**UNIT TEST IN PYTHON**

**Code (Main Program)**

from datetime import \*  
  
def validateCard(expdt):  
 if expdt>datetime.now().date():  
 *#print("Card Expiry Date - {}/{}/{}".format(expdt.day, expdt.month, expdt.year) + " - is Valid")* return 'Valid'  
 else:  
 *#print("Card Expiry Date - {}/{}/{}".format(expdt.day, expdt.month, expdt.year) + " - is InValid")* return 'InValid'  
  
  
*#vdate = date(input("Please enter Card Expirty Dateate : "))  
#vdate=date(2027,4,28)  
#print(validateCard(vdate))  
#vdate1=date(2023,4,28)  
#print(validateCard(vdate1))*

**Code (Test Program)**

import unittest  
from creditCardValidate import \*  
  
  
*# CLASS NAME CAN BE ANYTHING*class CCTest(unittest.TestCase):  
  
 def test\_validateCard(self):  
 result = validateCard(date(2027,4,28))  
 self.assertEqual('Valid', result)  
  
*#WHEN TE SCRIPT IS EXECUTED, THE PYTHON INTERPRETOR LOOKS AT THIS LINE AND IT INVOKES THE UNIT TEST.MAIN METHOD.  
# WHENEVER WE RUN A PYTHON SCRIPT, THAT BECOMES THE MAI VALUE.*if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

The above highlighted is optional in PyCharm, if we remove it and execute, this will work.

**Output(Valid - validateCard(date(2027,4,28))**

Testing started at 07:51 ...

Launching unittests with arguments python -m unittest C:\Users\kamal\PycharmProjects\pythonProjectdemo\creditCardValidatieTest.py in C:\Users\kamal\PycharmProjects\pythonProjectdemo

Ran 1 test in 0.002s

OK

A screenshot of a computer

Description automatically generated

**Output(Valid - validateCard(date(2023,4,28))**

Testing started at 07:58 ...

Launching unittests with arguments python -m unittest C:\Users\kamal\PycharmProjects\pythonProjectdemo\creditCardValidatieTest.py in C:\Users\kamal\PycharmProjects\pythonProjectdemo

Ran 1 test in 0.006s

FAILED (failures=1)

InValid != Valid

Expected :Valid

Actual :InValid

<Click to see difference>

Traceback (most recent call last):

File "C:\Users\kamal\PycharmProjects\pythonProjectdemo\creditCardValidatieTest.py", line 10, in test\_validateCard

self.assertEqual('Valid', result)

AssertionError: 'Valid' != 'InValid'

- Valid

+ InValid

? ++

A screenshot of a computer

Description automatically generated

**Code for 2 Tests (Valid and Invalid)**

import unittest  
from creditCardValidate import \*  
  
*# CLASS NAME CAN BE ANYTHING*class CCTest(unittest.TestCase):  
  
 *# VALID SCENARIO* def test\_validateCard\_valid(self):  
 result = validateCard(date(2027,4,28))  
 self.assertEqual('Valid', result)  
  
 *# INVALID SCENARIO* def test\_validateCard\_expired(self):  
 result = validateCard(date(2022,4,28))  
 self.assertEqual('InValid', result)  
  
*#WHEN TE SCRIPT IS EXECUTED, THE PYTHON INTERPRETOR LOOKS AT THIS LINE AND IT INVOKES THE UNIT TEST.MAIN METHOD.  
# WHENEVER WE RUN A PYTHON SCRIPT, THAT BECOMES THE MAI VALUE.*if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**Note:** The returned values in the test check (self.assertEqual) should be the same like the main file which is imported. If not will fail.

ie: return 'Valid' should be equal to self.assertEqual('Valid', result)

**Ouput for 2 Tests (Valid and Invalid)**

Ran 2 tests in 0.003s

OK

**Screen Shot (Valid and Invalid)**

A screenshot of a computer

Description automatically generated

If the value is return 'InValid' should be equal to self.assertEqual(Expired, result)

**The output will be** :

InValid != Expired

Expected :Expired

Actual :InValid

<Click to see difference>

Traceback (most recent call last):

File "C:\Users\kamal\PycharmProjects\pythonProjectdemo\creditCardValidatieTest.py", line 15, in test\_validateCard\_expired

self.assertEqual('Expired', result)

AssertionError: 'Expired' != 'InValid'

- Expired

+ InValid

Ran 2 tests in 0.006s

FAILED (failures=1)

**Code for Assert for Exceptions (validateCard(date) should be invalid one):**

import unittest  
from creditCardValidate import \*  
  
*# CLASS NAME CAN BE ANYTHING*class CCTest(unittest.TestCase):  
  
 *# VALID SCENARIO* def test\_validateCard\_valid(self):  
 result = validateCard(date(2027,4,28))  
 self.assertEqual('Valid', result)  
  
 *# INVALID SCENARIO* def test\_validateCard\_expired(self):  
 *#CODE FOR RETURN OBJECTS  
 #result = validateCard(date(2022,4,28))  
 #self.assertEqual('InValid', result)  
 #CODE FOR RETURN VALUE IS EXCEPTION USING RAISE* with self.assertRaises(RuntimeError):  
 validateCard(date(2023, 4, 28))  
  
  
*#WHEN TE SCRIPT IS EXECUTED, THE PYTHON INTERPRETOR LOOKS AT THIS LINE AND IT INVOKES THE UNIT TEST.MAIN METHOD.  
# WHENEVER WE RUN A PYTHON SCRIPT, THAT BECOMES THE MAI VALUE.*if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**Output:**

Ran 2 tests in 0.002s

OK

**Code for Assert for Exceptions (validateCard(date) should be valid one):**

import unittest  
from creditCardValidate import \*  
  
*# CLASS NAME CAN BE ANYTHING*class CCTest(unittest.TestCase):  
  
 *# VALID SCENARIO* def test\_validateCard\_valid(self):  
 result = validateCard(date(2027,4,28))  
 self.assertEqual('Valid', result)  
  
 *# INVALID SCENARIO* def test\_validateCard\_expired(self):  
 *#CODE FOR RETURN OBJECTS  
 #result = validateCard(date(2022,4,28))  
 #self.assertEqual('InValid', result)  
 #CODE FOR RETURN VALUE IS EXCEPTION USING RAISE* with self.assertRaises(RuntimeError):  
 validateCard(date(2027, 4, 28))  
  
  
*#WHEN TE SCRIPT IS EXECUTED, THE PYTHON INTERPRETOR LOOKS AT THIS LINE AND IT INVOKES THE UNIT TEST.MAIN METHOD.  
# WHENEVER WE RUN A PYTHON SCRIPT, THAT BECOMES THE MAI VALUE.*if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**Output:**

Ran 2 tests in 0.007s

FAILED (failures=1)

Failure

Traceback (most recent call last):

File "C:\Users\kamal\PycharmProjects\pythonProjectdemo\creditCardValidatieTest.py", line 18, in test\_validateCard\_expired

with self.assertRaises(RuntimeError):

AssertionError: RuntimeError not raised

If we want to setup some data and clean it up after the test run, then we can used using life cycle methods called set up and tear down.

Setup and Tear Down method is invoked before and after every tests.

These could be database connections, connecting to another applications and so on.

**Code for Assert for Exceptions with Setup and TearDown:**

import unittest  
from creditCardValidate import \*  
  
*# CLASS NAME CAN BE ANYTHING*class CCTest(unittest.TestCase):  
  
 *#SETUP METHOD* def setUp(self):  
 print("Setup Values !!!!")  
  
 *# VALID SCENARIO* def test\_validateCard\_valid(self):  
 result = validateCard(date(2027,4,28))  
 self.assertEqual('Valid', result)  
  
 *# INVALID SCENARIO* def test\_validateCard\_expired(self):  
 *#CODE FOR RETURN OBJECTS  
 #result = validateCard(date(2022,4,28))  
 #self.assertEqual('InValid', result)  
 #CODE FOR RETURN VALUE IS EXCEPTION USING RAISE* with self.assertRaises(RuntimeError):  
 validateCard(date(2023, 4, 28))  
  
 *# TEAR DOWN* def tearDown(self):  
 print("Tear Down to clear the values.... ")  
  
*#WHEN TE SCRIPT IS EXECUTED, THE PYTHON INTERPRETOR LOOKS AT THIS LINE AND IT INVOKES THE UNIT TEST.MAIN METHOD.  
# WHENEVER WE RUN A PYTHON SCRIPT, THAT BECOMES THE MAI VALUE.*if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**Output:**

Ran 2 tests in 0.003s

OK

Setup Values !!!!

Tear Down to clear the values....

Setup Values !!!!

Tear Down to clear the values....