1 print(“Pruebas de Software”)

Camilo Herrera Arcila, M.Sc.
Universidad de San Buenaventura – Medellín
Semestre 2025 – 1
Semana 06

```
def main():
    print("Tabla de
        Contenido")
```

1. Introducción a Pytest
2. Ejercicio de implementación

Introducción a Pytest

pytest

Generalidades

PyTest es un framework de testing para Python que permite escribir y ejecutar pruebas de manera fácil y eficiente. Es conocido por su simplicidad y flexibilidad, lo que lo hace popular tanto en proyectos pequeños como en aplicaciones más grandes y complejas.

Se caracteriza por:

- **Simplicidad:** No requiere una estructura de clases. Enfoque funcional.
- **Detección automática:** detecta los archivos y funciones de prueba que contengan en su nombre la palabra [Tt]est*.
- **Fixtures:** Proporciona una manera poderosa y flexible de configurar el estado antes de que las pruebas se ejecuten, y limpia después.
- **Parametrización:** Permite ejecutar una sola prueba con diferentes conjuntos de datos.
- **Plugins:** Tiene una gran variedad de plugins que extienden su funcionalidad, desde informes avanzados hasta integración continua.

PyTest** está basado en **Unittest

PyTest vs. Unittest

PyTest ofrece una experiencia de testing más ligera y flexible que unittest, facilitando la escritura y ejecución de pruebas sin la necesidad de boilerplate o estructuras rígidas.

1. Estructura de pruebas.

Mientras que ***unittest*** sigue un enfoque más tradicional basado en clases (similar a los frameworks de testing en otros lenguajes como Java), PyTest permite escribir pruebas como funciones simples sin la necesidad de crear clases.

2. Menor Boilerplate

Requiere menos código repetitivo. Por ejemplo, no es necesario usar self.assertEqual() para verificar condiciones, ya que PyTest puede evaluar directamente las expresiones booleanas usando simples assert.

3. Flexibilidad y facilidad de uso

Es más flexible y fácil de usar para desarrolladores que prefieren una sintaxis más concisa y menos verbosa, mientras que unittest es más formal y requiere más código para configuraciones similares.

Ejemplo

Instalación: $ pip install pytest
Ejecución: $ python –m pytest

suma.py

```python
def suma (a=None, b=None):
    if not(isinstance(a,(int,float))) or not(isinstance(b,(int,float))):
        raise ValueError("a and b must be a int or float")

    if not(a) or not(b):
        raise ValueError("a and b must not be empty.")

    return a + b
```

test_suma.py

```python
from suma import suma
import pytest

def test_suma_positivos():
    assert suma(2, 3) == 5

def test_suma_negativos():
    assert suma(-1, -1) == -2

def test_value_errors():
    with pytest.raises(ValueError):
        suma("1",2)
        suma()
```

```
└─>> python -m pytest
═══════════════ test session starts ═══════════════
platform win32 -- Python 3.10.11, pytest-8.3.2, pluggy-1.5
.0
rootdir: C:\Users\camil\Documents\USB\Pruebas de software\
Contenido\Semana 07\Ejemplo - PyTest
collected 3 items

test_suma.py ...                                  [100%]

═══════════════ 3 passed in 0.02s ═══════════════
```

Ejemplo

Aislar

$ python –m pytest **–v –k *positivos***

```
>> python -m pytest -v -k positivos
========== test session starts ==========
platform win32 -- Python 3.10.11, pytest-8.3.2, pluggy-1.5.0 -
- C:\Users\camil\AppData\Local\Microsoft\WindowsApps\PythonSof
twareFoundation.Python.3.10_qbz5n2kfra8p0\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\camil\Documents\USB\Pruebas de software\Cont
enido\Semana 07\Ejemplo - PyTest
collected 3 items / 2 deselected / 1 selected

test_suma.py::TestMultiples::test_suma_positivos PASSED [100%]

========== 1 passed, 2 deselected in 0.04s ==========
```

Con el parámetro ***–k*** puedo aislar o excluir pruebas dependiendo de mis necesidades.

Excluir

$ python –m pytest **–v –k *“not positivos”***

```
test_suma.py::TestMultiples::test_suma_negativos PASSED [ 50%]
test_suma.py::TestMultiples::test_value_errors PASSED  [100%]

========== 2 passed, 1 deselected in 0.01s ==========
```

Fixtures

La principal diferencia entre ***unittest*** y ***PyTest*** radica en cómo manejan los *fixtures*. Mientras que unittest como fixtures (***setUp***, ***tearDown***, ***setupClass***, etc) todavía se admiten a través de la clase TestCase cuando se utiliza pytest. Pytest trata de proporcionar un mayor desacoplamiento de las pruebas de los *fixtures*.

En *pytest*, se puede declarar un fixture utilizando el **decorador** ***pytest.fixture***. Cualquier función decorada con el decorador se convierte en un fixture.

Antes de continuar...

¿Sabes que es un decorador en Python?

Un ***decorador en Python*** es una función que modifica o extiende el comportamiento de otra función o método sin modificar su código fuente. Los decoradores son una forma de envolver una función para añadir funcionalidad adicional antes o después de su ejecución, o incluso para reemplazarla por completo.

Ejemplo de Python Decorator

Analicemos el siguiente código:

python_decorator.py

```python
def mi_decorador(func):
    def nueva_funcion():
        print("Algo se ejecuta antes de la función original")
        func()
        print("Algo se ejecuta después de la función original")
    return nueva_función

@mi_decorador
def funcion_a_decorar():
    print("Esta es la función original")

funcion_a_decorar()
```

```
>> python python_decorator.py
Algo se ejecuta antes de la función original
Esta es la función original
Algo se ejecuta después de la función original
```

¿Ya sabemos que es un decorador en Python?

Fixtures

Como lo evidenciamos, con los *fixtures* de PyTest podemos configurar los análogos de *SetUp()* y *TearDown()* de *Unittest*.

Analicemos el siguiente código:

SetUp()

TearDown()

```python
from suma import suma
import pytest

@pytest.fixture(scope="function")
def preparar_datos():
    print("\nPreparando datos...")
    yield
    print("\nEliminando datos de prueba...")

@pytest.mark.usefixtures("preparar_datos")
class TestMultiples:
    def test_suma_positivos(self):
        assert suma(2, 3) == 5
    def test_suma_negativos(self):
        assert suma(-1, -1) == -2
    def test_value_errors(self):
        with pytest.raises(ValueError):
            suma("1",2)
            suma()
```

```
test_suma_fixtures.py::TestMultiples::test_suma_positivos
Preparando datos...
PASSED
Eliminando datos de prueba...

test_suma_fixtures.py::TestMultiples::test_suma_negativos
Preparando datos...
PASSED
Eliminando datos de prueba...

test_suma_fixtures.py::TestMultiples::test_value_errors
Preparando datos...
PASSED
Eliminando datos de prueba...
```

Fixtures Scope

```python
@pytest.fixture(scope="class")
def preparar_datos():
    print("\nPreparando datos...")
    yield
    print("\nEliminando datos de prueba...")

@pytest.mark.usefixtures("preparar_datos")
class TestMultiples:
    def test_suma_positivos(self):
        assert suma(2, 3) == 5
    def test_suma_negativos(self):
        assert suma(-1, -1) == -2
    def test_value_errors(self):
        with pytest.raises(ValueError):
            suma("1",2)
            suma()
```

```
test_suma_fixtures.py::TestMultiples::test_sum
a_positivos
Preparando datos...
PASSED
test_suma_fixtures.py::TestMultiples::test_sum
a_negativos PASSED
test_suma_fixtures.py::TestMultiples::test_val
ue_errors PASSED
Eliminando datos de prueba...

========= 3 passed, 3 deselected in 0.04s =========
```

En cobertura ***"class"*** notamos que el setUp() y el TearDown() solo se ejecuta antes y después del caso de prueba, no de las pruebas individuales.

Ejercicio de implementación

pytest

Ejercicios

1. ***User Login:*** Retome el ejercicio de la semana 04 y reestructure sus pruebas ahora con PyTest. Tenga en cuenta que las funcionalidades de setUp() y tearDown() para crear la base de datos de usuarios temporales ahora es con el decorador pytest.fixture().

2. ***Reconocimiento de Django:*** Si nos queda tiempo, realizaremos un reconocimiento de Django básico con un “Hola Mundo”. Y vamos a testear una funcionalidad implementada en el framework.

Gracias

Pruebas de Software
Universidad de San Buenaventura Medellín
Camilo Herrera Arcila, M.Sc.
camilo.herreraa@tau.usbmed.edu.co