



FULL STACK

**Comenzamos en unos
minutos**

ACADEMY
by NUMEN


Javascript: Clase 6

CSS y Eventos en el DOM

Estilos con Javascript

A través de Javascript podemos acceder a estilos CSS, agregarlos, quitarlos y mucho más. A través de la propiedad style de la API del document, podemos ver toda la variedad de propiedades CSS de las que podremos disponer.

```
<!-- index.html -->  
<a href="#" class="enlace"></a>
```




```
// index.js  
const $enlace = document.querySelector(".enlace");  
  
console.log($enlace.style)
```

Método `getAttribute()`

Este método nos permite acceder al contenido del atributo `style` de una etiqueta HTML. Si este atributo posee varias propiedades, nos devolverá toda la colección de propiedades que posea.

```
<!-- index.html -->  
<a href="#" class="enlace" style="color: red"></a>
```




```
// index.js  
const $enlace = document.querySelector(".enlace");  
  
console.log($enlace.getAttribute("style"))
```

Método getComputedStyle()

Nos permite ver las propiedades de nuestra etiqueta en forma de posiciones de un arreglo. Incluso podemos visualizar los estilos por defecto que ya se le aplican a la etiqueta.

```
<!-- index.html -->  
<a href="#" class="enlace" style="color: red"></a>
```



```
// index.js  
const $enlace = document.querySelector(".enlace");  
  
console.log(getComputedStyle($enlace))
```

Método `getPropertyValue()`

Este método nos permite acceder al valor de una propiedad CSS. Veámoslo:

```
<!-- index.html -->  
<a href="#" class="enlace" style="color: red"></a>
```




```
// index.js  
const $enlace = document.querySelector(".enlace");  
  
console.log(getComputedStyle($enlace).getPropertyValue("color"))
```

Método setProperty()

Este método nos permite insertar una propiedad CSS al DOM. Veámoslo:

```
<!-- index.html -->  
<a href="#" class="enlace">Esto es un enlace</a>
```



```
// index.js  
const $enlace = document.querySelector(".enlace");  
  
$enlace.style.setProperty("text-decoration", "none")
```


Estilos por notación del punto

Otra forma en la que podríamos agregar un valor CSS es conectar con la propiedad CSS implícita de objeto `CSSStyleDeclaration`. Veámoslo:

```
<!-- index.html -->  
<a href="#" class="enlace">Esto es un enlace</a>
```



```
// index.js  
const $enlace = document.querySelector(".enlace");  
  
$enlace.style.backgroundColor = "pink"  
$enlace.style.color = "blue"  
$enlace.style.padding = "20px"
```

Variable CSS con Javascript

Tambien podemos manipular variables CSS. Veamos como sería esto.

```
<!-- index.html -->
<section>
  <h2>Título</h2>
  <p>Esto es un párrafo</p>
</section>
<section>
  <h2>Otro título</h2>
  <p>Esto es otro párrafo</p>
</section>
```



```
/* style.css */
:root {
  --yellow-color: yellow;
  --dark-color: black;
}
```



```
// index.js
const $html = document.documentElement,
      $body = document.body;

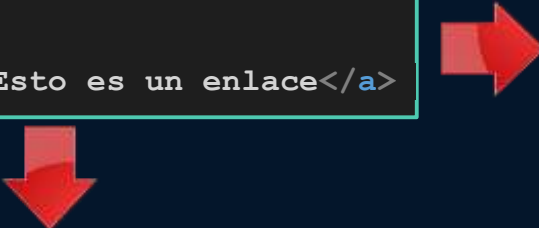
let varDarkColor = getComputedStyle($html).getPropertyValue("--dark-color"),
    varYellowColor = getComputedStyle($html).getPropertyValue("--yellow-color");

$body.style.backgroundColor = varDarkColor;
$body.style.color = varYellowColor;
```

Agregando y removiendo clases

Lo último que nos queda por saber sobre CSS es como agregar una clase adicional a una etiqueta. Para ello utilizaremos el método `add()`. Veámoslo:

```
<!-- index.html -->
<a href="#" class="enlace">Esto es un enlace</a>
```



```
// index.js
const $enlace = document.querySelector(".enlace");
console.log($enlace.classList)

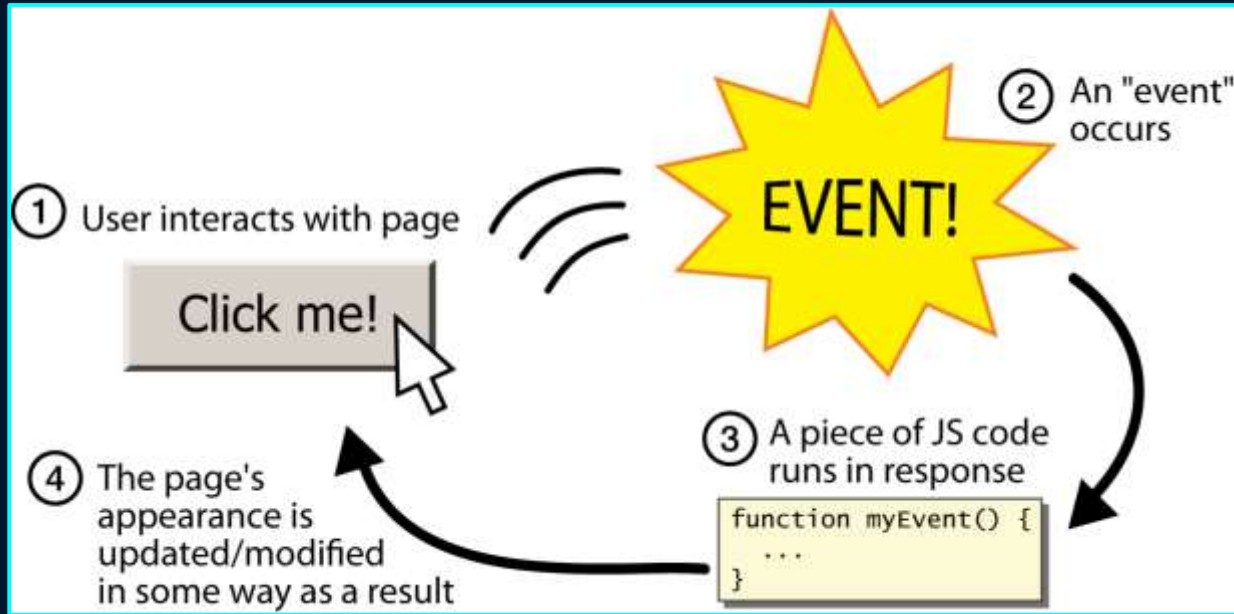
$enlace.classList.add("agregar-color")
$enlace.classList.remove("agregar-color")
```

```
/* style.css */
.enlace{
    text-decoration: none;
}

.agregar-color {
    color: red;
}
```

EVENTOS

Los eventos son un tipo de método que nos permite ejecutar funciones en determinadas situaciones, como por ejemplo, cuando el usuario interactúa con un botón.



Eventos como atributos

Una forma de manejar eventos, aunque no la recomendada, es a través de atributos de las etiquetas HTML. Veámos un ejemplo:


```
<!-- index.html -->  
<button onclick="alert('Alerta desde un evento')">Mi primer evento</button>
```

Llamando una función por atributo

En el ejemplo anterior ejecutamos una función de alerta, de esas que vienen colgadas del objeto window.

Ahora nos toca llamar una función que hayamos programado en nuestro archivo.js.

```
<!-- index.html -->  
<button onclick="holaMundo()">Mi primer evento</button>
```





```
// index.js  
function holaMundo() {  
    alert("Evento desde archivo.js")  
}
```

Eventos semánticos

Así como podíamos agregar propiedades CSS a través de la notación de punto, también es posible manejar eventos de la esta forma. Veamos un ejemplo:

```
<!-- index.html -->
<button id="evento">Mi primer evento</button>
```



```
// index.js
function holaMundo() {
    alert("Evento desde archivo.js")
    console.log(event)
}

const $evento = document.getElementById("evento")

$evento.onclick = holaMundo;
```


```
// index.js
const $evento = document.getElementById("evento")

$evento.onclick = function (e) {
    alert("Evento desde archivo.js")
}
```

Eventos Listeners (manejadores)

El `addEventListener` es un método que nos permite ejecutar múltiples eventos. Como parámetro recibe el evento elegido y la función a ejecutar.

```
<!-- index.html -->  
<button id="evento">Mi primer evento</button>
```




```
// index.js  
const $evento = document.getElementById("evento")  
  
function holaMundo() {  
    alert('Hola Mundo')  
}  
  
function adiosMundo() {  
    alert('Adios Mundo')  
}  
  
$evento.addEventListener("click", holaMundo)  
$evento.addEventListener("click", adiosMundo)
```


Función anónima como parámetro

Además de recibir una función declarada, `addEventListener()` puede recibir como parámetro una función anónima.

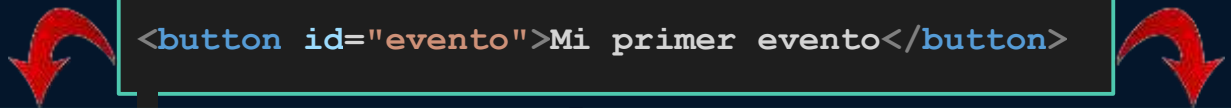
```
<!-- index.html -->  
<button id="evento">Mi primer evento</button>
```



```
// index.js  
const $evento = document.getElementById("evento")  
  
$evento.addEventListener("click", function(e) {  
    alert('Hola Mundo')  
}))
```

Escuchar funciones con parámetros

¿Qué pasaría si deseamos que la función que pasamos por parámetro en el `addEventListener` reciba parámetros? ¿Como lo programaríamos? Veámoslo:




```
<!-- index.html -->
<button id="evento">Mi primer evento</button>
```

```
// index.js
const $evento = document.getElementById("evento")

function holaMundo(nombre = "usuario") {
  alert(`Hola ${nombre}`)
}


$evento.addEventListener('click', holaMundo)
```



```
// index.js
const $evento = document.getElementById("evento")

function holaMundo(nombre = "usuario") {
  alert(`Hola ${nombre}`)
}

$evento.addEventListener('click', function (e) {
  holaMundo()
})
```



Flujo de Eventos (parte 1)

Los eventos en Javascript fluyen como si fuesen una burbuja. Parten desde el elemento más interno y se propagan hasta el elemento más externo. Aquí veremos un ejemplo:

```
<!-- index.html -->
<section class="eventos-flujo">
  <div class="uno">
    1
    <div class="dos">
      2
      <div class="tres">
        3
      </div>
    </div>
  </div>
</section>
```

```
/* styles.css */
.eventos-flujo div {
  padding: 4rem;
  font-size: 2rem;
  text-align: center;
}
.uno {
  background-color: yellow;
}
.dos {
  background-color: gold;
}
.tres {
  background-color: lightyellow;
}
```

Flujo de Eventos (parte 2)

```
// index.js

$divsEventos = document.querySelectorAll(".eventos-flujo div")

function flujoEventos(e) {
    console.log(`Hola, te saluda ${this.className}, el click lo originó ${e.target.className}`)
}

$divsEventos.forEach(div => {
    div.addEventListener("click", flujoEventos)
});
```

Deteniendo la propagación

Para detener la propagación podemos usar un método llamado `stopPropagation()` dentro de la función que queremos que deje de propagarse.

Probemos usarla en la función del ejemplo de las diapos de Flujo de Eventos.

```
// index.js  
e.stopPropagation()
```

preventDefault()

Existe otro método para detener la propagación. Este se utiliza principalmente para evitar que nuestro sitio se recargue al enviar los datos de un formulario. Veamos un sencillo ejemplo:

```
<!-- index.html -->  
<a href="https://www.google.com/?hl=es">Ir a Google</a>
```



```
// index.js  
const $a = document.querySelector("a")  
  
$a.addEventListener("click", function (e) {  
    alert(`¿A donde crees que vas?`);  
    e.preventDefault();  
})
```

Delegación de eventos

Para evitar declarar muchos `addEventListener()`, algo que podemos hacer es programar una delegación de eventos en un solo `addEventListener()` afectando directamente al `document`.

```
<!-- index.html -->
<section class="una-seccion">
  <h2>Título</h2>
  <article>
    <h3>Subtítulo</h3>
    <p>Lorem, ipsum dolor.</p>
    <a href="https://www.google.com/?hl=es">Ir a Google</a>
  </article>
</section>
```



```
// index.js
document.addEventListener("click", function (e) {
  console.log("Hiciste click en", e.target);

  if (e.target.matches(".una-seccion a")) {
    alert("A dónde crees que vas?");
    e.preventDefault()
  }
});
```

ACADEMY
by NUMEN