Vicente Elias

TDD

Framework utilizado para el testing: unittest

El tdd fue llevado acabo con todo el ciclo, fallo, funcionamiento y refactorización.

Código de Calculadora

```
import sys

class Calculadora:

    def valor(self):
        return self.total

    def suma(self, x, y):
        self.total = x+y
        print("La suma de los números es = ",self.total)

    def resta(self, x, y):
        self.total = x-y
        print("La resta de los números es = ",self.total)

    def multiplicacion(self, x, y):
        self.total = x*y
        print("La multiplicación de los números es = ",self.total)

    def division(self, x, y):
        self.total = x/y
        print("La división de los números es = ",self.total)

try:
    numero_1= int(input("Digame un numero: "))
except ValueError:
    print('Eso no es un numero')
    sys.exit()

try:
    numero_2= int(input("Digame otro numero: "))
except ValueError:
```

```
    print('Eso no es un numero')
    sys.exit()

print ("Que operacion quiere hacer:\n1) --> sumar\n2) -->
restar\n3) --> multiplicar\n4) --> dividir")
try:
    operador= int(input('Opcion: '))
except ValueError:
    print('Eso no es un numero')
    sys.exit()

if operador == 1:
    Calculadora().suma(numero_1, numero_2)

elif operador == 2:
    Calculadora().resta(numero_1, numero_2)

elif operador == 3:
    Calculadora().multiplicacion(numero_1, numero_2)

elif operador == 4:
    Calculadora().division(numero_1, numero_2)
```

Output

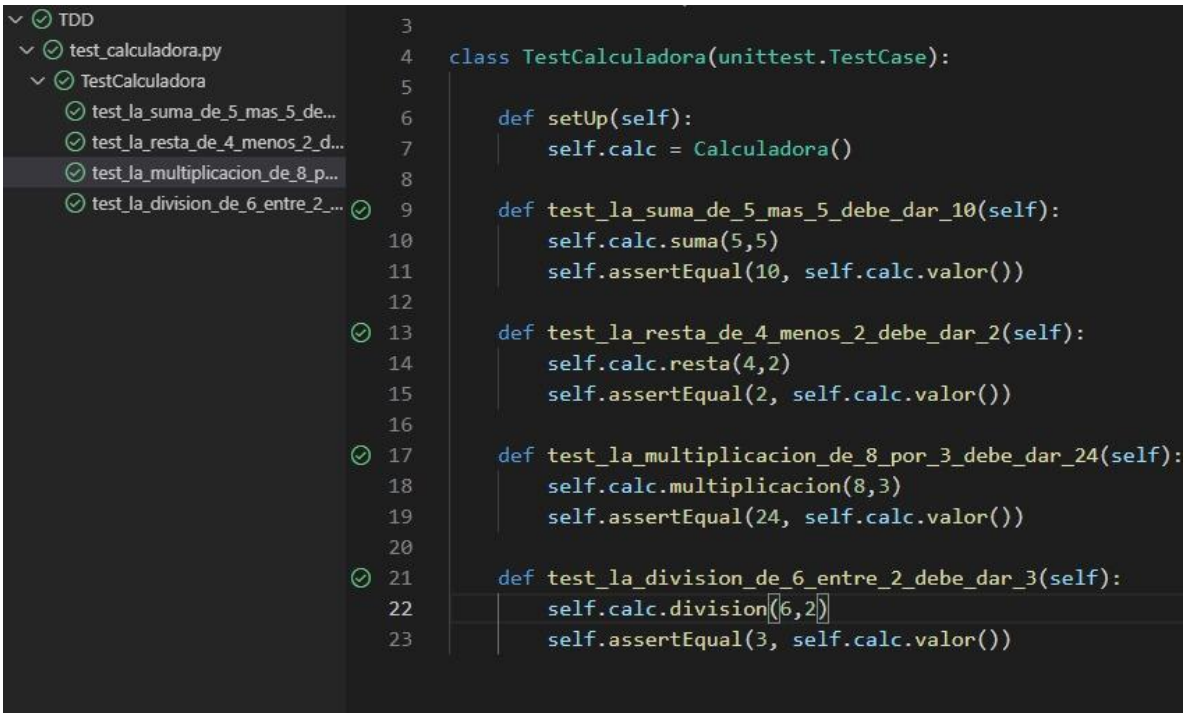
```
Digame un numero: 4
Digame otro numero: 3
Que operacion quiere hacer:
1) --> sumar
2) --> restar
3) --> multiplicar
4) --> dividir
Opcion: 1
La suma de los números es =  7
```

Código de Testeo

```
import unittest
from calculadora import Calculadora

class TestCalculadora(unittest.TestCase):
```

```python
    def setUp(self):
        self.calc = Calculadora()

    def test_la_suma_de_5_mas_5_debe_dar_10(self):
        self.calc.suma(5,5)
        self.assertEqual(10, self.calc.valor())

    def test_la_resta_de_4_menos_2_debe_dar_2(self):
        self.calc.resta(4,2)
        self.assertEqual(2, self.calc.valor())

    def test_la_multiplicacion_de_8_por_3_debe_dar_24(self):
        self.calc.multiplicacion(8,3)
        self.assertEqual(24, self.calc.valor())

    def test_la_division_de_6_entre_2_debe_dar_3(self):
        self.calc.division(6,2)
        self.assertEqual(3, self.calc.valor())
```

Evidencia de Testeo



Código Generador de Contraseñas aleatorias junto con pruebas (para hacer testing dentro del mismo código) estaba probando nuevas cosas.

```
import unittest
from random import randint

def CrearContrasena(numero):
    contra = []
    for i in range(numero):
        num = randint(33, 126)
        contra.append(chr(num))
        contrasena = ''.join(contra)
    print(contrasena)

n= int(input("De cuantos caracteres sera la contrasena?: "))
CrearContrasena(n)

class Test_Contrasena(unittest.TestCase):

    def test_la_contraseña_tiene_caracteres(self):
        self.assertFalse(self.CrearContrasena(8) != None)

    def
test_la_contraseña_tiene_caracteres_tiene_5_caracteres(self):
        self.assertEqual(8,self.CrearContrasena(8))
```

Outputs

```
De cuantos caracteres sera la contrasena?: 8
?0>9FqC_
```

```
De cuantos caracteres sera la contrasena?: 6
8+KzWj
```

```
De cuantos caracteres sera la contrasena?: 15
UIf$0)c0oZ5g!07
```