12-02-2026

TAREA 5.3 - Desarrollo de una función factorial usando TDD

Nombre y apellidos: Cristina Sandoval Laborde

Curso: 2ºDAM

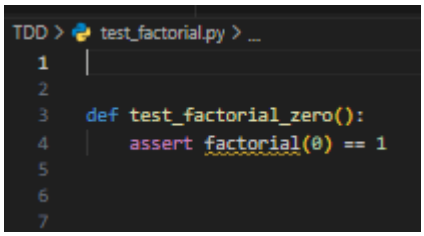
Asignatura: Desarrollo de interfaces

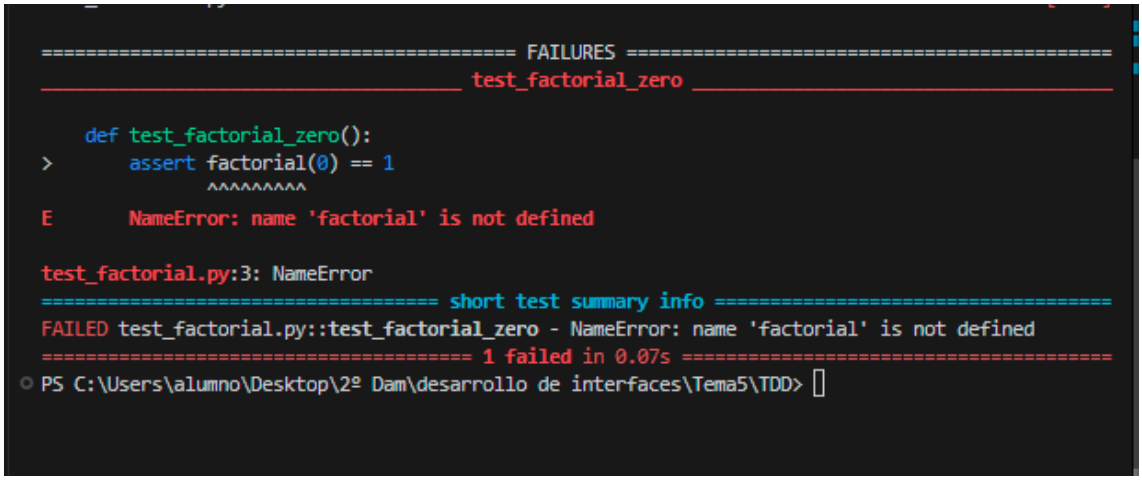
Índice

1.factorial(0) debe devolver 1

Fase RED- Test que falla

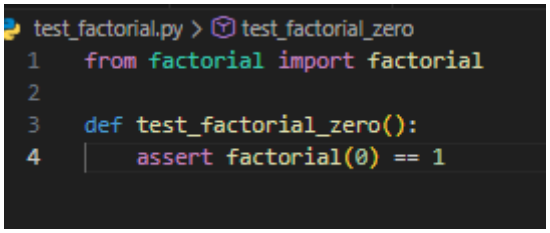
NameError: name “factorial” is not defined

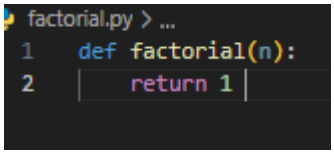


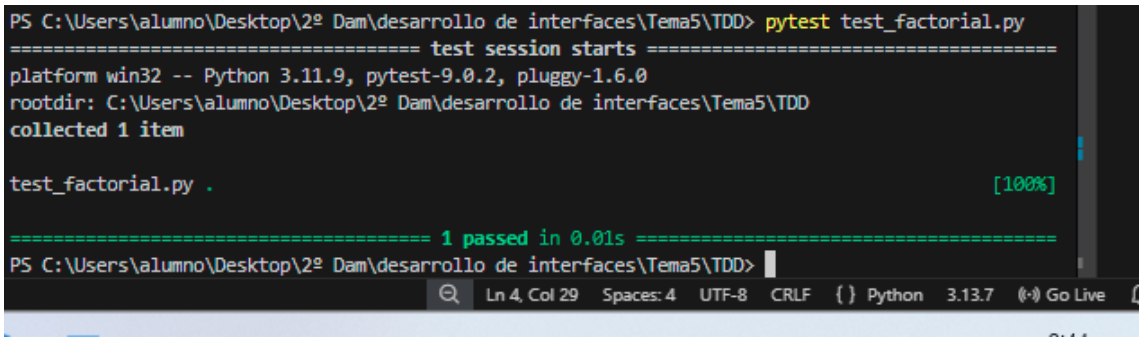


Fase GREEN – Mínimo código para que pase el test

Le añadimos el import en test_factorial y en factorial ponemos return 1

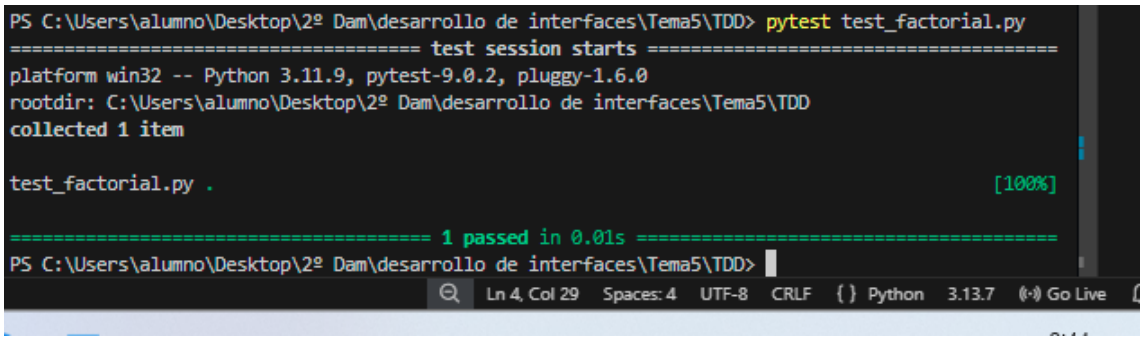






Fase Refactor

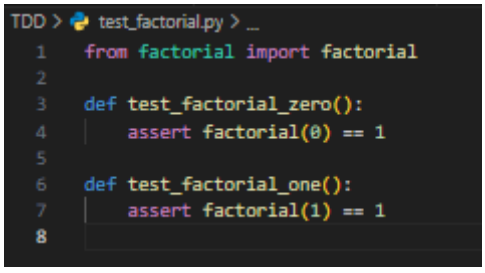
Se queda como lo tenemos porque el código es simple y no hay que refactorizar nada



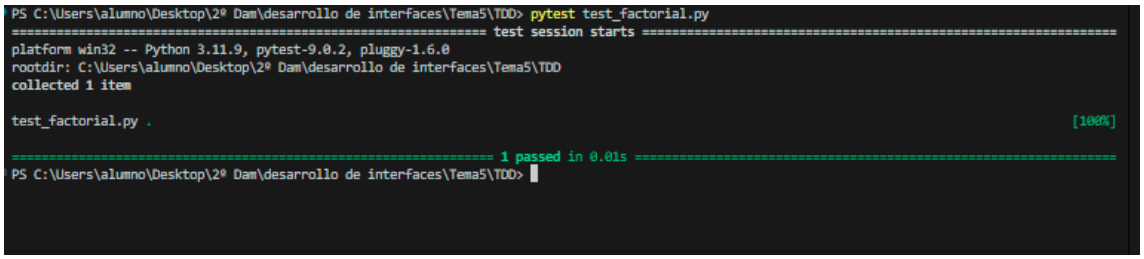
2.factorial(1) debe devolver 1

Fase Red

Añadimos el nuevo test debajo del anterior ahora lo ponemos == 1

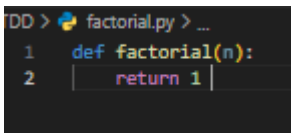


En este caso particular del TDD, el test pasa automáticamente



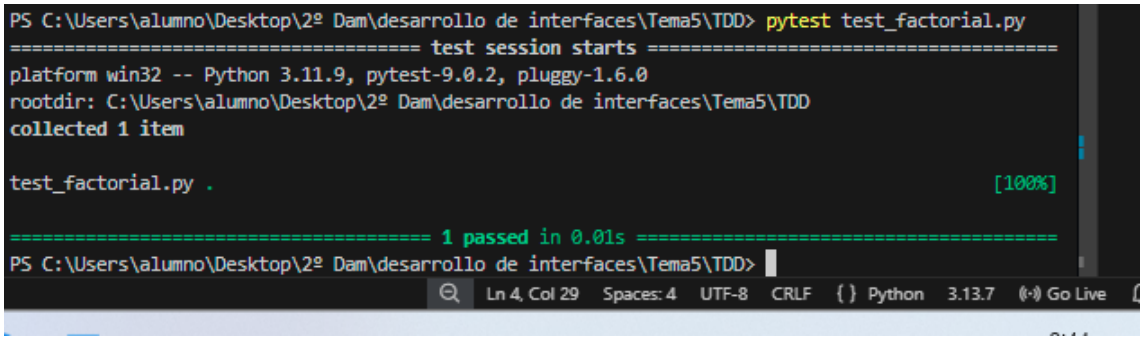
Fase green

Igual que la fase green anterior, porque como el test ya ha pasado con el código existente, no necesitamos añadir ni modificar nada en este momento



Fase refactor

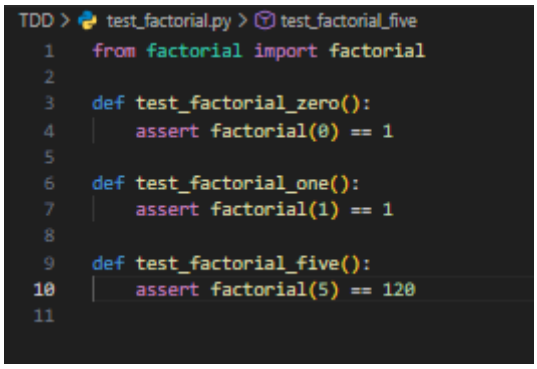
Se queda como lo tenemos porque el código es simple y no hay que refactorizar nada



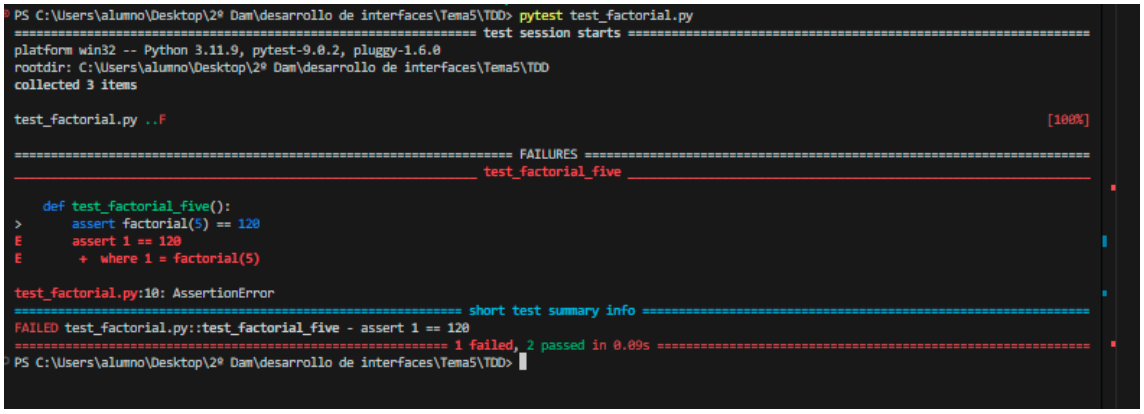
3.factorial(5) debe devolver 120

Fase red

Añadimos el nuevo test en test_factorial.py para comprobar el cálculo real de un factorial.

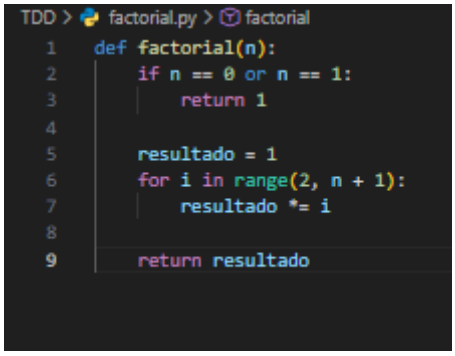


El test falla y la consola muestra un error en rojo: AssertionError: assert 1 == 120. Esto ocurre porque la función sigue devolviendo 1 para cualquier número, incluido el 5.

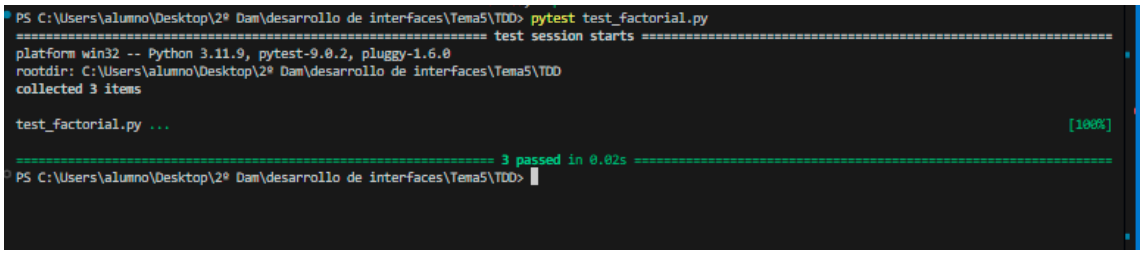


Fase green

Para que el test pase, debemos dejar de devolver un 1 fijo e implementar la lógica matemática real que calcule el factorial de cualquier número positivo.



La consola muestra que los tres tests (test_factorial_zero, test_factorial_one y test_factorial_five) pasan correctamente (en verde).



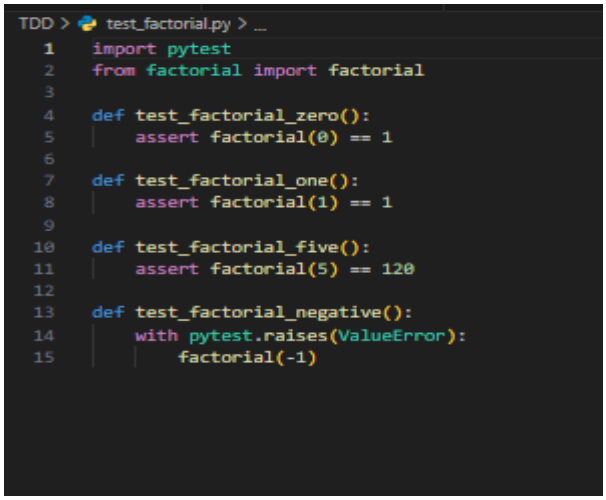
Fase refactor

Ya esta hecha al cambiar la clase factorial para quitar el error que daba al lanzar el pytest.

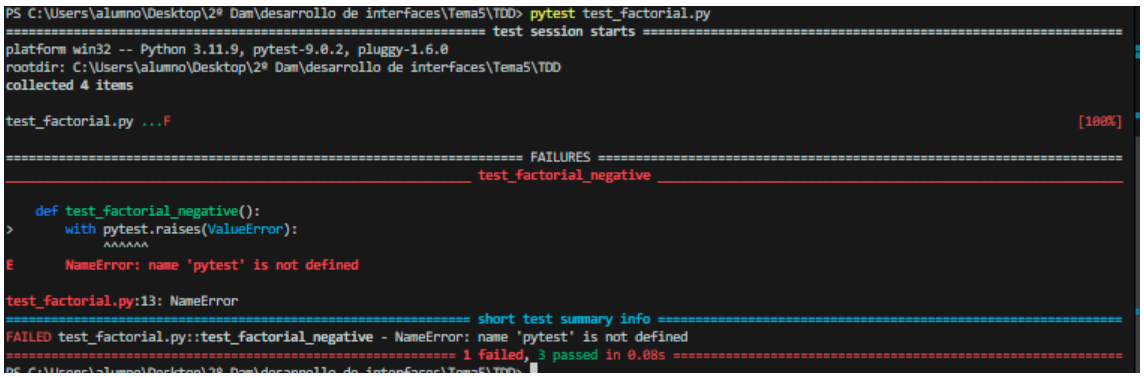
4.Si n es negativo, la función debe lanzar ValueError

Fase red

Para comprobar excepciones con pytest, es necesario importar la librería de pytest

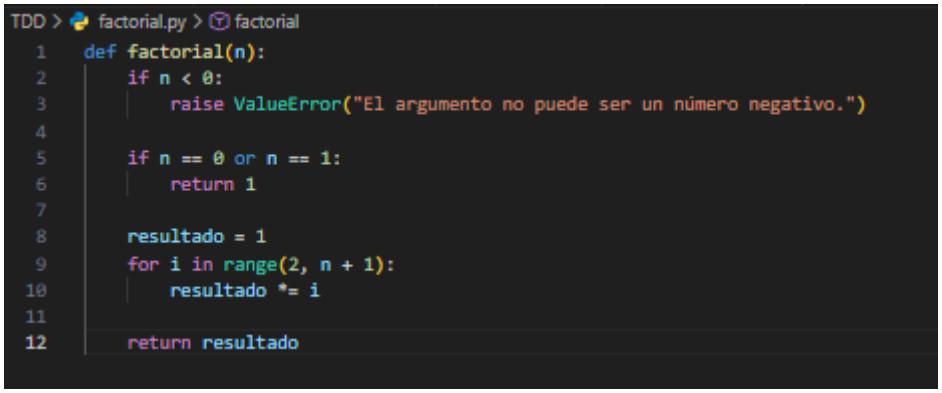


El test falla mostrando un error en la consola. Esto sucede porque la función actual no lanza ningún ValueError al recibir un número negativo

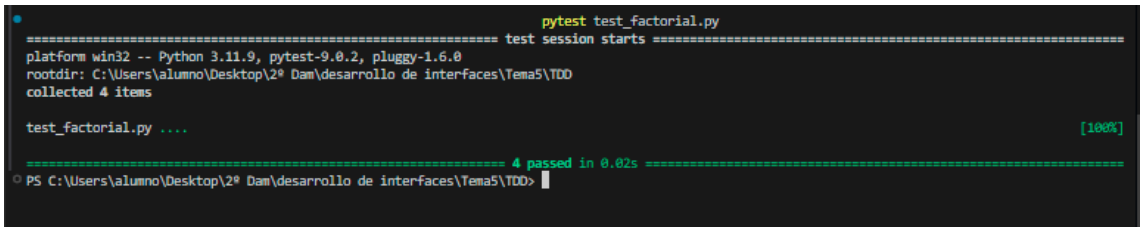


Fase green

Hay que modificar la función para detectar si el número ingresado es menor a cero y, en ese caso, lanzar la excepción requerida.



La consola muestra que los cuatro tests pasan correctamente.



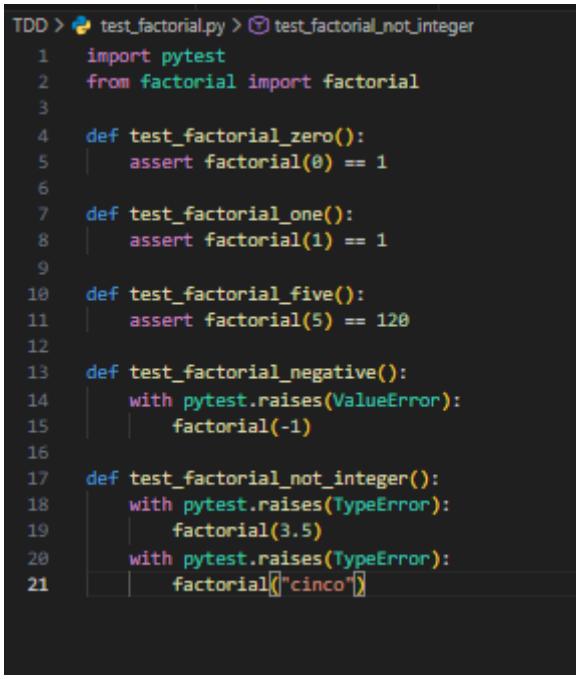
Fase refactor

El código cumple su propósito de forma clara. La validación del número negativo se ha colocado al principio de la función, lo cual es la práctica para evitar ejecutar código innecesario si los parámetros de entrada no son válidos.

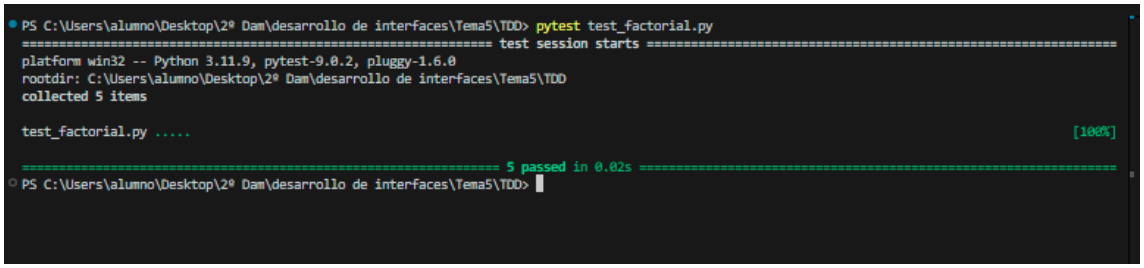
5.Si n no es un número entero, la función debe lanzar TypeError

Fase red

Añadimos el nuevo test para comprobar que ocurre cuando se envían tipos de datos no válidos.



Observamos que los test pasan

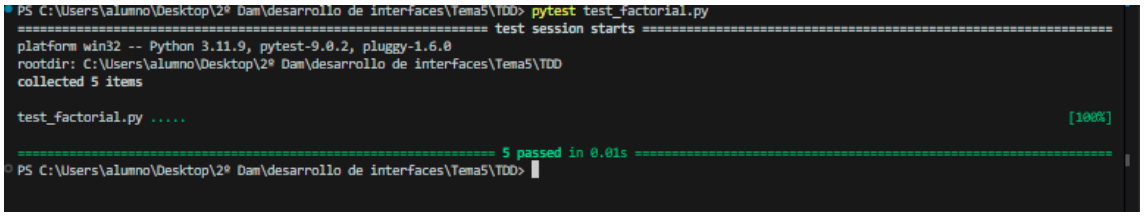


Fase green

Para asegurar que la función congrola el tipode dato desde el primer momento, añadimos un anueva clausula a factorial.py

```python
TDD > factorial.py > factorial
def factorial(n):
    if type(n) is not int:
        raise TypeError("El argumento debe ser un número entero.")

    if n < 0:
        raise ValueError("El argumento no puede ser un número negativo.")

    if n == 0 or n == 1:
        return 1

    resultado = 1
    for i in range(2, n + 1):
        resultado *= i

    return resultado
```

La consola muestra que los 5 tests pasan correctamente.



Fase refactor

No es necesario realizar más cambios, ya que cumple con los principios de código limpio y reunee todos los requisitos del enunciado.

Webgrafia

https://experts-deny-b9a.craft.me/y3eKOUPLrtEM6g