

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.....	1
2.factorial(1) debe devolver 1.....	2
3.factorial(5) debe devolver 120.....	3
4.Si n es negativo, la función debe lanzar ValueError.....	4
5.Si n no es un número entero, la función debe lanzar TypeError.....	6
Webgrafía.....	7

## 1.factorial(0) debe devolver 1

### Fase RED- Test que falla

NameError: name “factorial” is not defined

The terminal window shows the following code in `test_factorial.py`:

```
TDD > 🐍 test_factorial.py > ...
1
2
3 def test_factorial_zero():
4     assert factorial(0) == 1
5
6
7
```

And the resulting output:

```
=====
===== FAILURES =====
test_factorial_zero
=====
def test_factorial_zero():
>     assert factorial(0) == 1
      ^^^^^^^^
E     NameError: name 'factorial' is not defined

test_factorial.py:3: NameError
=====
===== short test summary info =====
FAILED test_factorial.py::test_factorial_zero - NameError: name 'factorial' is not defined
===== 1 failed in 0.07s =====
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> []
```

Fase GREEN – Mínimo código para que pase el test

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

The terminal window shows the code in `test_factorial.py` and `factorial.py`:

```
▶ test_factorial.py > 🐍 test_factorial_zero
1 from factorial import factorial
2
3 def test_factorial_zero():
4     assert factorial(0) == 1

▶ factorial.py > ...
1 def factorial(n):
2     return 1
```

And the resulting output:

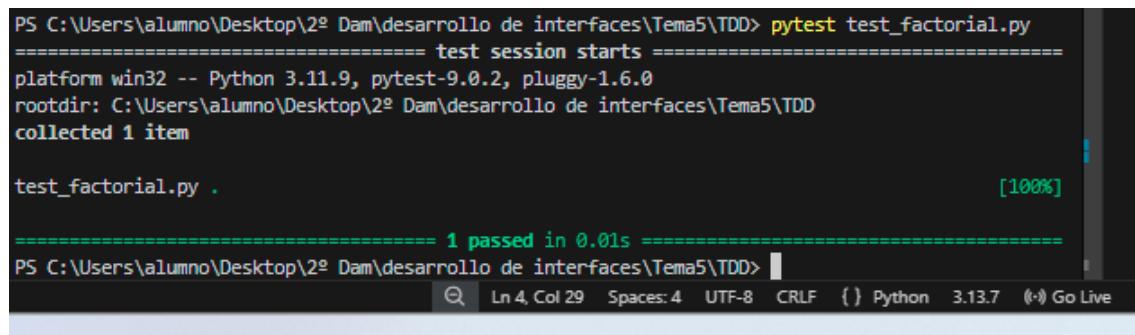
```
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> pytest test_factorial.py
=====
===== test session starts =====
platform win32 -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD
collected 1 item

test_factorial.py . [100%]

=====
===== 1 passed in 0.01s =====
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> []
```

## Fase Refactor

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



```
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> pytest test_factorial.py
=====
platform win32 -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD
collected 1 item

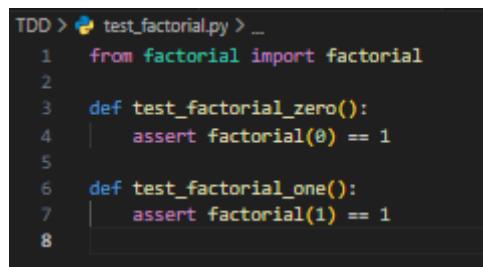
test_factorial.py .

===== 1 passed in 0.01s =====
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD>
```

## 2.factorial(1) debe devolver 1

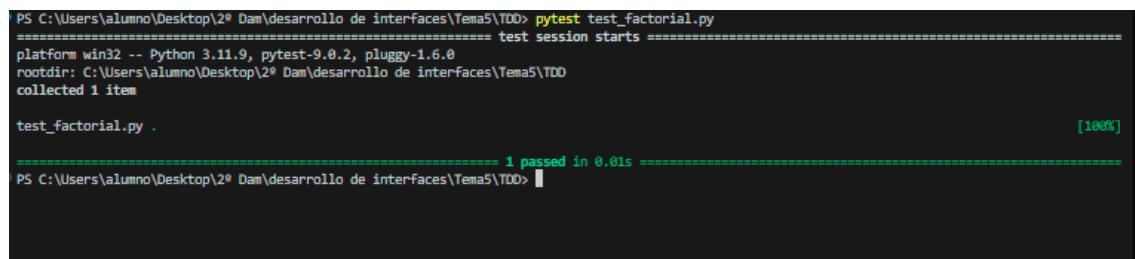
### Fase Red

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



```
TDD > 🐍 test_factorial.py > ...
1  from factorial import factorial
2
3  def test_factorial_zero():
4      assert factorial(0) == 1
5
6  def test_factorial_one():
7      assert factorial(1) == 1
8
```

En este caso particular del TDD, el test pasa automáticamente



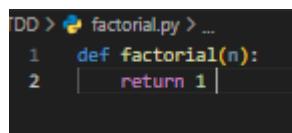
```
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> pytest test_factorial.py
=====
platform win32 -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD
collected 1 item

test_factorial.py .

===== 1 passed in 0.01s =====
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD>
```

### Fase green

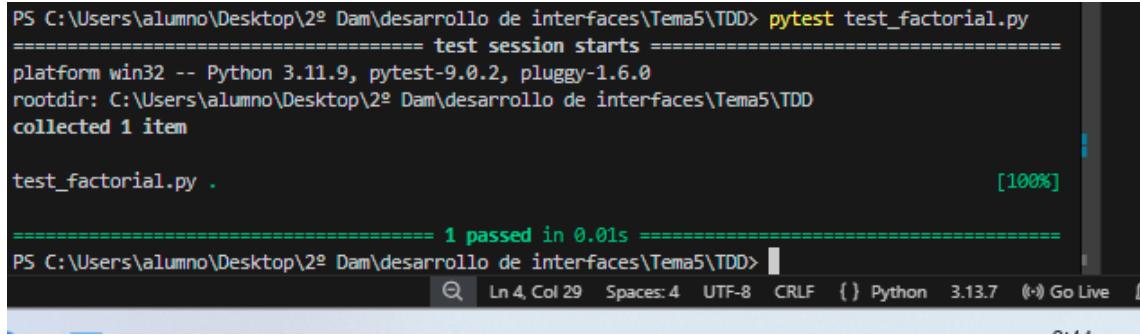
Igual que la fase green anterior, porque como el test ya ha pasado con el código existente, no necesitamos añadir ni modificar nada en este momento



```
TDD > 🐍 factorial.py > ...
1  def factorial(n):
2      return 1
```

## Fase refactor

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



```
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> pytest test_factorial.py
=====
test session starts =====
platform win32 -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD
collected 1 item

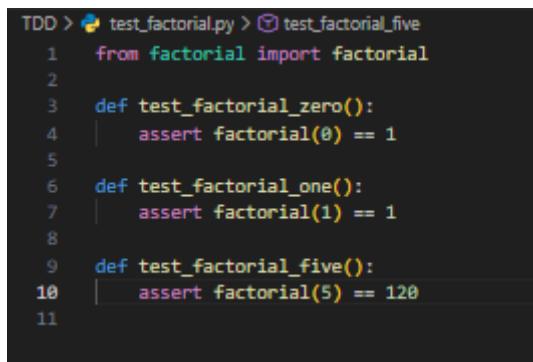
test_factorial.py . [100%]

===== 1 passed in 0.01s =====
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD>
```

## 3.factorial(5) debe devolver 120

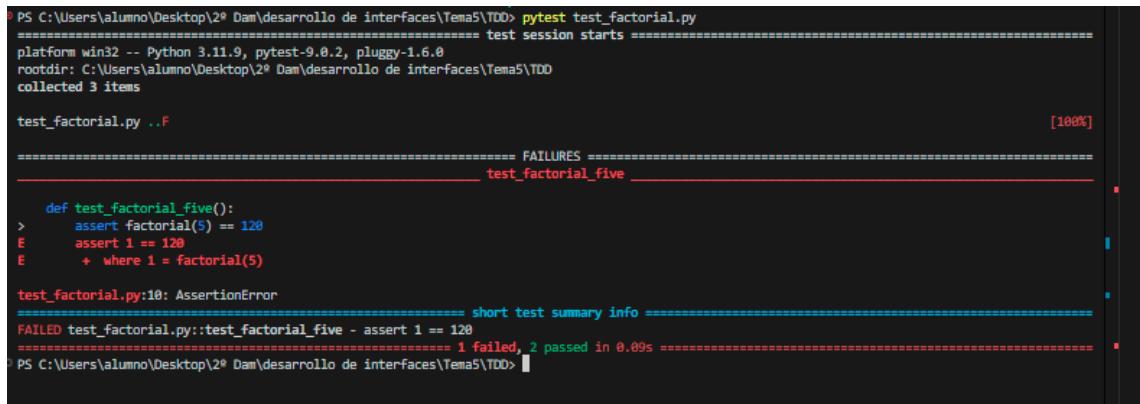
### Fase red

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



```
TDD > 🏃 test_factorial.py > 📄 test_factorial_five
1  from factorial import factorial
2
3  def test_factorial_zero():
4      assert factorial(0) == 1
5
6  def test_factorial_one():
7      assert factorial(1) == 1
8
9  def test_factorial_five():
10     assert factorial(5) == 120
11
```

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.



```
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> pytest test_factorial.py
=====
test session starts =====
platform win32 -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD
collected 3 items

test_factorial.py ..F [100%]

===== FAILURES =====
test_factorial.py:10: AssertionError
>     assert factorial(5) == 120
E     assert 1 == 120
E       + where 1 = factorial(5)

test_factorial.py:10: AssertionError
===== short test summary info =====
FAILED test_factorial.py::test_factorial_five - assert 1 == 120
===== 1 failed, 2 passed in 0.09s =====
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD>
```

## 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.

```
TDD > 🐍 factorial.py > ⚡ factorial
1  def factorial(n):
2      if n == 0 or n == 1:
3          return 1
4
5      resultado = 1
6      for i in range(2, n + 1):
7          resultado *= i
8
9      return resultado
```

La consola muestra que los tres tests (test\_factorial\_zero, test\_factorial\_one y test\_factorial\_five) pasan correctamente (en verde).

```
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> pytest test_factorial.py
=====
platform win32 -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD
collected 3 items

test_factorial.py ...
[100%]

===== 3 passed in 0.02s =====
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD>
```

## 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

```
TDD > 🐍 test_factorial.py > ...
1  import pytest
2  from factorial import factorial
3
4  def test_factorial_zero():
5      assert factorial(0) == 1
6
7  def test_factorial_one():
8      assert factorial(1) == 1
9
10 def test_factorial_five():
11     assert factorial(5) == 120
12
13 def test_factorial_negative():
14     with pytest.raises(ValueError):
15         factorial(-1)
```

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

```
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> pytest test_factorial.py
=====
platform win32 -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD
collected 4 items

test_factorial.py ...F [100%]

=====
===== FAILURES =====
test_factorial_negative
=====
def test_factorial_negative():
>     with pytest.raises(ValueError):
      ^^^^^^
E     NameError: name 'pytest' is not defined

test_factorial.py:13: NameError
===== short test summary info =====
FAILED test_factorial.py::test_factorial_negative - NameError: name 'pytest' is not defined
=====
===== 1 failed, 3 passed in 0.08s =====
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD>
```

## 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.

```
TDD > factorial.py > factorial
1 def factorial(n):
2     if n < 0:
3         raise ValueError("El argumento no puede ser un número negativo.")
4
5     if n == 0 or n == 1:
6         return 1
7
8     resultado = 1
9     for i in range(2, n + 1):
10        resultado *= i
11
12    return resultado
```

La consola muestra que los cuatro tests pasan correctamente.

```
●
=====
platform win32 -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD
collected 4 items

test_factorial.py .... [100%]

=====
===== 4 passed in 0.02s =====
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD>
```

## 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.

```
TDD > 🐍 test_factorial.py > ⚡ test_factorial_not_integer
  1 import pytest
  2 from factorial import factorial
  3
  4 def test_factorial_zero():
  5     assert factorial(0) == 1
  6
  7 def test_factorial_one():
  8     assert factorial(1) == 1
  9
 10 def test_factorial_five():
 11     assert factorial(5) == 120
 12
 13 def test_factorial_negative():
 14     with pytest.raises(ValueError):
 15         factorial(-1)
 16
 17 def test_factorial_not_integer():
 18     with pytest.raises(TypeError):
 19         factorial(3.5)
 20     with pytest.raises(TypeError):
 21         factorial(["cinco"])
```

Observamos que los test pasan

```
• PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> pytest test_factorial.py
=====
platform win32 -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD
collected 5 items
test_factorial.py ..... [100%]
=====
5 passed in 0.02s
○ PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD>
```

### Fase green

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

```
TDD > factorial.py > factorial
1  def factorial(n):
2      if type(n) is not int:
3          raise TypeError("El argumento debe ser un número entero.")
4
5      if n < 0:
6          raise ValueError("El argumento no puede ser un número negativo.")
7
8      if n == 0 or n == 1:
9          return 1
10
11     resultado = 1
12     for i in range(2, n + 1):
13         resultado *= i
14
15     return resultado|
```

La consola muestra que los 5 tests pasan correctamente.

```
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> pytest test_factorial.py
=====
platform win32 -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD
collected 5 items

test_factorial.py ..... [100%]

===== 5 passed in 0.01s =====
PS C:\Users\alumno\Desktop\2º Dam\desarrollo de interfaces\Tema5\TDD> |
```

## Fase refactor

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

## Webgrafía

<https://experts-deny-b9a.craft.me/y3eKOULrtEM6g>