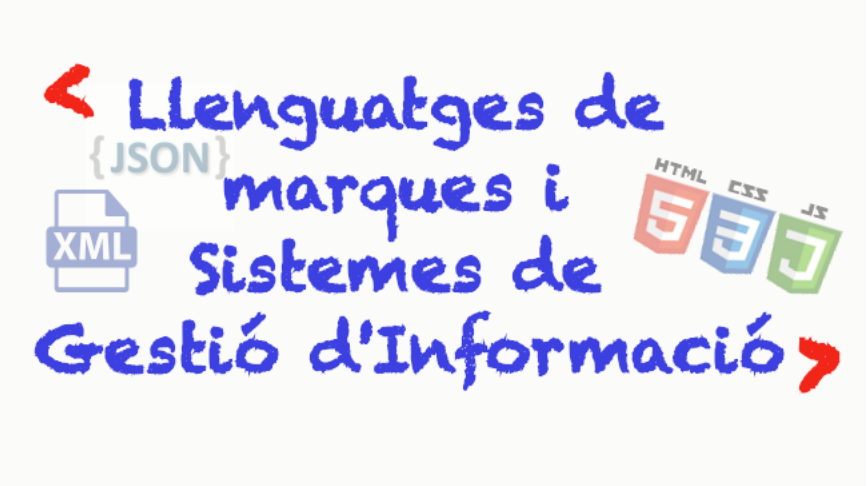
PROGRAMACIÓ AMB JAVASCRIPT

EVENTS



Contingut

1. Introducció

Quan un usuari interactua amb una aplicació, es produeixen una sèrie d'esdeveniments (de teclat, ratolí, etc.) que el nostre codi hauria de gestionar de manera adequada. Hi ha molts esdeveniments que poden ser capturats i processats. Veurem alguns dels més comuns.

1.1. Events de la pàgina

Aquests esdeveniments són produïts en el document HTML. Normalment afecten l'element **body**.

load → Aquest esdeveniment es llança quan el document HTML ha acabat de carregar-se. És útil per a realitzar accions que requerisquen que el DOM haja sigut completament carregat (com consultar o modificar el DOM).

unload → Ocorre quan el document és destruït, per exemple, després de tancar la pestanya on la pàgina estava carregada.

beforeunload → Ocorre just abans de tancar la pàgina. Per defecte, un missatge pregunta a l'usuari si vol realment eixir de la pàgina, però hi ha altres accions que poden ser executades.

resize → Aquest esdeveniment es llança quan la grandària del document canvia (normalment s'usa si la finestra es redimensiona)

1.2. Events del teclat

keydown → L'usuari pressiona una tecla. Si la tecla es manté premuda durant un temps, aquest esdeveniment es generarà de forma repetida.

keyup → Es llança quan l'usuari deixa de pressionar la tecla.

keypress → Més o menys el mateix que keydown. Acció de prémer i alçar.

1.3. Events del ratolí

click → Aquest esdeveniment ocorre quan l'usuari prem un element (pressiona i alça el dit del botó → mousedown + mouseup). També normalment es llança quan un esdeveniment tàctil de toc (tap) és rebut.

dblclick → Es llança quan es fa un doble clic sobre l'element

mousedown → Aquest esdeveniment ocorre quan l'usuari pressiona un botó del ratolí

mouseup → Aquest esdeveniment ocorre quan l'usuari alça el dit del botó del ratolí

mouseenter → Es llança quan el punter del ratolí entra en un element

mouseleave → Es llança quan el punter del ratolí ix d'un element

mousemove → Aquest esdeveniment es diu repetidament quan el punter d'un ratolí es mou mentre està dins d'un element

1.4. Events tàctils

Touchstart, touchend, touchmove, touchcancel

1.5. Events de formulari

focus → Aquest esdeveniment s'executa quan un element (no sols un element d'un formulari) té el focus (és seleccionat o està actiu).

blur → S'executa quan un element perd el focus.

change → S'executa quan el contingut, selecció o estat de la casella de selecció d'un element canvia (només <input>, <select>, i <textarea>)

input → Aquest esdeveniment es produeix quan el valor d'un element <input> o <textarea> canvia.

select → Aquest esdeveniment es llança quan l'usuari selecciona un text d'un <input> o <textarea>.

submit → S'executa quan un formulari és enviat (l'enviament pot ser cancel·lat).

2. Gestor d'events

Hi ha moltes maneres d'assignar un codi o funció a un determinat esdeveniment.

Veurem les dues formes possibles (per a ajudar a entendre codi fet per uns altres), però el recomanat (i la forma vàlida per a aquest curs) és usar **event listeners**.

2.1. Maneig d'esdeveniments clàssic (menys recomanat)

El primer de tot, podem posar codi JavaScript (o cridar a una funció) en un atribut d'un element HTML. Aquests atributs es nomenen com els esdeveniments, però amb el prefix 'on' (clic → onclick). Exemple:

```
<!DOCTYPE>
<html>
	<head>
		<title>Ejemplo JS</title>
	</head>
	<body>
		<div id="div1">
			<p>
				<input type="text" onclick="alert('!Me has pulsado!')" />
			</p>
		</div>
		<script src="./ejemplo1.js"></script>
	</body>
</html>
```

Si cridàrem a una funció, podem passar-li paràmetres (si és necessari). La paraula reservada this i event es poden usar per a passar l'element HTML (afectat per l'esdeveniment) i / o l'objecte amb informació de l'esdeveniment a la funció.

```
Archivo: ejemplo1.html

<input type="text" id="input1" onclick="inputClick(this, event)" />

Archivo: ejemplo1.js

function inputClick(element, event) {
   // Mostrará "Un evento click ha sido detectado en  #input1"
   alert("Un evento" + event.type + " ha sido detectado en #" + element.id);
}
```

1 Exemple codi JS. onclick

Podem afegir un manejador (funció) d'esdeveniment des de codi, accedint a la propietat corresponent (onclick, onfocus, …), o assignar-los un valor nul si volem deixar d'escoltar algun esdeveniment.

```
let input = document.getElementById("input1");
input.onclick = function(event) {
   // Dentro de esta función, 'this' se refiere al elemento
   alert("Un evento " + event.type + " ha sido detectado en " + this.id);
}
```

2 Exemple codi JS. onclick II

2.2. Event listeners (recomanat)

El mètode per a manejar esdeveniments explicat a dalt té alguns desavantatges, per exemple, no es poden gestionar diverses funcions manejadoras per a un mateix esdeveniment.

Per a afegir un event listener, usem el mètode addEventListener sobre l'element. Aquest mètode rep almenys dos paràmetres. El nom de l'esdeveniment (una cadena) i un manejador (funció anònima o nom d'una funció existent).

```
let input = document.getElementById("input1");
input.addEventListener('click', function(event) {
   alert("Un evento " + event.type + " ha sido detectado en " + this.id);
});
```

3 Exemple codi JS. Event listener

Podem afegir tants manejadores com vulguem. No obstant això, si volem eliminar un manejador, hem d'indicar quina funció estem eliminant.

```
let inputClick = function(event) {
        console.log("Un evento " + event.type + " ha sido detectado en " + this.id);
};
let inputClick2 = function(event) {
        console.log("Yo soy otro manejador para el evento click!");
};
let input = document.getElementById("input1");
// Afegim tots dos manejadores. En fer clic, es ejecutarian tots dos per ordre
input.addEventListener('click', inputClick);
input.addEventListener('click', inputClick2);
// Així és com s'elimina el manejador d'un esdeveniment
input.removeEventListener('click', inputClick);
input.removeEventListener('click', inputClick2);
```

2.3. Objecte de l'esdeveniment

L'objecte de l'esdeveniment és creat per JavaScript i passat al manejador com a paràmetre. Aquest objecte té algunes propietats generals (independentment del tipus d'esdeveniment) i altres propietats específiques (per exemple, un esdeveniment del ratolí té les coordenades del punter, etc.).

Aquestes són algunes **propietats generals** que tenen tots els esdeveniments:

target → L'element que llança l'esdeveniment (si va ser pres, etc...).

type → El nom de l'esdeveniment: 'click', 'keypress', …

cancelable → Retorna true o false. Si l'esdeveniment es pot cancel·lar significa que cridant a event.preventDefault() es pot anul·lar l'acció per defecte (L'enviament d'un formulari, el clic d'un link, etc…).

bubbles → Retorna cert o fals depenent de si l'esdeveniment s'està propagant

preventDefault() → Aquest mètode prevé el comportament per defecte (carregar una pàgina quan es prem un enllaç, l'enviament d'un formulari, etc.)

stopPropagation() → Prevé la propagació de l'esdeveniment.

stopImmediatePropagation() → Si l'esdeveniment té més d'un manejador, es diu a aquest mètode per a previndre l'execució de la resta de manejadors.

Depenent del tipus d'esdeveniment, l'objecte tindrà **diferents propietats**:

MouseEvent

button → Retorna el botó del ratolí que l'ha pres (0: botó esquerre, 1: la roda del ratolí, 2: botó dret).

clientX, **clientY** → Coordenades relatives del ratolí en la finestra del navegador quan l'esdeveniment va ser llançat.

pageX, pageY → Coordenades relatives del document HTML, si s'ha realitzat algun tipus de desplaçament (scroll), aquest serà afegit (usant clientX, clientY no s'afegeix)

screenX, screenY → Coordenades absolutes del ratolí en la pantalla.

detail → Indica quantes vegades el botó del ratolí ha sigut pres (un clic, doble, o triple clic).

KeyboardEvent

key → Retorna el nom de la tecla presa.

keyCode → Retorna el codi del caràcter Unicode en l'esdeveniment keypress, keyup o keydown.

altKey, ctrlKey, shiftKey, metaKey → Retornen si les tecles "alt", "control", "shift" o "meta" han sigut premudes durant l'esdeveniment (Bastant útil per a les combinacions de tecles com ctrl+c). L'objecte MouseEvent també té aquestes propietats.

3. Propagación de eventos (bubbling)

Moltes vegades, hi ha elements en una web que se solapen amb altres elements (estan continguts dins). Per exemple, si premem sobre un paràgraf que està contingut en un element , l'esdeveniment s'executarà en el paràgraf, en el o en tots dos? Quin s'executa primer?.

Per exemple, veurem què ocorre amb aquests dos elements (un element dins d'un altre) quan fem clic en ells.

Archivo: ejemplo1.html

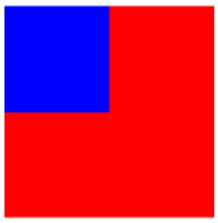
```
<div id="div1" style="background-color: red; width: 200px; height: 200px;">
  <div id="div2" style="background-color: blue; width: 100px; height: 100px;"></div>
</div>
```

Archivo: ejemplo1.js

```
let divClick = function(event) {
   console.log("Has pulsado: " + this.id);
};

let div1 = document.getElementById("div1");
let div2 = document.getElementById("div2");

div1.addEventListener('click', divClick);
div2.addEventListener('click', divClick);
```



4 Exemple codi HTML i JS

Si fem clic sobre l'element roig , s'imprimirà només “Has premut: div1”. No obstant això, si premem sobre l'element blau (el qual està dins de l'element roig) imprimirà tots dos missatges (#div2 primer). En conclusió, per defecte l'element el qual està al capdavant (normalment un element fill) rep l'esdeveniment primer i llavors passa a executar els manejadors que conté.

Normalment, la propagació d'esdeveniments va de pares a fills, però podem canviar aquest procés afegint un tercer paràmetre al mètode addEventListener i establir-lo a true.

Veurem un altre exemple:

Archivo: ejemplo1.html

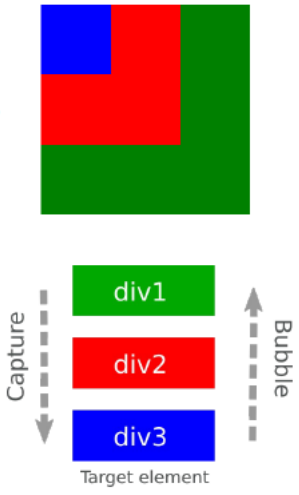
```
<div id="div1" style="background-color: green; width: 150px; height: 150px;">
    <div id="div2" style="background-color: red; width: 100px; height: 100px;">
        <div id="div3" style="background-color: blue; width: 50px; height: 50px;"></div>
    </div>
</div>
```

Archivo: ejemplo1.js

```
let divClick = function(event) {
   // eventPhase: 1 -> capture, 2 -> target (objetivo), 3 -> bubble
   console.log("Has pulsado: " + this.id + ". Fase: " + event.eventPhase);
};

let div1 = document.getElementById("div1");
let div2 = document.getElementById("div2");
let div3 = document.getElementById("div3");

div1.addEventListener('click', divClick);
div2.addEventListener('click', divClick);
div3.addEventListener('click', divClick);
```



5 Exemple propagació d'events

Per defecte, quan premem el blau (div3), imprimeix:

Has pulsado: div3. Fase: 2 → Target element

Has pulsado: div2. Fase: 3 → Bubbling

Has pulsado: div1. Fase: 3 → Bubbling

Si s'estableix un tercer paràmetre a true, s'imprimirà:

Has pulsado: div1. Fase: 1 → Propagation

Has pulsado: div2. Fase: 1 → Propagation

Has pulsado: div3. Fase: 2 → Target element

Podem cridar al mètode **stopPropagation** en l'esdeveniment, no continuarà la propagació (si és en la fase de captura) o en (la fase de propagació o bubbling).

```
let divClick = function(event) {
   // eventPhase: 1 -> capture, 2 -> target (clicked), 3 -> bubble
   console.log("Has pulsado: " + this.id + ". Fase: " + event.eventPhase);
   event.stopPropagation();
};
```

6 Exemple stopPropagation

Ara, quan fem clic l'element imprimirà només un missatge, "Has pulsado: div3. Fase: 2", si el tercer argument no s'ha establit (o s'ha posat a false), o "Has pulsado: div1. Fase: 1" si el tercer argument s'ha marcat a true (l'element pare prevé als fills de rebre'l en aquest cas).

4. Obtindre valors d'un formulari

Per a capturar l'enviament d'un **formulari** en JavaScript, hem de capturar l'esdeveniment **submit** d'aquest. És més recomanable això que usar l'esdeveniment clic d'un botó, ja que en pressionar la tecla enter també s'envia el formulari, per exemple.

A més, hem d'anul·lar el comportament per defecte de l'esdeveniment (recarregar la pàgina), ja que si no, no funcionarà.

```
const form = document.getElementById('formulario');

form.addEventListener('submit', e => {
   e.preventDefault();
   ...
});
```

7 preventDefault en submit

Podem obtindre fàcilment els valors d'un formulari, obtenint una referència al mateix i accedint als input a través del seu nom (atribut name).

Camp de text

```
<input type="text" id="nombre" name="nombre">
form.nombre.value
```

Checkbox

En aquest cas, sota el mateix nom tindrem una col·lecció d'inputs. Cada input té un atribut checked (booleà) que indica si està marcat, i el seu atribut value amb el corresponent valor. Podem filtrar els que estan marcats i obtindre un array amb els seus valors.

```
<input type="checkbox" id="deporte" name="aficiones" value="deporte">
<label for="deporte">Deporte</label>
<input type="checkbox" id="viajar" name="aficiones" value="viajar">
<label for="viajar">Viajar</label>
<input type="checkbox" id="comer" name="aficiones" value="comer">
<label for="comer">Comer</label>
const aficiones = Array.from(form.aficiones)
        .filter(input => input.checked)
        .map(input => input.value);
```

Radio

Encara que en aquest cas també tindrem una col·lecció d'inputs, com només es pot triar un, podem obtindre el valor seleccionat directament (value).

```
<input type="radio" id="rojo" name="color" value="rojo" checked>
<label for="rojo">Rojo</label>
<input type="radio" id="verde" name="color" value="verde">
<label for="verde">Verde</label>
<input type="radio" id="azul" name="color" value="azul">

<label for="azul">Azul</label>

form.color.value
```

File

En aquesta mena d'inputs, existeix un array anomenat files, on podrem accedir a la informació de l'arxiu (o arxius) seleccionats. No podrem accedir a la ruta, per exemple.

```
<input type="file" id="fichero" name="fichero">
if(formulario.fichero.files.length) { // Si hemos seleccionado un archivo
        const fichero = formulario.fichero.files[0];
        console.log(`Archivo: ${fichero.name}, tipo: ${fichero.type}, tamano: ${fichero.size}bytes`);
}
```

A més, podrem utilitzar classes com FileReader per a llegir el contingut del fitxer, i en aquest cas retornar-lo en format base64.

```
if(fichero.type.startsWith('image')) {
        let reader = new FileReader();
        reader.readAsDataURL(fichero); // Llegir en base64
        reader.addEventListener('load', e => {
                // Visualitza la imagen en un <img> en el HTML
                document.getElementById("imgPreview").src = reader.result;
        });
}
```

5. Enllaços d'interés

Events: https://www.w3schools.com/jsref/dom_obj_event.asp

6. Bibliografia

Bernal Mayordomo, Arturo. Curso Programación con JavaScript. Cefire 2020

W3Schools - JavaScript Tutorial