|  |  |
| --- | --- |
|  | Chapter 10 Practice Questions – Debugging |
| **Q1** | **Write an assert statement that triggers an AssertionError if the variable spam is an integer less than 10** |
| A | assert spam > 10, ‘The value for spam needs to be less than 10’  assert(spam >=10, ‘The spam variable is less than 10.’) |
| **Q2** | **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)** |
| A | assert eggs.lower() =! bacon.lower(), ‘bacon and egg are not the same’  assert(eggs.lower() != bacon.lower(), ‘The eggs and bacon variables are the same!’)  Note: Assertion is always triggered on a False condition |
| **Q3** | **Write an assert statement that always triggers an AssertionError** |
| A | assert <condition>,<error message>  assert(False, ‘This assertion always triggers.’) |
| **Q4** | **What are the two lines that your program must have in order to be able to call logging.debug()?** |
| A | import logging  logging.basicConfig(level=loggin.DEBUG, format=’ %(asctime)s - %(levelname)s - %(message)s’) |
| **Q5** | **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?** |
| A | import logging  logging.basicConfig(filename=’programLog.txt’, level=logging.DEBUG, format=’%(asctime)s - %(levelname)s - %(message)s’) |
| **Q6** | **What are the five logging levels?** |
| A | * DEBUG * INFO * WARNING * ERROR * CRITICAL |
| **Q7** | **What line of code can you add to disable all logging messages in your program?** |
| A | Pass logging.disable() a logging level, and it will suppress all log messages at that level or lower e.g.  logging.disable(logging.CRITICAL) |
| **Q8** | **Why is using logging messages better than using print() to display the same message()?** |
| A | You can add as many debug messages you like throughout your program and can disable all of them later with a single line: logging.disable(logging.CRITICAL)  You can selectively disable lower-level logging messages.  Logging messages provide timestamp |
| **Q9** | **What are the differences between the Step, Over, and Out buttons in the Debug Control Window** |
| A | Upon entering the debugger (enabled by Debug>Debugger) you can:   * Step (button): Will cause the debugger to execute the next line of code and then pause again * Over (button): Will cause the debugger to execute the next line of code and then pause again except for if the next line is a function it will execute without stepping through the function code and pause when the function returns it’s value * Out (button): allows one to step out of a function if stepped into with the step button (which causes line by line execution of the function) |
| **Q10** | **After you click Go in the Debug Control window, when will the debugger stop?** |
| A | Clicking the Go button will cause the program to execute normally until it terminates or reaches a breakpoint |
| **Q11** | **What is a breakpoint?** |
| A | A breakpoint can be set on a specific line of code and forces the debugger to pause whenever the program execution reaches that line |
| **Q12** | **How do you set a breakpoint on a line of code in IDLE?** |
| A | To set a breakpoint right-click the line in the file editor and select Set Breakpoint |