Assignment_11

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

```
spam = -1
assert spam >= 0
```

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).

```
eggs = "hello"
bacon = "HellO"
assert eggs.lower() != bacon.lower()
```

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

```
assert False
```

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)
```

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.DEBU
G)
```

6. What are the five levels of logging?

- DEBUG: This level is used to log information that is useful for debugging.
- INFO: This level is used to log general information about the program's execution.
- WARNING: This level is used to log warnings about potential problems.
- ERROR: This level is used to log errors that have occurred.
- CRITICAL: This level is used to log critical errors that have caused the program to terminate.

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

`logging.disable(logging.CRITICAL)` This line of code sets the logging level to CRITICAL. This means that no logging messages will be logged, not even critical errors.

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

There are several reasons why using logging messages is better than using print() to display the same message.

- **Logging messages are more structured**: When you use print(), the output is just a string. This can be difficult to parse and understand, especially if you have a lot of logging messages. When you use logging, the output is a structured object that contains information about the message, such as the level, the source, and the timestamp. This makes it easier to filter, search, and analyze the logging messages.
- **Logging messages can be redirected**: By default, logging messages are printed to the console. However, you can also redirect logging messages to a file, a database, or an email address. This allows you to keep a record of your logging messages, even if your program crashes or terminates unexpectedly.
- **Logging messages can be filtered**: You can filter logging messages based on their level, source, or other criteria. This allows you to focus on the logging messages that are most relevant to you.
- **Logging messages can be formatted**: You can format logging messages to make them more readable and easier to understand. For example, you can format the messages with timestamps, colors, or other special characters.

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 in the debugger are used to control the flow of execution of your program.

- **Step Over** executes the next line of code, but it does not step into any function calls. This means that if the next line of code is a function call, the debugger will not enter the function. Instead, it will execute the next line of code after the function call.
- **Step In** executes the next line of code, and it steps into any function calls. This means that if the next line of code is a function call, the debugger will enter the function and execute the first line of code in the function.
- **Step Out** executes the current function until it returns. This means that if you are currently in a function, the Step Out button will execute the rest of the code in the function, and then it will return to the line of code that called the function.

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

> After we click Continue, the debugger will stop when it reaches the end of the program, or when it encounters a breakpoint. A breakpoint is a line of code that you can set in your program to tell the debugger to stop execution.

11. What is the concept of a breakpoint?

> A ***breakpoint*** is a line of code that you can set in your program to tell the debugger to stop execution. When the debugger reaches a breakpoint, it will pause execution and you will be able to inspect the values of variables and the state of the program.