file.

6. What are the five levels of logging?

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

DEBUG: Detailed information, typically useful only for diagnosing problems. These messages are primarily for developers and are usually disabled in production environments.

INFO: Confirmation that things are working as expected. This level of logging is used to record information on general events in the program.

WARNING: Indication that something unexpected happened, or an indication of a potential problem. The software is still working as expected.

ERROR: Designates error conditions that may be handled by the application. These are usually logged when something has gone wrong, but the application can still continue to run.

CRITICAL: Indicates a serious error where the program may not be able to continue running. This level of logging is typically reserved for fatal errors.

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

To disable all logging messages in your software, you can set the logging level to a higher level than CRITICAL, such as logging.CRITICAL + 1. Here's the line of code you would add:

```
logging.disable(logging.CRITICAL + 1)
```

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

Using logging messages provides better control, flexibility, and consistency in managing and analyzing your application's output compared to using print() statements.

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

Step Over: Executes the current line and moves to the next line in the current context.
Step In: Enters the function called by the current line, allowing you to step through its code.
Step Out: Completes the execution of the current function and returns to the caller context.

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

Clicking "Continue" allows the program to run uninterrupted until it encounters a breakpoint, raises an exception, or completes execution.

11. What is the concept of a breakpoint?

A breakpoint is a debugging tool used in programming to temporarily pause the execution of a program at a specific line of code. When a breakpoint is set, the debugger interrupts the program's execution when it reaches that line, allowing the developer to inspect the program's state, variables, and environment at that point in time.