



# Introducción a JavaScript Unit Testing

**Objetivo:** Entender qué son las pruebas unitarias, por qué son útiles y cómo aplicarlas con ejemplos simples.

# ¿Qué es el Unit Testing?



## Prueba la "unidad" más pequeña

Se enfoca en componentes aislados de tu código, como funciones o módulos, es decir, verifica que una parte pequeña de tu código (una función, por ejemplo) se comporte como esperas.



## Verifica comportamiento esperado

Asegura que cada parte del código funcione exactamente como se diseñó.



## Se ejecutan de forma automática

Permite una integración continua y detección temprana de problemas sin intervención manual.



# ¿Por Qué Testear? Beneficios Clave



## Reduce Errores

Captura bugs en fases tempranas del desarrollo, ahorrando tiempo y costos.



## Documenta Código

Los tests sirven como ejemplos claros del uso y propósito de cada unidad.



## Facilita Refactorización

Permite realizar cambios con confianza, sabiendo que las pruebas detectarán regresiones.



## Genera Confianza

Aumenta la seguridad en el equipo y en la estabilidad del proyecto.

# Conceptos Clave del Testing



## Test (Prueba)

Un escenario específico para validar una funcionalidad.

## Assertion (Aserción)

Verifica un resultado esperado. Ejemplo:  
`expect(resultado).toBe(5).`

## Test Runner

La herramienta que ejecuta tus pruebas. Ej: Jest.

## Test Suite

Colección organizada de pruebas relacionadas entre sí.



# Herramientas: Conoce Jest

Jest es un framework (librería) de testing de JavaScript desarrollado por Meta. Es conocido por su facilidad de configuración y su enfoque en la simplicidad.

- Fácil de configurar y usar (CLI simple)
- Ideal para principiantes en JS
- Incluye test runner, assertions y mocks



Jest se integra perfectamente con proyectos React, Vue, Angular y Node.js.

# Test Runner

Significa: “el que corre los tests”.

Jest automáticamente:

Busca todos los archivos que terminan en .test.js

Ejecuta cada prueba

Muestra en pantalla si pasó o falló

```
bash
npm test
```

```
bash
npx jest nombreDelArchivo.test.js
```

```
$ npm test
> 13-js-unit-testing@1.0.0 test
> jest

PASS  ./mayor.test.js
PASS  ./contarVocales.test.js
PASS  ./invertirCadena.test.js
PASS  ./esPar.test.js
PASS  ./sum.test.js
```

```
Test Suites: 5 passed, 5 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        1.033 s
Ran all test suites.
```

# ASSERTIONS

Son afirmaciones sobre lo que esperas que pase.

En **código**, usamos cosas como:

```
js

function suma(a, b) {
  return a + b;
}

const resultado = suma(2, 2);

if (resultado === 4) {
  console.log("✓ Pasa");
} else {
  console.log("✗ Falla");
}
```

En **test**, usamos cosas como:

```
js

test('suma 2 + 2 = 4', () => {
  expect(2 + 2).toBe(4);
});
```

Aquí estás afirmando que esperas que  $2 + 2$  sea igual a 4.

Ej: Si pides una hamburguesa sin cebolla, esperas que venga sin cebolla. Esa es tu "assertion". Si trae cebolla, el test falla.

# MOCKS

"Mock" en inglés significa "fingir" o "simular".

En programación, los mocks son funciones falsas que simulan otras reales.

Las usamos para probar cosas sin necesidad de que ocurran de verdad.

*Ejemplo.* Supongamos que haces una app que envía emails. No quieres que cada test mande un correo real, ¿verdad?

En su lugar, creas una función falsa (mock) que simula que manda el correo.

Así puedes probar: ¿Se llamó la función?, ¿Cuántas veces?, ¿Con qué datos?

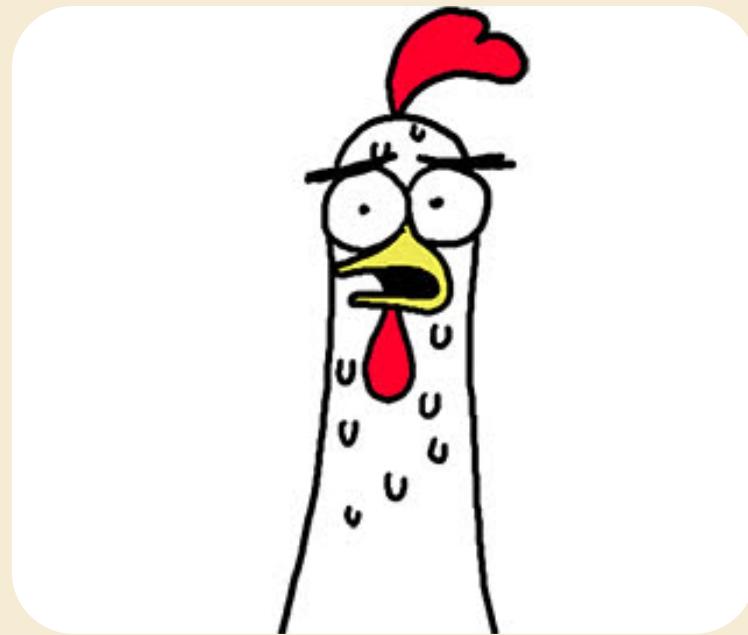
```
js

const enviarCorreo = jest.fn();

test('enviarCorreo se llamó una vez', () => {
  enviarCorreo();
  expect(enviarCorreo).toHaveBeenCalledTimes(1);
});
```

*Aquí no se manda ningún email, solo se simula que lo mandamos, y comprobamos que se haya usado la función.*

# ¿EMPEZAMOS?



# PASO 0. Requisitos Previos

- Tener instalado **Node.js**

## ¿Qué es **Node.js**?

es un programa que permite ejecutar JavaScript fuera del navegador. Es como un “motor” que va a interpretar y ejecutar tu código, y también te da acceso a una herramienta llamada npm.

## ¿Qué es **npm**?

npm significa "Node Package Manager". Es un sistema para descargar e instalar librerías de JavaScript, como Jest, que te ayudan a programar sin tener que escribir todo desde cero.

# ¿Comprobamos si tenemos Node.js?

1.-Abre tu terminal

2.- Escribe el siguiente comando:

```
bash
```

```
node -v
```

3.-Si aparece algo como v18.17.1 o similar, ¡tienes Node.js instalado!

4.- Escribe el siguiente comando:

```
bash
```

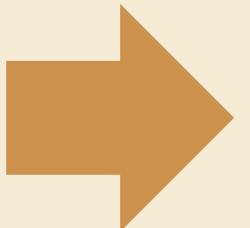
```
npm -v
```

5.-Si te aparece algo como 9.6.7, ¡tienes npm instalado!

# ¿NO LO TIENES?



**NO PASA NADA, HAGAMOSLO JUNTAS!!!!**



# INSTALAR NODE.JS

1. Ve al sitio oficial de [Node.js](#)
2. Descarga la versión LTS (Long Term Support).  
Es la más estable y recomendada.
3. Instálala como cualquier otro programa (haz clic en "Siguiente" hasta que termine).
4. Vuelve a abrir la terminal y repite los comandos **node -v** y **npm -v** para comprobar que está bien instalado.

# PASO 1. Crear el Proyecto

1. Creamos una carpeta para la práctica dentro de píldoras:

*Ej: js-test*

2. Vamos a VSCode

*code .*

# PASO 2. Inicializar el proyecto con npm

Esto crea un archivo especial llamado package.json, que guarda toda la información del proyecto (como sus dependencias, scripts, etc.).

```
bash
npm init -y
```

*El -y acepta todas las opciones por defecto para hacerlo más rápido.*

# ¿Qué ha ocurrido? ¿Qué es package.json?

```
json                                         ⌂ Copiar ⌋ Editar

{

  "name": "13-js-unit-testing", // Nombre del proyecto
  "version": "1.0.0",           // Versión inicial
  "main": "index.js",          // Archivo principal (puedes ignorarlo si no haces un módulo)
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],                // Palabras clave (útil en proyectos públicos)
  "author": "",                 // Puedes poner tu nombre
  "license": "ISC",              // Tipo de licencia (ISC está bien si no sabes)
  "description": "",            // Descripción breve del proyecto
  "devDependencies": {
    "jest": "^30.0.2"           // Jest instalado como dependencia de desarrollo
  }
}
```

Es un archivo de configuración que describe tu proyecto.

Es como el “DNI” de tu proyecto Node.js.



→ Información básica (nombre, versión, autor...).  
Qué librerías (dependencias) necesita tu proyecto.  
Qué comandos puedes ejecutar con npm run.

# PASO 3. Instalar Jest

herramienta que nos permite escribir pruebas para comprobar que nuestro código funciona correctamente.

bash

```
npm install --save-dev jest
```

*npm install: estás instalando una librería.*

*--save-dev: significa que esta librería es solo para desarrollo (no la usaremos en producción)*  
*jest: es el nombre de la librería.*

# PASO 4. ¿Se instaló bien Jest?

1. Mira en la carpeta: debe haberse creado una nueva llamada node\_modules
2. En tu archivo package.json debe aparecer algo así:

```
json  
  
"devDependencies": {  
  "jest": "^29.0.0"  
}
```

3. Puedes ejecutar el siguiente comando para comprobarlo :

```
bash  
  
npx jest --version
```

# PASO 5: Configurar el script para ejecutar pruebas

Abre el archivo package.json y busca esta parte:

```
json

"scripts": {
  "test": "echo \\"Error: no test specified\\" && exit 1"
}
```

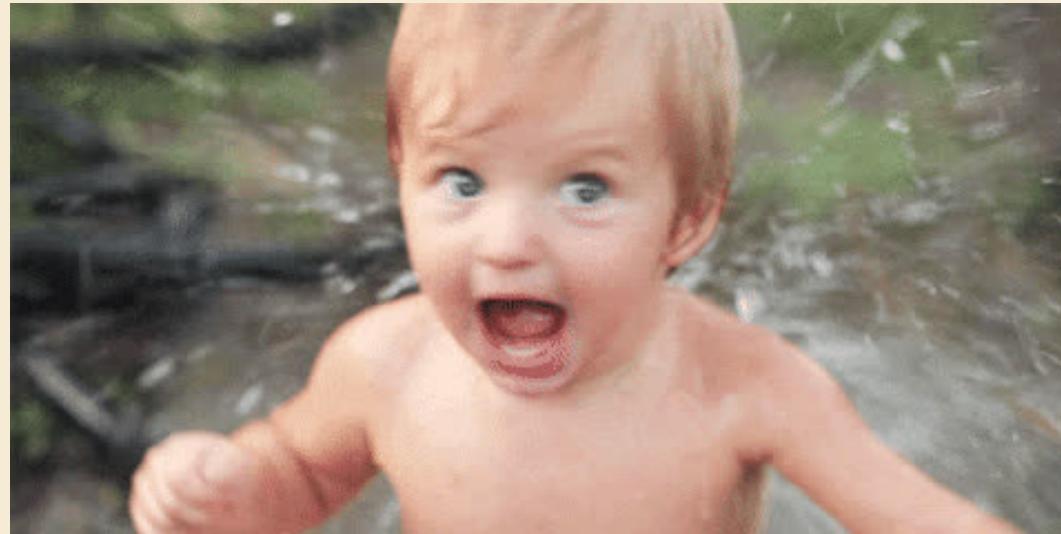
Cambiala por:

```
json

"scripts": {
  "test": "jest"
}
```

*Esto nos permitirá correr las pruebas escribiendo simplemente npm test*

**YA ESTAMOS LISTAS PARA....**



**TESTEAR!!!**

## EJERCICIO1.

1.Crea un archivo llamado *sum.js* con esta función:

```
js

function sum(a, b) {
    return a + b;
}

module.exports = sum;
```

*function sum(a, b) {...} → Es una función normal en JavaScript. Recibe dos valores, a y b.*

*return a + b → Devuelve el resultado de sumarlos.*

*module.exports = sum; → Esta línea "exporta" la función para que otro archivo (como el de pruebas) pueda importarla y usarla.*

# EJERCICIO1.

2.Crea otro llamado `sum.test.js` :

```
js

const sum = require('./sum');

test('suma 1 + 2 debe ser igual a 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

1. `const sum = require('./sum');`

Importamos la función `sum` desde el archivo `sum.js` para poder probarla.

`require('./sum')` le dice a `Node.js`: "Busca el archivo llamado `sum.js` que está en la misma carpeta".

2. `test('suma 1 + 2 debe ser igual a 3', () => { ... })`

Este es un bloque de prueba de `Jest`.

`test` es una función que viene con `Jest` y que se usa para definir una prueba.

El primer parámetro ('suma 1 + 2 debe ser igual a 3') es una descripción de lo que estamos probando.

El segundo parámetro es una función que contiene la lógica de la prueba.

3. `expect(sum(1, 2)).toBe(3);`

`expect(...)`: Es una forma de decir "esperamos que el resultado sea..."

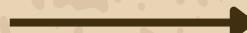
`sum(1, 2)`: Llamamos a la función `sum` con los valores 1 y 2.

`.toBe(3)`: Esperamos que el resultado sea exactamente igual a 3.

# EJERCICIO1.

3.Escribe en la terminal:

```
bash  
  
npm test
```



```
$ npm test  
  
> 13-js-unit-testing@1.0.0 test  
> jest  
  
PASS ./sum.test.js  
  ✓ suma 1 + 2 debe ser igual a 3 (3 ms)  
  
Test Suites: 1 passed, 1 total  
Tests:       1 passed, 1 total  
Snapshots:   0 total  
Time:        0.532 s  
Ran all test suites.
```



# MUCHAS GRACIAS POR LA ATENCIÓN!

Instalación Node.js

Documentación oficial-JEST

Introducción al Testing desde cero con JEST

