Assignment 11 Solutions

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

ANS:assert spam >0

In [1]:

```
spam = -45
assert spam >=0, 'Variable Spam should not be a -ve number'
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12168/3323081260.py in <module>
      1 spam = -45
----> 2 assert spam >=0, 'Variable Spam should not be a -ve number'

AssertionError: Variable Spam should not be a -ve number
```

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 [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'
```

In [3]:

```
raise_assert('hello','HELLO')
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12168/1731780412.py in <module>
----> 1 raise_assert('hello','HELLO')

~\AppData\Local\Temp/ipykernel_12168/3500650028.py in raise_assert(egg, baco
n)
      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 [4]:

```
raise_assert('goodbye','GOODbye')
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12168/4052838577.py in <module>
----> 1 raise_assert('goodbye','GOODbye')

~\AppData\Local\Temp/ipykernel_12168/3500650028.py in raise_assert(egg, baco
n)
      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.

ANS: `assert False` , 'This Assertion Always Shows Assertion Error'

In [5]:

```python
def assert_always():
    assert False, 'Always Shows Assertion Error'
assert_always()
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12168/1503883157.py in <module>
      1 def assert_always():
      2     assert False, 'Always Shows Assertion Error'
----> 3 assert_always()

~\AppData\Local\Temp/ipykernel_12168/1503883157.py in assert_always()
      1 def assert_always():
----> 2     assert False, 'Always Shows Assertion Error'
      3 assert_always()

AssertionError: Always Shows Assertion Error
```

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

ANS: To be able to call logging.debug(), you must have these two lines at the start of your program:

In [7]:

```python
import logging
logging.basicConfig(filename = 'application_log.txt',level=logging.DEBUG, format=' %(as
```

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?

ANS: To be able to send logging messages to a file named programLog.txt with logging.debug(), you must have these two lines at the start of your program:

In [9]:

```python
import logging
logging.basicConfig(filename = 'application_log.txt',level=logging.DEBUG, format=' %(as
logging.debug("Data Inserted Successfully")
logging.debug('Connection Closed Successfully')
```

In [10]:

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

```
2022-11-23 01:13:51,317 - DEBUG - Data Inserted Successfully

2022-11-23 01:13:51,317 - 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 [12]:

```
#ANS:
logging.disable = True
```

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

ANS: Post devlopment of our code, we can disable logging messages without removing the logging function, whereas we need to manually remove print() statements, which would become a tedious activity. Also, print is used when we want to display any particular message or help whereas logging is used to record all events such as 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

The `Step In` button will move the debugger into a function call.

The `Step Over` button will quickly execute the function call without stepping into it.

The `Step Out` button will quickly execute the rest of the code until it steps out of the function it currently is in.

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

ANS: After you click Continue, the debugger will stop when it has reached the end of the program or a line with a breakpoint.

11. What is the concept of a breakpoint?

ANS: A breakpoint is a setting on a line of code that causes the debugger to pause when the program execution reaches the line.