

## Ejecutar TEST PASO A PASO

1. El primer paso al no tener el package.json es crearlo para eso vamos a ejecutar el comando **npm init -y**. El package.json es necesario ya que aquí quedan guardado todos los scripts y configuraciones que vamos a utilizar dentro del proyecto
2. Tenemos que instalar jest, para eso ejecutamos: **npm install --save-dev jest**. Recordar que por cada proyecto hay que instalar jest, ya que el ámbito del mismo es el proyecto. Se instala en el entorno de **dev** porque es una dependencia que no hace falta en producción.
3. En el archivo package.json dentro de "script" ingresar: **"test": "jest"** si va a trabajar/testear cosas del DOM entonces ingresar **"test": "jest --env=jsdom"** Para revisar cobertura dentro de "script" ingresar: **"test:coverage": "jest --coverage"** y si es con el DOM **"test:coverage": "jest --coverage --env=jsdom"**

```
"scripts": {  
  "test": "jest --env=jsdom",  
  "test:coverage": "jest --coverage --env=jsdom"  
}
```

**--env=jsdom** nos sirve para acceder a otros elementos del DOM que no se encuentran en el archivo a testear. Generalmente se lo usa para trabajar con elementos HTML

Notas: En caso de que no funcione hay una alternativa

```
"jest":{  
  "testEnvironment": "jsdom"  
}
```

4. Crear la carpeta **\_\_test\_\_** en la raíz del proyecto (opcional, pero es una buena práctica)
5. Dentro de la carpeta creada en el paso anterior, crear el archivo **NombreDelArchivoAtestear.test.js** (para que JEST reconozca los archivos de prueba deben terminar en **.test.js**)

En el ejemplo de la calculadora: **funcionalidad.test.js**

6. Exportar el código/funciones a probar:

**exports.function = () =>{function();}**

En el ejemplo de la [calculadora](#):

```
exports.limpiar= () =>{  
  limpiar();  
}
```

RESOLUCION  
DEL EJ:

```

exports.init= () =>{
    init();
}
exports.resetear= () =>{
    resetear();
}
exports.resetear= () =>{
    resetear();
}

```

7. Importar en el archivo que creamos dentro de la carpeta `__test__` el código/funciones a probar.

**`const {nombre_funcion} = require('../archivoAtestear.js')`**

En el ejemplo de la calculadora:

```

const fs = require("fs");
document.body.innerHTML = fs.readFileSync("./calculadora.html");
const { init, limpiar, resetear, resolver} = require('../funcionalidad');

```

En este ejemplo de la calculadora se utilizó el `readFileSync()`, para poder acceder a cada uno de los elementos html ya creados en `calculadora.html`

**En estos test lo que hacemos es simular el DOM y crear objetos que nos van a servir para testear las funciones**

8. En el anterior archivo escribir las pruebas a ejecutar.

```

describe('Prueba para function init', ()=>{
    test('Init', ()=>{
        init()
    })
    test('evento uno', () => {
        limpiar();
        init();
        uno.onclick() ;
        expect( resultado.textContent).toBe("1");
    })
}

```

En la función **`init()`** lo que queremos validar es el comportamiento de cada número y operando al hacer click en ellos. La lógica es la siguiente:

- **`limpiar()`** llamamos a la función limpiar sino cada vez que ejecutemos el test de un número va a quedar en `resultado.textContent` el valor de ese número. Tengan en cuenta que funciona como una calculadora, si hago click en 1 y después en 2 se me concatena ambos números formando el 12 (hagan pruebas quitando la función `limpiar()` para validarlo)
- **`init()`** llamamos a la función que queremos testear
- **`uno.onclick()`** llamamos al evento que hace click en el botón uno

- `expect(resultado.textContent).toBe("1")` validamos que luego de hacer click en uno, el objeto resultado tenga como contenido realmente 1

Nota: Gracias al DOM, podemos trabajar con los diferentes elementos HTML dentro del archivo calculadora.HTML, es por ello, que vamos llamando a las funciones y validando los datos dentro de esos elementos. Este es un ejemplo diferente al de lista, al cual enviabamos parámetros. **En este caso simulamos los elementos y llamamos a las funciones para ver el comportamiento de esos elementos.**

```
describe('Pruebas de limpiar', () => {  
  test('resultado vacío', () => {  
    limpiar();  
    expect(resultado.textContent).toBe("");  
  });  
});
```

**limpiar()** la función limpiar borra el contenido del objeto resultado, entonces tenemos que validar que después de ejecutarla el resultado sea vacío ""

Y así seguimos la lógica para las demás funciones....

9. Ejecuta pruebas con instrucciones:  
"npm run test" para las suites o pruebas individuales  
"npm run test:coverage" para cobertura