



Pytest Progress - Training Test for All Days

Python 3.13 Pytest 8.4.2

*Repositorio que realice para entrenar un como los test automaticos **Pytest** y **Pytest-Mock** pueden apoyarse en el para mejorar habilidades en testing automatizado con Python.*



Tabla de Contenidos

Descripción

Tecnologías

Instalación

Guía de Uso

Conceptos Desarrollados

Ejemplos de Tests

Comandos Útiles




Recursos



Descripción

Este proyecto documenta mi experiencia realizando **testing automatizado** con Pytest. Incluye ejemplos prácticos de:

- ✓ Tests básicos con pytest
- ✓ Uso de fixtures
- ✓ Mocking con pytest-mock

-  Tests de APIs y requests
-  Tests de bases de datos
-  Buenas prácticas de testing

Tecnologías

- **Python 3.13** - Lenguaje de programación
- **Pytest 8.4.2** - Framework de testing
- **Pytest-Mock 3.15.1** - Plugin para mocking
- **Requests 2.32.5** - Librería HTTP
- **Flask 3.1.2** - Framework web (para ejemplos)
- **UV** - Gestor de paquetes rápido code de Rast, mejora el indexado de dependencias y el mas rapido que pip

Instalación

Prerrequisitos

- Python 3.13+
- [UV](#) (recomendado) o pip

Clonar el repositorio

```
git clone https://github.com/ycaballero12315/pytest-progress.git
cd pytest-progress
```

Instalar dependencias

Con UV (recomendado - 10x más rápido)

```
# Instalar UV si no lo tienes
curl -LsSf https://astral.sh/uv/install.sh | sh

# Instalar dependencias del proyecto
uv pip install -e .
```

Con pip (alternativa)

```
pip install -e .
```



Guía de Uso

1 Ejecutar todos los tests

```
pytest
```

2 Ejecutar tests con verbose (detallado)

```
pytest -v
```

3 Ejecutar un archivo específico

```
pytest test_basic.py
```

4 Ejecutar un test específico

```
pytest test_basic.py::test_suma
```

5 Ejecutar con coverage

```
pytest --cov=src --cov-report=html
```

Luego abre `htmlcov/index.html` en tu navegador.

6 Ejecutar solo tests que fallaron

```
pytest --lf
```

7 Ver información detallada de errores

```
pytest -vv --tb=short
```

Conceptos Desarrollados

1. Tests Básicos

Los tests más simples verifican que una función retorna el valor esperado.

```
def test_suma():  
    assert 2 + 2 == 4  
  
def test_string():  
    assert "hello".upper() == "HELLO"
```

Detalle clave: Usar `assert` para verificar resultados esperados.

2. Fixtures

Los fixtures son funciones que proporcionan datos o configuraciones reutilizables para tus tests.

```
import pytest

@pytest.fixture
def user_manager():
    return UserManager() # Instancia limpia para cada test

def test_add_user(user_manager):
    assert user_manager.add_user('Ana', 'ana@test.com') == True
```

Notas claves:

- Los fixtures se inyectan automáticamente como parámetros
- Cada test recibe una instancia nueva y limpia
- Evita duplicación de código de setup

3. Mocking con pytest-mock

El mocking simula objetos o comportamientos sin ejecutar código real (APIs, BDs, archivos).

```
def test_api_call(mock):
    # Simular una llamada HTTP
    mock_get = mock.patch('requests.get')
    mock_get.return_value.json.return_value = {'temperature': 25}

    result = weather_city('Uruguay')

    assert result['temperature'] == 25
    mock_get.assert_called_once_with('https://api.weather.com/v1/Uruguay')
```

Notas claves:

- No necesitas servicios reales para testear
- Puedes simular cualquier comportamiento
- Verificas que se llamen las funciones correctamente

4. Tests de Base de Datos

Mockear operaciones de BD para no depender de una BD real.

```
def test_save_user(mockeer):
    mock_connect = mockeer.patch('db_other.sqlite3.connect')
    mock_conn = mock_connect.return_value
    mock_cursor = mock_conn.cursor.return_value

    save_user('Yoe', 41)

    mock_connect.assert_called_once_with('/data/users.db')
    mock_cursor.execute.assert_called_once_with(
        "INSERT INTO users(name, age) VALUES (?, ?)", ("Yoe", 41)
    )
    mock_conn.commit.assert_called_once()
```

Notas claves:

- Tests rápidos sin BD real
- Verificar que se llamen los métodos correctos
- Aislar la lógica de negocio de la infraestructura

5. Manejo de Excepciones

Verificar que el código lanza las excepciones esperadas.

```
def test_duplicate_user(user_manager):  
    user_manager.add_user('Ana', 'ana@test.com')  
  
    with pytest.raises(ValueError, match='User already exists'):  
        user_manager.add_user('Ana', 'otra@test.com')
```

Notas claves:

- Usa `pytest.raises()` para capturar excepciones
- Puedes verificar el mensaje de error con `match`

Ejemplos de Tests

Ejemplo 1: Test de UserManager

Código a testear (`user_easy.py`):

```
class UserManager:  
    def __init__(self):  
        self.users = {}  
  
    def add_user(self, username, email):  
        if username in self.users:  
            raise ValueError('User already exists')  
        self.users[username] = email  
        return True  
  
    def get_user(self, username):  
        return self.users.get(username)
```

Test (`test_user_easy.py`):

```
import pytest
from user_easy import UserManager

@pytest.fixture
def user_manager():
    return UserManager()

def test_add_user(user_manager):
    result = user_manager.add_user('Yoeny', 'yoeny@test.com')
    assert result == True
    assert user_manager.get_user('Yoeny') == 'yoeny@test.com'

def test_duplicate_user(user_manager):
    user_manager.add_user('Yoeny', 'yoeny@test.com')
    with pytest.raises(ValueError, match='User already exists'):
        user_manager.add_user('Yoeny', 'otro@test.com')
```

Ejemplo 2: Test con Mock de API**Código a testear (`mock_requests.py`):**

```
import requests

def weather_city(city):
    response = requests.get(f'https://api.weather.com/v1/{city}')
    if response.status_code == 200:
        return response.json()
    else:
        raise ValueError('Could not fetch weather data')
```

Test (`test_mock_requests.py`):


```
import pytest
from mock_requests import weather_city

def test_get_weather(mockeer):
    mock_get = mockeer.patch('mock_requests.requests.get')
    mock_get.return_value.status_code = 200
    mock_get.return_value.json.return_value = {
        'temperature': 25,
        'condition': 'Sunny'
    }

    result = weather_city('Uruguay')

    assert result['temperature'] == 25
    assert result['condition'] == 'Sunny'
    mock_get.assert_called_once_with('https://api.weather.com/v1/Uruguay')
```

Ejemplo 3: Test de Base de Datos

Código a testear (`db_other.py`):

```
import sqlite3

def save_user(name, age):
    conn = sqlite3.connect('/data/users.db')
    cursor = conn.cursor()
    cursor.execute("INSERT INTO users(name, age) VALUES (?, ?)", (name, age))
    conn.commit()
    conn.close()
```

Test (`test_save_user.py`):

```
from db_other import save_user

def test_save_user(mockeer):
    mock_connect = mockeer.patch('db_other.sqlite3.connect')
    mock_conn = mock_connect.return_value
    mock_cursor = mock_conn.cursor.return_value

    save_user('Yoe', 41)

    mock_connect.assert_called_once_with('/data/users.db')
    mock_cursor.execute.assert_called_once_with(
        "INSERT INTO users(name, age) VALUES (?, ?)", ("Yoe", 41)
    )
    mock_conn.commit.assert_called_once()
    mock_conn.close.assert_called_once()
```

Comandos Útiles

Testing

```
# Ejecutar todos los tests
pytest

# Verbose mode
pytest -v

# Muy verbose (más información)
pytest -vv

# Ejecutar un test específico
pytest tests/test_basic.py::test_suma

# Ejecutar solo tests que fallaron
pytest --lf

# Detener al primer fallo
```

```
pytest -x

# Ejecutar tests en paralelo (requiere pytest-xdist)
pytest -n auto
```

Coverage

```
# Generar reporte de coverage
pytest --cov=src

# Reporte en HTML
pytest --cov=src --cov-report=html

# Ver líneas no cubiertas
pytest --cov=src --cov-report=term-missing
```

Debugging

```
# Mostrar prints en los tests
pytest -s

# Entrar en debugger cuando falla
pytest --pdb

# Mostrar más contexto en errores
pytest --tb=long
```



Buenas Prácticas



Estructura de un test

Sigue el patrón **AAA (Arrange-Act-Assert)**:

```
def test_example():  
    # Arrange: Preparar datos  
    user_manager = UserManager()  
  
    # Act: Ejecutar la acción  
    result = user_manager.add_user('Ana', 'ana@test.com')  
  
    # Assert: Verificar resultado  
    assert result == True
```

✓ Nombres descriptivos

```
# ✗ Mal  
def test_1():  
    pass  
  
# ✓ Bien  
def test_add_user_successfully():  
    pass  
  
def test_add_duplicate_user_raises_error():  
    pass
```

✓ Un concepto por test

```
# ✗ Mal - prueba muchas cosas  
def test_user_operations():  
    assert add_user() == True  
    assert delete_user() == True  
    assert update_user() == True  
  
# ✓ Bien - un test por operación  
def test_add_user():
```

```
    assert add_user() == True

def test_delete_user():
    assert delete_user() == True
```

✅ Usar fixtures para setup

```
# ❌ Mal - duplicación
def test_1():
    user_manager = UserManager()
    user_manager.add_user('Ana', 'ana@test.com')
    # test...

def test_2():
    user_manager = UserManager()
    user_manager.add_user('Ana', 'ana@test.com')
    # test...

# ✅ Bien - usar fixture
@pytest.fixture
def user_with_data():
    um = UserManager()
    um.add_user('Ana', 'ana@test.com')
    return um

def test_1(user_with_data):
    # test...

def test_2(user_with_data):
    # test...
```



Progreso del Proyecto

- ✅ Tests básicos con assert

- ☒ Fixtures para reutilización
- ☒ Mocking con pytest-mock
- ☒ Tests de APIs con requests
- ☒ Tests de base de datos
- ☒ Manejo de excepciones
- ☒ Tests parametrizados
- ☐ Tests asíncronos
- ☐ Integration tests
- ☒ CI/CD con GitHub Actions

Recursos

Documentación Oficial

- [Pytest Documentation](#)
- [Pytest-Mock Documentation](#)
- [Python unittest.mock](#)

Tutoriales Recomendados

- [Real Python - Pytest Tutorial](#)
- [Test & Code Podcast](#)

Libros

- "Python Testing with pytest" by Brian Okken
- "Test-Driven Development with Python" by Harry Percival

Contribuciones

Este es un repositorio de entrenamiento personal, pero si encuentras errores o tienes sugerencias, siéntete libre de abrir un issue o PR.

 **Autor**

Yoeny Caballero

GitHub: @ycaballero12315

★ Si este repositorio te ayudó a aprender pytest, ¡dale una estrella!

Próximos Pasos

1. Agregar más ejemplos de tests complejos
2. Continuar con el CI/CD en GitHub Actions
3. Agregar tests de integración
4. Documentar más patrones de testing
5. Crear ejemplos con diferentes frameworks (FastAPI)

Espero lo disfruten!!!

Happy Testing!  