

Germán Martínez Cagigas

DIW
TS

INDICE

Sumario

INDICE.....	2
INTRODUCION.....	3
EJERCICIO.....	4
Instalación Node.js.....	4
Requisitos.....	4
Código en ts y compilador.....	4
Función que realice una suma.....	6
Interfaces y Objetos.....	10
Clases y Métodos.....	11
Manipulación del DOM.....	14
BIBLIOGRAFÍA.....	18

INTRODUCION

Actividades con typescript, interfaces, funciones, dom y clases.

EJERCICIO

Instalación Node.js

Primero que todo tenemos que instalar en nuestro equipo nodejs.org.

Requisitos.

Una vez instalado, creamos una carpeta, abrimos el terminal en ella y ejecutamos `npm init`.

Luego instalamos ts con `npm install typescript --save-dev`.

Esto creará la estructura que vemos en la figura 1, excepto la carpeta `src`, que ahí es donde ejecutaré ts.

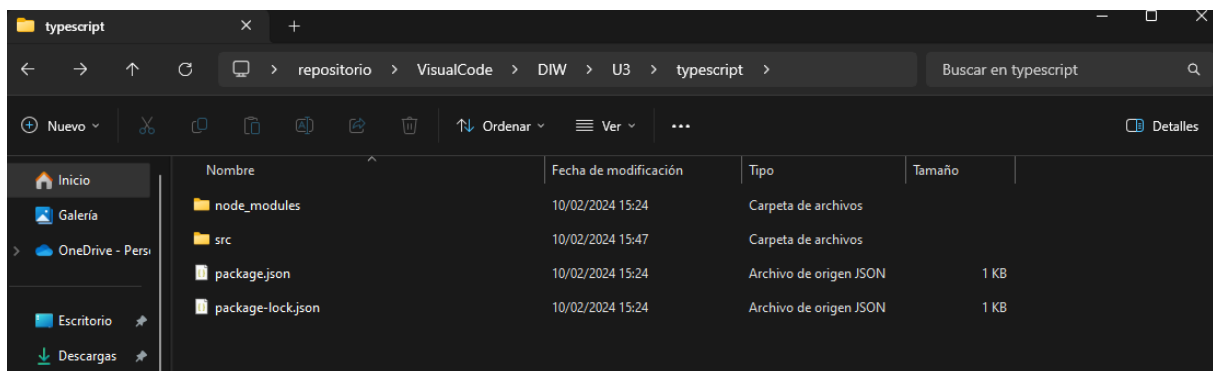


Figura 1

Ahora si, en mi caso creo la carpeta `src` para tenerlo organizado y dentro de la carpeta creo un archivo `main.ts`.

Código en ts y compilador.

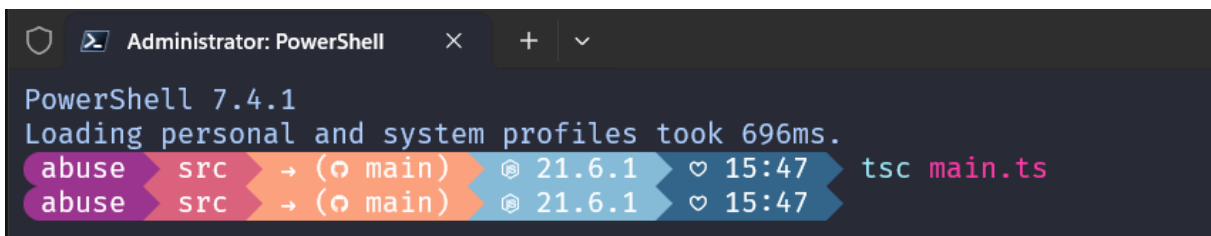
Dentro de `main.ts` creamos las variables que nos piden figura 2.

```
let s: string = "Hola mundo";
let n: number = 1;
let boolean: boolean = true;

console.log("Valor de string:", s);
console.log("Valor de n:", n);
console.log("Valor de bool:", boolean);
```

Figura 2

Ahora pasamos a ejecutar la terminal donde se ubica nuestro archivo main.ts el comando: `tsc main.ts` como se ve en la figura 3.



```
Administrator: PowerShell
PowerShell 7.4.1
Loading personal and system profiles took 696ms.
abuse src → (🔌 main) @ 21.6.1 ♡ 15:47 tsc main.ts
abuse src → (🔌 main) @ 21.6.1 ♡ 15:47
```

Figura 3

Esto hará que compile el código y lo traduzca a un archivo de javascript (figura 4).

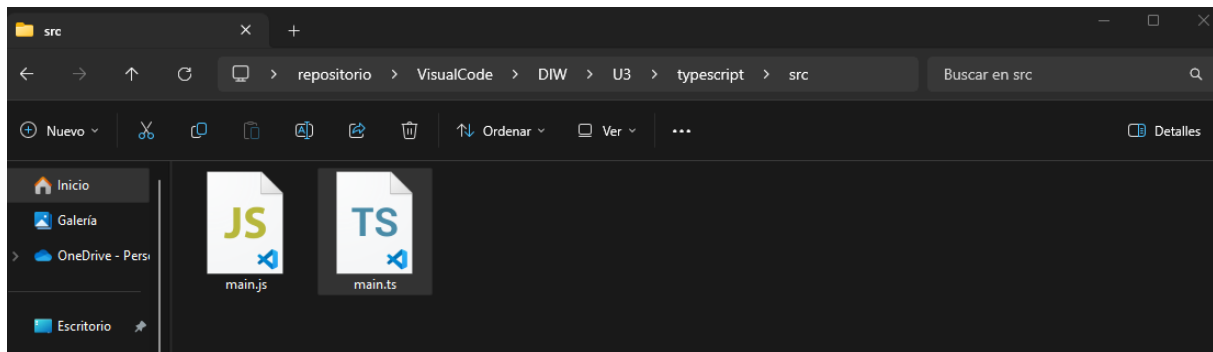


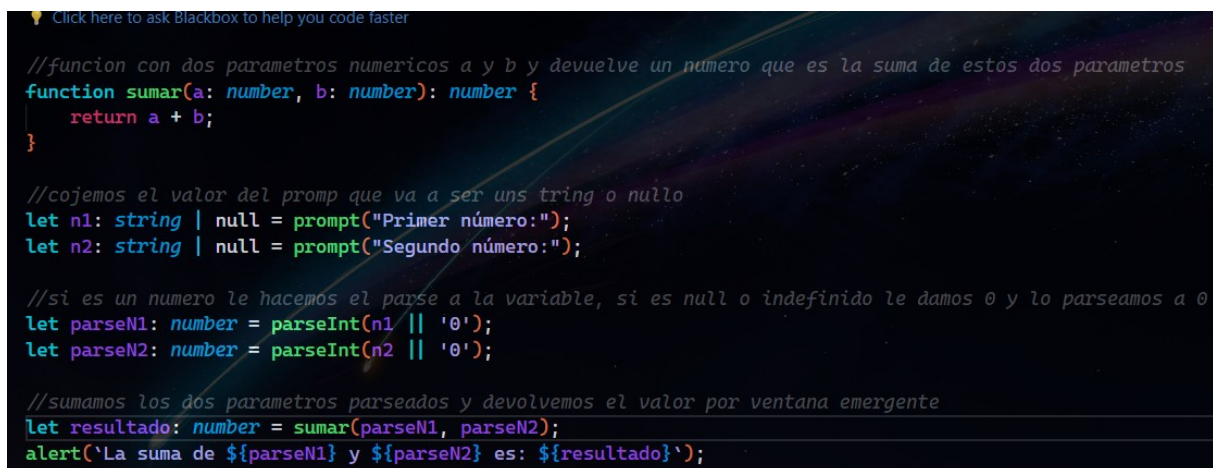
Figura 4

```
Click here to ask Blackbox to help you code faster
var s = "Hola mundo";
var n = 1;
var boolean = true;
console.log("Valor de string:", s);
console.log("Valor de n:", n);
console.log("Valor de bool:", boolean);
```

Figura 5

Y en la figura 5 vemos como sería nuestro archivo de javascript tras la compilación.

Función que realice una suma.



```

Click here to ask Blackbox to help you code faster

//funcion con dos parametros numericos a y b y devuelve un numero que es la suma de estos dos parametros
function sumar(a: number, b: number): number {
    return a + b;
}

//cojemos el valor del prompt que va a ser un string o nullo
let n1: string | null = prompt("Primer número:");
let n2: string | null = prompt("Segundo número:");

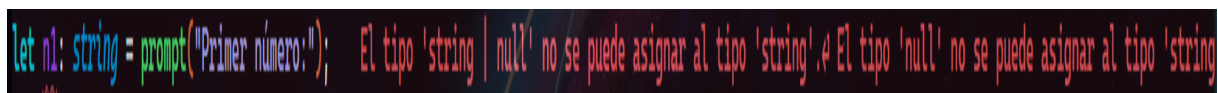
//si es un numero le hacemos el parse a la variable, si es null o indefinido le damos 0 y lo parseamos a 0
let parseN1: number = parseInt(n1 || '0');
let parseN2: number = parseInt(n2 || '0');

//sumamos los dos parametros parseados y devolvemos el valor por ventana emergente
let resultado: number = sumar(parseN1, parseN2);
alert(`La suma de ${parseN1} y ${parseN2} es: ${resultado}`);

```

Figura 6

En la figura 6 ya le añadí los comentarios para que sea autoexplicativo lo que hace cada línea, el null lo añadí porque me tilteaba ver rojo básicamente.



```

let n1: string = prompt("Primer número:"); El tipo 'string | null' no se puede asignar al tipo 'string'. El tipo 'null' no se puede asignar al tipo 'string'

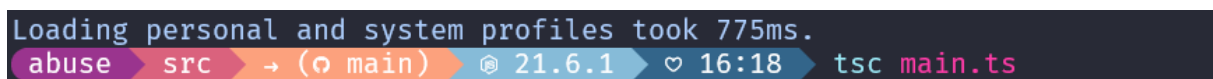
```

Figura 7

En la figura 7 se ve el error que me encontré en primera estancia, añadí el null y es simplemente por si no se mete ningún valor.

Al igual que cuando se hace el parse, si el valor es null o indefinido le meto manualmente el string 0 para que lo parsee.

Volvemos a abrir la terminal en el archivo y ejecutamos el comando figura 8.



```

Loading personal and system profiles took 775ms.
abuse src → (main) 21.6.1 16:18 tsc main.ts

```

Figura 8

```
//funcion con dos parametros numericos a y b y devuelve un numero que es la suma de estos dos parametros
function sumar(a, b) {
    return a + b;
}
//cojemos el valor del prompt que va a ser un string o nullo
var n1 = prompt("Primer número:");
var n2 = prompt("Segundo número:");
//si es un numero le hacemos el parse a la variable, si es null o indefinido le damos 0 y lo parseamos a 0
var parseN1 = parseInt(n1 || '0');
var parseN2 = parseInt(n2 || '0');
//sumamos los dos parametros parseados y devolvemos el valor por ventana emergente
var resultado = sumar(parseN1, parseN2);
alert("La suma de ".concat(parseN1, " y ").concat(parseN2, " es: ").concat(resultado));
```

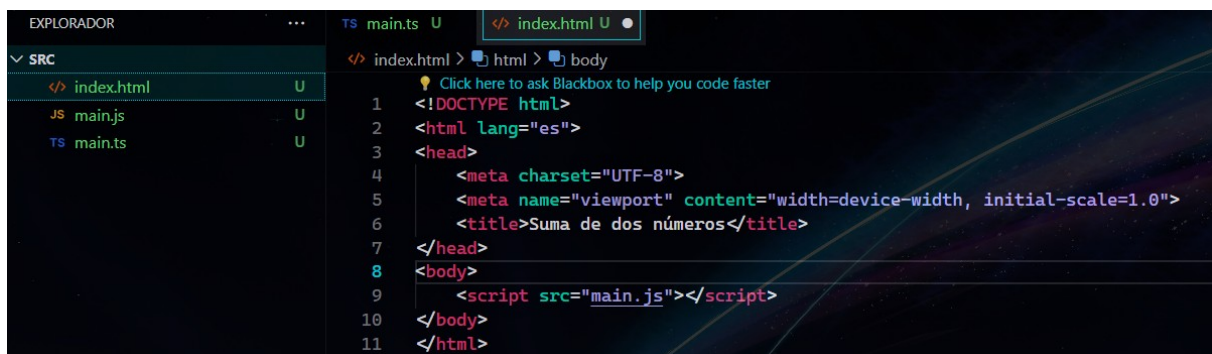
Figura 9

La figura 9 muestra el archivo js generado, y lo primero que me choca es que se genera un prompt normal, en ts definimos si es string o null, pero ahora ya js lo entiende como un prompt normal aunque haya que hacer ese pasito extra en ts.

El parse pues el de toda la vida, si la primera opción es falsa entonces le pone un string de 0 para parsear.

Y por último el dólar y variable es .concat, no tenía ni idea de ese metodo.

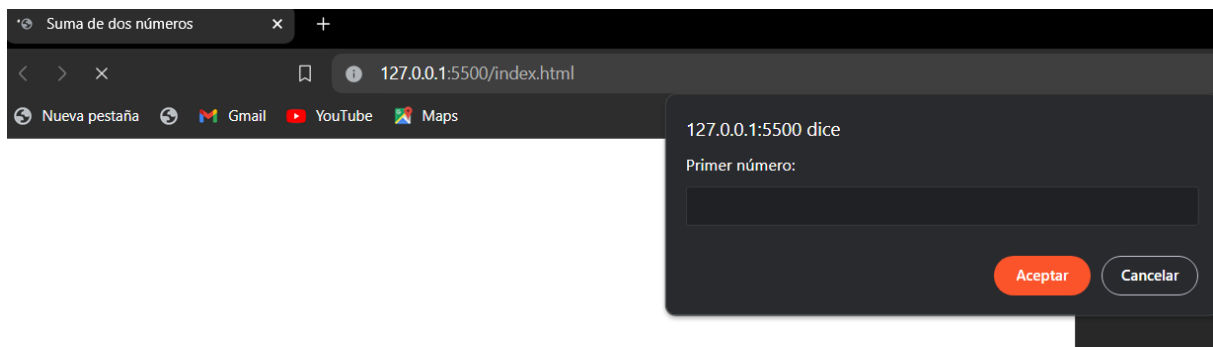
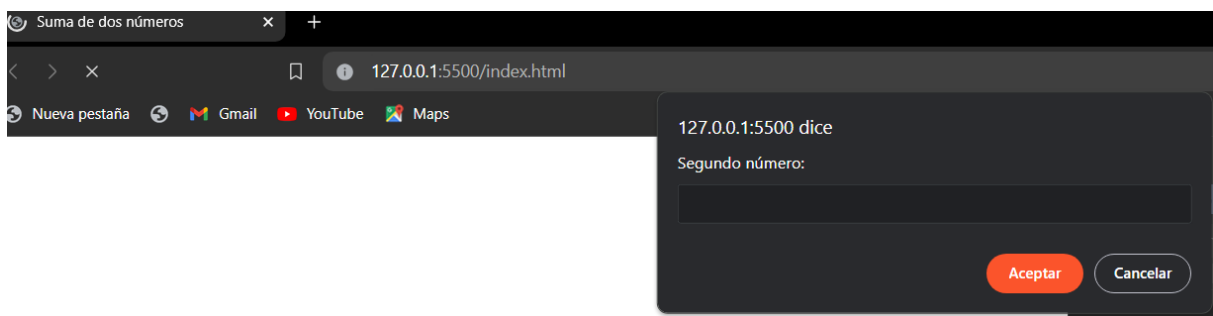
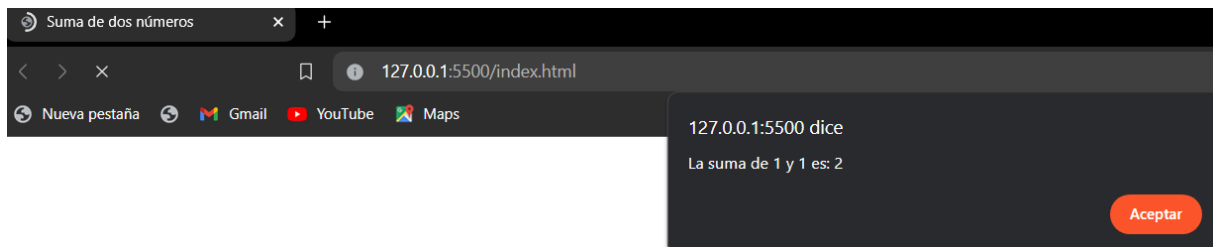
Ahora ya si, creamos nuestro html para llamar a esta función y probar su funcionamiento figura 10.



```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Suma de dos números</title>
7 </head>
8 <body>
9   <script src="main.js"></script>
10 </body>
11 </html>
```

Figura 10

Y vamos al live server figura 11 , 12 y 13.

*Figura 11**Figura 12**Figura 13*

Este ejercicio pedía pegarlo por consola, pero para la foto preferí darle un alert, en vez de alert, se pone ``console.log`` y listo para verlo por consola.

Interfaces y Objetos.

Vamos a crear el constructor de libro, luego creamos un objeto libro, y lo pintamos por pantalla con un for each.

```
interface Libro{  
  titulo: string;  
  auto: string;  
  anioPublicacion: number;  
}
```

Figura 14

En la figura 14 definimos nuestro constructor para Libro.

```
const libro: Libro = {  
  titulo: "Libro 1",  
  auto: "Autor 1",  
  anioPublicacion: 2020  
}
```

Figura 15

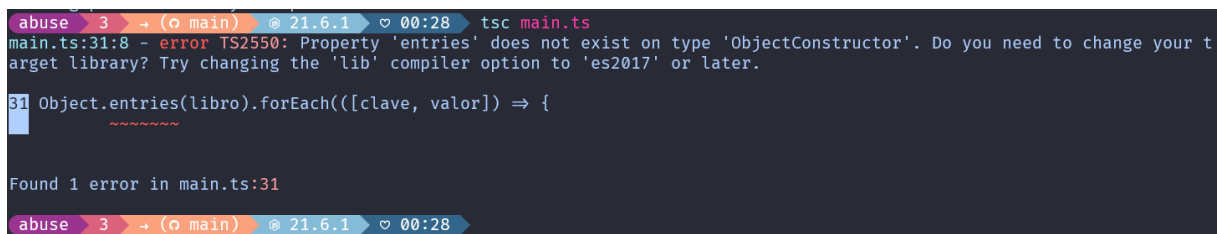
En la figura 15 creamos un objeto libro y le damos propiedades.

```
Object.entries(libro).forEach(([key, value]) => {  
  console.log(`${key}: ${value}`);  
})
```

Figura 16

Iteramos sobre el objeto con el método entries de la clase objeto que crea una matriz de clave haciendo referencia a la propiedad y valor al atributo de dicha propiedad en el objeto.

Al compilar me da este error en la figura 17.

A screenshot of a code editor showing a TypeScript error. The error message is: "main.ts:31:8 - error TS2550: Property 'entries' does not exist on type 'ObjectConstructor'. Do you need to change your target library? Try changing the 'lib' compiler option to 'es2017' or later." The code line is: "Object.entries(libro).forEach(([clave, valor]) => {". The error is highlighted in red. The status bar at the bottom shows "abuse 3" and "21.6.1".

```
abuse 3 → (main) 21.6.1 00:28 tsc main.ts
main.ts:31:8 - error TS2550: Property 'entries' does not exist on type 'ObjectConstructor'. Do you need to change your target library? Try changing the 'lib' compiler option to 'es2017' or later.

31 Object.entries(libro).forEach(([clave, valor]) => {
    ~~~~~

Found 1 error in main.ts:31

abuse 3 → (main) 21.6.1 00:28
```

Figura 17

Lo que tendría que hacer es según he encontrado crear un archivo de configuración para el compilador, pero al probar los resultados en la figura 18, veo que se muestra así que lo dejo mencionado y no me apetece tampoco borrarlo y simplemente poner un console log e ir haciendo libro.titulo, libro.autor, etc, que ni me moleste en probarlo pero seguro que también sirve de esa manera.

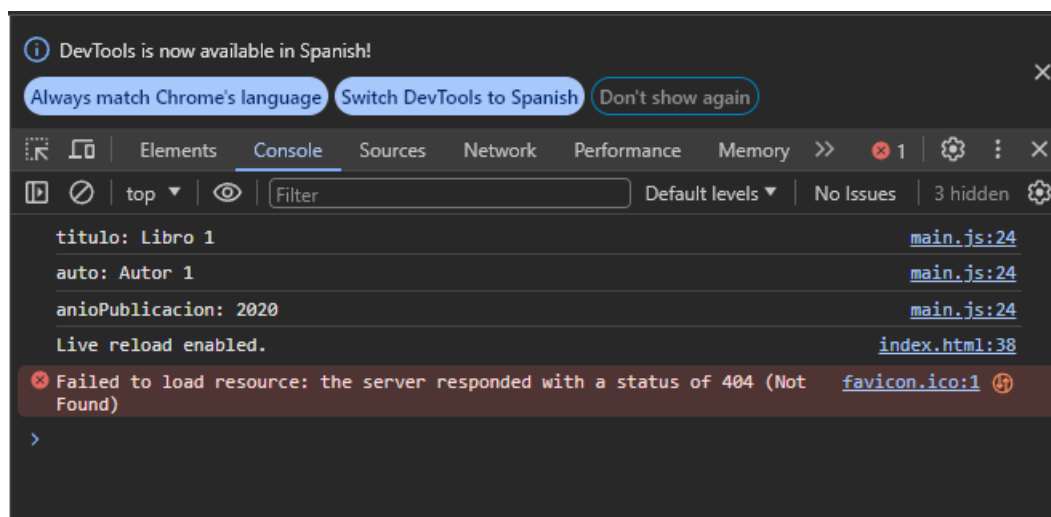


Figura 18

Si bien es una interfaz y el constructor no es un constructor como tal ni creo un objeto usando el constructor, no se como se le llamaría a un objeto de la interfaz ni si tiene un nombre como tal la definición de la interfaz.

Clases y Métodos.

Aquí si que vamos a usar un constructor y clase como tal de java, además de un método.

```
class Rectangulo {  
    base: number;  
    altura: number;  
  
    constructor(base: number, altura: number) {  
        this.base = base;  
        this.altura = altura;  
    }  
}
```

Figura 19

En la figura 19 veo la clase rectangulo con el constructor, exactamente igual que en Java nada que decir aquí.

```
public get area(): number {  
    return this.calcularArea();  
}  
  
private calcularArea(): number {  
    return this.base * this.altura;  
}
```

Figura 20

En la figura 20 probé a llamar a la función de calcularArea a través de un getter público y dejé el método en privado, aunque vaya a crear ahora una instancia del objeto en la misma hoja, entiendo que lo suyo sería dejar esta hoja de plantilla por así decirlo y luego en el main crear las instancias y llamar a los métodos públicos de la clase como los getters, aunque lo puse principalmente para ver que me encuentro en javascript la verdad.

```
//creo la instancia  
const miRectangulo = new Rectangulo(321, 444);  
  
//accedo al getter de la instancia  
console.log("El área es: ", miRectangulo.area);
```

Figura 21

En la figura 21 ya simplemente creo una instancia de rectangulo usando el constructor paso los valores, y luego accedo al getter de la instancia que llama internamente a la función de calcular el área, compilemos.

```
var Rectangulo = /** @class */ (function () {
    function Rectangulo(base, altura) {
        this.base = base;
        this.altura = altura;
    }
    Object.defineProperty(Rectangulo.prototype, "area", {
        /**
         * public getFullName(): string {
         *     return `${this.firstName} ${this.lastName}`;
         * }
         */
        get: function () {
            return this.calcularArea();
        },
        enumerable: false,
        configurable: true
    });
    Rectangulo.prototype.calcularArea = function () {
        return this.base * this.altura;
    };
    return Rectangulo;
})();
//creo la instancia
var miRectangulo = new Rectangulo(321, 444);
//accedo al getter de la instancia
console.log("El área es: ", miRectangulo.area);
```

Figura 22

Resultado del java script tras la compilación.

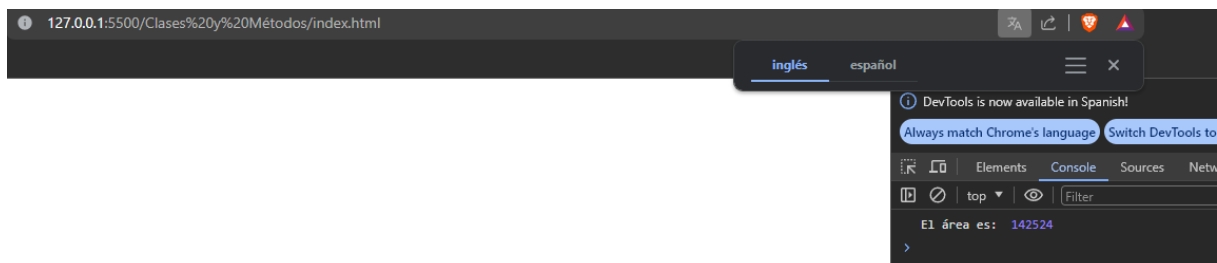


Figura 23

Manipulación del DOM.

```
const button = document.getElementById('button');
const message = document.getElementById('p1');

if (button && message) {
  button.addEventListener('click', () => {
    message.style.color = 'red';
    message.textContent = 'El párrafo ha sido modificado por DOM';
  })
} else {
  console.error('El valor de button o message es null');
}
```

Figura 24

No hay mucho que decir es igual que en javascript, hice un if en la figura 24 para comprobar si el valor era null, que me tiltea que se vean mensajes de advertencia como se vé en la figura 25.

```
button.addEventListener('click', () => {  "button" es posiblemente "null".
  message.style.color = 'red';           "message" es posiblemente "null".
  message.textContent = 'El párrafo ha sido modificado por DOM';  "message" es posiblemente "null".
})
```

Figura 25

Por lo demás no hay nada que decir, es exactamente igual que en javascript.

```
Document.getElementById

The definition for this method is as follows:

getElementById(elementId: string): HTMLElement | null;
```

Figura 26

En la figura 26 estuve buscando otras maneras para ver si hay algo que se me escapa, pero eso es parte de la documentación de typescript, y ahí el DOM lo trabajan como en la figura 27 que es igual que en javascript.

Let's explore a TypeScript script that adds a `<p>Hello, World!</p>` element to the `#app` element.

```
// 1. Select the div element using the id property
const app = document.getElementById("app");

// 2. Create a new <p></p> element programmatically
const p = document.createElement("p");

// 3. Add the text content
p.textContent = "Hello, World!";

// 4. Append the p element to the div element
app?.appendChild(p);
```

Figura 27

Bueno, compilamos.

```
1  var button = document.getElementById('button');
2  var message = document.getElementById('p1');
3  if (button && message) {
4      button.addEventListener('click', function () {
5          message.style.color = 'red';
6          message.textContent = 'El párrafo ha sido modificado por DOM';
7      });
8  }
9  else {
10     console.error('El valor de button o message es null');
11 }
12
```

Figura 28

En la figura 28, lo único que cambia y no tendría porque es que la función anónima la define como function, la función anónima la uso siempre en los eventos en javascript así que no entiendo porque al compilar hace eso, pero vamos, que es exactamente igual en este ejemplo el ts que el js.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=
    , initial-scale=1.0">
  <title>Document</title>
</head>
<style>
  body{
    text-align: center;
    margin-top: 20%;
  }
</style>
<body>
  <button id="button">Click me</button>
  <div>
    <p id = "p1">Soy un párrafo al que todavía no se le ha modificado</p>
  </div>
  <script src="main.js"></script>
</body>
</html>
```

Figura 29

Figura 29 muestra mi html y la 30 como se ve en el browser.



Figura 30

La figura 31 muestra como se ve tras clickar.



Figura 31

BIBLIOGRAFÍA

<https://bobbyhadz.com/blog/typescript-object-foreach>

<https://www.tutorialesprogramacionya.com/angularya/detalleconcepto.php?punto=23&codigo=23&inicio=20>

<https://www.typescripttutorial.net/typescript-tutorial/typescript-getters-setters/>

<https://www.typescriptlang.org/docs/handbook/dom-manipulation.html>