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

Answer:

assert spam >= 0, “spam cannot be a negative integer”

>>> AssertionError: spam cannot be 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, 'hello' and 'hello' are considered the same, and 'goodbye' and 'GOODbye' are also considered the same).

Answer:

eggs:str = "hello"

bacon:str = "HellO"

assert not eggs.lower() == bacon.lower(), "Both are same"

print("Both are not same")

>>> AssertionError: equal

--------------------------------------------------------------------------------------------------------------------------------------

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

Answer:

assert False, "Throws error always"

print("You will never get this printed")

>>> AssertionError: Throws error always

--------------------------------------------------------------------------------------------------------------------------------------

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

Answer:

import logging

logging.basicConfig(filename=”example.log”, level=logging.DEBUG, format=’%(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?

Answer:

import logging

logging.basicConfig(filename=” programLog.txt”, level=logging.DEBUG, format=’%(levelname)s | %(message)s’)

--------------------------------------------------------------------------------------------------------------------------------------

6. What are the five levels of logging?

Answer:

1. DEBUG
2. INFO
3. WARNING
4. ERROR
5. CRITICAL

--------------------------------------------------------------------------------------------------------------------------------------

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

Answer: Logging.disable(level=CRITICAL)

--------------------------------------------------------------------------------------------------------------------------------------

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

Answer:

Logging records the events and errors during the execution whereas print is used to display information in the console. Logging provides some features like Log levels, custom one time formatting, filtering, etc whereas print does not have any such features. Logging is mainly used for production environments whereas print is used for debugging.

--------------------------------------------------------------------------------------------------------------------------------------

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

Answer:

Step Over – Executes a scope in one go and moves to the next scope

Step In – Executes the code line-by-line

Step Out – Executes the remaining lines of a code in current scope and moves to higher scope where the scope is returned.

--------------------------------------------------------------------------------------------------------------------------------------

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

Answer:

Continue execution stops only when a breakpoint is encountered else it will execute the whole code.

--------------------------------------------------------------------------------------------------------------------------------------

11. What is the concept of a breakpoint?

Answer:

Breakpoints are important techniques in debugging a project. It pauses the debugger execution wherever it is set, allowing us to check the values of the variable at that point, state of the application when it is paused, etc.

--------------------------------------------------------------------------------------------------------------------------------------