1. Write an assert statement that triggers an AssertionError if the variable spam is an integer less than 10. Answer: assert(spam >= 10, 'The spam variable is less than 10.')

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: 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!')

3. Write an assert statement that always triggers an AssertionError. Answer: assert(False, 'This assertion always triggers.')

4. What are the two lines that your program must have in order to be able to call logging.debug()? Answer: To be able to call logging.debug(), you must have these two lines at the start of your program: import logging logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(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=' %(asctime)s - %(levelname)s - %(message)s')

6. What are the ﬁve logging levels? Answer: DEBUG, INFO, WARNING, ERROR, and CRITICAL

7. What line of code can you add to disable all logging messages in your program? Answer: logging.disable(logging.CRITICAL)

8. Why is using logging messages better than using print() to display the same message? Answer: You can disable logging messages without removing the logging function calls. You can selectively disable lower-level logging messages. You can create logging messages. Logging messages provides a timestamp.

9. What are the differences between the Step, Over, and Out buttons in the Debug Control window? Answer: The Step button will move the debugger into a function call. The Over button will quickly execute the function call without stepping into it. The Out button will quickly execute the rest of the code until it steps out of the function it currently is in.

10. After you click Go in the Debug Control window, when will the debugger stop? Answer: click Go, the debugger will stop when it has reached the end of the program or a line with a breakpoint.

11.What is a breakpoint? Answer: breakpoint is a setting on a line of code that causes the debugger to pause when the program execution reaches the line

12. How do you set a breakpoint on a line of code in IDLE? Answer: set a breakpoint in IDLE, right-click the line and select Set Breakpoint from the context menu.