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

Ans->spam = 0

assert spam < 0, 'if the variable spam is a negative integer'

print('spam is a positive number.')

1. Create an assert statement that throws an AssertionError if the variables eggs and bacon contain strings that are the same, even if their cases are different (for example, 'hello' and 'hello' are considered the same, as are 'goodbye' and 'GOODbye').

Ans-: eggs = ’hello’

Bacon = ‘goodbye’

assert eggs.lower()! =bacon.lower(), ‘eggs/bacon should not be the same!’

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

Ans-:assert True, ‘Always triggers an AssertionError.’

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

Ans-:

Import logging

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

Logging.debug(‘This is a test message.’)

1. What are the two lines that must be present in your software in order for it to log? Is it possible for logging.debug() to log a message to a file called programLog.txt?

Ans-: Import logging

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

Logging.debug(‘This is a test message.’)

1. What are the five stages of logging?

Ans-:

|  |
| --- |
| logging.debug() - variable's state and small details |
| logging.info() - general events, confirm a program is working |
| logging.warning() - potiental problem to work on in the future |
| logging.error() - record an error that caused program to fail to do something |
| logging.critical() - fatal error that has caused |

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

Ans-:

|  |  |
| --- | --- |
|  | logging.disable(logging.DEBUG) |
|  |  |
|  |  |

8.Why is it easier to use logging messages instead of print() to show the same message?

|  |  |
| --- | --- |
| Ans-: | Because with print, when your program is ready for production, you still |
|  | have to "remove" or comment it out. Verses logging message, you can toggle |
|  | the setting on/off or write to a file (send to a server). It is more flexible |
|  | especially with logging level 1-5 |

9. What is the difference between the debugger's Step Over, Step In, and Step Out buttons?

Ans-:

|  |
| --- |
| StepIn - one line execution at a time |
| Over - excecute the next line of code, but if it is a program, it will |
| complete the entire function call. |
| out - execute the lines of code until it returns from the current function. |
| (out is useful when you stepped into a function call). |

10.When will the debugger stop after you press Continue?

Ans-: Go runs until the program terminate or reaches a breakpoint set.

11. What is the concept of a breakpoint?

Ans-:

|  |
| --- |
| When you have Debugger enabled and you can right click on any lines |
| to create a breakpoint. During Go - it will stop there and await your next |
| command. |

12. In Mu, how do you place a breakpoint on a line of code?

Ans-:Run **to** a **breakpoint** in **code**

**To** set a simple **breakpoint** in your **code**, click the far left margin next **to** the **line of code** where **you** want **to** suspend execution. **You can** also select the **line** and press F9, select Debug > Toggle **Breakpoint**, or right-click and select **Breakpoint** > **Insert Breakpoint**.

|  |
| --- |
|  |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |