Desarrollador Frontend

clase 2 | modelo orientado a eventos

Estructurando un proyecto pequeño

Cuando no usamos frameworks, la estructura de una aplicación javascript puede pensarse como un **conjunto de funciones** independientes que reaccionan a determinados **eventos del DOM** (algunos generados por el usuario, otros que suceden de forma programática).
La secuencia de nuestro archivo sería:

- seleccionar un objeto del DOM
- vincularnos a alguno de sus eventos con una función
- las funciones pueden llamarse entre sí

Este tipo de estructura no es recomendada para aplicaciones de mediana o alta complejidad, pero es un buen comienzo para ordenar nuestros scripts.

Podemos encontrar una primer aproximación a este tipo de proyectos en el siguiente link

Eventos más usados

Los elementos del DOM nos permiten ejecutar scripts *cuando cierto evento sucede*.

Algunos de los eventos más usados son:

- onclick: al realizar un click sobre el elemento
- onfocus: al comenzar a interactuar con el elemento*
- onchange: al cambiar el valor del elemento*
- onkeydown / onkeyup: al presionar una tecla dentro del elemento
- onblur: al abandonar el elemento*
- onsubmit: al accionar un formulario
- onload: evento automático que se ejecuta cuando el elemento está cargado**

* generalmente <input> <select> <textarea>
** dependiendo del elemento, por ejemplo, window.onload se dispara al cargar toda la página, document.onload al terminarse de armar el DOM, imagen.onload al cargarse la imagen

Podemos encontrar un listado completo de los eventos DOM en w3schools o en Mozilla Developer

Eventos como atributo html

Una primer forma de vincularnos a un evento es incluir código javascript directamente en el atributo:

```
<button onclick="alert('me clickearon'); var a = 22;">Tocame</button>
```

Puedo también ejecutar una función que haga las dos cosas

```
<button onclick="miFuncion()">Tocame</button>
<script>
            function miFuncion(){
            alert('me clickearon');
            var a = 22;
      }
</script>
```

```
<button onclick="miFuncion()">Tocame</button>
<script src="funciones.js"></script>

//funciones.js
function miFuncion(){
      alert('me clickearon');
      var a = 22;
}
```

Si bien el ejemplo de la derecha es más ordenado y permite reutilizar funciones, no es una buena práctica incluir javascript dentro del HTML (similar al uso del atributo style en vez de combinar class + CSS)

Vincular funciones a eventos desde javascript

Esta será la forma recomendada: seleccionar el elemento DOM y vincularlo a una función

```
<button id="elBoton">Tocame</button>
<script src="programa.js"></script>
```

```
//programa.js
var btn = document.querySelector('#elBoton');
btn.onclick = miFuncion; //notar ausencia de ()

function miFuncion(){
        alert('me clickearon');
        var a = 22;
}
```

Desde el punto de vista del HTML, no puedo saber qué interacciones están asociadas al botón (se desacopla la estructura de la lógica). De hecho, si me vinculo a otro js pasará otra cosa.

La función **miFuncion** es igual a cualquier otra, pero al vincularse al evento de onclick decimos que es un **callback**, ya que será ejecutada (llamada) cuando el evento de **onclick** se dispare.

Mecanismo de suscripción y propagación

Las versiones modernas de los navegadores (IE9+) disponen de manejo de *listeners*

```
<button id="elBoton">Tocame</button>
<script src="programa.js"></script>
```

```
//programa.js
var btn = document.querySelector('#elBoton');
btn.addEventListener('click', miFuncion);
btn.addEventListener('click', miFuncion2);

function miFuncion(event){
      console.log(event.target);
}
```

Como vimos en el ejemplo anterior, para HTML es transparente qué método utilizamos!

Utilizar **addEventListener** permite suscribir varias funciones al mismo evento (onclick solo permite una asignación). La función callback ahora recibe un parámetro con información del evento.

Funciones anónimas

Un callback puede ser directamente una función en vez de una referencia

```
//programa.js método clásico
var btn = document.querySelector('#elBoton');
btn.onclick = function(){
             alert('tocaron el boton');
}
```

```
//programa.js método listeners
var btn = document.querySelector('#elBoton');
btn.addEventListener('click', function(event){
       console.log(event.target);
});
```

Esta manera de programar tiende a generar un código más ilegible.
También perdemos la posibilidad de reutilizar la función con otro evento.
En el caso de **addEventListener** perdemos la referencia original para poder remover el callback.

Remover callbacks

Un callback puede ser eliminado, para dejar de ejecutarse al surgir el evento

```
//programa.js método clásico
var btn = document.querySelector('#elBoton');
var n = 0;
btn.onclick = sumar;
function sumar(){
            console.log('tocaron el boton ',
n++);
            if(n == 3) btn.onclick = null;
}
```

```
//programa.js método listeners
var btn = document.querySelector('#elBoton');
var n = 0;
btn.addEventListener('click', sumar);
function sumar(){
  console.log('tocaron el boton ', n++);
  if(n == 3) {
    btn.removeEventListener('click', sumar);
  }
}
```

En el método clásico, se asigna null para eliminar la referencia al callback.

La ventaja de **removeEventListener** es que sólo elimina un callback en particular.

Cancelar la propagación de un evento

Si el callback devuelve false el evento no continúa su curso

```
<a href="http://algunlado.com">Link</a>
<script src="programa.js"></script>
```

```
//programa.js
var link = document.querySelector('a');
link.addEventListener('click', cancelar);
function cancelar(){
  return false;
}
```

Sin la intervención de javascript, al tocar la palabra Link seremos dirigidos a la URL definida en href.

En este caso el callback impide que el click continúe, por lo que el link nunca irá a **http://algunlado.com**. Esta técnica será útil para realizar validaciones.

Validaciones

En versiones anteriores de HTML, se utilizaba javascript para realizar todo tipo de validaciones. Hoy en día, HTML5 soporta nativamente las siguientes validaciones:

- Campos requeridos (a través del atributo required)
- Tipo de campos (email, url, number)
- Mínimos y máximos
- Expresiones regulares (a través del atributo pattern)

Seguimos utilizando javascript para realizar validaciones de negocio, o situaciones como “repetir contraseña”, comparaciones entre campos, validaciones contra algún listado de valores.

La idea siempre es la misma: cancelar el evento asociado si no se cumplen las condiciones requeridas (generalmente el **onsubmit** del formulario, o el **onclick** de alguna acción).

Ejercicios en clase

- En el siguiente link se encuentran los ejercicios para trabajar en clase, con sus soluciones. Plunker nos permite ejecutar código online y compartirlo, sin necesidad de tener nada instalado más que el navegador web.