

# JAVASCRIPT



**JS**

# Contenido

## ★ Eventos

- Del mouse
- Del teclado
- De la ventana
- Listeners

## ★ Timers

- `setTimeout`
- `setInterval`
- `cleaners`

¿Qué es un evento?

x

Un evento es algo que pasa en el navegador o que es ejecutado por el usuario.

### Algunos ejemplos:

- La página terminó de cargar
- El input de un formulario cambió
  - Hicieron click en un botón

# Eventos más comunes

- ★ onclick
- ★ ondblclick
- ★ onmouseover
- ★ onmouseout
- ★ onmousemove
- ★ onscroll
- ★ onkeydown
- ★ onload
- ★ onfocus
- ★ onblur
- ★ onchange
- ★ onsubmit

# Eventos

## Existen 2 formas de asignar eventos

La primera es estableciendo una propiedad en el objeto que deseamos sea objetivo del evento:

```
element.onNameEvent = function () {  
    // Lo que sucede al disparar el evento  
}
```

```
btn.onclick = function () {  
    alert('Dont push me bro');  
}
```

# Eventos

## Existen 2 formas de asignar eventos

La segunda es utilizando un *listener*.

```
element.addEventListener(event, fn);
```

**event:** evento que deseamos ejecutar (*string*).

**fn:** callback que se invoca al dispararse el evento (*function*).



```
btn.addEventListener('click', sayHello);

function sayHello() {
    alert('Hello. Is anybody out there?');
}
```

Podemos utilizar la palabra reservada **this** que en este contexto hace referencia al objeto que ejecutó el evento.

```
btn = document.querySelector('.my_button');
```

```
btn.addEventListener('click', sayHello);
```

```
function sayHello() {  
    alert('Hello. Is anybody out there?');  
    console.log(this); // ¿?  
}
```

The background is a solid yellow color with a pattern of various geometric shapes in a lighter yellow shade. These shapes include squares, circles, and crosses, some of which are rotated or tilted. A small, solid yellow circle is positioned directly behind the 'VS' text.

onclick  
vs  
addEventListener

# onclick vs addEventListener

## ¿Existen diferencias?

La principal diferencia entre **onclick** y **addEventListener**, es que el *listener* nos permite que un mismo elemento pueda tener muchos canales escuchándolo, es decir, muchos *listeners*.

El **onclick** en cambio solamente tendrá un solo evento relacionado.



# preventDefault()

¿qué podemos hacer si queremos evitar que se dispare el evento por defecto de un elemento?

# preventDefault()

```
// html
<a href="http://google.com">Cliiick oooooon meeeeee</a>

// javascript
var btn = document.querySelector('a');

btn.onclick = function () {
    alert('Hi there!');
}
```



Al dar click sobre el elemento `<a>` se disparará la función que llama al **alert()**.

Sin embargo, cuando se termina la ejecución de éste método, se dispara el evento por defecto, redirección del **href**.



Para evitar esto, lo primero que tenemos que hacer es **CAPTURAR** al evento que viene por defecto con el elemento **(href)**.

# preventDefault()

```
// html
<a href="http://google.com">Click on me</a>

// javascript
var btn = document.querySelector('a');

btn.onclick = function (event) {
    alert('Hi there!');
}
```

Acto seguido ejecutar sobre éste **EVENTO** el **preventDefault()**.

# preventDefault()

```
// html
<a href="http://google.com">Click on me</a>

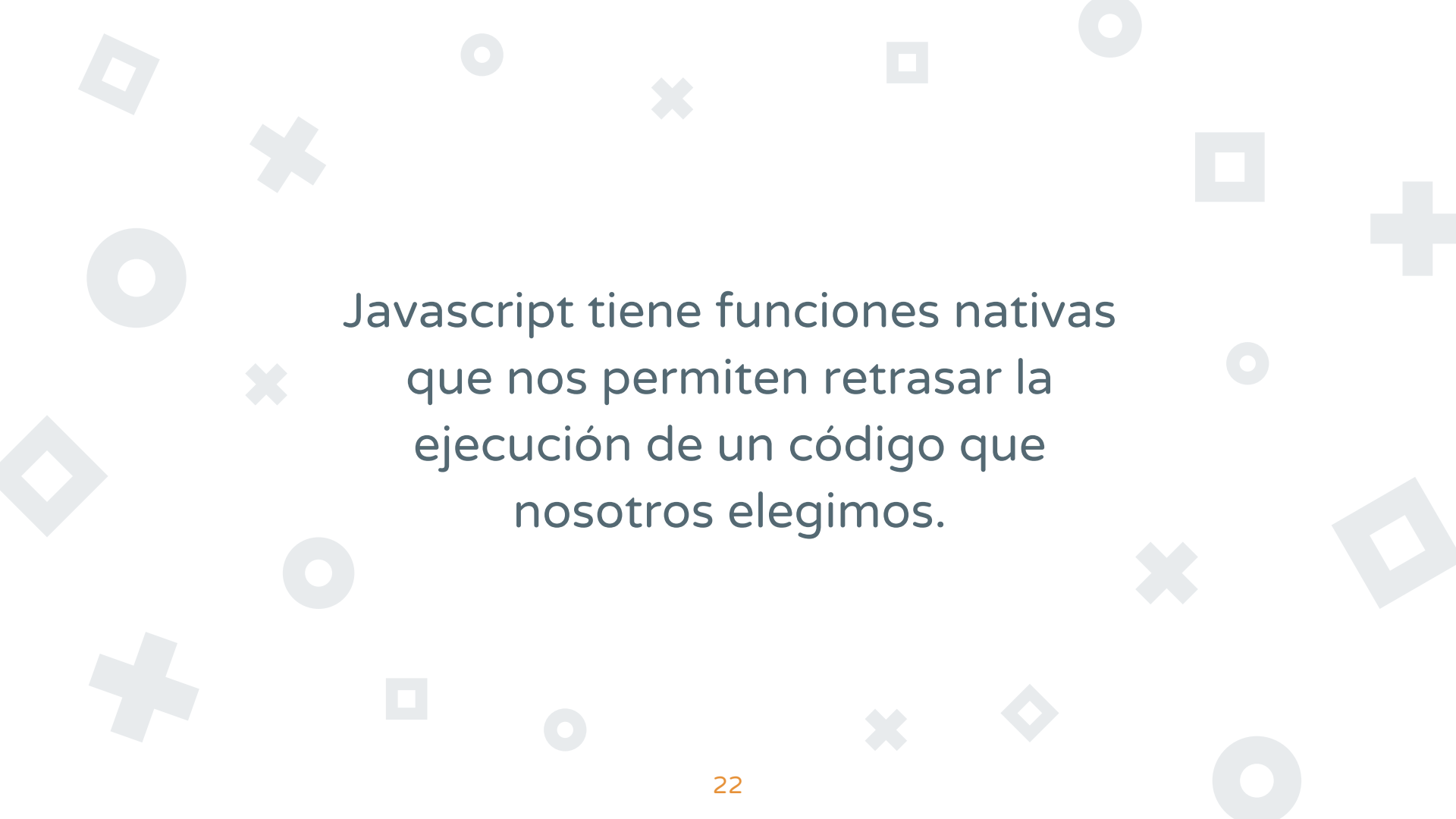
// javascript
var btn = document.querySelector('a');

btn.onclick = function (event) {
    alert('Hi there');
    event.preventDefault();
}
```



# TIMERS

♪ Do you remember the time  
When we fell in love ♪

The background is white and decorated with various light gray geometric shapes scattered across the page. These shapes include squares, circles, and crosses, some of which are hollow and others are solid. The shapes are of different sizes and are oriented in various directions, creating a subtle pattern.

Javascript tiene funciones nativas  
que nos permiten retrasar la  
ejecución de un código que  
nosotros elegimos.

# Timers

## setTimeout

Se utiliza cuando queremos que nuestro código se ejecute una sola vez, pasado un tiempo establecido.

```
setTimeout(fn, delay);
```

**fn:** callback que se invoca al dispararse el evento (*function*).

**delay:** tiempo de espera para ejecutar el callback, en milisegundos (*number*).



```
var myOwnTimeout = setTimeout(areYouThere, 3000);
```

```
function areYouThere() {  
    alert('Are you there bro?');  
}
```



# Timers

## setInterval

Se utiliza cuando queremos que nuestro código se ejecute MUCHAS VECES, pasado un tiempo establecido.

```
setInterval(fn, delay);
```

**fn:** callback que se invoca al dispararse el evento (*function*).

**delay:** tiempo de espera para ejecutar el callback, en milisegundos (*number*).



```
var myOwnInterval = setInterval(everySecond, 3000);
```

```
function everySecond() {  
    alert('Hello, how are you?');  
}
```





¡NO!

Podemos detener un timer  
cuando lo deseemos.



```
var myOwnTimeout = setTimeout(areYouThere, 3000);
```

```
function areYouThere() {  
    alert('Are you there bro?');  
}
```

```
clearTimeout(myOwnTimeout);
```



```
var myOwnInterval = setInterval(everySecond, 3000);
```

```
function everySecond() {  
    alert('Hello, how are you?');  
}
```

```
clearInterval(myOwnInterval);
```

**¡A practicar!**

Práctica Integradora



```
// To Do  
console.log("Practice Time");
```