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

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

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[1], line 2
      1 spam =-1
----> 2 assert spam >= 0

AssertionError:
```

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).

```
def eggs_bacon(eggs,bacon):
    assert eggs.lower() != bacon.lower()
```

```
eggs_bacon('hello','goodbye')
```

```
eggs_bacon('hello','hello')
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[3], line 1
----> 1 eggs_bacon('hello','hello')

Cell In[2], line 2, in eggs_bacon(eggs, bacon)
      1 def eggs_bacon(eggs,bacon):
----> 2     assert eggs.lower() != bacon.lower()

AssertionError:
```

```
eggs_bacon('goodbye','GOODbye')
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[4], line 1
----> 1 eggs_bacon('goodbye','GOODbye')

Cell In[2], line 2, in eggs_bacon(eggs, bacon)
      1 def eggs_bacon(eggs,bacon):
----> 2     assert eggs.lower() != bacon.lower()

AssertionError:
```

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

```
assert False
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[7], line 1
----> 1 assert False

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)
```

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)
```

6. What are the five levels of logging?

The five levels of logging are:

1. **DEBUG**: Detailed information, typically useful for debugging purposes.
2. **INFO**: General information about the program's operation, used to confirm that things are working as expected.
3. **WARNING**: Indicates a potential issue or situation that might lead to problems in the future, but the program can still continue running.
4. **ERROR**: Indicates an error or exceptional condition that should be addressed. The program may not be able to continue normally.
5. **CRITICAL**: Indicates a critical error or failure that usually leads to the termination of the program.

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?

Logging is better than print because of following :

1.Logging supports different severity levels (debug, info, warning, error, critical), allowing you to categorize and filter messages based on importance.

2. Logging can include additional context, metadata, and structured data, enhancing the quality of debugging and monitoring.

3. Logging allows dynamic configuration of output targets (files, console, network), making it suitable for both development and production environments.

4. Using logging promotes consistent message formatting and ensures proper separation of concerns.

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, treating functions as a single step. Step In navigates into the current function call to debug its internal code. Step Out continues execution until the current function is complete and returns to its caller.

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

Debugger will stop only when a predetermined stopping condition is met such as encountering a breakpoint, error or completion of the program.

11. What is the concept of a breakpoint?

A breakpoint is a debugging tool that lets programmers pause program execution at a chosen line of code. This enables inspecting variables, tracing code step by step, and pinpointing errors. It aids in understanding program behavior, identifying issues, and testing specific scenarios for effective debugging