Assignment 11

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

In [3]:
```
spam = -22
assert spam >=0
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [3], in <cell line: 2>()
      1 spam = -22
----> 2 assert spam >=0

AssertionError:
```

2. 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).

In [4]:
```
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)
Input In [4], in <cell line: 5>()
      3     bacon = bacon.upper()
      4     assert not(egg == bacon), 'Eggs/Bacon should not be same, which are same now'
----> 5 raise_assert('hello','HELLO')

Input In [4], in raise_assert(egg, bacon)
      2 egg = egg.upper()
      3 bacon = bacon.upper()
----> 4 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
```

In [5]:
```
raise_assert('goodbye','GOODbye')
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [5], in <cell line: 1>()
----> 1 raise_assert('goodbye','GOODbye')

Input In [4], in raise_assert(egg, bacon)
      2 egg = egg.upper()
      3 bacon = bacon.upper()
----> 4 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. Create an assert statement that throws an AssertionError every time.

In [7]:
```
def assert_always():
    assert False, 'Always Shows Assertion Error'
assert_always()
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [7], in <cell line: 3>()
      1 def assert_always():
      2     assert False, 'Always Shows Assertion Error'
----> 3 assert_always()

Input In [7], in assert_always()
      1 def assert_always():
----> 2     assert False, 'Always Shows Assertion Error'

AssertionError: Always Shows Assertion Error
```

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

In [10]:
```
import logging
logging.basicConfig(filename = 'application_log.txt',level=logging.DEBUG, format=' %(asctime)s - %(levelname)s
```

5. 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?

In [11]:
```
import logging
logging.basicConfig(filename = 'application_log.txt',level=logging.DEBUG, format=' %(asctime)s - %(levelname)s
logging.debug("Data Inserted Successfully")
logging.debug('Connection Closed Successfully')
```

In [12]:
```
file = open("./application_log.txt","r")
for record in file.readlines():
    print(record)
```

```
 2023-06-14 00:46:30,808 - DEBUG - Data Inserted Successfully

 2023-06-14 00:46:30,808 - DEBUG - Connection Closed Successfully
```

6. What are the five levels of logging?

Ans: The Five levels of Logging provided by python's logging module are CRITICAL(50), ERROR(40), WARNING(30), INFO(20, DEBUG(10), NOTSET(0)

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

In [13]:
```
logging.disable = True
```

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

Ans: Post devlopment 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.

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

Ans: 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.

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

Ans: 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?

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