CLASE 17
Testing con pytest

Guia de Estudio y Referencia Rapida

Contenido:

- Que es testing y por que importa
- pytest - Instalacion y uso
- Anatomia de un test (patron AAA)
- Cheat Sheet de assertions
- Casos edge comunes
- Buenas practicas
- 8 ejercicios adicionales con soluciones

1. Que es testing y por que importa

Testing es el proceso de verificar que tu codigo funciona correctamente. En vez de probar manualmente (ejecutar y ver con los ojos), escribimos codigo que prueba nuestro codigo automaticamente.

Beneficios del testing

- **Confianza:** Puedes cambiar codigo sin miedo a romper algo
- **Documentacion:** Los tests muestran como se usa tu codigo
- **Deteccion temprana:** Encuentras bugs antes de que lleguen a produccion
- **Refactoring seguro:** Puedes mejorar tu codigo y verificar que sigue funcionando
- **Trabajo en equipo:** Otros pueden modificar tu codigo con confianza

Testing en entrevistas de trabajo

Muchas empresas preguntan sobre testing en entrevistas tecnicas. Saber pytest te diferencia de otros candidatos y demuestra profesionalismo.

> **TIP:** Un programador junior que sabe testing vale mas que uno senior que no lo hace.

2. pytest - Instalacion y uso

Instalacion

```
# Instalar pytest
pip install pytest

# Verificar instalacion
pytest --version
```

Estructura de archivos

```
mi_proyecto/
calculadora.py # Tu codigo (modulo)
test_calculadora.py # Tus tests
validadores.py # Otro modulo
```

```
test_validadores.py # Tests de validadores
```

TIP: El archivo de tests DEBE empezar con **test_** y las funciones de test tambien DEBEN empezar con **test_**. Si no, pytest no los encuentra.

Comandos basicos

Comando	Descripcion
`pytest`	Ejecuta todos los tests del directorio
`pytest -v`	Modo verbose (detallado, muestra cada test)
`pytest test_archivo.py`	Ejecuta tests de un archivo especifico
`pytest test_archivo.py::test_funcion`	Ejecuta UN test especifico
`pytest -x`	Para en el primer fallo
`pytest --tb=short`	Traceback corto (menos detalle)
`pytest --tb=long`	Traceback largo (mas detalle)

---

3. Anatomia de un test - Patron AAA

Todo test sigue el patron **AAA** (Arrange, Act, Assert):

```
def test_suma_positivos():
# ARRANGE (Preparar) - datos de entrada
a = 5
b = 3

# ACT (Actuar) - ejecutar la funcion
resultado = suma(a, b)

# ASSERT (Verificar) - comprobar resultado
assert resultado == 8
```

Paso	Que hace	Ejemplo
ARRANGE	Preparar datos de entrada	a = 5, b = 3
ACT	Ejecutar la funcion a testear	resultado = suma(a, b)
ASSERT	Verificar el resultado	assert resultado == 8

Para tests simples, el patron AAA puede estar en una sola linea:

```
def test_suma_simple():
assert suma(2, 3) == 5 # Arrange+Act+Assert en una linea
```

Nombres descriptivos

El nombre del test debe describir QUE se testea y BAJO QUE condicion:

```
# BUENOS nombres:
test_suma_con_positivos
test_suma_con_cero
test_division_por_cero_lanza_error
test_email_sin_arroba_es_invalido

# MALOS nombres:
test_1
test_funcion
test_a
```

4. Cheat Sheet de Assertions

assert evalua una expresion. Si es True, el test pasa. Si es False, el test falla.

Igualdad

```
assert resultado == 5 # Igual a
assert resultado != 0 # Diferente de
```

Booleanos

```
assert es_par(4) # Es True
assert not es_par(7) # Es False
assert es_par(4) == True # Tambien funciona
assert es_par(7) == False # Tambien funciona
```

Contenido (in / not in)

```
assert "hola" in texto # String contiene "hola"
assert "error" not in mensaje # No contiene "error"
assert 5 in [1, 3, 5, 7] # Elemento en lista
assert "clave" in diccionario # Clave en diccionario
```

Comparaciones

```
assert edad >= 18 # Mayor o igual
assert precio < 100 # Menor que
assert len(lista) > 0 # Lista no vacia
assert len(texto) <= 50 # Texto no muy largo
```

Tipo de dato

```
assert isinstance(resultado, int) # Es entero
assert isinstance(nombre, str) # Es string
assert isinstance(items, list) # Es lista
```

Excepciones con pytest.raises()

```
import pytest

# Verificar que se lance una excepcion
def test_division_por_cero():
with pytest.raises(ValueError):
dividir(10, 0)

# Verificar el mensaje de la excepcion
def test_division_por_cero_mensaje():
with pytest.raises(ValueError) as exc_info:
dividir(10, 0)
assert "cero" in str(exc_info.value)
```

Valores aproximados (floats)

```
# Los floats pueden tener errores de precision
assert 0.1 + 0.2 == 0.3 # FALLA!
assert 0.1 + 0.2 == pytest.approx(0.3) # PASA
```

TIP: pytest.approx() es util cuando trabajas con numeros decimales que pueden tener pequenas diferencias de precision.

5. Casos edge comunes

Los **casos edge** (borde) son valores limite donde el codigo puede fallar. Son los tests MAS importantes porque ahi es donde viven los bugs.

Tipo de edge case	Valores a testear	Ejemplo
Cero	0	suma(0, 0), dividir(0, 5)
Negativos	-1, -100	suma(-5, -3), edad=-1
Strings vacios	""	validar("")
Listas vacias	[]	promedio([])
None	None	funcion(None)
Limites	17, 18, 19	es_adulto(17), es_adulto(18)
Muy grandes	999999, 10**6	suma(999999, 999999)
Tipos inesperados	"abc" donde esperan int	suma("2", 3)

Ejemplo completo: testear una funcion con casos edge

```
def calcular_promedio(numeros):
"""Calcula el promedio de una lista de numeros."""
if not numeros:
raise ValueError("La lista no puede estar vacia")
return sum(numeros) / len(numeros)

# Tests con casos edge
def test_promedio_normal():
assert calcular_promedio([10, 20, 30]) == 20

def test_promedio_un_elemento():
assert calcular_promedio([5]) == 5

def test_promedio_negativos():
assert calcular_promedio([-10, 10]) == 0

def test_promedio_lista_vacia():
with pytest.raises(ValueError):
calcular_promedio([])
```

6. Buenas practicas

Hacer

- **Un test = un caso especifico.** Si falla, sabes exactamente que caso fallo.
- **Nombres descriptivos.** test_suma_con_negativos, no test_1.
- **Testear casos edge primero.** Ahi estan los bugs.
- **Ejecutar tests frecuentemente.** Despues de cada cambio.
- **Tests independientes.** Cada test debe funcionar solo.

Evitar

- Tests que dependen de otros tests
- Tests que modifican archivos reales o bases de datos
- Usar print() para verificar (usar assert)
- Ignorar tests que fallan
- Tests sin assert (no verifican nada)

Regla de oro

> **TIP:** Cuando encuentres un bug: 1) Escribe un test que falle. 2) Arregla el bug. 3) Verifica que el test pase. Asi el bug nunca vuelve.

7. Ejercicios adicionales

Ejercicio 1 (Principiante): Tests para es_par

Escribe tests para esta funcion:

```
def es_par(n):
return n % 2 == 0
```

Tests a crear: con par, con impar, con cero, con negativo par, con negativo impar.

Ejercicio 2 (Principiante): Tests para encontrar maximo

Escribe tests para esta funcion:

```
def encontrar_maximo(lista):
"""Retorna el elemento mas grande de la lista."""
if not lista:
raise ValueError("Lista vacia")
return max(lista)
```

Tests: lista normal, un elemento, negativos, mixtos, lista vacia (ValueError).

Ejercicio 3 (Principiante): Tests para contar vocales

Escribe tests para esta funcion:

```
def contar_vocales(texto):
"""Cuenta las vocales en un texto."""
vocales = "aeiouAEIOU"
return sum(1 for c in texto if c in vocales)
```

Tests: texto normal, sin vocales ("xyz"), string vacio, mayusculas, solo vocales.

Ejercicio 4 (Intermedio): Tests para convertir temperatura

Escribe tests para esta funcion:

```
def celsius_a_fahrenheit(celsius):
return celsius * 9/5 + 32
```

```
def fahrenheit_a_celsius(fahrenheit):
return (fahrenheit - 32) * 5/9
```

Tests: 0C=32F, 100C=212F, -40C=-40F (ambos iguales), conversion ida y vuelta.

Ejercicio 5 (Intermedio): Tests para validar password

Escribe tests para esta funcion:

```
def validar_password(password):
"""
Password valido si:
- Al menos 8 caracteres
- Al menos una mayuscula
- Al menos un numero
"""
if len(password) < 8:
return False
if not any(c.isupper() for c in password):
return False
if not any(c.isdigit() for c in password):
return False
return True
```

Tests: valido, corto, sin mayusculas, sin numeros, vacio, exactamente 8 chars.

Ejercicio 6 (Intermedio): Tests para calcular IMC

Escribe tests para esta funcion:

```
def calcular_imc(peso_kg, altura_m):
"""Calcula el Indice de Masa Corporal."""
if peso_kg <= 0 or altura_m <= 0:
raise ValueError("Peso y altura deben ser positivos")
return round(peso_kg / (altura_m ** 2), 1)
```

Tests: caso normal, peso 0 (error), altura 0 (error), negativos (error).

Ejercicio 7 (Avanzado): Tests para fizzbuzz

Escribe tests para esta funcion:

```
def fizzbuzz(n):
"""
Retorna 'Fizz' si n es multiplo de 3,
'Buzz' si es multiplo de 5,
'FizzBuzz' si es multiplo de ambos,
o el numero como string.
"""
if n % 15 == 0:
return "FizzBuzz"
if n % 3 == 0:
return "Fizz"
if n % 5 == 0:
return "Buzz"
return str(n)
```

Tests: multiplo de 3, multiplo de 5, multiplo de 15, no multiplo de ninguno, 0, negativo.

Ejercicio 8 (Avanzado): Tests para gestionar inventario

Escribe tests para estas funciones:

```
inventario = {}

def agregar_producto(nombre, cantidad, precio):
if cantidad < 0 or precio < 0:
raise ValueError("Cantidad y precio deben ser >= 0")
inventario[nombre] = {"cantidad": cantidad, "precio": precio}

def obtener_valor_total():
return sum(p["cantidad"] * p["precio"]
for p in inventario.values())
```

Tests: agregar producto, cantidad negativa (error), valor total, inventario vacio.

> **TIP:** Recuerda limpiar el inventario entre tests para que sean independientes.

8. Soluciones

Solucion Ejercicio 1

```
def test_par_con_par():
assert es_par(4) == True

def test_par_con_impar():
assert es_par(7) == False

def test_par_con_cero():
assert es_par(0) == True

def test_par_negativo_par():
assert es_par(-6) == True

def test_par_negativo_impar():
assert es_par(-3) == False
```

Solucion Ejercicio 2

```
def test_maximo_normal():
assert encontrar_maximo([3, 7, 2, 9, 1]) == 9

def test_maximo_un_elemento():
assert encontrar_maximo([42]) == 42

def test_maximo_negativos():
assert encontrar_maximo([-5, -1, -8]) == -1

def test_maximo_mixtos():
assert encontrar_maximo([-3, 0, 5]) == 5

def test_maximo_lista_vacia():
with pytest.raises(ValueError):
encontrar_maximo([])
```

Solucion Ejercicio 3

```
def test_vocales_normal():
assert contar_vocales("hola mundo") == 4
```

```
def test_vocales_sin_vocales():
assert contar_vocales("xyz") == 0

def test_vocales_vacio():
assert contar_vocales("") == 0

def test_vocales_mayusculas():
assert contar_vocales("AEIOU") == 5

def test_vocales_solo_vocales():
assert contar_vocales("aeiou") == 5
```

Solucion Ejercicio 4

```
def test_cero_celsius():
assert celsius_a_fahrenheit(0) == 32

def test_100_celsius():
assert celsius_a_fahrenheit(100) == 212

def test_menos_40():
assert celsius_a_fahrenheit(-40) == -40

def test_ida_y_vuelta():
temp = 25
f = celsius_a_fahrenheit(temp)
c = fahrenheit_a_celsius(f)
assert c == pytest.approx(temp)
```

Solucion Ejercicio 5

```
def test_password_valido():
assert validar_password("Abc12345") == True

def test_password_corto():
assert validar_password("Abc1") == False

def test_password_sin_mayuscula():
assert validar_password("abc12345") == False

def test_password_sin_numero():
assert validar_password("Abcdefgh") == False

def test_password_vacio():
```

```
assert validar_password("") == False

def test_password_exacto_8():
assert validar_password("Abcdefg1") == True
```

Solucion Ejercicio 6

```
def test_imc_normal():
assert calcular_imc(70, 1.75) == 22.9

def test_imc_peso_cero():
with pytest.raises(ValueError):
calcular_imc(0, 1.75)

def test_imc_altura_cero():
with pytest.raises(ValueError):
calcular_imc(70, 0)

def test_imc_negativos():
with pytest.raises(ValueError):
calcular_imc(-70, 1.75)
```

Solucion Ejercicio 7

```
def test_fizzbuzz_fizz():
assert fizzbuzz(3) == "Fizz"
assert fizzbuzz(9) == "Fizz"

def test_fizzbuzz_buzz():
assert fizzbuzz(5) == "Buzz"
assert fizzbuzz(10) == "Buzz"

def test_fizzbuzz_fizzbuzz():
assert fizzbuzz(15) == "FizzBuzz"
assert fizzbuzz(30) == "FizzBuzz"

def test_fizzbuzz_numero():
assert fizzbuzz(1) == "1"
assert fizzbuzz(7) == "7"

def test_fizzbuzz_cero():
assert fizzbuzz(0) == "FizzBuzz" # 0 es multiplo de todo
```

Solucion Ejercicio 8

```
def test_agregar_producto():
inventario.clear()
agregar_producto("Laptop", 5, 1000)
assert "Laptop" in inventario
assert inventario["Laptop"]["cantidad"] == 5

def test_cantidad_negativa():
inventario.clear()
with pytest.raises(ValueError):
agregar_producto("Mouse", -1, 50)

def test_valor_total():
inventario.clear()
agregar_producto("Laptop", 2, 1000)
agregar_producto("Mouse", 10, 25)
assert obtener_valor_total() == 2250

def test_valor_total_vacio():
inventario.clear()
assert obtener_valor_total() == 0
```

Referencia Rapida - pytest

Concepto	Codigo / Comando
Instalar	`pip install pytest`
Ejecutar todos	`pytest`
Ejecutar verbose	`pytest -v`
Ejecutar un archivo	`pytest test_archivo.py`
Test basico	`assert suma(2,3) == 5`
Test excepcion	`with pytest.raises(ValueError):`
Test aproximado	`assert x == pytest.approx(0.3)`
Agrupar en clase	`class TestSuma:`
Patron	`Arrange -> Act -> Assert`

TIP: Guarda este documento como referencia. Te sera util en las proximas clases cuando empecemos OOP y necesites testear clases y objetos.