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

Answer> assert spam >= 10, 'The spam variable is less than 10.'

1. 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> assert eggs.lower() != bacon.lower(), 'The eggs and bacon variables are the same!' or assert eggs.upper() != bacon.upper(), 'The eggs and bacon variables are the same!'

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

Answer> assert False, 'This assertion always triggers.'

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

Answer> To call logging.debug(), these two lines at the start of Program

import logging

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

1. 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> To be able to send logging messages to a file named programLog.txt with logging.debug(), these two lines must have at the start of your program:

import logging

logging.basicConfig(filename='programLog.txt', level=logging.DEBUG, format=' %(asctime)s - %(levelname)s - %(message)s')

1. What are the five levels of logging?

Answer> DEBUG, INFO, WARNING, ERROR, and CRITICAL

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

Answer> logging.disable(logging.CRITICAL)

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

Answer> Logging messages provide a trail that can help us figure out when things started to go wrong. Writing the logging messages to a file will keep screen clear and store the messages so we can read them after running the program.

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

Answer> Step In - Step In button will cause the debugger to execute the next line

of code and then pause again. If the next line of code is a function call, the

debugger will “step into” that function and jump to the first line of code of

that function

Step Over - Step Over button will execute the next line of code, similar to

the Step In button. if the next line of code is a function call, the

Step Over button will “step over” the code in the function. The function’s

code will be executed at full speed, and the debugger will pause as soon as

the function call returns.

Step Out - Step Out button will cause the debugger to execute lines of

code at full speed until it returns from the current function

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

Answer> After you click Go, the debugger will stop when it has reached the end of the program or a line with a breakpoint.

1. What is the concept of a breakpoint?

Answer> A breakpoint is a setting on a line of code that causes the debugger to pause when the program execution reaches the line.