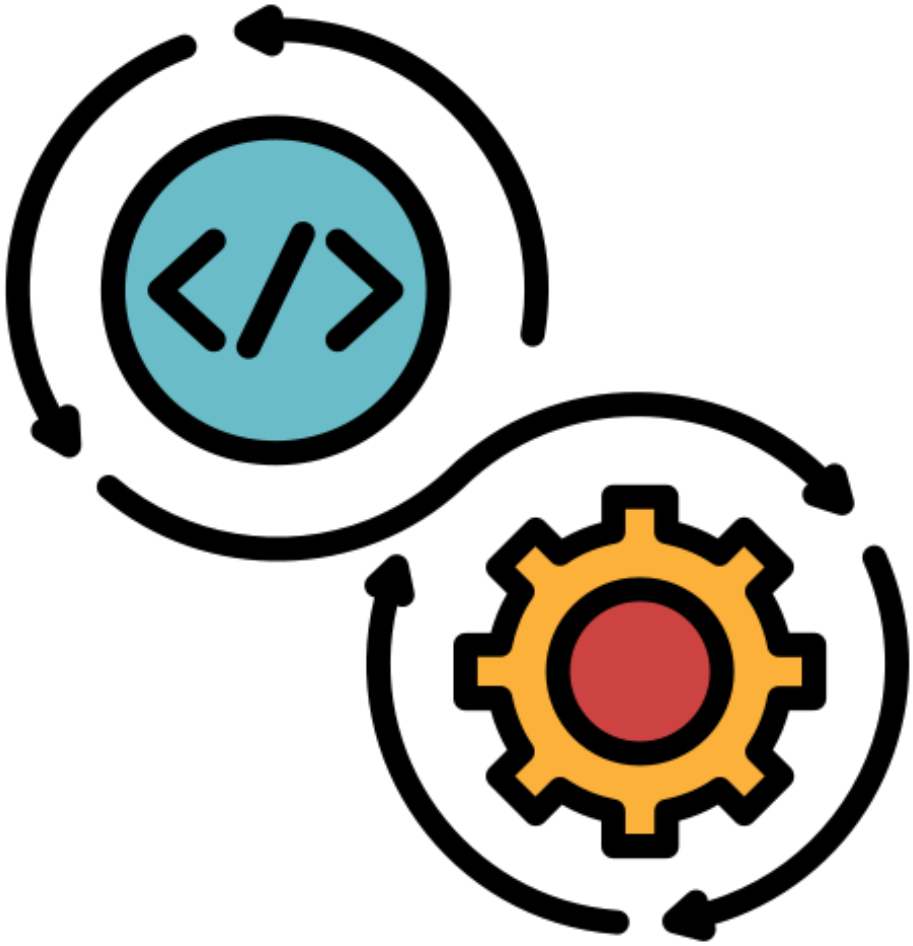
DevOps

Sprint 1 Lab 5



Felipe Izquierdo Romero

Índice

1. Descripción del problema

Para el laboratorio de DevOps se deberá realizar un sencillo módulo de Python con cuatro funciones (suma, resta, multiplicación y división). Estos cuatro módulos deberán estar subidos a un repositorio de GitHub.

Además, habrá que realizar unas pruebas unitarias para cada función y automatizarlas en el repositorio de GitHub con la herramienta GitActions.

2. Desarrollo

Empezaremos con la creación del repositorio en GitHub

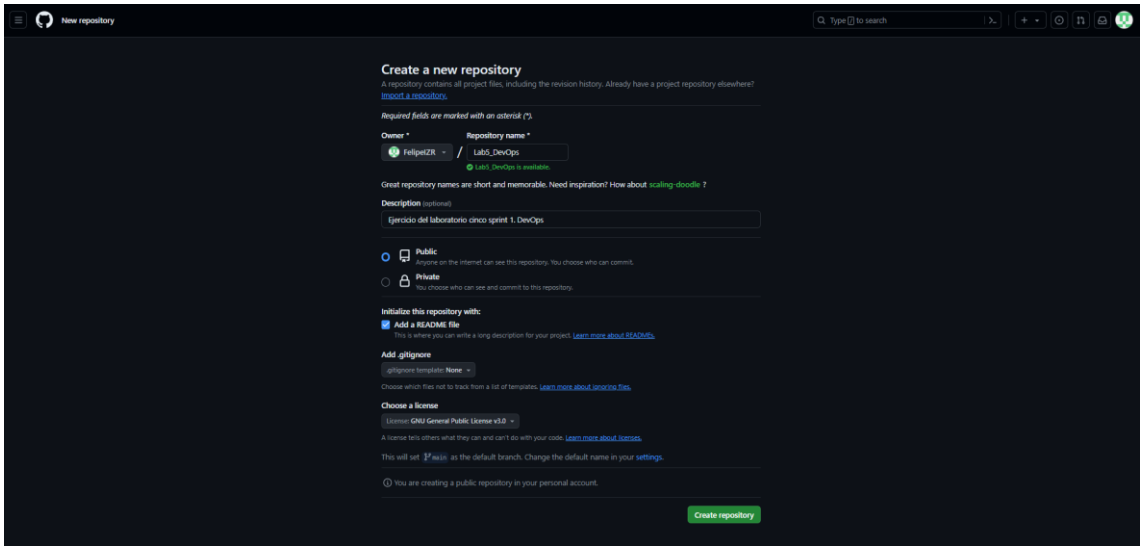


Ilustración 1- Creación de repositorio en GitHub

A continuación, clonamos el repositorio en local

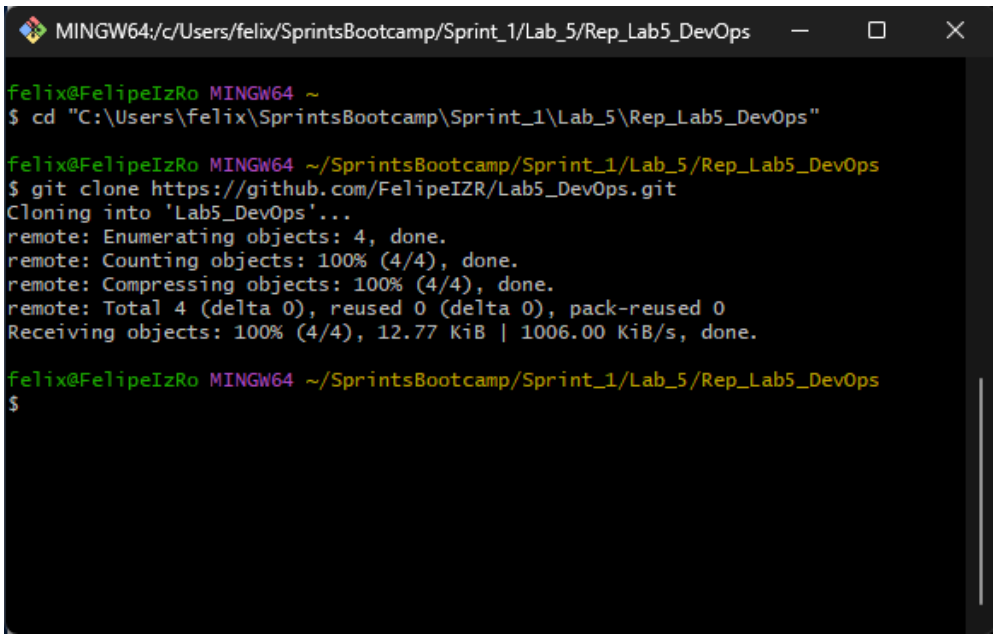


Ilustración 2- Clonación del repositorio

Como paso intermedio crearemos una plantilla del archivo .gitignore en la web https://www.toptal.com/developers/gitignore

Ilustración 3- Generador web de .gitignore

Después en el repositorio creado añadimos un archivo en la raíz con el nombre .gitignore y copiamos el texto que nos generó esta web.

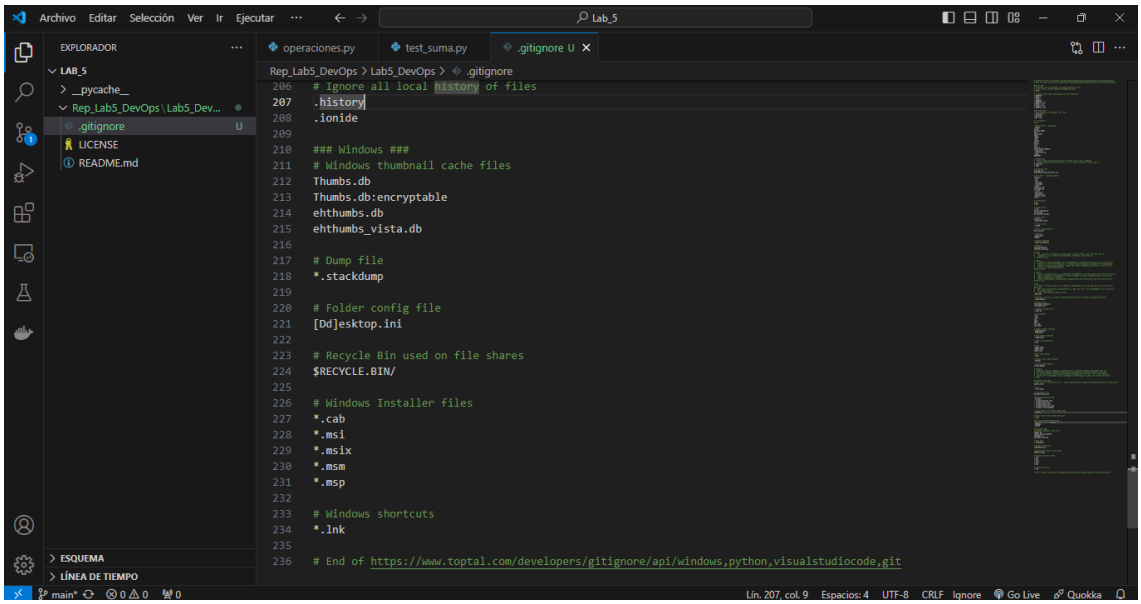


Ilustración 4- Creación de .gitignore

En el repositorio local crearemos el módulo de operaciones.py y test.py.

En operaciones estará el código necesario para crear las cuatro funciones necesarias para completar el laboratorio.

```python
def suma (a,b):
    try:
        if type(a) == bool or type(b) == bool:
            return 'error'
        return float(a) + float(b)
    except:
        return 'error'
def resta (a,b):
    try:
        if type(a) == bool or type(b) == bool:
            return 'error'
        return float(a) - float(b)
    except:
        return 'error'

def multiplicacion(a,b):
    try:
        if type(a) == bool or type(b) == bool:
            return 'error'
        return float(a) * float(b)
    except:
        return 'error'
def division(a,b):
    try:
        if type(a) == bool or type(b) == bool:
            return 'error'
        return float(a) / float(b)
    except:
        return 'error'

```

Ilustración 5- Operaciones

En test.py los test unitarios que deberán pasar mis cuatro funciones

```python
import unittest
import operaciones as operaciones

class TestSumar (unittest.TestCase):
    def test_suma(self):
        # Prueba de sumar
        self.assertEqual(operaciones.suma(1,1),2)
        self.assertEqual(operaciones.suma(5,-1),4)
        self.assertEqual(operaciones.suma(-3,-3),-6)
        self.assertEqual(operaciones.suma(3,-4),-1)
        # Prueba parametros incorrectos
        self.assertEqual(operaciones.suma("a",-4),'error')
        self.assertEqual(operaciones.suma(True,-4),'error')

    def test_resta(self):
        # Prueba de restar
        self.assertEqual(operaciones.resta(1,1),0)
        self.assertEqual(operaciones.resta(5,-1),6)
        self.assertEqual(operaciones.resta(-1,-1),0)
        self.assertEqual(operaciones.resta(5,10),-5)
        # Prueba parametros incorrectos
        self.assertEqual(operaciones.resta("a",10),'error')
        self.assertEqual(operaciones.resta(True,10),'error')

    def test_multiplicacion(self):
        # Prueba de multiplicar
        self.assertEqual(operaciones.multiplicacion(1,1),1)
        self.assertEqual(operaciones.multiplicacion(2,5),10)
        self.assertEqual(operaciones.multiplicacion(3,-3),-9)
        self.assertEqual(operaciones.multiplicacion(20,0),0)
        # Prueba parametros incorrectos
        self.assertEqual(operaciones.multiplicacion("a",2),'error')
        self.assertEqual(operaciones.multiplicacion(True,4),'error')

    def test_division(self):
        # Prueba de dividir
        self.assertEqual(operaciones.division(1,1),1.0)
        self.assertEqual(operaciones.division(5,2),2.5)
        self.assertEqual(operaciones.division(20,10),2.0)
        # Prueba parametros incorrectos
        self.assertEqual(operaciones.division(1,0),'error')
        self.assertEqual(operaciones.division("a",1),'error')
        self.assertEqual(operaciones.division(True,1),'error')

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

Ilustración 6- Test de operaciones

Después crearemos dos subdirectorios .github en la raíz del repositorio y dentro workflows para que GitHub reconozca el fichero de configuración YAML que será el encargado de definir el WorkFlow que queremos que pase nuestro código en GitHub de manera automática.

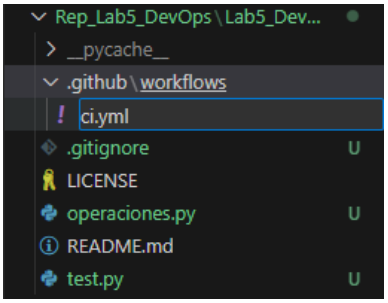


Ilustración 7- Árbol de directorios

El fichero YAML contiene las siguientes instrucciones

```yaml
name: Python CI

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout repository
      uses: actions/chechout@v4

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: 3.11.16

    - name: Run tests
      run:
        python -m unittest test.py

```

Ilustración 8- Fichero ci.yml

Para probar la ejecución del workflow bastaría con hacer un commit y hacer un push al repositorio de GitHub, para ver el proceso haz click en el link

Link Video