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

spam=-1

assert spam>=0

output:

**AssertionError** Traceback (most recent call last)

Input **In [1]**, in <cell line: 2>**()**

1 spam=-1

**----> 2** **assert** spam>=0

**AssertionError**:

1. 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).

def func1(egg,bacon):

assert not(egg==bacon)

func1(‘hello’,’hello’)

output:

**AssertionError** Traceback (most recent call last)

Input **In [4]**, in <cell line: 1>**()**

**----> 1** func1('hello','hello')

Input **In [3]**, in func1**(egg, bacon)**

1 **def** func1(egg,bacon):

2 ## egg = egg.upper()

3 ##bacon = bacon.upper()

**----> 4** **assert** **not**(egg == bacon)

**AssertionError**:

func1(

In [ ]:



​

def func1(egg,bacon):

egg = egg.upper()

bacon = bacon.upper()

assert not(egg == bacon)

func1("goodbye","GOODbye")

ouput:

**AssertionError** Traceback (most recent call last)

Input **In [7]**, in <cell line: 5>**()**

3 bacon = bacon.upper()

4 **assert** **not**(egg == bacon)

**----> 5** func1("goodbye","GOODbye")

Input **In [7]**, in func1**(egg, bacon)**

2 egg = egg.upper()

3 bacon = bacon.upper()

**----> 4** **assert** **not**(egg == bacon)

**AssertionError**:

In [ ]:



​

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

def func2():

assert False, 'Always Shows Assertion Error'

func2()

output:

**AssertionError** Traceback (most recent call last)

Input **In [8]**, in <cell line: 3>**()**

1 **def** func2():

2 **assert** **False**, 'Always Shows Assertion Error'

**----> 3** func2()

Input **In [8]**, in func2**()**

1 **def** func2():

**----> 2** **assert** **False**, 'Always Shows Assertion Error'

**AssertionError**: Always Shows Assertion Error

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

**import** logging

logging**.**basicConfig(filename **=** 'Test.log',level**=**logging**.**DEBUG, format**=**' %(asctime)s - %(levelname)s - %(message)s')

1. 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?

import logging

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

logging.debug("Insetion completed")

file = open("./programLog.txt","r")

for i in file.readlines():

print(i)

output:

2022-07-08 10:03:15,450 - DEBUG - Insetion completed

1. What are the five levels of logging?

DEBUG-10

INFO-20

WARNING-30

ERROR-40

CRITICAL-50

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

Logging.disable=True

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

Logging has different levels of severity that allows you to display log messages according to the level you want. A print statement does not give you that flexibility

Logging allows you to direct the log messages to separate files that can then be used for post analysis while the same is not easily available with print statement

You can set different log levels at individual code file level as well - some files may have INFO level while some may have DEBUG level

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

Step in - Step In button will cause the debugger to execute the next line of code and then pause again.

Step Over - Step Over button will execute the next line of code, similar to the Step In button. However, if the next line of code is a function call, the Step Over button will “step over” the code in the function. The function’s code will be executed at full speed, and the debugger will pause as soon as the function call returns.

Step out - Step Out button will cause the debugger to execute lines of code at full speed until it returns from the current function.

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

This will cause the program to continue running normally, without pausing for debugging untill it terminates or reaches a breakpoint.

11. What is the concept of a breakpoint?

The Python breakpoint() built-in function is **a tool that allows developers to set points in code at which a debugger is called**.