JavaScript:
BOM y Eventos

Alicia Vázquez
@aliciaFPInf

01

Eventos

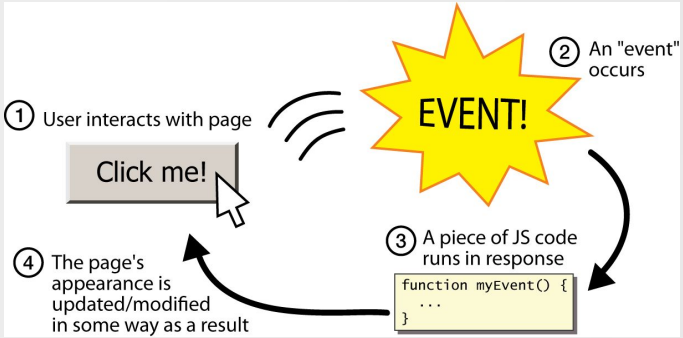
Eventos

Los eventos son la manera que tenemos en Javascript de controlar las acciones de los usuarios y definir un comportamiento de la página cuando se produzca cada uno de ellos.

Para entender los eventos necesitamos conocer algunos conceptos básicos:

- **Evento**: Es algo que ocurre cuando el usuario interactúa con el documento, pero también cuando la página se ha cargado, o cuando una imagen que no se puede cargar porque no esté disponible, etc.
- **Tipo de evento**: Es el tipo del suceso que ha ocurrido, por ejemplo, un clic sobre un botón, pulsación de una tecla, scroll o el envío de un formulario. Para cada tipo de elemento de la página hay diversos tipos de eventos de Javascript.
- **Handler** (Manejador de evento)**:** es el comportamiento que nosotros podemos asignar como respuesta a un evento. Será una función Javascript, que asociaremos a un tipo de evento en concreto.
- **Listener:** Es el elemento que “escucha” los eventos que definimos y ejecutará el manejador especificado.

Eventos



Evento	Manejador Evento	Descripción	Elemento para los que está definido
blur	Onblur	Seleccionar el elemento	<button> <input> <label> <select> <textarea> <body>
change	Onchange	Deseleccionar un elemento que se ha modificado	<input> <select> <textarea>
click	Onclick	Al hacer click en el mouse	Todos los elementos
dblclick	Ondblclick	Al hacer doble click en el mouse	Todos los elementos
focus	Onfocus	Darle atención a un elemento	<button> <input> <label> <select> <textarea> <body>
keydown	Onkeydown	Pulsar una tecla, sin soltar	Elemento de formularios y <body>
keypress	Onkeypress	Pulsar una tecla	Elemento de formulario y <body>
keyup	Onkeyup	Soltar una tecla pulsada	Elemento de formulario y <body>

Eventos

Definimos las acciones (funciones) que queremos realizar.

```
//Handler
function clickTitulo(){
   console.log("Se ha hecho click en el titulo");
}
```

Podemos añadir los “escuchadores” en varios lugares.

- Dentro del elemento HTML del que queremos detectar la acción y directamente en el HTML.

```
<h2 id="titulo" onclick="clickTitulo()">Formulario</h2>
```

- Directamente en el JS.

```
const titulo = document.getElementById('titulo')
titulo.onclick = clickTitulo; //Asignamos, no estamos invocando!!
```

.addEventListener()

Para poder asignar varias acciones a un mismo evento usaremos el método **addEventListener()** que tiene todos los elementos del DOM.

addEventListener("evento",handler,true/false)

- Nombre del evento
- Función u objeto que se ejecutará en el momento que se produzca el evento que debe capturarse.
- Booleano, que por defecto es false, indica si la escucha debe hacer a todos los elementos del DOM o solo al asignado.


```
titulo.addEventListener("click",clickTitulo);
titulo.addEventListener("click",() => console.log("Se ha hecho click en el titulo ARROW"));
```

Eventos

Otra cosa interesante de usar **addEventListener()**, es que podemos borrar la función que tenemos asignada a un evento usando el método **removeEventListener()**, lo que nos da más control a la hora de añadir o eliminar eventos a un elemento y agiliza la programación.

```
titulo.removeEventListener("click", clickTitulo);
//No puedo eliminar la función que he definido como arrow, es anónima.
```

El standard de JS me ofrece una serie de eventos predefinidos y puedo querer otros.

```
//Evento que nos indica que el DOM ha sido cargado y por lo tanto ya tenemos acceso a él.
document.addEventListener("DOMContentLoaded", () => console.log("Página cargada!"));
```

Object Event

A menudo necesitamos acceder a nuestro **evento** desde la función *handler*, este parámetro es pasado por defecto sin necesidad de especificarlo en el *addEventListener*.

```
const button = document.querySelector("button");
button.addEventListener("click", (event) => {
  console.log(event);
});
```

Nota: JS considera que los eventos de click los hace el mouse, por ello nos dice que es un **PointerEvent**

Object Event: Propiedades de posicionamiento

Propiedad	Tipo de Dato	Descripción
`clientX, clientY`	`number`	Posición X o Y del puntero respecto a la **ventana** (viewport).
`pageX, pageY`	`number`	Posición X o Y del puntero respecto a **toda la página**, incluyendo scroll.
`screenX, screenY`	`number`	Posición X o Y del puntero respecto a la **pantalla** del usuario.
`offsetX, offsetY`	`number`	Posición X o Y del puntero relativa al **elemento** que disparó el evento.
`movementX, movementY`	`number`	Diferencia en X o Y del puntero respecto al evento anterior.
`relatedTarget`	`EventTarget`	Elemento relacionado en eventos como `mouseenter` y `mouseleave`.
`ctrlKey`	`boolean`	`true` si la tecla **Ctrl** estaba presionada durante el evento.
`shiftKey`	`boolean`	`true` si la tecla **Shift** estaba presionada durante el evento.
`altKey`	`boolean`	`true` si la tecla **Alt** estaba presionada durante el evento.
`metaKey`	`boolean`	`true` si la tecla **Meta** (⌘ en Mac, Windows en PC) estaba presionada.

Ejercicio 2: Probando eventos

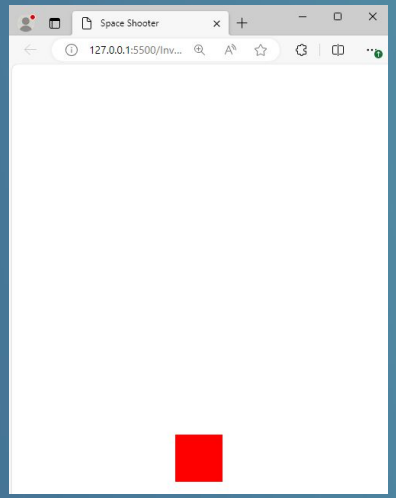
Crea una página HTML con varios elementos: botón, capa, imagen, un campo de texto.

Añade unos escuchadores de eventos para:

- Al hacer clic en el botón se lance un mensaje al usuario.
- A hacer doble clic salte otra alerta.
- Al detectar que el ratón pasa por encima del elemento se vaya escribiendo en consola su posición X,Y
- Al presionar “Enter” salte una alerta.

Ejercicio 3: Space shooter

Crea una página que contenga un elemento abajo de la página y que se mueva según el movimiento del ratón capturando la **coordenada X** y moviéndose igual. Usa css y que inicialmente sea una div, luego ya pondrás una imagen. (getBoundingClientRect())



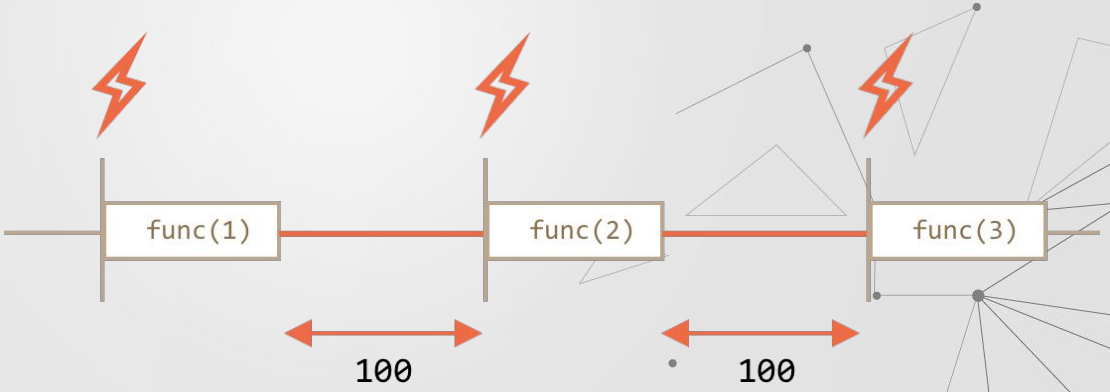
03

Timmers

Timmers

Los eventos, como ya hemos comentado, no siempre vienen generados por el usuario. Nosotros también podemos querer generar eventos al pasar X tiempo o cada X tiempo. Para ello JS nos ofrece dos funciones que generan un evento e interrumpen el hilo de ejecución.

- `setTimeout`
- `setInterval`



Timmers

Estas funciones permiten ejecutar un trozo de código en el futuro (cuando transcurran los milisegundos indicados). Para definirlo tenemos dos opciones:

- **`Identificador = setTimeout(function, miliseconds)`**: ejecuta la función indicada una sóla vez, cuando transcurran los milisegundos
- **`Identificador = setInterval(function, miliseconds):`** ejecuta la función indicada cada vez que transcurran los milisegundos, hasta que sea cancelado el timer.

A ambas se le pueden pasar más parámetros tras los milisegundos y serán los parámetros que recibirá la función a ejecutar. Ambas funciones devuelven un identificador que nos permitirá cancelar la ejecución del código, con:

- **clearTimeout(identificador)**
- **clearInterval(identificador)**

Timmers

```
const idTimeout = setTimeout(() => alert('Timeout que se ejecuta al cabo de 5 seg.'), 5000);

//Creamos una función y pasamos parametro
function showMessage(msg) {
 alert(msg)
}

//especificamos la función que qeuremos que se ejecute, no la ejecitamos --> No ponemos ();
const idTimeoutF = setTimeout(showMessage, 1000, 'Timeout que se ejecuta al cabo de 1 seg.');

let i = 1;
const idInterval = setInterval(() => {
 alert('Interval cada 3 seg. Ejecución n°: '+ i++);
 if (i === 5) {
   clearInterval(idInterval);
   alert('Fin de la ejecución del Interval');
 }
}, 3000);
```

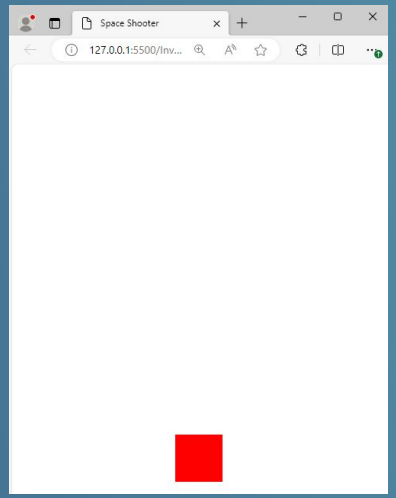
Haz pruebas:

Ejecuta el código anterior.

¿Serías capaz de abrir y crear una ventana cada X tiempo? ¿O de ir cambiando el color de fondo de negro a blanco cada 2 segundos?

Space shooter

Añade un evento a la barra espaciadora que lance una “bala” que salga desde la posición de la “nave” y suba hacia arriba, hasta desaparecer.



Discutamos, ¿como se podría implementar el Space invaders”? ¿Qué eventos crees que hay que tener en cuenta?

Hagamos un esquema juntos.