what is testing?
---
the process of identify the bugs/errors in our program/source code is known as a testing.

in software industry,to test our application in two ways,they are

1).unit testing

2).integration testing.

how to meassure our software quality?
---
we can meassure our software quality by using testing.

unit testing:
---
to test the each and every modules/a small piece of code in our program is nothing but a unit testing.

the unit testing is performed by the developer.

python provides various libraries to perform the unit testing,they are unittest,pytest,...

unittest framework is a builtin unittesting framework in python.

integration testing:
---
to test entire applicaltion is properly working or not,is known as a integration testing.

the integration testing is performed by the tester.

if we want to perform integration testing by using python to required seleneium and robotframework

working with unittest framework?
---
unittest is a builtin python framework for unittesting.

ex1:
---
sample.py
---
```
import unittest
class mytest(unittest.TestCase):
    def test_1(self):
        pass
```

```
if __name__=='__main__':
    unittest.main()


        output:
        -------
C:\Users\DELL\Desktop>python sample.py
.
----------------------------------------
Ran 1 test in 0.001s

OK

        (or)

C:\Users\DELL\Desktop>python sample.py -v
test_1 (__main__.mytest) ... ok

----------------------------------------
Ran 1 test in 0.002s

OK


ex2:
----
        sample.py
        ---------
import unittest
class mytest(unittest.TestCase):
    def test_1(self):
        pass
    def test_2(self):
        pass
    def test_3(self):
        pass

if __name__=='__main__':
    unittest.main()

        output
        ------
C:\Users\DELL\Desktop>python sample.py
...
----------------------------------------
Ran 3 tests in 0.001s

OK
```

```
        (or)

C:\Users\DELL\Desktop>python sample.py -v
test_1 (__main__.mytest) ... ok
test_2 (__main__.mytest) ... ok
test_3 (__main__.mytest) ... ok

----------------------------------------
Ran 3 tests in 0.001s

OK

ex3:
----
        sample.py
        ----------
import unittest
class mytest(unittest.TestCase):
    def test_1(self):
        pass
    def test_2(self):
        pass
    def m1(self):
        pass
    def m2_test(self):
        pass

if __name__=='__main__':
    unittest.main()

        output
        ------
C:\Users\DELL\Desktop>python sample.py
..
----------------------------------
Ran 2 tests in 0.003s

OK

        (or)

C:\Users\DELL\Desktop>python sample.py -v
test_1 (__main__.mytest) ... ok
test_2 (__main__.mytest) ... ok

----------------------------------------
Ran 2 tests in 0.002s

OK
```

note:
----
our test case method always starts with 'test'

if we want to write the test cases by using "AAA" model

Arrange : to arrange the what are the prerequsites are needed to write the test case.

Act:    to perform the operation/to run the test case

Assert: to verify the result

ex4:
---
```
import unittest
def add(x,y):
    z=x+y
    return z

class mytest(unittest.TestCase):
    def test_add(self):
        #Arrange
        self.a=10
        self.b=20
        #Act
        res=add(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a+self.b)

if __name__=='__main__':
    unittest.main()
```

output:
--------
```
C:\Users\DELL\Desktop>python sample.py
.
----------------------------------------
Ran 1 test in 0.003s

OK
```

(or)

```
C:\Users\DELL\Desktop>python sample.py -v
test_add (__main__.mytest) ... ok

----------------------------------------
Ran 1 test in 0.001s
```

OK

ex5:
---
```
import unittest
def add(x,y):
    z=x+y
    return z
def sub(x,y):
    z=x-y
    return z
def mul(x,y):
    z=x*y
    return z
def div(x,y):
    z=x/y
    return z
class mytest(unittest.TestCase):
    def test_add(self):
        #Arrange
        self.a=10
        self.b=20
        #Act
        res=add(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a+self.b)
    def test_sub(self):
        #Arrange
        self.a=10
        self.b=20
        #Act
        res=sub(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a-self.b)
    def test_mul(self):
        #Arrange
        self.a=10
        self.b=20
        #Act
        res=mul(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a*self.b)
    def test_div(self):
        #Arrange
        self.a=10
        self.b=20
        #Act
        res=div(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a/self.b)
```

```
if __name__=='__main__':
    unittest.main()

                (or)

import unittest
def add(x,y):
    z=x+y
    return z
def sub(x,y):
    z=x-y
    return z
def mul(x,y):
    z=x*y
    return z
def div(x,y):
    z=x/y
    return z
class mytest(unittest.TestCase):
    def setUp(self):
        #Arrange
        self.a=10
        self.b=20
    def test_add(self):
        #Act
        res=add(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a+self.b)
    def test_sub(self):
        #Act
        res=sub(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a-self.b)
    def test_mul(self):
        #Act
        res=mul(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a*self.b)
    def test_div(self):
        #Act
        res=div(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a/self.b)

if __name__=='__main__':
    unittest.main()

        output
        ------
```

```
C:\Users\DELL\Desktop>python sample.py
....
------------------------------------
Ran 4 tests in 0.001s

OK

        (or)

C:\Users\DELL\Desktop>python sample.py -v
test_add (__main__.mytest) ... ok
test_div (__main__.mytest) ... ok
test_mul (__main__.mytest) ... ok
test_sub (__main__.mytest) ... ok

----------------------------------------
Ran 4 tests in 0.001s

OK
```

ex6:

```
        calculation.py
        --------------
def add(x,y):
    z=x+y
    return z
def sub(x,y):
    z=x-y
    return z
def mul(x,y):
    z=x*y
    return z
def div(x,y):
    z=x/y
    return z


        sample.py
        ---------
import unittest
from calculation import add,sub,mul,div
class mytest(unittest.TestCase):
    def setUp(self):
        #Arrange
        self.a=10
        self.b=20
    def test_add(self):
        #Act
        res=add(self.a,self.b)
```

```
        #Assert
        self.assertEqual(res,self.a+self.b)
    def test_sub(self):
        #Act
        res=sub(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a-self.b)
    def test_mul(self):
        #Act
        res=mul(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a*self.b)
    def test_div(self):
        #Act
        res=div(self.a,self.b)
        #Assert
        self.assertEqual(res,self.a/self.b)

if __name__=='__main__':
    unittest.main()

        output:
        ------
C:\Users\DELL\Desktop>python sample.py -v
test_add (__main__.mytest) ... ok
test_div (__main__.mytest) ... ok
test_mul (__main__.mytest) ... ok
test_sub (__main__.mytest) ... ok

----------------------------------------
Ran 4 tests in 0.003s

OK

ex7:
----
        strreverse.py
        -------------
def rev(x):
    y=''
    i=len(x)-1
    while i>=0:
        y+=x[i]
        i-=1
    return y
if __name__=='__main__':
    print(rev("siva"))


        sample.py
```

```
        ---------
import unittest
from strreverse import rev
class mytest(unittest.TestCase):
    def setUp(self):
        #Arrange
        self.a="siva"
    def test_rev(self):
        #Act
        res=rev(self.a)
        #Assert
        self.assertEqual(res,self.a[::-1])
if __name__=='__main__':
    unittest.main()

        output
        ------
C:\Users\DELL\Desktop>python sample.py -v
test_rev (__main__.mytest) ... ok

----------------------------------------
Ran 1 test in 0.003s

OK

        (or)

C:\Users\DELL\Desktop>python sample.py
.
----------------------------------------
Ran 1 test in 0.000s

OK

note:
-----
. means Success

F means Failure

E means Error

working with pytest framework
-----------------------------
        if we want to working with pytest frame work to install pytest frame work
manually.

        pip install pytest

in pytest,to write the testcases by using functions concept instead of writeing the
```

classes in unittesting framework.

our test function should starts with test or ends with test.

ex1:
---

```
        demo.py
        -------
def test_1():
    pass

        output
        -------
C:\Users\DELL\Desktop>pytest demo.py
=========== test session starts ================
platform win32 -- Python 3.10.2, pytest-7.0.0,
pluggy-1.0.0

rootdir: C:\Users\DELL\Desktop
collected 1 item

demo.py .                                    [100%]

========== 1 passed in 0.16s ======================

               (or)

C:\Users\DELL\Desktop>pytest demo.py -v
========== test session starts =================
platform win32 -- Python 3.10.2, pytest-7.0.0,
pluggy-1.0.0 -- C:\Python310\python.exe

cachedir: .pytest_cache
rootdir: C:\Users\DELL\Desktop
collected 1 item

demo.py::test_1 PASSED                       [100%]

========== 1 passed in 0.10s ====================
```

note:
---
it is recommended to save the test modules starts with test or endswith test.

ex2:
---

```
        test_sample.py
        --------------
def test_a():
    pass
```

```
        demo_test.py
        ------------
def test_b():
    pass


        output
        ------
C:\Users\DELL\Desktop>pytest
========= test session starts ================
platform win32 -- Python 3.10.2, pytest-7.0.0,
pluggy-1.0.0

rootdir: C:\Users\DELL\Desktop
collected 2 items

demo_test.py .                          [ 50%]
test_sample.py .                        [100%]

========== 2 passed in 2.07s ==================

        (or)

C:\Users\DELL\Desktop>pytest -v
========= test session starts ================
platform win32 -- Python 3.10.2, pytest-7.0.0,
pluggy-1.0.0 -- C:\Python310\python.exe

cachedir: .pytest_cache
rootdir: C:\Users\DELL\Desktop
collected 2 items

demo_test.py::test_b PASSED               [ 50%]
test_sample.py::test_a PASSED             [100%]

========== 2 passed in 1.76s ==================

ex3:
----
        test_sample.py
        --------------
def add(x,y):
    z=x+y
    return z

def test_add():
    #Arrange
    a=10
```

```
    b=20
    #Act
    res=add(a,b)
    #Assert
    res==a+b


        demo_test.py
        -------------
def sub(x,y):
    z=x-y
    return z

def test_sub():
    #Arrange
    a=10
    b=20
    #Act
    res=sub(a,b)
    #Assert
    res==a-b

        output
        ------
C:\Users\DELL\Desktop>pytest -v
=========== test session starts ==================
platform win32 -- Python 3.10.2, pytest-7.0.0,
pluggy-1.0.0 -- C:\Python310\python.exe

cachedir: .pytest_cache
rootdir: C:\Users\DELL\Desktop
collected 2 items

demo_test.py::test_sub PASSED               [ 50%]
test_sample.py::test_add PASSED             [100%]

========== 2 passed in 1.96s ======================

        (or)

C:\Users\DELL\Desktop>pytest
========= test session starts ====================
platform win32 -- Python 3.10.2, pytest-7.0.0,
pluggy-1.0.0

rootdir: C:\Users\DELL\Desktop
collected 2 items

demo_test.py .                              [ 50%]
test_sample.py .                            [100%]
```

```
========== 2 passed in 1.66s ========================
```