PYTHON ASSIGNMENT – 11

1. Create an assert statement that throws an AssertionError if the variable spam is a 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, &#39;hello&#39; and &#39;hello&#39; are

considered the same, and &#39;goodbye&#39; and &#39;GOODbye&#39; are also considered the same).

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

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

6. What are the five levels of logging?

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

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

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

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

11. What is the concept of a breakpoint?

SOLUTIONS

1.Assert Statement for Negative Integer:

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

python

2. Assert Statement for Strings with Different Cases:

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

3.Always Raising AssertionError:

assert False, "This assertion always raises an AssertionError"

4.Lines for calling logging.debug():

import logging

logging.basicConfig(level=logging.DEBUG)

5.Lines for logging.debug() to File:

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

6. Five Levels of Logging:

- **DEBUG**
- **INFO**
- **WARNING**
- **ERROR**
- **CRITICAL**

7. Disable All Logging Messages:

```
logging.disable(logging.CRITICAL)
```

8. Advantages of Logging over print():

- Logging allows you to control the verbosity of messages through different levels.
- It provides a flexible way to redirect messages to different outputs (console, file, etc.).
- You can selectively enable or disable logging based on the severity of messages.

9. Debugger Buttons:

a. **Step Over:** Executes the current line and stops at the next line in the current function.
b. **Step In:** If the current line contains a function call, it will enter the function and stop at the first line of that function.
c. **Step Out:** Executes the remaining lines of the current function and stops when the function returns.

10. Debugger Stop After Continue:

- The debugger will stop when it encounters a breakpoint, an unhandled exception, or when explicitly paused by the user.

11. Concept of a Breakpoint:

- A breakpoint is a point in your code where the debugger will pause its execution, allowing you to inspect variables, step through code, and analyze program state. Breakpoints are useful for debugging specific parts of your code without stopping at every line.