INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS GUADALAJARA



Caso #100.

Automatización de aplicaciones en Python con GitHub Actions y pytest

Objetivo:

Que el estudiante comprenda y aplique la automatización de aplicaciones con GitHub Actions y pytest

pytest es un framework de pruebas para Python que se utiliza para escribir y ejecutar pruebas automatizadas de forma sencilla, flexible y poderosa. Es ampliamente utilizado en el desarrollo de software para garantizar que el código funcione correctamente.

Características principales de pytest:

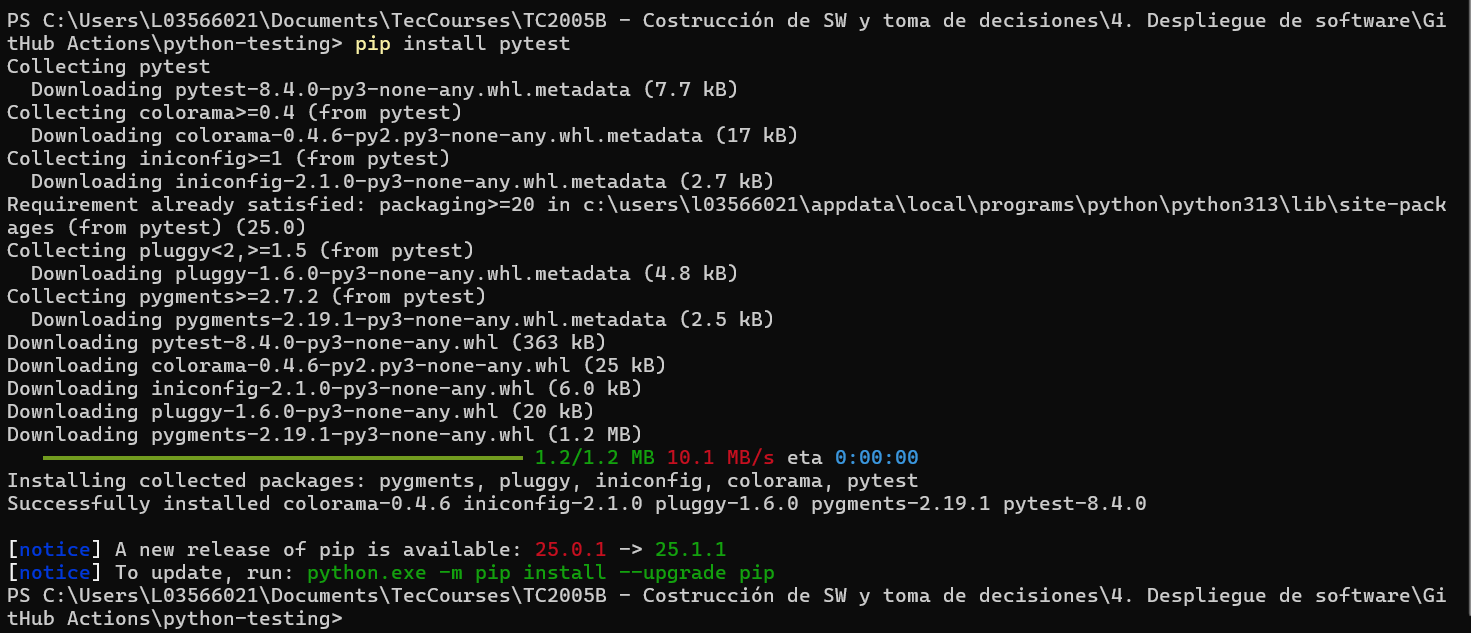
1. **Sencillo de usar:** Nosotros vamos a comenzar con funciones de prueba simples.
2. **Detección automática de pruebas:** Esta es la parte principal de la operación de pytest. Busca automáticamente archivos que comiencen o terminen con “***test\_***” y dentro de los archivos, busca funciones que comiencen con “***test\_”***.
3. **Gestión de errores**: Despliega mensajes de error legibles y detallados cuando una prueba falla.
4. **Ampliable:** Cuenta con muchos plugins (como pytest-django, pytest-cov, etc.) y permite crear plugins personalizados.
5. **Integración con herramientas CI/CD:** Se integra bien con herramientas como Github Actions, Jenkins, Travis CI, etc.

Requerimiento para realizar la práctica

* Tener instalado Docker Desktop
* Tener una cuenta de GitHub
* Instalar Git en tu computadora.
* Instalar pytest en tu computadora.

Primero hay que instalar ***pytest*** con el siguiente comando:

***pip install pytest***



Antes de utilizar ***pytest*** es importante entender lo siguiente.

pytest se utiliza para la calidad de software y por lo tanto detecta posibles fallas en el software (PERO TU, que conoces tu software, que sabes en que parte del código puede haber una falla, debes realizar las *funciones de testing* para detectar fallas. Si se te pasa una función de testing entonces tu software seguramente fallará). Por convención ***pytest*** detecta automáticamente y ejecuta los archivos que cumplen con alguna de estas reglas:

El nombre del *programa de testing* debe comenzar con ***test\_*** o terminar con ***\_test.py***.

Por ejemplo, los siguientes nombres sí son válidos:

test\_areas.py

areas\_test.py

Pero estos nombres no son detectados automáticamente por *pytest*:

probar\_areas.py

testes\_areas.py

Y de igual forma el nombre de las funciones dentro de los *programas python de testing*, pytest solo reconoce aquellas funciones, cuyo nombre comienza con ***test\_.***

Ejemplos válidos:

def test\_area\_triangulo()

assert valida\_base(base) > 0

Recomendación: Para tu aplicación puedes usar una estructura como la siguiente:

Mi aplicación/

test\_areas.py

tests/

test\_area\_triangulo.py

test\_area\_cuadrado.py

donde:

dentro de la carpeta tests/ están todos los *programas de testing* que quieres hacer a la aplicación test\_areas.py

Para efectos meramente didácticos, vamos a usar el siguiente programa Python sumar dos

números.

Recuerda la convención de ***pytest*** para reconocer archivos y funciones.

*test\_suma.py*

# archivo: test\_suma.py

def sumar(a, b):

return a + b

def test\_sumar\_correcto():

assert sumar(2, 3) == 5

# Este test fallará y pytest me lo indica

def test\_sumar\_falla():

assert sumar(2, 2) == 5

Después en la carpeta donde se encuentra la aplicación, abrimos una terminal de comandos y tecleamos el comando:

***pytest***

A screen shot of a computer

AI-generated content may be incorrect.

¿Cómo interpretar los resultados ?

En la imagen anterior el resultado arroja ***test\_suma.py .F***

Pongan atención, el ***.F*** significa que hubo dos *funciones de testing* que se ejecutaron dentro de ***test\_suma.py***. El . significa que la primer función paso sin problema. La F significa que la segunda función no paso (hubo un error). Los posibles valores para el resultado de un ***pytest*** son:

* . = passed
* F = failed
* s = skipped

pytest se puede combinar con **GitHub Actions** y automatizar las pruebas cada vez que se hace algún cambio en tu repositorio en github (por ejemplo, un *push* o una *pull request*). Vamos a utilizar el siguiente ejemplo para entender esto:

1. La estructura de la aplicación debe ser la siguiente:

test-suma/

|--- test.suma.py

|--- requirements.txt

|--- .github/

|--- workflows/

|--- python-tests.yaml

1. El archivo test-suma.py es el que me interesa automatizar la prueba.

sumardosnumeros.py

# archive de la app

def sumar(a, b):

return a + b

testsuma.py

# Archivo con las funciones de testing

# Importamos la app sumar

from sumardosnumeros import sumar

def test\_sumar\_correcto():

assert sumar(2, 3) == 5

# Esto fallará

def test\_sumar\_falla():

assert sumar(2, 2) == 5

Nota importante: La palabra clave ***assert*** se usa en Python (*y en pytest*) para verificar que una condición sea verdadera durante la ejecución del código. Si la condición no se cumple, Python lanza una excepción (AssertionError) y, en el contexto de testing, el test falla.

1. En un archivo ***requirements.txt*** incluyo ***pytest*** como dependencia.

requirements.txt

pytest

1. Creamos el archivo yaml python-tests.yaml para ***GitHub Actions*** y lo ubicamos dentro de la ruta: .github/workflows. El *yaml* permite que siempre que ejecutemos un *push o pull* de la aplicaciónen github, los programas de *testing* se ejecutan automaticamente.

python-tests.yaml

name: Python Tests

on:

push:

branches: [main]

pull\_request:

branches: [main]

jobs:

test:

runs-on: ubuntu-latest

steps:

- name: Clonar el repositorio

uses: actions/checkout@v4

- name: Configurar Python

uses: actions/setup-python@v5

with:

python-version: '3.11'

- name: Instalar dependencias

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Ejecutar pruebas con pytest

run: |

pytest

1. Después, subimos la aplicación a GitHub, para esto, abrimos una terminal de comandos en la carpeta de la aplicación y usamos los siguientes comandos:

git-init

git add .

git commit -m "Primer commit"

git branch -M main

git remote add origin https://github.com/tu\_usuario/tu\_repositorio.git

git push -u origin main

Estos comandos suben tu aplicación a ***github.***

A screenshot of a computer

AI-generated content may be incorrect.

Despues, dentro de la carpeta en github, damos click en ***“Actions”***

A screenshot of a computer

AI-generated content may be incorrect.

Al dar click en “Actions” se ejecutan los programas de testing.

A screenshot of a computer

AI-generated content may be incorrect.

Y nos da el resultado, en este caso nos dice que el testing falló.

Damos un click en las “Annotations” y nos dirá la falla del testing.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.