

Calidad

Ricardo Pérez López

IES Doñana, curso 2025/2026



Generado el 2025/07/10 a las 15:55:00

1. Pruebas

1. Pruebas

1.1. doctest

doctest

- ▶ `doctest` es una herramienta que permite realizar pruebas de forma automática sobre una función.
- ▶ Para ello, se usa la *docstring* de la función.
- ▶ En ella, se escribe una *simulación* de una pretendida ejecución de la función desde el intérprete interactivo de Python.
- ▶ La herramienta comprueba si la salida obtenida coincide con la esperada según dicta la *docstring* de la función.
- ▶ De esta forma, la *docstring* cumple dos funciones:
 - Documentación de la función.
 - Especificación de casos de prueba de la función.

```
# ejemplo.py
def factorial(n):
    """Devuelve el factorial de n, un número entero >= 0.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000
    >>> factorial(-1)
    Traceback (most recent call last):
      ...
    ValueError: n debe ser >= 0
    """

    import math
    if not n >= 0:
        raise ValueError("n debe ser >= 0")
    result = 1
    factor = 2
    while factor <= n:
        result *= factor
        factor += 1
    return result
```

```
$ python -m doctest ejemplo.py
$ python -m doctest ejemplo.py -v
Trying:
    [factorial(n) for n in range(6)]
Expecting:
    [1, 1, 2, 6, 24, 120]
ok
Trying:
    factorial(30)
Expecting:
    265252859812191058636308480000000
ok
Trying:
    factorial(-1)
Expecting:
    Traceback (most recent call last):
      ...
    ValueError: n debe ser >= 0
ok
1 items had no tests:
    ejemplo
1 items passed all tests:
   3 tests in ejemplo.factorial
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
```

1.2. pytest

pytest

- ▶ **pytest** es una herramienta que permite realizar pruebas automáticas sobre una función o programa Python, pero de una manera más general que con **doctest**.
- ▶ La forma más sencilla de usarla es crear una función llamada **test_⟨nombre⟩** por cada función **⟨nombre⟩** que queramos probar.
- ▶ Esa función **test_⟨nombre⟩** será la encargada de probar automáticamente el funcionamiento correcto de la función **⟨nombre⟩**.
- ▶ Dentro de la función **test_⟨nombre⟩**, usaremos la orden **assert** para comprobar si se cumple una determinada condición.
- ▶ En caso de que no se cumpla, se entenderá que la función **⟨nombre⟩** no ha superado dicha prueba.
- ▶ En Python 3, la herramienta se llama **pytest-3** y se instala mediante:

```
$ sudo apt install python3-pytest
```

```
# test_ejemplo.py
def inc(x):
    return x + 1

def test_respuesta():
    assert inc(3) == 5
```

```
$ pytest-3
===== test session starts =====
platform linux -- Python 3.8.5, pytest-4.6.9, py-1.8.1, pluggy-0.13.0
rootdir: /home/ricardo/python
collected 1 item

test_ejemplo.py F [100%]

===== FAILURES =====
_____ test_respuesta _____

    def test_respuesta():
>         assert inc(3) == 5
E         assert 4 == 5
E         + where 4 = inc(3)

test_ejemplo.py:7: AssertionError
===== 1 failed in 2.48 seconds =====
```

- ▶ `pytest` sigue la siguiente estrategia a la hora de localizar pruebas:
 - Si no se especifica ningún argumento, empieza a buscar recursivamente empezando en el directorio actual.
 - En esos directorios, busca todos los archivos `test_*.py` o `*_test.py`.
 - En esos archivos, localiza todas las funciones cuyo nombre empiece por `test`.

1.3. Desarrollo conducido por pruebas

Desarrollo conducido por pruebas

- ▶ El **desarrollo conducido por pruebas** o **TDD** (del inglés, *test-driven development*) es una práctica de ingeniería de software que agrupa otras dos prácticas:
 - Escribir las pruebas primero (*test first development*).
 - Refactorización (*refactoring*).
- ▶ Para escribir las pruebas generalmente se utilizan **pruebas unitarias** (*unit test*).
- ▶ El propósito del desarrollo guiado por pruebas es lograr un código limpio que funcione.
- ▶ La idea es que los requisitos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que el software cumple con los requisitos que se han establecido.

Ciclo de desarrollo

- ▶ En primer lugar se debe definir una lista de requisitos y después se ejecuta el siguiente ciclo de siete pasos:
 1. **Elegir un requisito:** Se elige el que nos dará mayor conocimiento del problema y que además sea fácilmente implementable.
 2. **Escribir una prueba:** Se comienza escribiendo una prueba para el requisito, para lo cual el programador debe entender claramente las especificaciones de la funcionalidad a implementar.
 3. **Verificar que la prueba falla:** Si la prueba no falla es porque el requisito ya estaba implementado o porque la prueba es errónea.

4. **Escribir la implementación:** Se escribe el código más sencillo que haga que la prueba funcione.
5. **Ejecutar las pruebas automatizadas:** Se verifica si todo el conjunto de pruebas se pasa correctamente.
6. **Refactorizar:** Se modifica el código para hacerlo más mantenible con cuidado de que sigan pasando todas las pruebas.
7. **Actualizar la lista de requisitos:** Se tacha el requisito implementado y se agregan otros nuevos si hace falta.

- ▶ Todo este ciclo se resume en que, por cada requisito, hay que hacer:
 1. **Rojo:** el test falla
 2. **Verde:** se pasa el test
 3. **Refactorizar:** se mejora el código

2. Bibliografía

Bibliografía

Python Software Foundation. n.d. "Sitio Web de Documentación de Python."
<https://docs.python.org/3>.