

# Matchers más usados con Jest

DigitalHouse>



**Certified Tech  
Developer**

The Ultimate Degree

# Matchers en Jest

Jest usa los **matchers** para probar los diferentes valores que puede tener el código. Vamos a partir de un ejemplo de una porción de código desarrollada que permite realizar las operaciones matemáticas básicas y aplicarle diferentes matchers:

```
export const sumar = (a, b) => a + b;  
export const restar = (a, b) => a - b;  
export const multiplicar = (a, b) => a * b;  
export const dividir = (a, b) => a / b;
```

## .toBe

Usado para comparar valores primitivos (enteros, flotantes, etc.).

```
describe('Operaciones matemáticas', () => {  
  test('Realizamos la suma', () => {  
    expect(sumar(1,1)).toBe(2);  
  });  
  test('Realizamos la resta', () => {  
    expect(restar(1,1)).toBe(0);  
  });  
});
```

A yellow square containing the letters 'JS' in a large, bold, dark grey font, representing JavaScript.

# .toEqual

Usado para comparar objetos y todas sus propiedades:

```
describe('Common matchers', () => {  
  const datos = {  
    nombre: 'Persona 1',  
    edad: 10  
  }  
  
  const datos2 = {  
    nombre: 'Persona 1',  
    edad: 10  
  }  
  
  test('Comprobamos que los objetos son iguales', () => {  
    expect(datos).toEqual(datos2);  
  });  
});
```

A yellow square containing the letters 'JS' in a large, bold, dark grey font.

## **.toBeLessThan**

El valor es menor que:

```
test('Resultado menor que...', () => {  
    expect(restar(5,3)).toBeLessThan(3);  
});
```

JS

## **.toBeLessThanOrEqual**

El valor es menor o igual que:

```
test('Resultado menor o igual que...', () => {  
    expect(restar(5,3)).toBeLessThanOrEqual(2);  
});
```

JS

## .toBeGreaterThan

El valor es mayor que:

```
test('Resultado mayor que...', () => {  
  expect(sumar(5,5)).toBeGreaterThan(9);  
});
```

JS

## .toBeGreaterThanOrEqual

El valor es mayor o igual que:

```
test('Resultado mayor o igual que...', () => {  
  expect(multiplicar(2,5)).toBeGreaterThanOrEqual(10);  
});
```

JS

## Más ejemplos

A nuestro código de ejemplo le vamos a agregar otras operaciones de lógica, las cuales nos permitirán comparar valores booleanos, indefinidos y nulos.

```
export const sumar = (a, b) => a + b;  
export const restar = (a, b) => a - b;  
export const multiplicar = (a, b) => a * b;  
export const dividir = (a, b) => a / b;  
export const isNull = null;  
export const isFalse = false;  
export const isTrue = true;  
export const isUndefined = undefined;
```

## **.toBeTruthy**

El valor es verdadero:

```
test('Resultado True', () => {  
  expect(isTrue).toBeTruthy();  
});
```

JS

## **.toBeFalsy**

El valor es falso:

```
test('Resultado False', () => {  
  expect(isFalse).toBeFalsy();  
});
```

JS



## **.toBeUndefined**

El valor es undefined:

```
test('Resultado Undefined...', () => {  
    expect(isUndefined).toBeUndefined();  
});
```

JS

## **.toBeNull**

El valor es null:

```
test('Resultado Null...', () => {  
    expect(isNull).toBeNull();  
});
```

JS

# Arrays y strings

Para continuar, también veremos algunos matchers que se utilizan para trabajar con arrays y strings. Para eso a nuestro código le agregamos las siguientes líneas:

```
const provincias = ['Álava', 'Girona', 'Huelva', 'Jaén', 'La Rioja', 'Madrid', 'Navarra'];
const dias = ['Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes', 'Sabado', 'Domingo'];
const expReg = {
  responseOK: 'Response OK',
  responseFAIL: 'Response FAIL',
  email: 'test@test.com',
  telefono: '919784852'
}

export const arrProvincias = () => provincias;
export const arrDias = () => dias;
export const objExpReg = () => expReg;
```

## .toBeContain

Contiene el elemento dentro del array:

```
test('Madrid existe en el array', () => {  
  expect(arrProvincias()).toContain('Madrid');  
});
```

JS

## .toHaveLength (array)

El array tiene la longitud:

```
test('El array días tiene 7 elementos', () => {  
  expect(arrDias()).toHaveLength(7);  
});
```

JS

## .toHaveLength (string)

También podemos usar este matcher para ver la longitud de un string:

```
const exp = objExpReg();  
test('Comprobamos longitud del string', () => {  
  expect(exp.responseFAIL).toHaveLength(13);  
});
```

JS

## .toMatch

Comprueba que un texto coincida con una expresión regular:

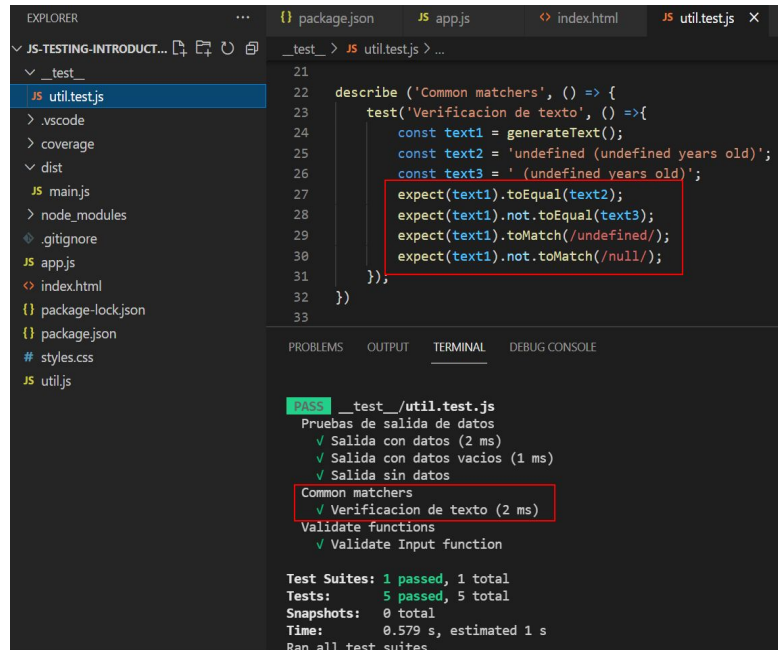
```
const exp = objExpReg();  
test('Comprobamos formato del email', () => {  
  expect(exp.email).toMatch(/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/);  
});
```

JS

# ¡Sigamos aprendiendo!

Si queremos conocer más detalles o conocer toda la lista de matchers que se puede usar con Jest, podemos verlo en la documentación oficial:

<https://jestjs.io/docs/expect>.



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files like `package.json`, `app.js`, `index.html`, and `util.js`. The code editor shows the content of `util.js`, which includes a `describe` block for 'Common matchers' and a `test` block for 'Verificacion de texto'. The test block contains several `expect` calls using Jest matchers. The bottom panel shows the output of the Jest test run, indicating that all tests passed.

```
21
22 describe ('Common matchers', () => {
23   test('Verificacion de texto', () =>{
24     const text1 = generateText();
25     const text2 = 'undefined (undefined years old)';
26     const text3 = ' (undefined years old)';
27     expect(text1).toEqual(text2);
28     expect(text1).not.toEqual(text3);
29     expect(text1).toMatch(/undefined/);
30     expect(text1).not.toMatch(/null/);
31   });
32 })
33
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

**PASS** \_\_test\_\_/util.test.js

Pruebas de salida de datos

- ✓ Salida con datos (2 ms)
- ✓ Salida con datos vacios (1 ms)
- ✓ Salida sin datos

Common matchers

- ✓ Verificacion de texto (2 ms)

Validate functions

- ✓ Validate Input function

Test Suites: 1 passed, 1 total

Tests: 5 passed, 5 total

Snapshots: 0 total

Time: 0.579 s, estimated 1 s

Ran all test suites.

DigitalHouse>  
Coding School