**ANSWERS**

**Q1. Create an assert statement that throws an AssertionError if the variable spam is a negative integer.**

**Ans.** Here's an example of an assert statement that throws an **AssertionError** if the variable **spam** is a negative integer:

assert spam >= 0, "spam should be a non-negative integer"

If the value of **spam** is a negative integer, the assert statement will raise an **AssertionError** with the specified error message.

**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).**

**Ans.** An assert statement that triggers an **AssertionError** if the variables **eggs** and **bacon** contain strings that are the same, regardless of their case, can be written as follows:

assert eggs.lower() != bacon.lower(), "eggs and bacon should not be the same"

The **lower()** method is used to convert the strings to lowercase before comparison. If the lowercase versions of **eggs** and **bacon** are the same, the assert statement will raise an **AssertionError** with the specified error message.

**Q3. Create an assert statement that throws an AssertionError every time.**

**Ans.** An assert statement that throws an **AssertionError** every time can be written like this:

‘assert False, "This assert statement always throws an error"

Since the condition **False** is always false, the assert statement will raise an **AssertionError** with the specified error message whenever it is encountered.

**Q4. What are the two lines that must be present in your software in order to call logging.debug()?**

**Ans.** In order to call **logging.debug()**, the following two lines need to be present in your software:

import logging

logging.basicConfig(level=logging.DEBUG)

The first line imports the **logging** module, and the second line configures the basic logging settings to the **DEBUG** level. This ensures that the **logging.debug()** statements will be processed and displayed according to the configured logging level.

**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?**

**Ans.** To make **logging.debug()** send a logging message to a file named **programLog.txt**, the following two lines should be added to your program:

import logging

logging.basicConfig(filename='programLog.txt', level=logging.DEBUG)

The **filename** argument in **basicConfig()** specifies the name of the file to which the logging messages will be written.

**Q6. What are the five levels of logging?**

**Ans.** The five levels of logging, in increasing order of severity, are:

* **DEBUG**: Detailed information for diagnostic purposes.
* **INFO**: General information about the program's execution.
* **WARNING**: Indication of a potential issue or unexpected behavior.
* **ERROR**: Indicates an error that caused a specific operation to fail.
* **CRITICAL**: Indicates a critical error that may lead to program termination.

**Q7. What line of code would you add to your software to disable all logging messages?**

**Ans.** To disable all logging messages, you can add the following line of code to your software:

logging.disable(logging.CRITICAL)

This line sets the logging level to **CRITICAL**, which disables all logging messages below the **CRITICAL** level. Therefore, no logging messages will be displayed.

**Q8.Why is using logging messages better than using print() to display the same message?**

**Ans.** Using logging messages is better than using **print()** for displaying messages because:

* Logging provides different log levels that allow you to categorize and filter messages based on their severity. This can be useful for troubleshooting and debugging purposes.
* Logging messages can be easily enabled or disabled or redirected to different outputs (e.g., console, file) without modifying the code. This allows for more flexibility in controlling the logging behavior.
* Logging provides more control over the formatting of the log messages, including timestamps, log levels, and additional contextual information.

**Q9. What are the differences between the Step Over, Step In, and Step Out buttons in the debugger?**

**Ans.** The differences between the Step Over, Step In, and Step Out buttons in the debugger are as follows:

* Step Over: Executes the current line of code and advances to the next line in the current function. If the current line contains a function call, the debugger will not step into the called function but will execute it as a single step.
* Step In: If the current line of code contains a function call, the debugger will step

**Q10.After you click Continue, when will the debugger stop?**

**Ans.** The debugger will stop again when it encounters the next breakpoint, encounters an unhandled exception, or reaches the end of the program.

**Q11. What is the concept of a breakpoint?**

**Ans.** A breakpoint is a point in your code where the debugger will pause execution so that you can examine the state of the program and step through the code line by line.