JavaScript II - HAB (https://remoto.hackaboss.com/trainer/course/id:134)

4. Eventos

Los eventos son acciones iniciadas por los usuarios o que ocurren automáticamente cuando un documento web se muestra en un navegador. Mediante Javascript y el DOM podremos programar respuestas a eses eventos.

Tipos de eventos

Cuando un documento web carga en un navegador se generan eventos de muchos tipos. Estos eventos son generados por múltiples elementos y muchos de ellos son causados por una acción del usuario (otros no). Por ejemplo:

- **Un botón:** al ser clicado por un usuario.
- **El tab actual:** al finalizar la carga de la página.
- **Un video:** al finalizar la reprodución.
- **Un párrafo:** cuando el usuario pasa el ratón por encima.
- **El tab actual:** acaba de ocurrir un error.
- **Un input:** cuando el usuario acaba de escribir algo.

Ante todos estos eventos podemos programar respuestas asignando ***event listeners*** también llamados ***event handlers***. La lista completa de eventos está aquí (https://developer.mozilla.org/en-US/docs/Web/Events).

Creando Event Listeners

- **Método tradicional:** hai dos maneras principales de escuchar eventos. Tradicionalmente se usaban propiedades de los ***Elements*** que representaban a cada uno de los eventos:

```
const button = document.querySelector('button');

button.onclick = function() {
  console.log('Acabas de hacer click en el botón');
};

button.onmouseover = function() {
  console.log('Acabas de pasar con el cursor por encima del botón');
};

button.onfocus = function() {
  console.log('Acabas de hacer zoom con el botón');
};
```

Todos los elementos que seleccionamos mediante el DOM tienen una serie de propiedades con esta forma: **on + nombre-del-evento** a las que les podemos asignar funciones y estas funciones se ejecutarán cada vez que ocurra uno de estos eventos.

- **Método moderno (recomendado):** en lugar de usar las propiedades de eventos mencionadas anteriormente podemos usar dos métodos de los elementos mucho máis adecuados para asignar eventos:
  - **addEventListener**
  - **removeEventListener**

Para asignar un evento:

```
const button = document.querySelector('button');

// Usando una función anónima.
button.addEventListener('click', function() {
   console.log('Acabas de hacer click');
});

// Referenciando una función.
const handleButtonMouseOver = () => console.log('acabas de pasar con el curso
r por arriba del botón');
button.addEventListener('mouseover', handleButttonMouseOver);
```

La ventaja principal de este méntodo es que podemos asignar múltiples respuestas a cada evento:

```
const button = document.querySelector('button');

// Definimos las funciones que se van ejecutar (podemos usar cualquier forma
de definir funciones).
const doSomething = () => console.log('¡Hola!');

function doSomethingElse() {
   console.log('¡Hola otra vez!');
}

button.addEventListener('click', doSomething);
button.addEventListener('click', doSomthingElse);
```

Si queremos eliminar un **event listener** podemos usar el método **removeEventListener** (siempre que no usáramos una función anónima para definir la respuesta al evento):

```
// Continuando el ejemplo anterior.
button.removeEventListener('click', doSomethingElse);

// Al clicar en el botón ya no mostrará '¡Hola otra vez!' en la consola.
```

Event Objects

Todas las funciones que definimos como **event listeners** reciben por defecto un objeto que define el evento, este objeto tiene una serie de propiedades y métodos útiles:

```
const button = document.querySelector('button');

const handleClick = function(e) {
  // Imprime información sobre el evento.
  console.log(e);
};

button.addEventListener('click', handleClick);
```

La lista de propiedades y métodos podemos verla aquí (https://developer.mozilla.org/en-US/docs/Web/API/Event).

Probablemente la propiedad que más usemos es **target** que contiene una referencia al elemento que generó el evento. Esto es muy útil si queremos programar respuestas que afecten a ese elemento y no queremos o no podemos seleccionarlo previamente:

```
/*
  <ul>
    <li>0 clicks</li>
    <li>0 clicks</li>
    <li>0 clicks</li>
    <li>0 clicks</li>
  </ul>
*/

const listItems = document.querySelectorAll('ul li');

const handleItemClick = function(e) {
   // Referencia al elemento clicado.
   const item = e.target;

   if (!item.hasAttribute('data-count')) {
      item.setAttribute('data-count', 0);
   }

   const updatedCount = Number(item.getAttribute('data-count')) + 1;

   item.textContent = `${updatedCount} clicks`;
   item.setAttribute('data-count', updatedCount);
};

for (const item of listItems) {
   item.addEventListener('click', handleClickItem);
}
```

Veremos más adelante la propiedad **currentTarget** y como se diferencia con esta.

Cancelando el comportamiento por defecto

De vez en cuando los eventos tienen asignada una acción por defecto, por ejemplo, cuando hacemos click en un link, la página carga el nuevo documento o cuando hacemos click en un botón dentro de un formulario este se envía.

Podemos cancelar este comportamiento por defecto usando el método **.preventDefault( ):**

```
const links = document.querySelectorAll('a');

const handleLinkClick = (e) => {
  // Cancelamos el comportamiento por defecto del link.
  e.preventDefault();
  console.log(`Cancelada la navegación a: ${e.target.getAttribute('href')}`);
}

for(const link of links) {
  link.addEventListener('click', handleLinkClick);
}
```

Event bubbling

Cuando determinados eventos ocurren en un elemento, especialmente los generados por el usuario estos también ocurren en los elementos padres. Por ejemplo, cuando hacemos clic en un **<li>** también ocurre un evento de tipo clic en el **<ul>** padre, y en el padre de ese **<ul>** y así sucesivamente hasta el elemento raíz **<html>**.

A esto se le llama **event bubbling** y a las veces puede ser molesto si temos asignados **event listeners** similares a los elementos padres. Si queremos cancelar este **Event bubbling** y evitar que los eventos se propaguen a los padres de los elementos podemos usar el método **.stopPropagation( )** del objeto de evento.

```
const ul = document.querySelector('ul');
const lis = document.querySelectorAll('ul li');
const handleListClick = function(e) {
  console.log('Hiciste clic en el <ul>');
};

const handleItemClick = function(e) {
  // La siguiente línea impedirá que el evento se propague al <ul> padre.
  e.stopPropagation();
  console.log('Hiciste clic en el <li>');
};

for (const li of lis) {
  li.addEventListener('click', handleItemClick);
}

ul.addEventListener('click', handleListClick);
```

Event delegation

El **Event bubbling** permite hacer cosas muy útiles como delegar eventos a elementos padre. Por ejemplo, si tenemos una lista en la que modificamos dinámicamente sus elementos mediante los métodos conocidos:

```
/*
  <button class="add">Add item</button>
  <ul class="items">

  </ul>
*/

const add = document.querySelector('button.add');
const ul = document.querySelector('ul.items');

const handleAddClick = e => {
   const newItem = document.createElement('li');
   const itemText = document.createTextNode(`Ítem ${ul.children.length + 1} `);
   const deleteButton = document.createElement('button');
   deleteButton.textContent = 'Bórrame';
   deleteButton.classList.add('delete');
   newItem.appendChild(itemText);
   newItem.appendChild(deleteButton);
   ul.appendChild(newItem);
};

const handleListClick = e => {
   const target = e.target;

   // "e.target" siempre va a ser el elemento clicado.
   // "e.currentTarget" siempre va a ser el elemento asociado a este evento (en este caso el <ul>)

   if (target.matches('button.delete')) {
      const item = target.parentElement;
      item.parentElement.removeChild(item);
   }
};
```

```
add.addEventListener('click', handleAddClick);
ul.addEventListener('click', handleListClick);
```

Vemos que asignamos un listener **handleListClick** que se ejecuta cuando se hace click en la lista, y como sabemos por el **bubbling** del que hablamos antes, los clics a los elementos de la lista se van a propagar, por lo que:

- la propiedad **target** siempre va hacer referencia al elemento clicado que generó el evento.
- Comprobamos si ese target es un **button** con la clase **.delete** usando el método **.matches( )** que comprueba si el elemento coincide con un selector.
- Si es así borramos el ítem de la lista.
- Si no el evento no hará nada.

Completado. Continuamos.