1. Create an assert statement that throws an AssertionError if the variable spam is 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, &#39;hello&#39; and &#39;hello&#39; are
considered the same, and &#39;goodbye&#39; and &#39;GOODbye&#39; are also considered the same).
3. Create an assert statement that throws an AssertionError every time.
4. What are the two lines that must be present in your software in order to call 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?
6. What are the five levels of logging?
7. What line of code would you add to your software to disable all logging messages?
8.Why is using logging messages better than using print() to display the same message?
9. What are the differences between the Step Over, Step In, and Step Out buttons in the debugger?
10.After you click Continue, when will the debugger stop ?
11. What is the concept of a breakpoint?

1. assert spam >= 0, "spam should be a non-negative integer"
2. assert eggs.lower() != bacon.lower(), "eggs and bacon should have different strings"
3. assert False, "This assertion always fails"
4. import logging logging.basicConfig(level=logging.DEBUG)
5. import logging logging.basicConfig(filename='programLog.txt', level=logging.DEBUG)
6. The five levels of logging, in increasing order of severity, are: DEBUG, INFO, WARNING, ERROR, and CRITICAL.
7. logging.disable(logging.CRITICAL)
8. Using logging messages is better than using print() because logging messages can be easily turned on and off or redirected to different outputs without changing the code, and can include more detailed information such as the severity level and the module where the message originated.
9. Step Over runs the current line and stops at the next line, Step In goes to the next line of a function being called, and Step Out runs until the current function returns and stops at the line after the function call.
10. The debugger will stop again either when it reaches another breakpoint, encounters an unhandled exception, or the program completes successfully.
11. A breakpoint is a designated point in the code where the debugger will pause execution, allowing the developer to inspect variables and step through the code one line at a time.