1.spam **=** **-**22

**assert** spam **>=**0, 'Variable Spam should not be a -ve number'

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

**<ipython-input-2-f0ddd2c96b62>** in <module>

1 spam **=** **-22**

**----> 2 assert** spam **>=0,** **'Variable Spam should not be a -ve number'**

**AssertionError**: Variable Spam should not be a -ve num

**2.def** raise\_assert(egg,bacon):

egg **=** egg**.**upper()

bacon **=** bacon**.**upper()

**assert** **not**(egg **==** bacon), 'Eggs/Bacon should not be same, which are same now'

raise\_assert('hello','HELLO')

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

**<ipython-input-12-66993e9183bf>** in <module>

**----> 1** raise\_assert**('hello','HELLO')**

**<ipython-input-11-fa875d6def57>** in raise\_assert**(egg, bacon)**

6 egg **=** egg**.**upper**()**

7 bacon **=** bacon**.**upper**()**

**----> 8 assert** **not(**egg **==** bacon**),** **'Eggs/Bacon should not be same, which are same now'**

**AssertionError**: Eggs/Bacon should not be same, which are same now

raise\_assert('goodbye','GOODbye')

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

**<ipython-input-13-5322ce220e06>** in <module>

**----> 1** raise\_assert**('goodbye','GOODbye')**

**<ipython-input-11-fa875d6def57>** in raise\_assert**(egg, bacon)**

6 egg **=** egg**.**upper**()**

7 bacon **=** bacon**.**upper**()**

**----> 8 assert** **not(**egg **==** bacon**),** **'Eggs/Bacon should not be same, which are same now'**

**AssertionError**: Eggs/Bacon should not be same, which are same now

**3.def** assert\_always():

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

assert\_always()

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

**<ipython-input-9-b25c0da8a33f>** in <module>

1 **def** assert\_always**():**

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

**----> 3** assert\_always**()**

**<ipython-input-9-b25c0da8a33f>** in assert\_always**()**

1 **def** assert\_always**():**

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

3 assert\_always**()**

**AssertionError**: Always Shows Assertion Error

**4.import** logging

logging**.**basicConfig(filename **=** 'application\_log.txt',level**=**logging**.**DEBUG, format**=**' %(asctime)s - %(levelname)s - %(mes

**5.import** logging

logging**.** basicConfig(filename **=** 'application\_log.txt',level**=**logging**.**DEBUG, format**=**' %(asctime)s - %(level name)s - %(message)s')

logging**.** Debug ("Data Inserted Successfully")

logging**.** Debug ('Connection Closed Successfully')

file **=** open("./application\_log.txt","r")

**for** record **in** file**.** read lines():

print(record)

2021-10-10 16:32:15,102 - DEBUG - Data Inserted Successfully

2021-10-10 16:32:15,102 - DEBUG - Connection Closed Successfully

**6Ans:** The Five levels of Logging provided by python's logging module are **CRITICAL (50)**, **ERROR (40)**, **WARNING (30)**, **INFO (20**, **DEBUG (10)**, **NOTSET**

7.logging**.** Disable **=** **True**

**8Ans:** Post development of your code, you can disable logging messages without removing the logging function, whereas you need to manually remove print () statements, which is tedious activity. and also print is used when you want to display any particular message or help whereas logging is used to record all events like error, info, debug messages, timestamps.

**9Ans:** The Differences between Step Over, Step In, Step Out buttons in debugger are:

1. Step in - Step In button will cause the debugger to execute the next line of code and then pause again.
2. 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.
3. Step out - Step Out button will cause the debugger to execute lines of code at full speed until it returns from the current function.

**10Ans:** This will cause the program to continue running normally, without pausing for debugging until it terminates or reaches a breakp

**11Ans:** Breakpoint is a setting on a line of code that causes the debugger to pause when the program execution reaches the line