Javascript

Carrera Programador full-stack

Parte II

Mostrar/Ocultar detalles

Carrera Programador full-stack

Ejemplo

Crear un botón “Ver Más”, que muestre/oculte el contenido de un div.

El botón debe poder reutilizarse y funcionar de manera independiente del resto de los botones de la página.

Lo que veamos como captura de pantalla corresponde al ejemplo de la clase:

```
let btn= document.getElementById("btn1");//seleccionamos el elemento html
```

Qué vamos a aprender

¿Qué vamos a aprender?

- Desacoplar HTML y JS
  - funciones anónimas
- Obtener múltiples elementos del DOM
- Recorrer el DOM
- Asignar clase a un elemento del DOM

Eventos en ES6

Asignar handlers en HTML es mala práctica porque mezcla HTML y TypeScript.

```
<button type="button" class="btn" onclick="verificarFormulario();">Enviar</button>
```

No recomendado

Debería recordar las funciones escritas en TS.

ECMAScript 6

ECMAScript 6 (ES6), también conocido como ECMAScript 2015, es una versión del estándar de scripting ECMAScript en la que se basan los lenguajes como JavaScript. Fue publicado en junio de 2015 por ECMA International y es una de las actualizaciones más importantes del lenguaje JavaScript, introduciendo una serie de características nuevas y mejoras que hacen que el lenguaje sea más poderoso y fácil de usar.

ECMAScript 6

Principales Características de ES6:

1. **Let y Const**:
   - **`let`** permite la declaración de variables con un ámbito de bloque, lo que mejora la gestión del alcance de las variables.
   - **`const`** permite la declaración de constantes, es decir, variables cuyo valor no puede cambiar una vez asignado.
2. **Arrow Functions**:
   - Las funciones flecha (**`=>`**) proporcionan una sintaxis más corta para escribir funciones y no tienen su propio **`this`** contexto, lo que facilita trabajar con el alcance léxico.
3. **Template Literals**:
   - Plantillas de cadena de texto que permiten incrustar expresiones dentro de cadenas utilizando la sintaxis **`${expresión}`**.
4. **Desestructuración**:
   - Permite extraer valores de arreglos u objetos y asignarlos a variables de manera concisa.
5. **Promesas**:
   - Introduce el objeto **`Promise`** para manejar operaciones asíncronas, mejorando la gestión de callbacks y el flujo de trabajo asíncrono

Eventos en ES6

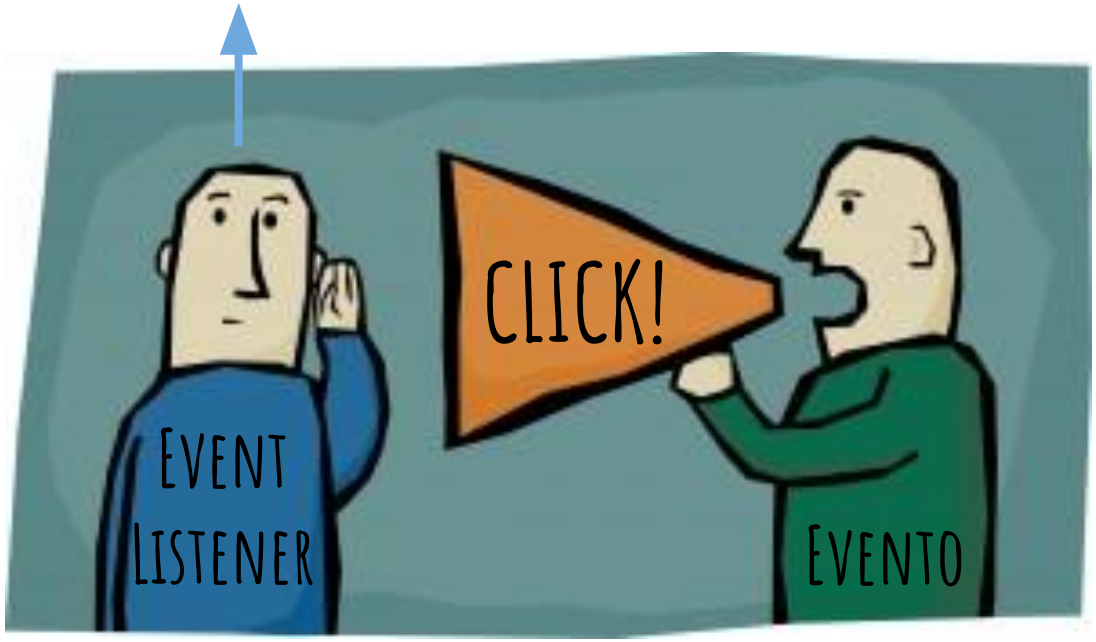
En **ES6**, primero buscamos el elemento y luego le asociamos el handler

HTML

```html
<button type="button" id="btn-enviar">Enviar</button>
```

JS

```js
let btn = document.getElementById("btn-enviar");
btn.addEventListener("click", verificarFormulario);
```



Está pendiente de escuchar un evento para ejecutarlo solo en el momento en que suceda

Eventos en ES6

```
let btn= document.getElementById("btn1");//seleccionamos el elemento html
btn.addEventListener("click",verificar);//le asignamos el addEventListener al boton
//con el evento click y la funcion verificar asociada
```

Prestar especial atención a cómo asociamos la función, sin paréntesis.
La función **verificar** se pasa como referencia al método addEventListener sin paréntesis porque queremos que la función se ejecute en respuesta al evento de click y no en el momento de agregar el evento.
Esto significa que **verificar** será llamada por el navegador cuando ocurra el evento "click".
Si escribimos **verificar()**, estamos llamando a la función inmediatamente y pasando el resultado de esa llamada

Eventos en ES6 con parámetros

Si no podemos ponerle () a la función ¿como hacemos para pasar parámetros?

Debemos pasar los parametros pero luego que el evento suceda.

Entonces necesitamos encapsular la función con otra y pasar los parámetros

Eventos en ES6 con parámetros

```
function verificar(event){
    event.preventDefault();
    let param1=document.getElementById("inpText1").value;
    let param2=document.getElementById("inpText2").value;
verificarConParametros(param1,param2)//usamos esta función
  //con la que si podemos pasar parametros
}
```

Eventos en ES6

Utilizamos una **función anónima** sin parámetros que encapsula a la función que si pasa parámetros

JS

```
let inputEmail, inputConsulta...
...
btn.addEventListener("click", verificarSinParametros)

function verificarSinParametros(e){
 verificarFormulario(inputEmail,inputConsulta)
}
function verificarFormulario(email, consulta)
{
    ...
```

El parámetro **"e"** lo pasa TS, tiene info del evento (el click en este caso)

DEMO

https://codepen.io/webUnicen/pen/qmZOaV

Eventos en ES6 - Funciones anónimas

- Se usan para no crear tantas funciones que se usan en un solo lugar
- Es una función sin nombre que se escribe directamente donde la quería pasar de parámetro
- En este caso encapsula a la función que si pasa parámetros

Una nueva función anónima que llama a verificarFormulario con sus parámetros, es igual al **verificarSinParametros**

```
btn.addEventListener("click", function(e){
 verificarFormulario(inputEmail,inputConsulta)
} )
function verificarFormulario(email, consulta)
{...
```

DEMO https://codepen.io/webUnicen/pen/qmZOaV

Eventos en ES6 - Funciones anónimas

En nuestro caso:

```
let btn= document.getElementById("btn1");//seleccionamos el elemento html
btn.addEventListener("click",function(event){
    event.preventDefault();
    let param1=document.getElementById("inpText1").value;
    let param2=document.getElementById("inpText2").value;
  verificarConParametros(param1,param2)
});
```

Nos ahorramos una función de paso, es decir solo es una función que ejecuta con el click. verificarConParametros en cambio podría ser reutilizada

Obtener nodos del DOM

- Se pueden obtener elementos del DOM consultando por un ID, nombre, clase o un selector.
- Podemos obtener como resultado de uno o múltiples elementos del DOM

Retorna un nodo

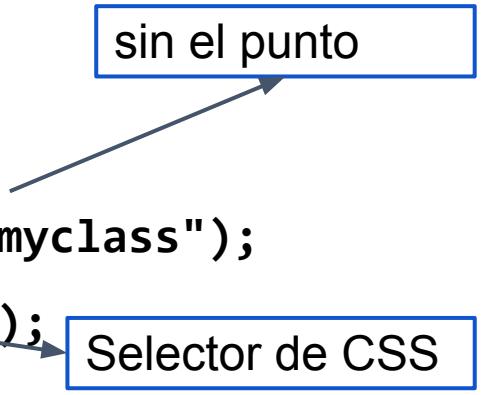
```
let elem = document.getElementById("identificador");

let singleElem = document.querySelector(".myclass");
```

Retorna uno o más

```
let manyElements = document.getElementsByClassName("myclass");

let manyElems = document.querySelectorAll(".myclass");
```



Más info

https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector

Obtener múltiples nodos del DOM

Obteniendo elementos del DOM con la misma clase

```
let manyElements = document.getElementsByClassName("myclass");
```

```
let manyElements = document.querySelectorAll(".myclass");
```

`manyElements` es un **arreglo** con los elementos que poseen la clase

`manyElements.length` largo del arreglo y cantidad de nodos con esa clase

`manyElements[0]` es el primer elemento con clase .myclass

Recorrer el árbol DOM

Los elementos del DOM se pueden recorrer como un árbol y ser localizados:

- element.children, encuentra los elementos hijos
- element.parentElement , encuentra el elemento padre
- element.nextElementSibling , encuentra el siguiente hermano
- element.previousElementSibling , encuentra el hermano anterior
- element.firstElementChild , encuentra el primer hijo
- element.lastElementChild , el último hijo

Editar estilo desde Javascript

```
// div es una referencia a un elemento <div>
div.classList.add("clase");
div.classList.remove("clase");
div.classList.toggle("clase");

alert(div.classList.contains("clase"));

// agregar o quitar múltiples clases
div.classList.add("clase-1", "clase-2", "clase-3");
```

Buena Práctica!
Cambiar estilos con clases

```
// estilos vía TS (Mala práctica)
document.getElementById("id").style.font-size = "20px";
```

https://codepen.io/webUnicen/pen/qmVoMV

this

En el contexto de Eventos *this* representa el elemento involucrado en el evento

```
let el =
document.getElementById('miDiv');
el.addEventListener('click', function(e){
  this.classList.toggle("clase");
  //toggle de clase del div miDiv click
  console.log(e); //ver E
});
```

https://codepen.io/webUnicen/pen/odNvKK

Resolver el problema

Debemos localizar todos los elementos que correspondan a una clase, y luego asignarle a cada uno el evento.

Teniendo en cuenta el código anterior nos damos cuenta de que si tenemos muchos botones vamos a terminar repitiendo mucho código.

La forma de evitar eso puede ser como el ejemplo que sigue.

Resolver el problema

// Búsqueda de todos los botones con una clase

```
let btns = document.querySelectorAll('.btn');
//un querySelectorAll devuelve un NodeList.
//Cuando usamos métodos como:
let ele1= document.getElementsByClassName('mi-clase');
let ele2 = document.getElementsByTagName('div');
//obtendremos un HTMLCollection
```

```
// asignación de evento a todos los elementos
for(let i = 0; i < btns.length; i++) {
    btns[i].addEventListener('click', miFuncion);
}
```

Resolver el problema

Luego, mediante una función anónima individualizamos el botón que dispara el evento y buscamos su hermano en el DOM.

```
for(let i = 0; i < btns.length; i++) {
    btns[i].addEventListener('click', function(e){
    //busca el hermano inmediato

    let el = this.nextElementSibling;
    //toggle de clase del hermano

    el.classList.toggle("ver");
  });
}
```

https://codepen.io/webUnicen/pen/gzOYaN

Resumen: conexión entre HTML, CSS y TS

- La unica conexion debieran ser las clases
- Las clases son el contrato entre los tres lenguajes
- En lo único que se tienen que poner de acuerdo es en qué significa cada clase
  - HTML le va a poner las clases a lo que corresponda
  - CSS va a hacer que se vea como dice el acuerdo
  - TS va a hacer que se comporte como dice el acuerdo
- El nombre de la clase debe ser representativo de este contrato

Ejercicios

Carrera Programador full-stack

Ejercicio 1

Utilizando lo visto en esta clase, crear una función Javascript que oculte y muestre un div que contiene información.

Ejercicio 2

Analizar cómo modificar el ejercicio anterior para que sea un código reutilizable (poder poner muchos botones que oculten o muestren un div respectivo)