**Para este módulo necesitamos lo siguiente:**

1. **Instalar navegadores (Chrome/Firefox)**
2. **Editor de código (Visual Studio Code) :**

[**https://code.visualstudio.com/Download**](https://code.visualstudio.com/Download)

Los editores de código son programas que nos ayudan a gestionar el código fuente de nuestros proyectos.Lo que hacen los editores de código es facilitarnos la tarea, de maneras distintas según cada uno de ellos. Por ejemplo, es muy habitual que tenga funciones de autocompletar, de gestión de versiones, que cambien los colores de cada parte del texto según el contexto, que comprueben la sintaxis del código, o que a partir de plugins se pueda ejecutar el código directamente para probarlo.

1. **Node.JS (Nos permite ejecutarcodigo de JavaScript en nuestra computadora)**

[**https://nodejs.org/en/download/**](https://nodejs.org/en/download/) **>** Version LTS

**Actividad Clase 13 A -Calculadora**

**Resuelto**

**function suma(a,b){**

**if (isNaN(a) || isNaN(b)) {**

**return false;**

**}**

**else{**

**return a+b;**

**}**

**}**

**function resta(a,b){**

**if (isNaN(a) || isNaN(b)) {**

**return false;**

**}**

**else{**

**return a-b;**

**}**

**}**

**function multiplicacion(a,b){**

**if (isNaN(a) || isNaN(b)) {**

**return false;**

**}**

**else{**

**return a\*b;**

**}**

**}**

**function division(a,b){**

**if (isNaN(a) || isNaN(b) || b==0 || a==null || b==null) {**

**return false;**

**}**

**else{**

**return a/b;**

**}**

**}**

**function calcular(operando1, operando2, op){**

**let result = 0;**

**switch(op){**

**case "suma":**

**result = suma(operando1,operando2);**

**break;**

**case "resta":**

**result = resta(operando1,operando2);**

**break;**

**case "division":**

**result = division(operando1,operando2);**

**break;**

**case "multiplicacion":**

**result = multiplicacion(operando1,operando2);**

**break;**

**default:**

**console.log("Error: No es una operación válida");**

**}**

**if (!result){**

**console.log("Error: no es posible realizar la operación: " + op)**

**}**

**else{**

**console.log(" Operando1: " + operando1 +**

**"\n Operando2: " + operando2 +**

**"\n Operacion: " + op +**

**"\n Resultado: " + result);**

**}**

**}**

**// let operacionSuma = calcular(2,3,"suma");**

**// let operacionResta = calcular(10,3,"resta");**

**// let operacionMultiplicacion = calcular(5,3,"multiplicacion");**

**// let operacionDivision = calcular(20,4,"division");**

**let operacionSuma = calcular(2,null,"suma");**

**let operacionResta = calcular("a",3,"resta");**

**let operacionMultiplicacion = calcular(5,2.3,"multiplicacion");**

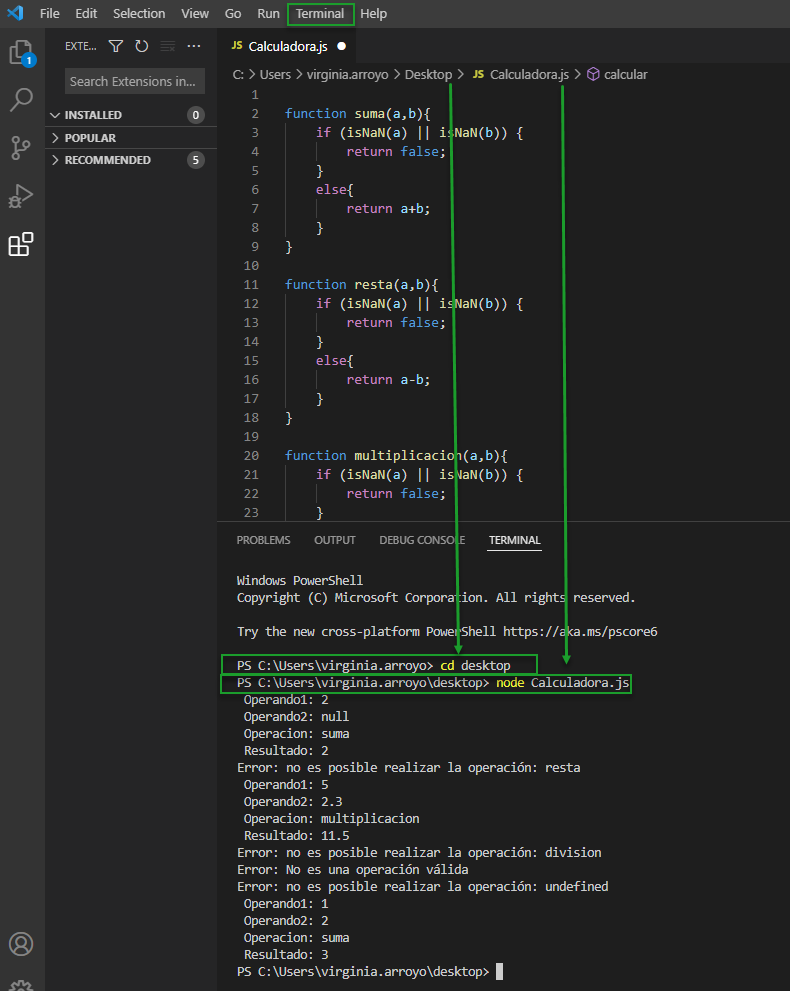
**let operacionDivision = calcular(20,0,"division");**

**let operacionSinDatos = calcular();**

¿Dónde ver los resultados?

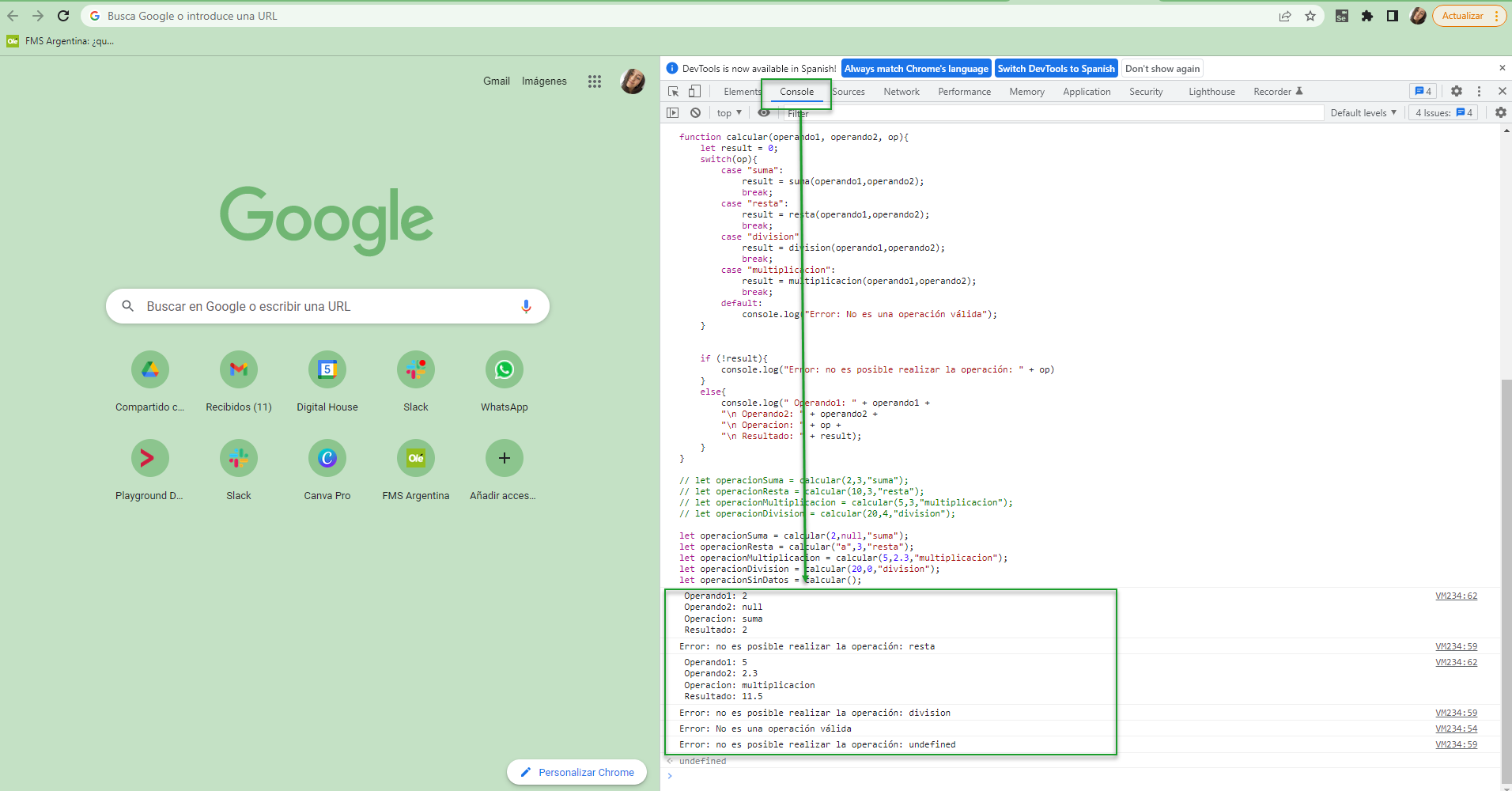
Opción **Visual Studio Code -Node.js**

1. Seleccionar terminal > New Terminal
2. Ir a la carpeta donde tenemos nuestro archivo
3. Tirar el comando node {nombre de nuestro archivo}

****

**Opción DevTools**

1. Abrir Chrome
2. Abrir el devTools > Fn+F12
3. Seleccionar la opcion “Consola”
4. Copiar el código > presionar Enter

****

**Actividad Pokemon**

**Resuelto**

**const jsonData= require('./response\_ability.json');**

**function search(textToSearch)**

**{**

**//var textToSearch='static';**

**for(let pokeName of jsonData.results){**

**if (pokeName.name == textToSearch) {**

**//pm.expect(pokeName.name).to.eql(textToSearch);**

**console.log("ID found: " + pokeName.name);**

**console.log("URLd: " + pokeName.url);**

**break;**

**}**

**else**

**{**

**console.log("Not found: " + pokeName.name);**

**}**

**}**

**}**

**search('static');**

**let poke1 = jsonData.results.find(a => a.name === 'static');**

**console.log("ID found with find in array: " + poke1.name);**

**console.log("URL: " + poke1.url)**

**console.log(typeof(poke1.name));**

En este ejercicio utilizamos la función require

**const jsonData= require('./response\_ability.json');**

Esta es una declaración de node.js incorporada y se usa más comúnmente para incluir módulos de otros archivos separados.

En este ejercicio lo utilizamos para incluir nuestro archivo sin necesidad de copiar todo el contenido. Es beneficioso, porque en caso de hacer una modificación, lo hacemos en el archivo Json.

**let poke1 = jsonData.results.find(a => a.name === 'static');**

El método find ejecuta la función callback una vez por cada índice del array hasta que encuentre uno en el que el callback devuelva un valor verdadero. Si es así, find devuelve inmediatamente el valor del elemento. En caso contrario, find devuelve undefined

**Debugging**

Para hacer debugging en el ejercicio de la calculadora con la opción

Opción **Chrome**

1. Abrimos la siguiente carpeta <https://drive.google.com/drive/folders/1HXIBzXFYxe09RVe3umFO4z1UOYsN8sf8>
2. Ejecutamos el archivo html en el navegador de chrome
3. Abrir el devTools > Fn+F12
4. Seleccionar la opción “Source”
5. Seleccionar el archivo funcionalidad
6. Marcar el breakpoint en la línea correspondiente (En este caso en la opción function resolver()
7. Y recorremos linea por linea hasta encontrar el error

Resultado

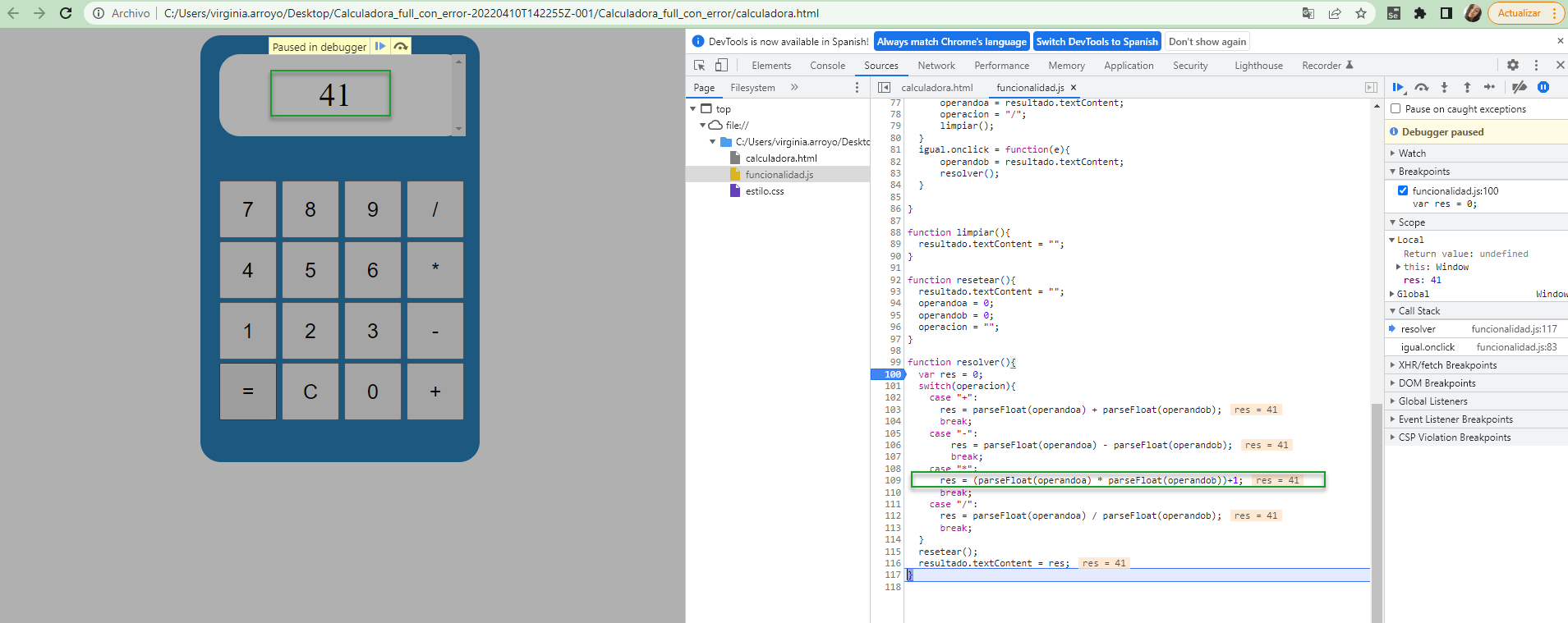
Defecto 1:

La multiplicación no realiza el cálculo según lo esperado.

● Resultado esperado: 5\*8 = 40

● Resultado actual: 5\*8 = 41

Esto se debe a que en la linea 109 se esta incrementando +1 El resultado de la multiplicación:



Opción **Visual Studio Code**

**Problemas comunes que suceden con Visual code:**

No se me quedan los breakpoint en el debugger de Visual Code

**Solución:**

Tienes que ejecutar el start debugging, quitar los breakpoint y volverlos a poner hasta que te funcione (siempre con el navegador abierto).

Anda ejecutando la app hasta que ingrese al breakpoint, luego que das start debugging se abre el navegador, bueno no lo cierres. Quitas el breakpoint y lo vuelves a agregar (es muy probable que no lo veas, solo haces un click para quitarlo y otro para agregarlo). Y volvemos al navegador a correr la app, debería entrar al breakpoint.

1. Abrir Visual Code
2. Ir a la función a depurar (Explore de soluciones)
3. Marcar el breakpoint en la línea correspondiente
4. Seleccionar la opción run > start debugging >Chrome
5. Aquí cambiamos nuestra url (Para eso copiamos la dirección de nuestro archivo index y la pegamos en la url del archivo que se generó después de darle al start debugging ) IMPORTANTE > La url debe ser doble barra porque caso contrario lo toma como un error
6. Ahora seleccionamos el breakpoint en la línea correspondiente
7. Seleccionar la opción run > start debugging >Chrome
8. Ejecutar el defecto y analizar las linea

**Actividad Clase 14**

**Resuelto**

function onClick() {

if (inputsAreEmpty()) {

label.textContent = 'Error: one or both inputs are empty.';

return;

}

if (inputsAreNumber()){

label.textContent = 'Error: one or both inputs are not a number.';

return;

}

updateLabel();

}

function inputsAreEmpty() {

if (getNumber1() === '' getNumber2() === '') {

return true;

} else {

return false;

}

}

function inputsAreNumber() {

if (isNaN(parseInt(getNumber1())) isNaN(parseInt(getNumber2()))) {

return true;

} else {

return false;

}

}

function updateLabel() {

var addend1 = getNumber1();

var addend2 = getNumber2();

var sum = parseInt(addend1) + parseInt(addend2);

//var sum = addend1 + addend2;

label.textContent = addend1 + ' + ' + addend2 + ' = ' + sum;

}

function getNumber1() {

return inputs[0].value;

}

function getNumber2() {

return inputs[1].value;

}

var inputs = document.querySelectorAll('input');

var label = document.querySelector('p');

var button = document.querySelector('button');

button.addEventListener('click', onClick);

## **UNIT TEST**

Herramientas necesarias:

1. NodeJS (<https://nodejs.org/>)
2. Un IDE (el recomendado es Visual Studio Code <https://code.visualstudio.com/>)

Por defecto, VS Code se instala en la siguiente ubicación: C:\users\{username}\AppData\Local\Programs\Microsoft VS Code.

1. JEST (<https://jestjs.io/>)

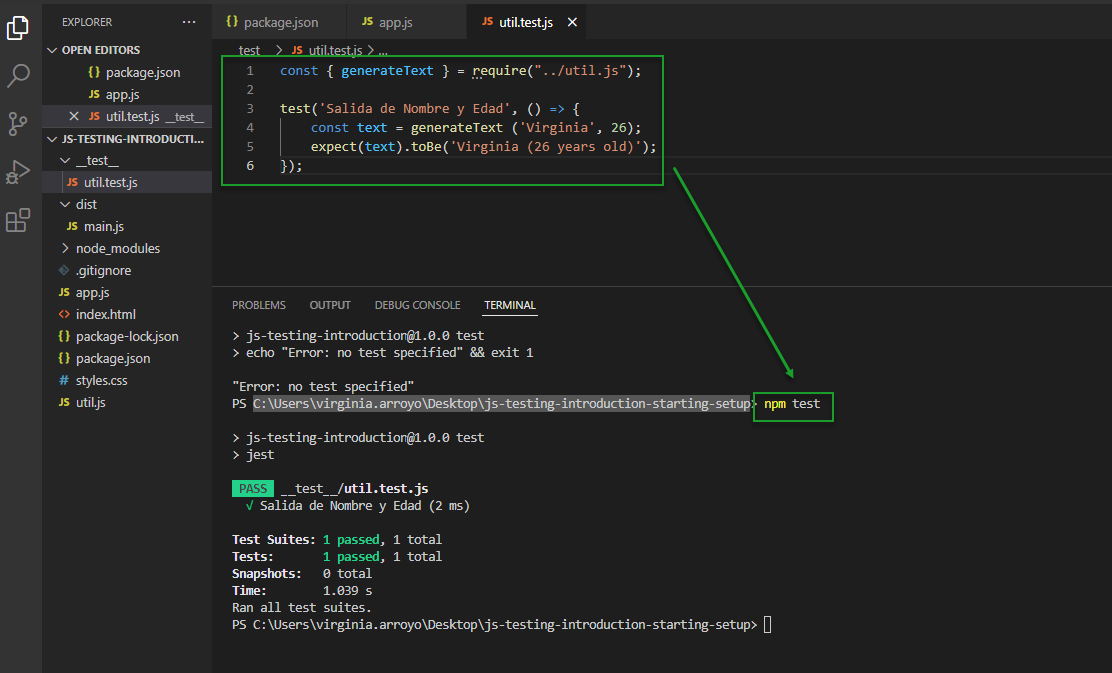
Paso a paso para **Instalar Jest**

1. Abrimos la carpeta del proyecto que se va a probar en Visual Studio Code > <https://github.com/academind/js-testing-introduction/tree/starting-setup>
2. Abrimos terminal
3. Ejecutamos el comando **npm init** (Lo que hace este comando es generar nuestro primer package json > archivo de arranque)
4. Ahora, tiramos el comando **npm install --save-dev jest** (Guarda nuestra dependencia jest como una dependencia de desarrollo) –En caso de que este comando no funcione, utilizaremos el comando **npm install -g jest** (instala jest de manera global)
5. Del archivo package.json remplazar “echo “Error: no test specified” && exit 1” por “jest”
6. Crear una carpeta "\_\_test\_\_" (dos guiones bajos + palabra test + dos guiones bajos EN LA CARPETA DEL PROYECTO
7. Dentro de la carpeta \_\_test\_\_, crear un archivo "útil.test.js".
8. Dentro del archivo, Importar el código a utilizar con la función Require
   1. const {generateText} = require ('../util.js');
9. Código de unit test

Las funciones claves a utilizar son: **test() o it() - expect():**

1. test('Salida de Nombre y Edad', () => {
2. const text= generateText ('Virginia', 26);
3. expect(text).toBe('Virginia (26 years old)');
4. });

10. Para ejecutar el test -> Utilizar el comando **npm test**

****

11. Se puede configurar que los tests se ejecuten cada vez que se genera un cambio en el código la linea **jest --watchAll** en el nodo “test” que está dentro de “scripts”

12. Para generar agrupaciones de unit tests se utiliza la siguiente palabra clave: **describe().** Esto será útil para organizar las pruebas de acuerdo a las funcionalidades a probar. Un describe puede ser 1 test suit que adentro puede tener muchos test (expected), su suit ahora la representa el archivo test, no él describe en sí (Es como que cada archivo es una suit y cada describe es un test y cada test es un expected)

**Coverage**

1. Para revisar cobertura dentro de "script" ingresar: "test:coverage": "jest --coverage", y si es con el DOM "test:coverage": "jest --coverage --env=jsdom"
2. Correr desde la terminal -> npm run test:coverage

Se debería haber creado una carpeta “coverage” y dentro otra carpeta “Icov-report”. Dentro de la carpeta “Icov-report” vamos a tener un archivo index.html. Mostrará la misma información que la que obtuvimos por terminal cuando hacemos “npm run test:coverage”

• Stmts: statements – sentencias

• Funciones

• Lineas

• Branches: si tenemos if clauses

Abrir en el navegador el archivo index.html ubicado dentro de la carpeta “Icov-report”. Hacer clic en link del util.js y se nos mostrará las sentencias a mejorar.

Para mejorar la cobertura, deberíamos agregar pruebas que agreguen los otros dos métodos que no se utilizaron.

**Actividad Clase 16**

**function suma(a,b){**

**return parseFloat(a) + parseFloat(b);**

**}**

**function resta(a,b){**

**return parseFloat(a) - parseFloat(b);**

**}**

**function multiplicacion(a,b){**

**return parseFloat(a) \* parseFloat(b);**

**}**

**function division(a,b){**

**return parseFloat(a) / parseFloat(b);**

**}**

**module.exports = {suma, resta, multiplicacion, division};**

**Resuelto**

const {suma,resta,multiplicacion,division} = require("../calc.js");

describe("test\_positivo", () =>{

const a= 4;

const b= 2;

test("Realizamos suma", () =>{

expect(suma(a,b)).toBe(6)

});

test("Realizamos resta", () =>{

expect(resta(a,b)).toBe(2)

});

test("Realizamos multiplicacion", () =>{

expect(multiplicacion(a,b)).toBe(8)

});

test("Realizamos division", () =>{

expect(division(a,b)).toBe(2)

});

});

describe("test\_negativo", () =>{

test("Realizamos suma", () =>{

expect(suma("a",2)).toBe(NaN)

});

test("Realizamos resta", () =>{

expect(resta(2,"a")).toBe(NaN)

});

test("Realizamos multiplicacion", () =>{

expect(multiplicacion("a","b")).toBe(NaN)

});

test("Realizamos division", () =>{

expect(division("a",2)).toBe(NaN)

});

});