

# Esdeveniments. Manejament de formularis

Xavier Garcia Rodriguez



# Índex

<b>Introducció</b>	<b>5</b>
<b>Resultats d'aprenentatge</b>	<b>7</b>
<b>1 Programació d'events</b>	<b>9</b>
1.1 Què és un event? . . . . .	9
1.2 Gestió d'events . . . . .	13
1.2.1 Afegir funcions als events . . . . .	15
1.2.2 Events del ratolí ('MouseEvents') . . . . .	19
1.2.3 Events del teclat ('keyboardEvents') . . . . .	25
1.2.4 Altres events del DOM . . . . .	31
1.2.5 Events d'HTML5 i HTML5 media . . . . .	38
1.2.6 Events de les Web API estàndard . . . . .	41
<b>2 Programació amb formularis</b>	<b>45</b>
2.1 Què és un formulari? . . . . .	45
2.2 Estructura d'un formulari . . . . .	46
2.2.1 Elements d'un formulari . . . . .	48
2.2.2 Element 'form' . . . . .	50
2.2.3 Element 'input' . . . . .	52
2.2.4 Element 'textarea' . . . . .	59
2.2.5 Element 'button' . . . . .	63
2.2.6 Element 'label' . . . . .	64
2.2.7 Element 'datalist' . . . . .	65
2.3 Modificació d'aparença i comportament . . . . .	65
2.3.1 Modificació de l'aparença d'un formulari . . . . .	65
2.3.2 Modificació del comportament d'un formulari . . . . .	67
2.4 Validació . . . . .	69
2.4.1 Validació d'HTML5 . . . . .	70
2.4.2 Validació per a JavaScript . . . . .	73
2.5 Expressions regulars . . . . .	76
2.5.1 Grups de captura . . . . .	79
2.5.2 Altres mètodes d'utilització de les expressions regulars ('String' i 'RegExp') . . . . .	81
2.6 Utilització de galetes i Web Storage . . . . .	84
2.6.1 Llei de cookies . . . . .	85
2.6.2 Utilització de les galetes . . . . .	86
2.6.3 Utilització de Web Storage . . . . .	91



## Introducció

Una de les principals necessitats que cobreixen les aplicacions en l'entorn client és la de gestionar l'entrada de dades a través de les interfícies web i, consegüentment, per poder portar a terme aquestes tasques, és imprescindible ser capaços de gestionar els esdeveniments que es produeixen a l'aplicació, així com de gestionar l'entrada de dades.

En aquesta unitat veureu com mitjançant la detecció d'*events* es pot controlar quan l'usuari ha modificat un camp d'un formulari, ha clicat sobre un botó o ha finalitzat un element d'una pàgina; mentre que l'entrada de dades a través de formularis permet a les aplicacions comunicar-se amb altres components interns o servidors remots.

En l'apartat "Programació d'events" aprendreu què és un *event* en JavaScript i quina diferència hi ha amb un *esdeveniment*. Seguidament, descriurem alguns dels *events* reconeguts pels diferents components de JavaScript:

- *Events* disparats per accions del ratolí
- *Events* disparats per accions del teclat
- Altres *events* disparats pel DOM
- *Events* afegits a HTML5

També es farà una presentació dels web API més utilitzats, juntament amb els *events* que hi estiguin relacionats.

L'apartat "Programació amb formularis" comença fent una introducció als formularis en l'entorn client. Seguidament, aprendreu com s'estructura un formulari i quins elements d'HTML en formen part. També aprendreu a modificar-ne l'aparença i el comportament.

A continuació, veureu com s'han de validar els formularis tant a través de les noves característiques afegides a HTML5 com fent servir les vostres pròpies implementacions amb JavaScript o expressions regulars.

Per acabar, mostrarem com es treballa amb galetes i com podeu crear les vostres pròpies funcions per guardar i recuperar la informació que hi ha emmagatzemada, així com a utilitzar una de les Web API introduïda a HTML5: Web Storage.

Per assimilar els continguts d'aquesta unitat es molt important provar tots els exemples en el vostre equip o a CodePen i modificar-los per comprovar com afecten aquests canvis al codi. A més a més es recomana consultar la web Mozilla Developer Network ([www.developer.mozilla.org](http://www.developer.mozilla.org)) per aprofundir en els diferents aspectes tractats.



## Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

**1.** Desenvolupa aplicacions web interactives integrant mecanismes de maneig d'esdeveniments.

- Reconeix les possibilitats del llenguatge de marques relatives a la captura dels esdeveniments produïts.
- Identifica les característiques del llenguatge de programació relatives a la gestió dels esdeveniments.
- Diferencia els tipus d'esdeveniments que es poden manejar.
- Crea un codi que capturi i utilitzi esdeveniments.
- Reconeix les capacitats del llenguatge relatives a la gestió de formularis web.
- Valida formularis web utilitzant esdeveniments.
- Utilitza expressions regulars per facilitar els procediments de validació.
- Prova i documenta el codi.

**2.** Desenvolupa aplicacions web analitzant i aplicant les característiques del model d'objectes del document.

- Reconeix el model d'objectes del document d'una pàgina web.
- Identifica els objectes del model, les seves propietats i els seus mètodes.
- Identifica les diferències que presenta el model en diferents navegadors.
- Crea i verifica un codi que accedeixi a l'estructura del document.
- Crea nous elements de l'estructura i en modifica elements ja existents.
- Associa accions als esdeveniments del model.
- Programa aplicacions web de manera que funcionin en navegadors amb diferents implementacions del model.
- Independitza les tres facetes (contingut, aspecte i comportament), en aplicacions web.





## 1. Programació d'events

Dominar la programació d'*events* és una destresa imprescindible per qualsevol desenvolupador web, ja que tots els elements d'HTML disposen de diversos *events* que poden ser detectats per la vostra aplicació i executar diferents funcions.

Gràcies a aquest sistema és possible crear aplicacions interactives que responguin a les accions de l'usuari, o l'estat de l'aplicació. Per exemple, és possible detectar quan s'ha fet clic sobre un botó o un enllaç, quan ha finalitzat la descàrrega d'una imatge, o quan s'ha mogut el cursor del ratolí sobre una àrea concreta de la pàgina.

Altres tipus d'*events* menys habituals, però que és interessant conèixer, són els relacionats amb les Web API, ja que en molts casos són imprescindibles per poder implementar correctament les aplicacions que les utilitzen. Per exemple, la Web API encarregada de gestionar les peticions AJAX (peticions asíncrones enviades a un servidor) depèn dels esdeveniments per determinar quin codi executar una vegada es rep la resposta del servidor.

### 'Events' o esdeveniments?

Encara que la traducció al català seria 'esdeveniment', ens hi referim amb el seu nom en anglès perquè no hi hagi confusions.

### 1.1 Què és un event?

Per entendre en què consisteix el sistema d'*events* caldria fer una petita introducció als **patrons de disseny** i, més específicament, al patró **observador** (en anglès, *observer*).

Els patrons de disseny són solucions reutilitzables a problemes que ocorren sovint. No es tracta d'algorismes ni implementacions concretes, més aviat és una explicació clara de com resoldre un problema concret. Per exemple, en el llibre original sobre patrons de disseny podem trobar una descripció del problema, una solució en forma de diagrama UML i una implementació en C++ o Smalltalk.

No es tracta, com en un receptari, d'agafar el codi que posen i enganxar-lo al nostre programa, sinó d'entendre quin és el problema, com aplicar una solució provada que funciona i quins són els pros i contres d'aplicar-la-hi.

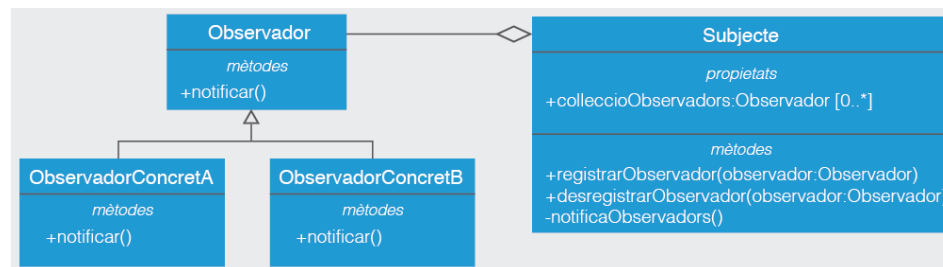
Un altre avantatge que té conèixer els patrons de disseny és que és molt útil per comunicar-se; per exemple: en lloc de descriure el problema i la solució que hi has aplicat, que generalment seria una explicació molt llarga, pots dir simplement: "He aplicat el patró de disseny X"; i si l'altra persona també els coneix, tindrà clar a què et refereixes sense haver d'afegir més detalls.

El cas que ens ocupa, el sistema d'*events*, és una implementació del patró observador. En la figura 1.1, podeu veure amb més detall una versió simplificada de com funciona aquest patró.

A les referències bibliogràfiques, trobareu uns llibres que tracten a fons els patrons de disseny.

### Patró Publicador-Subscriber

El patró *Publicador-Subscriber* és una variant del patró observador i en molts casos es poden fer servir de forma intercanviable.

**FIGURA 1.1.** Diagrama del patró observador

Aquest patró s'utilitza quan necessitem un sistema capaç d'indicar quan s'ha produït un esdeveniment i de comunicar-lo només als interessats, que s'han de poder afegir i eliminar dinàmicament.

Tot i que el diagrama no indica que es passi cap paràmetre a l'operació `notificar()`, normalment es passarà la informació del subjecte als observadors fent servir aquest paràmetre.

#### Exemple d'aplicació del patró observador

Si volem seguir l'IOC (@ioc) a Twitter, el que faríem és registrar-nos com a observadors al compte de l'IOC, que seria el subjecte. Llavors, quan l'@ioc publiqués un nou tuit (*tweet*), recorreria la seva llista d'observadors registrats i faria arribar aquesta informació a través de l'operació `notificar()` de cadascun dels observadors, passant el tuit com a paràmetre.

Dins d'aquesta operació, cada observador farà una cosa o altra amb aquesta piulada segons la seva implementació. Per exemple, en el cas dels usuaris, rebrien una notificació indicant que l'@ioc ha publicat una piulada; un administrador podria afegir-lo a una llista històrica, o en el cas d'un programari especialitzat d'automatització, podria retuitejar-lo (tornar-lo a publicar).

Si més endavant volem deixar de rebre notificacions, el que faríem és desregistrar-nos com a observador del subjecte, de manera que ja no rebriem més notificacions.

Aquesta seria una implementació d'aquest patró en JavaScript:

```

1 let subjecte = {
2   observadorsRegistrats: [],
3
4   registrarObservador: function(observador) {
5     this.observadorsRegistrats.push(observador);
6   },
7
8   desregistrarObservador: function(observador) {
9     let index = this.observadorsRegistrats.indexOf(observador);
10    this.observadorsRegistrats.splice(index, 1);
11  },
12
13  notificaObservadors: function(missatge) {
14    console.log("Iniciem la notificació del missatge als observadors");
15    for (let i = 0; i < this.observadorsRegistrats.length; i++) {
16      this.observadorsRegistrats[i].notificar(missatge);
17    }
18    console.log("Fi de les notificacions");
19  },
20
21  saludar: function() {
22    this.notificaObservadors('Hola mon');
23  }
24 }
25
26 let observador1 = {

```

```
27   notificar: function(missatge) {
28       console.log("Soc l'observador 1 i he rebut el missatge: " + missatge);
29   }
30 }
31
32 let observador2 = {
33     notificar: function(missatge) {
34         console.log("Soc l'observador 2 i també mostro el missatge : " + missatge);
35     }
36 }
37
38 // Enregistrem els dos observadors del subjecte
39 subjecte.registrarObservador(observador1);
40 subjecte.registrarObservador(observador2);
41
42 // Comprovem que funciona enviant un missatge
43 subjecte.saludar();
44
45 // Desenregistrem a un observador i tornem a saludar
46 subjecte.desregistrarObservador(observador1);
47 subjecte.saludar();
48
49 // Desenregistrem a un observador i tornem a saludar; com que ja no hi ha més
    observadors, no farà res
50 subjecte.desregistrarObservador(observador2);
51 subjecte.saludar();
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/vNrpXx?editors=1011](https://codepen.io/ioc-daw-m06/pen/vNrpXx?editors=1011). Tingueu en compte que si voleu provar l'exemple amb Internet Explorer, haureu de gravar la pàgina web al disc per després obrir-lo amb aquest navegador.

Una cosa semblant és el que fa el sistema d'*events*, però en aquest cas els Subjectes són els elements de la pàgina, per exemple els botons, les imatges, els paràgrafs... I els observadors, en lloc de ser objectes complets, només són funcions que compleixen el mateix paper que l'operació notificar; en lloc de cridar aquesta operació, el que fa el subjecte és invocar aquestes funcions.

En JavaScript i en altres llenguatges, aquestes funcions que passen a altres objectes es diuen *callbacks*. Sempre que vegeu aquest terme heu de tenir en compte que el que s'espera és una funció que serà executada en un altre punt del programa.

En el cas dels navegadors, un *event* és el que es produeix quan el navegador detecta determinats esdeveniments; per exemple, quan es clica a un enllaç, s'activa un botó o es carrega una pàgina.

Actualment hi ha una gran quantitat d'*events* estàndards associats als elements HTML; els hem agrupat de la manera següent:

- **DOM:** *events* activats per accions sobre el DOM (Document Object Model), per exemple, si s'ha enviat un formulari, si s'ha canviat de camp o si s'ha produït cap canvi en un text que havíem introduït. Podem classificar-los en:
  - **MouseEvent:** *events* que es produeixen en detectar-se accions relacionades amb el ratolí.

#### Funcions: objectes de primera classe

En JavaScript les funcions són objectes de primera classe i, com a tals, es poden tractar igual que altres objectes i guardar-los en variables i *arrays*.

#### Llista d'events

Podeu trobar una llista exhaustiva dels *events* disponibles a [www.goo.gl/sSVjnG](http://www.goo.gl/sSVjnG).

- **KeyboardEvent:** *events* produïts en detectar-se accions de teclat.
- **UIEvent:** *events* relacionats amb els canvis a la interfície, com és el fet de carregar una pàgina o canviar la mida de la finestra.
- **HTML5:** *events* específics d'HTML5 que no són compatibles amb versions anteriors; ens permeten treballar amb *events* addicionals, que fan que puguem detectar quan es comença a arrossegar un element.
- **HTML5 media:** aquests *events* els produeixen els elements *media*, com l'àudio i el vídeo, i permeten conèixer el seu estat (si està a l'espera, si està cercant, si ha finalitzat...).
- **Web API:** a banda dels *events* proporcionats que formen part de l'especificació d'HTML5, s'han creat moltes API que són admeses pels navegadors i afegeixen funcionalitats molt importants. Aquestes API també fan servir el sistema d'*events* per comunicar-nos quan es produeixen canvis a l'emmagatzematge local, si s'ha rebut un missatge d'un *web worker* indicant que ha finalitzat la tasca, o si s'han detectat canvis a la càrrega de la bateria.

#### Web worker

Un *web worker* és un tipus de procés que permet realitzar tasques en paral·lel a l'execució principal.

#### Les Web API

Encara que existeixen moltes Web API, s'ha de tenir en compte que no totes són estàndard; per exemple vosaltres mateixos en podríeu crear una. Les estàndard són les que es troben a les especificacions web oficials. A l'enllaç següent, podeu veure quins són aquests *events* estàndard i, en cada cas, a quina especificació de Web API pertanyen: [www.goo.gl/wXGpNN](http://www.goo.gl/wXGpNN).

Encara que aquestes Web API es troben a la llista d'especificacions del W3C, es desenvolupen de manera paral·lela a l'estàndard HTML5.

Alguns exemples de Web API són:

- **Web Workers:** tot i que els navegadors no permeten la programació multifil amb suport per múltiples nuclis, amb aquesta API es poden programar tasques que s'executen de manera concurrent.
- **Web Storage:** ens permet emmagatzemar dades en el nostre navegador de manera que no s'han de descarregar cada vegada; és semblant al que es fa amb les galetes però amb un límit molt més gran (el volum depèn del navegador).
- **XMLHttpRequest:** aquesta és l'API que ens permet fer servir crides AJAX des del navegador.
- **WebSocket:** fent servir aquesta API és possible mantenir una connexió bidireccional amb servidors, cosa que permet crear aplicacions tipus xat o jocs “multijugador” al navegador.

#### AJAX

AJAX és un conjunt de tecnologies que permeten actualitzar els continguts d'una pàgina web sense haver de carregar una altra pàgina.

Podeu veure amb més detall el funcionament d'AJAX a la secció “API XMLHttpRequest” d'aquest mateix apartat.

## 1.2 Gestió d'events

Un dels problemes de JavaScript és que es poden fer les mateixes coses de moltes maneres i, com passa habitualment en aquests casos, algunes són millors que d'altres. En aquesta secció veurem les diverses maneres de treballar amb *events* amb JavaScript.

Començarem veient com podem registrar-nos com a observadors d'un *event*, en aquest cas el paper de l'operació `notificar()` el realitza la funció `alert`; el subjecte és el botó que ens notificarà el moment que es dispari l'*event* `click`:

```
1 <button onclick="alert('clic a l'element');">
2   Fes clic aquí!
3 </button>
```

Fixeu-vos que s'ha afegit a l'element `button` un atribut, `onclick` amb el valor `alert('clic a l'element')`. Aquest valor és un fragment de codi JavaScript que serà executat en detectar-se l'*event* `click` sobre el botó. Aquesta és la manera més simple d'afegir la detecció d'*events* a un element, tot i que no és la més adient.

Tot i que és possible afegir codi per reaccionar a *events* directament incrustats al codi HTML, **es considera una mala pràctica**. Sempre hem d'intentar separar el codi CSS, HTML i JavaScript en blocs diferents i no fer-los *inline*. A més, és preferible carregar-los com a fitxers externs.

Veieu a continuació un altre exemple més complexe on s'ha substituït el codi JavaScript per una crida a una funció pròpia:

```
1 <script>
2   let i = 0;
3   function mostraMissatge() {
4     ++i;
5     alert('Has fet clic a l'element ' + i + ' vegades');
6   }
7 </script>
8 <button onclick="mostraMissatge();">
9   Fes clic aquí!
10 </button>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/YyeJLP?editors=1010](https://codepen.io/ioc-daw-m06/pen/YyeJLP?editors=1010).

Si analitzem aquest exemple pas a pas, el funcionament seria el següent:

1. Quan el navegador processa el codi HTML i troba l'atribut `onclick` a l'element `button`.
2. El navegador enregistra aquesta funció (`mostraMissatge()`) per ser executada quan es dispari l'*event* `click`. És a dir, la funció serà invocada quan es detecti un clic sobre aquest element.

3. Quan el navegador detecta que s'ha disparat un *event* de tipus `click` sobre el botó:
  - (a) Recorre la llista d'observadors registrats per l'*event* `click` d'aquest subjecte.
  - (b) Els “notifica” invocant a la funció corresponent passada com argument (el *callback*), en aquest cas `mostrarMissatge()`, amb la informació de l'*event* com a paràmetre.

Aquesta és una versió lleugerament modificada per comprovar que, efectivament, la funció `mostrarMissatge()` rep la informació que li passa el navegador:

```

1 <script>
2   function mostraMissatge(e) {
3     console.log(e);
4     alert('Detectat un event de tipus: ' + e.type);
5   }
6 </script>
7 <button onclick="mostraMissatge(event);">
8   Fes clic aquí!
9 </button>

```

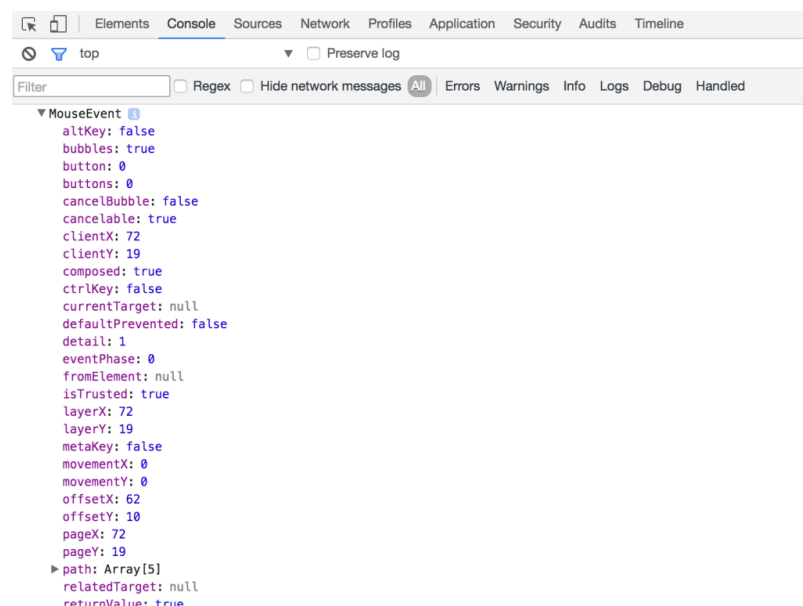
Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/qOxJZL?editors=1011](https://codepen.io/ioc-daw-m06/pen/qOxJZL?editors=1011).

### Eines per a desenvolupadors

La informació que li passa el navegador es mostra a través de la consola. S'hi accedeix amb les eines per a desenvolupadors que faciliten els navegadors. Cada navegador té les seves pròpies eines, però algunes, com la consola -que és fonamental-, les inclouen tots els navegadors.

En aquest cas hem fet servir `console.log(e)` per mostrar, per la consola del navegador (a la qual es pot accedir des de les eines per a desenvolupadors), el contingut total de l'*event* que hem rebut; com es pot veure a la figura 1.2.

**FIGURA 1.2.** Exemple d'informació de l'*event* 'click'



La funció `document.getElementById()` no és pròpia de JavaScript, sinó que és afegida per la interfície DOM que implementen els navegadors.

### 1.2.1 Afegir funcions als events

Una manera correcta d'afegir aquestes funcions sense haver de contaminar el codi HTML és la següent:

1. El primer pas és obtenir una referència als elements. En aquest cas, per simplificar, farem servir la propietat `id`. Aquesta propietat no pot repetir-se en cap altre element de la pàgina.
2. Podem tractar aquests elements com a objectes i, per tant, podem definir-hi un mètode i assignar-los una funció (anònima en aquest cas). Per exemple, creem un mètode `onclick` tant al primer com al segon element.
3. Quan es dispari un *event* de tipus `click` sobre els subjectes, aquests ho notificaran als seus observadors i s'invocarà la funció associada amb la informació de l'*event*.

```
1 <script>
2 // Guardem els elements en variables
3 let primerElement = document.getElementById('primer'),
4     segonElement = document.getElementById('segon');
5
6 // Assignem a cada element una funció que serà cridada quan s'hi faci clic
7 primerElement.onclick = function(e) {
8     alert('Clic al primer element de tipus: ' + e.type);
9 }
10
11 segonElement.onclick = function(e) {
12     alert('Clic al segon element de tipus: ' + e.type);
13 }
14
15 </script>
16 <button id="primer">
17     Fes clic al primer element!
18 </button>
19 <button>
20     Aquí no fa res, no té cap id per referir-nos
21 </button>
22 <button id="segon">
23     Fes clic al segon element!
24 </button>
25 <button id="tercer">
26     Aquí sí que tenim id, però no hi hem afegit cap funció
27 </button>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/ojEQbb?editors=1010](https://codepen.io/ioc-daw-m06/pen/ojEQbb?editors=1010).

Fixeu-vos que hem afegit més elements dels necessaris a l'HTML. Això ho hem fet per comprovar que, efectivament, l'element que no té associada cap "id" i l'element que sí que la té (però que no l'hem lligat a cap funció) no responen als clics que fem sobre ells.

Vegeu el mateix exemple, però usant la notació fletxa de les funcions. Aquesta notació és molt útil sobretot quan el cos de les funcions és curt. Al material s'utilitzaran les dues notacions de manera indiferent:

```
1 <script>
2   // Guardem els elements en variables
3   let primerElement = document.getElementById('primer'),
4     segonElement = document.getElementById('segon');
5
6   // Assignem a cada element una funció que serà cridada quan s'hi faci clic
7   primerElement.onclick = e => alert('Clic al primer element de tipus: ' + e.type);
8
9   segonElement.onclick = e => alert('Clic al segon element de tipus: ' + e.type);
10
11 </script>
12 <button id="primer">
13   Fes clic al primer element!
14 </button>
15 <button>
16   Aquí no fa res, no té cap id per referir-nos
17 </button>
18 <button id="segon">
19   Fes clic al segon element!
20 </button>
21 <button id="tercer">
22   Aquí sí que tenim id, però no hi hem afegit cap funció
23 </button>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/jOBjByG?editors=1010](https://codepen.io/ioc-daw-m06/pen/jOBjByG?editors=1010).

La manera d'assignar events vista als exemples anteriors, encara que sigui correcta, presenta un inconvenient: només hi pot haver un observador per a cada tipus d'*event*. Si intentem afegir-n'hi un altre, l'anterior queda eliminat; com podeu provar amb aquest exemple:

```
1 <script>
2   // Guardem l'element en una variable
3   let primerElement = document.getElementById('primer');
4
5   // Assignem a l'element una funció que serà cridada quan s'hi faci clic
6   primerElement.onclick = () => alert('Primera funció');
7
8   // Assignem una altra funció l'element que també serà cridada quan s'hi faci clic
9   primerElement.onclick = e => alert('Segona funció');
10 </script>
11 <button id="primer">
12   Fes clic aquí!
13 </button>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/QjQJpV?editors=1010](https://codepen.io/ioc-daw-m06/pen/QjQJpV?editors=1010).

Com podeu veure, no funciona correctament: la segona funció sobreescriu la primera, així que només es notifica la segona.



Si en lloc d'usar una funció anònima per a l'assignació, voleu assignar-li una funció creada en alguna altra part del codi, cal posar el nom d'aquesta funció sense els parèntesis, ja que si no, en comptes d'assignar la funció a l'esdeveniment, el que faria seria executar la funció i assignar el resultat d'aquesta.

És a dir:

```
1 <script>
2   // Guardem l'element en una variable
3   let primerElement = document.getElementById('primer');
4
5   // Assignem a l'element una funció que serà cridada quan s'hi faci clic
6   primerElement.onclick = notificacio;
7
8   function notificacio() {
9     alert('Primera funció');
10  }
11
12 </script>
13 <button id="primer">
14   Fes clic aquí!
15 </button>
```

### Afegir múltiples observadors per a un mateix subjecte

És molt possible que necessitem afegir dos o més observadors a un subjecte pel mateix *event* o que ho hàgim de fer més endavant, així que la **forma més recomanable** de fer-ho és **amb el mètode** `addEventListener()`, com us mostrem en l'exemple següent:

```
1 <script>
2   // Guardem l'element en una variable
3   let primerElement = document.getElementById('primer');
4
5   // Creem les funcions que volem que siguin cridades
6   function mostrarMissatge1() {
7     alert("primera funció");
8   }
9
10  function mostrarMissatge2() {
11    alert("segona funció");
12  }
13
14  // Subscriuim l'element perquè respongui amb les dues funcions
15  primerElement.addEventListener('click', mostrarMissatge1);
16  primerElement.addEventListener('click', mostrarMissatge2);
17 </script>
18 <button id="primer">
19   Fes clic aquí!
20 </button>
```

#### **`addEventListener()` a IE 8 i versions anteriors**

La funció `addEventListener()` no s'ha afegit al navegador de Microsoft fins a la versió 9. Per a versions anteriors cal fer servir `attachEvent()`, que té la mateixa funcionalitat, però Internet Explorer 11 ja no l'admet.

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/qOxQXZ?editors=1010](https://codepen.io/ioc-daw-m06/pen/qOxQXZ?editors=1010).

La signatura del mètode `addEventListener()` és la següent:

```
1 target.addEventListener(type, listener[, useCapture]);
```

- `target` és l'element al qual s'ha de disparar l'*event* perquè aquest sigui detectat pel **subjecte**. És a dir, si volem detectar quan s'ha disparat l'*event* `click` sobre un botó, el `target` seria el botó. Fixeu-vos que amb aquest sistema es pot observar qualsevol element del DOM i molts dels elements de les Web API, no només dels elements HTML.
- `type` és una cadena amb el tipus d'*event* que volem escoltar, per exemple: `click`, `dblclick`, `load`.
- `listener` aquesta és la funció *callback* que serà cridada quan es dispari l'*event* observat. Fixeu-vos que aquesta funció és la que és **notificada** quan es produeix l'*event* al subjecte. Correspon a la funció `notificar()` en la figura 1.2.
- `useCapture` és un paràmetre opcional que no farem servir en aquests materials, per tant, el considerem sempre fals. En cas que fos cert, canviaria la forma amb què es despatxen els *events*.

En podeu trobar més informació al següent enllaç: [www.goo.gl/a8LqGS](http://www.goo.gl/a8LqGS).

Ara ja hem vist com podeu afegir la detecció d'*events* a un element i com podeu invocar la funció que us interessi. Segurament us deu haver adonat que descriure els processos és una mica enrevessat. Per aquest motiu, a partir d'ara ens referirem als elements i a les diferents accions de la següent manera:

- **Event**. Pot fer referència tant a un tipus com a a un objecte:
  - Un tipus, com pot ser `click`, `dblclick` o `load`.
  - Un objecte que conté tota la informació de l'esdeveniment que s'ha disparat.
- **Disparar un event** (en anglès *trigger* o *fire*). Significa que s'ha produït un esdeveniment. Per exemple, quan el navegador finalitza la càrrega del codi HTML es produeix l'esdeveniment `load`. En aquest cas diríem que s'ha disparat l'*event* `load`, i aquesta informació seria notificada als observadors registrats.
- **Despatxar un event** (en anglès *dispatch*). Continuem amb el mateix exemple: quan el navegador finalitza la càrrega de la pàgina, despatxa l'*event* `load`. És a dir, quan quelcom és responsable de comunicar que s'ha produït algun esdeveniment, el que fa és despatxar l'*event*. Quan es despatxa un *event* es passa un objecte com a paràmetre que conté tota la informació corresponent. Per exemple, en el cas dels *events* del ratolí, els botons clicats o la posició del cursor es passen a la pantalla com a coordenades cartesianes. Quan es despatxa un *event* el que es fa és notificar tots els observadors registrats per aquest tipus d'*event*.

Per eliminar la detecció d'*events* d'un element cal invocar el mètode `removeEventListener()` i passar com a arguments el tipus d'*event* i la funció que es vol eliminar.

### 1.2.2 Events del ratolí ('MouseEvents')

Aquests són els *events* que es disparen quan canvia l'estat del ratolí (vegeu la taula 1.1), per exemple, quan es prem un botó del dispositiu o quan es deixa anar.

**TAULA 1.1.** Events relacionats amb els botons del ratolí

Tipus	Operació	Descripció
click	onclick	Es dispara en fer clic amb qualsevol botó del ratolí
dblclick	ondblclick	Es dispara en fer doble clic amb qualsevol botó del ratolí
mousedown	onmousedown	Es dispara en prémer qualsevol botó del ratolí
mouseup	onmouseup	Es dispara en deixar anar qualsevol botó del ratolí

Per comprovar com funciona podríeu escriure un codi així:

```

1 <button onclick="console.log('click');">test clic</button>
2 <button ondblclick="console.log('dblclick');">test doble clic</button>
3 <button onmousedown="console.log('mousedown');">test prémer botó del ratolí</
  button>
4 <button onmouseup="console.log('mouseup');">test deixar anar botó del ratolí</
  button>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/BoVxGO?editors=1000](https://codepen.io/ioc-daw-m06/pen/BoVxGO?editors=1000).

Fixeu-vos que cada botó mostra el missatge a la consola sota unes condicions diferents.

Un altre punt interessant és saber en quin ordre es disparen aquests *events*; què passa si hem registrat un observador per a cada *event* d'un sol botó?

En aquest cas, recordeu que haurem de fer servir forçosament `addEventListener()`, així que els afegirem mitjançant JavaScript.

```

1 <script>
2   let button = document.getElementById('test');
3
4   button.addEventListener('click', notificaObservadorClick);
5   button.addEventListener('dblclick', notificaObservadorDbClick);
6   button.addEventListener('mousedown', notificaObservadorMouseDown);
7   button.addEventListener('mouseup', notificaObservadorMouseUp);
8
9   function notificaObservadorClick() {
10     console.log("S'ha disparat l'event Click");
11   }
12
13   function notificaObservadorDbClick() {
14     console.log("S'ha disparat l'event DbClick");

```

Per detectar el botó del ratolí que s'ha pulsat es pot fer servir la propietat `button` de l'objecte `event`, de manera que:

- Botó esquerre:  
`event.button = 0`
- Botó central:  
`event.button = 1`
- Botó dret:  
`event.button = 2`

Aquesta propietat només funciona amb els events `mouseup` i `mousedown`.

```
15 }
16
17 function notificaObservadorMouseDown() {
18     console.log("S'ha disparat l'event Mouse Down");
19 }
20
21 function notificaObservadorMouseUp() {
22     console.log("S'ha disparat l'event Mouse Up");
23 }
24 </script>
25
26 <button id="test">test de múltiples events</button>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/bVKMJz?editors=1011](https://codepen.io/ioc-daw-m06/pen/bVKMJz?editors=1011).

Fixeu-vos en quin ordre es disparen els *events*:

1. El primer de ser disparat és mousedown.
2. A continuació es dispara mouseup, quan es deixa anar el botó.
3. I finalment es dispara click. És a dir, fins que no es deixi anar el botó no es confirmarà que s'ha produït un clic sobre l'element.

Si el que fem és un doble clic, fixeu-vos com canvia la seqüència:

1. mousedown
2. mouseup
3. click
4. mousedown
5. mouseup
6. click
7. dblclick

Això s'ha de tenir molt en compte, perquè si tenim un element al qual hem lligat la detecció d'*events* com click o mousedown al mateix temps que dblclick, també es dispararan quan es faci un doble clic, i segurament no és la nostra intenció.

#### Objectes com a paràmetres

En JavaScript és molt freqüent fer servir objectes que només tenen propietats per passar-les a mètodes o funcions. En altres llenguatges, això s'anomenaria *diccionari de dades*, *array associatiu* o simplement *array*. Un exemple d'aquests tipus d'objectes seria {x: 10, y:20}.

Vegem ara una variant del codi anterior. Aquesta vegada, a les funcions de notificació els arriba un paràmetre; en lloc de fer servir una funció per a cada *event*, farem servir la mateixa funció `notificarObservador()` per a totes.

El subjecte sempre ens enviarà la informació de l'*event* com un objecte. És important entendre que aquest paràmetre **l'envia el subjecte que observem**, i que no importa el nom que li posem nosaltres, perquè aquest nom només ens serveix de referència a la nostra funció. **Si no el fem servir, no cal declarar-lo com a paràmetre**, simplement serà ignorat.

```

1 <script>
2   let button = document.getElementById('test');
3
4   button.addEventListener('click', notificaObservador);
5   button.addEventListener('dblclick', notificaObservador);
6   button.addEventListener('mouseup', notificaObservador);
7   button.addEventListener('mousedown', notificaObservador);
8
9   function notificaObservador(e) {
10     console.log("S'ha disparat un event de tipus: ", e.type, "amb aquesta
        informació: ", e);
11   }
12 </script>
13 <html>
14 <button id="test">test de múltiples events</button>
15
16 </html>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/WQyJqW?editors=1011](https://codepen.io/ioc-daw-m06/pen/WQyJqW?editors=1011).

En aquest exemple podeu veure que obtenim el tipus d'*event* de la propietat `type` de l'*event* que ens arriba per paràmetre, i a continuació es mostra tota la informació que conté en aquest objecte (com poden ser les coordenades, en quin element s'ha produït l'*event*, si s'està prement cap tecla modificadora -majúscules o control, per exemple-...).

## Variacions segons el navegador

Segons en quin navegador s'executi el codi **alguns comportaments poden ser diferents**. Per exemple, en el cas del navegador Chrome, si es fa clic amb el botó dret del ratolí, només es dispara l'*event* `mousedown`, en canvi, en Firefox es dispara també l'*event* `mouseup`.

Aquesta discrepància es produeix perquè en fer clic amb el botó dret tots dos disparen l'*event* `contextmenu` (vegeu la taula 1.2), com s'indica a l'especificació d'HTML5, però disparar o no l'*event* `mouseup` o `keyup` -segons el cas- depèn del navegador i de la plataforma que faci servir l'usuari. Inclús un mateix navegador, en diferents sistemes operatius, pot tenir comportaments diferents perquè l'especificació només obliga que es dispari l'*event* `contextmenu`.

### Especificació HTML5

Podeu trobar el document actualitzat amb l'especificació d'HTML5 en què es basa la implementació dels navegadors web a: [bit.ly/3tMEtTr](https://bit.ly/3tMEtTr)

**TAULA 1.2.** Events relacionats amb el menú contextual

Tipus	Operació	Descripció
contextmenu	oncontextmenu	Es dispara quan es mostra el menú contextual del navegador

Si proveu aquest exemple en Chrome, veureu que efectivament la resta d'*events* es disparen, ja que la seqüència s'interromp quan es llença el menú contextual. En canvi, si ho proveu amb Firefox, l'*event* `mouseup` es dispara a continuació de l'*event* `contextmenu`:

```

1 <script>
2   let button = document.getElementById('test');

```

```
3
4   button.addEventListener('click', notificaObservador);
5
6   button.addEventListener('contextmenu', notificaObservador);
7
8   button.addEventListener('mouseup', notificaObservador);
9
10  button.addEventListener('mousedown', notificaObservador);
11
12  function notificaObservador(e) {
13    console.log("S'ha disparat un event de tipus: ", e.type, "amb aquesta
14      informació: ", e);
15  }
16  </script>
17  <button id="test">test de múltiples events</button>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/NGzzPV?editors=1011](https://codepen.io/ioc-daw-m06/pen/NGzzPV?editors=1011).

Aquí, la seqüència per a Chrome:

1. mousedown
2. contextmenu

Mentre que per a Firefox serà aquesta:

1. mousedown
2. contextmenu
3. mouseup

### jQuery

jQuery és una biblioteca molt popular que inclou moltes funcionalitats que s'usen habitualment com a crides AJAX, manipulació del DOM (Document Object Model), enregistrament a *events* i animacions d'elements. Les versions més recents han deixat de banda el suport per a navegadors antics.

S'ha de tenir en compte que l'especificació d'HTML5 no és exhaustiva. Per això hi ha casos com aquests en què el comportament serà diferent segons el navegador que s'utilitzi. En aquests casos hem de decidir entre:

- Evitar fer servir elements que puguin presentar conflictes entre navegadors i plataformes.
- Fer una implementació que cobreixi tots els casos per als navegadors i les plataformes en què vulguem que el nostre codi funcioni correctament.
- Fer servir una llibreria que s'encarregui de normalitzar aquests comportaments, per exemple *jQuery*.

### Moviments del ratolí

A banda dels *events* que es disparen en prémer els botons del ratolí, també podem detectar els que es produeixen en moure'l. Aquesta és la llista de tipus d'*events* observables (vegeu la taula 1.3).

**TAULA 1.3.** Altres events relacionats amb els botons del ratolí

Tipus	Operació	Descripció
mouseenter	onmouseenter	Es dispara quan el cursor entra dins de l'àrea ocupada pel subjecte
mouseleave	onmouseleave	Es dispara quan el cursor abandona l'àrea ocupada pel subjecte
mousemove	onmousemove	Es dispara cada vegada que el cursor es mou per sobre del subjecte
mouseout	onmouseout	Es dispara quan el cursor abandona l'àrea ocupada pel subjecte o per algun dels seus fills
mouseover	onmouseover	Es dispara quan el cursor entra dins de l'àrea ocupada pel subjecte o per algun dels seus fills

Com podeu veure, la diferència principal entre `onmouseenter` i `onmouseover` és que el primer no diferencia els fills del subjecte i el segon sí que ho fa, i passa el mateix amb `onmouseout` i `onmouseleave`.

Fixeu-vos en aquest exemple; el quadre més petit és fill del quadre més gran, i hi podeu veure:

```

1 <style>
2   div {
3     width: 150px;
4     height: 150px;
5     background-color: red;
6     margin: 50px auto;
7   }
8
9   div>div {
10    position: relative;
11    top: -25px;
12    width: 50px;
13    height: 50px;
14    background-color: blue;
15  }
16 </style>
17 <script>
18   let element = document.getElementById('test');
19   element.addEventListener('mouseenter', notificaObservador);
20   element.addEventListener('mouseleave', notificaObservador);
21   element.addEventListener('mousemove', notificaObservador);
22   element.addEventListener('mouseout', notificaObservador);
23   element.addEventListener('mouseover', notificaObservador);
24   function notificaObservador(e) {
25     console.log("S'ha disparat un event de tipus: ", e.type);
26   }
27 </script>
28 <div id="test">
29   <div></div>
30 </div>

```

Podeu veure aquest exemple en l'enllaç següent: <https://codepen.io/ioc-daw-m06/pen/ZbRRbQ?editors=1111>

En entrar al quadre gran o al petit des de fora es disparen:

1. `mouseover`
2. `mouseenter`

3. `mousemove`, múltiples vegades, mentre el cursor estigui en moviment sobre el quadre.
4. `mouseout` en sortir del quadre gran
5. `mouseleave` en sortir del quadre gran

La diferència la trobem quan tenim el cursor a sobre d'un dels dos quadres i passem a l'altre; per exemple, amb el cursor sobre el quadre vermell passem al quadre blau. En aquest cas, veurem a la consola la següent seqüència (ignorant els `mousemove` és clar):

1. `mouseout`
2. `mouseover`

És a dir, es detecta la transició entre els dos elements, cosa que no passa amb `mouseenter` i `mouseleave`.

## Events per a pantalla tàctil

Els **TouchEvent** són similars als **MouseEvent** però pertanyen a una altra especificació diferent (vegeu la taula 1.4). Són els *events* disparats quan s'interactua tocant una pantalla tàctil, com podria ser la d'un mòbil o una tauleta.

### Combinar els EventTouch i MouseEvent

Com que aquest és un problema molt comú, generalment ja està resolt o es poden trobar llibreries que adaptin el funcionament que necessitem. Per exemple: els dispositius mòbils interpreten els tocs de pantalla com a clics automàticament, però si voleu afegir coses més específiques com el reconeixement de textos, haureu de fer servir una llibreria especialitzada.

**TAULA 1.4.** Events relacionats amb l'especificació TouchEvent

Tipus	Operació	Descripció
<code>touchstart</code>	<code>ontouchstart</code>	Es dispara quan es toca la pantalla encara que ja s'estigui tocant en un altre punt
<code>touchend</code>	<code>ontouchend</code>	Es dispara quan es deixa de tocar la pantalla en un punt, encara que hi hagi altres punts pressionats
<code>touchmove</code>	<code>ontouchmove</code>	Es dispara quan un dels dits es mou sobre la pantalla
<code>touchcancel</code>	<code>ontouchcancel</code>	Es dispara quan es produeix una interrupció d'una manera específica, com podria ser que hi haguessin més punts tocats dels que admet el dispositiu

L'exemple següent canviarà de color quan es disparin els *events* de toc a la pantalla; el `touchstart` farà que es posi verd, el `touchmove` de color groc i `touchend` de color gris. L'*event* `touchcancel` normalment no es dispara.

```

1 <style>
2   div {
3     position: absolute;
4     top: 0;

```



```

5     bottom: 0;
6     left: 0;
7     right: 0;
8     background-color: orange;
9   }
10 </style>
11 <script>
12   let element = document.getElementById('test');
13
14   element.addEventListener('touchcancel', notificaObservador);
15   element.addEventListener('touchstart', notificaObservador);
16   element.addEventListener('touchend', notificaObservador);
17   element.addEventListener('touchmove', notificaObservador);
18
19   function notificaObservador(e) {
20
21     switch (e.type) {
22       case 'touchstart':
23         element.style['background-color'] = 'green';
24         break;
25
26       case 'touchend':
27         element.style['background-color'] = 'grey';
28         break;
29
30       case 'touchmove':
31         element.style['background-color'] = 'yellow';
32         break;
33
34       case 'touchcancel':
35         element.style['background-color'] = 'red';
36         break;
37     }
38   }
39 </script>
40 <div id="test"></div>
41

```

Podeu provar-ne l'exemple en aquest URL: [codepen.io/ioc-daw-m06/pen/avKKJy](https://codepen.io/ioc-daw-m06/pen/avKKJy). Però en aquest cas ho haureu de fer des de un dispositiu amb pantalla tàctil, com un mòbil o una tauleta, ja que a l'ordinador aquests *events* no es dispren.

Podem fer servir una única funció per gestionar totes les notificacions i llavors, depenent del tipus d'*event* (`Event.type`), fer una cosa o altra amb un selector `switch-case`; com per exemple canviar el color de la pantalla.

### 1.2.3 Events del teclat ('keyboardEvents')

Els *events* de teclat són els que ens permeten reaccionar quan es dispara un *event* relacionat amb el teclat, com per exemple prémer una tecla o deixar-la anar.

**TAULA 1.5.** Events relacionats amb el teclat

Tipus	Operació	Descripció
keydown	onkeydown	Es dispara quan es prem una tecla
keypress	onkeypress	Es dispara quan es prem una tecla, i la tecla produeix un valor corresponent a un caràcter
. . . . .		

TAULA 1.5 (continuació)

Tipus	Operació	Descripció
keyup	onkeyup	Es dispara quan es deixa anar la tecla

D'entrada això pot semblar una mica estrany; per exemple, es pot observar un element de tipus `<div>` per l'*event* `keydown`, però en quines circumstàncies es rebria la notificació? Vegem-ho amb un primer exemple amb dos elements, un és un `<div>` i l'altre, un botó:

```

1 <script>
2   let element1 = document.getElementById('test1');
3   let element2 = document.getElementById('test2');
4
5   element1.addEventListener('keypress', notificaObservador);
6   element1.addEventListener('keydown', notificaObservador);
7   element1.addEventListener('keyup', notificaObservador);
8
9   element2.addEventListener('keypress', notificaObservador);
10  element2.addEventListener('keydown', notificaObservador);
11  element2.addEventListener('keyup', notificaObservador);
12
13  function notificaObservador(e) {
14    console.log("S'ha disparat un event de tipus: ", e.type);
15  }
16 </script>
17 <div id="test1">Aquest element no pot rebre el focus, així que no dispara
   events de teclat</div>
18 <button id="test2">Quan tinc el focus puc disparar els events de teclat</button
   >

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/vNraNq?editors=1011](https://codepen.io/ioc-daw-m06/pen/vNraNq?editors=1011).

Com podeu veure no és possible fer que l'element `<div>` dispari els *events* de teclat; en canvi, veiem que quan el botó té el *focus*, aquest sí que els dispara i mostra els missatges a la consola. Però si el *focus* és en un altre element de la pàgina, deixa de disparar-los.

### Focus

L'element que té el *focus* és el que està seleccionat en un moment determinat i només pot haver-hi un element amb *focus*. Es distingeix normalment perquè surt envoltat per una brillantor (botons, quadres de text...) o per una línia puntejada (enllaços). L'estil depèn del navegador.

Vegem un altre cas, aquesta vegada amb un element capaç de rebre el *focus* dins del `<div>`; per exemple, un altre botó:

```

1 <script>
2   let element1 = document.getElementById('test1');
3   let element2 = document.getElementById('test2');
4
5   element1.addEventListener('keypress', notificaObservador1);
6   element1.addEventListener('keydown', notificaObservador1);
7   element1.addEventListener('keyup', notificaObservador1);
8
9   element2.addEventListener('keypress', notificaObservador2);
10  element2.addEventListener('keydown', notificaObservador2);
11  element2.addEventListener('keyup', notificaObservador2);
12
13  function notificaObservador1(e) {
14    console.log("Des del <div> s'ha disparat un event de tipus: ", e.type);
15  }
16
17  function notificaObservador2(e) {
18    console.log("Des del <button> s'ha disparat un event de tipus: ", e.type);
19  }
20 </script>

```

```

21 <div id="test1">Aquest és el div observat que conté un botó <button>Aquest es
    un botó que no està essent observat</div>
22 <button id="test2">Aquest és el botó que està essent observat</button>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/bVKjew?editors=1011](https://codepen.io/ioc-daw-m06/pen/bVKjew?editors=1011).

En aquest cas, encara que el botó que hi ha a dins del `<div>` no estigui essent observat, sí que es disparen els *events* corresponents al `<div>`: quan aquest botó té el *focus* i es prem alguna tecla (feu un clic a sobre perquè rebí el *focus* i premeu qualsevol tecla). De tot això podem deduir quines són les regles perquè es puguin detectar o no si s'ha premut alguna tecla.

Per poder disparar *events* de teclat, un element ha de complir alguna de les següents condicions:

- Ha de ser capaç de rebre el *focus*.
- Ha de contenir un element fill capaç de rebre el *focus*.

Si s'acompleixen alguna d'aquestes condicions i l'element, o algun dels seus descendents, és el *focus* llavors dispararà els *events* de teclat.

## L'element 'body'

Un cas interessant és el de l'element `<body>`: encara que no tingui el *focus*, si s'afegeix un observador per als *events* de teclat, són capturats igualment.

En l'exemple següent podeu veure que no cal incloure cap altre element per fer que es detectin els *events* de teclat:

```

1 <script>
2   let element = document.getElementById('test');
3
4   element.addEventListener('keypress', notificaObservador);
5   element.addEventListener('keydown', notificaObservador);
6   element.addEventListener('keyup', notificaObservador);
7
8   function notificaObservador(e) {
9     console.log("Des del <body> s'ha disparat un event de tipus: ", e.type);
10  }
11 </script>
12 <body id="test">
13 </body>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/MaXBbr?editors=1011](https://codepen.io/ioc-daw-m06/pen/MaXBbr?editors=1011).

Aprofitem aquest exemple per analitzar el comportament d'aquests *events*:

- Si premem la tecla *a* (o qualsevol altre caràcter imprimible) i la deixem premuda, es dispararan alternativament els *events* `keydown` i `keypress` fins que deixem de prémer-la.

### Capturar events

Quan un element dispara un *event* i un altre és notificat, direm que l'*event* ha estat capturat. El fet de ser notificat és el mateix que capturar l'*event*.

- Si premem la tecla *control*, només es dispararà l'*event* *keydown*.
- Quan deixem de prémer qualsevol tecla, es dispararà l'*event* *keyup*.

Fixeu-vos que encara que *keydown* i *keypress* es repeteixin mentre la tecla estigui premuda, no passa el mateix amb *keyup*, que només es dispara una vegada per tecla.

Llavors, què passa si estem observant un element i al mateix temps afegim un altre observador per a algun dels seus fills? Es cridaran tots dos? Es cridarà només un d'ells? Vegem-ho amb un exemple ampliat:

```

1 <script>
2   let element1 = document.getElementById('test1');
3   let element2 = document.getElementById('test2');
4
5   element1.addEventListener('keypress', notificaObservador1);
6   element1.addEventListener('keydown', notificaObservador1);
7   element1.addEventListener('keyup', notificaObservador1);
8
9   element2.addEventListener('keypress', notificaObservador2);
10  element2.addEventListener('keydown', notificaObservador2);
11  element2.addEventListener('keyup', notificaObservador2);
12
13  function notificaObservador1(e) {
14    console.log("Des del <body> s'ha disparat un event de tipus: ", e.type);
15  }
16
17  function notificaObservador2(e) {
18    console.log("Des del <textarea> s'ha disparat un event de tipus: ", e.type);
19  }
20 </script>
21 <body id="test1">
22   <textarea id="test2"></textarea>
23 </body>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/ZbRjex?editors=1011](https://codepen.io/ioc-daw-m06/pen/ZbRjex?editors=1011).

Efectivament, es dispararà l'*event* a tots dos elements, tant al `<textarea>` com al `<body>`, i sempre en aquest ordre. Aquest efecte es produeix gràcies a l'*efervescència* (*event bubbling* en anglès): quan a un element es dispara un *event*, aquest es propaga (per exemple, com les bombolles d'un cava) cap als diferents observadors i cap als elements que els contenen. És a dir, quan es dispara l'*event* al `<textarea>` aquest és enviat també a l'element que el conté, l'element `<body>`.

Per solucionar aquest problema només cal invocar el mètode `Event.stopPropagation()` de l'*event* que arriba com a paràmetre.

En aquest exemple veiem que, en interrompre la propagació de l'*event* al `textarea`, no arriba mai a ser notificat l'element `<body>`:

```

1 <script>
2   let element1 = document.getElementById('test1');
3   let element2 = document.getElementById('test2');
4
5   element1.addEventListener('keypress', notificaObservador1);
6   element1.addEventListener('keydown', notificaObservador1);
7   element1.addEventListener('keyup', notificaObservador1);
```

### El flux d'events

Podeu trobar més informació sobre el flux d'*events* a [www.goo.gl/vFVGS3](http://www.goo.gl/vFVGS3) (DOM2) i [www.goo.gl/zJBRCf](http://www.goo.gl/zJBRCf) (Esborrany DOM3)

```

8
9  element2.addEventListener('keypress', notificaObservador2);
10 element2.addEventListener('keydown', notificaObservador2);
11 element2.addEventListener('keyup', notificaObservador2);
12
13 function notificaObservador1(e) {
14     console.log("Des del <body> s'ha disparat un event de tipus: ", e.type);
15 }
16
17 function notificaObservador2(e) {
18     console.log("Des del <textarea> s'ha disparat un event de tipus: ", e.type)
19     ;
20     e.stopPropagation();
21 }
22 </script>
23 <body id="test1">
24     <textarea id="test2"></textarea>
25 </body>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/WQyKOj?editors=1011](https://codepen.io/ioc-daw-m06/pen/WQyKOj?editors=1011).

## Detecció de pulsacions de tecles

A banda de detectar si s'ha pres o s'ha deixat anar una tecla, per saber quina és la tecla en qüestió hem de comprovar la propietat `EventKeyboard.key` a Google Chrome i Safari, o `EventKeyboard.keyIdentifier` a Internet Explorer i Firefox:

```

1 <script>
2   let element = document.getElementById('test');
3
4   element.addEventListener('keypress', notificaObservador);
5   element.addEventListener('keydown', notificaObservador);
6   element.addEventListener('keyup', notificaObservador);
7
8   function notificaObservador(e) {
9       console.log("Disparat ", e.type, " per la tecla amb codi: ", e.key || e.
10         keyIdentifier);
11   }
12 </script>
13 <body id="test">
14 </body>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/vNraWE?editors=1011](https://codepen.io/ioc-daw-m06/pen/vNraWE?editors=1011).

El codi retornat serà, per exemple, en prémer la tecla 'a' us mostrarà per la consola:

1. Disparat `keydown` per la tecla amb codi: `a`
2. Disparat `keypress` per la tecla amb codi: `a`
3. Disparat `keyup` per la tecla amb codi: `a`

## Formats de codificació

Un format de codificació és el que ens permet identificar un caràcter a partir d'un codi. Dos dels més utilitzats són el codi ASCII i Unicode.

### Event.keyCode

Abans es podia fer servir la propietat `keyCode` d'un *event* per determinar quina tecla s'havia premut, però no tots els navegadors retornaven el mateix codi; finalment, aquesta propietat s'ha declarat obsoleta.

### Operador lògic ||

En JavaScript es pot fer servir l'operador lògic `||`, de manera que si el primer valor és `null` o `undefined` ho prova amb el següent valor fins que en troba un que sigui vàlid, i llavors retorna o s'assigna aquest valor.

La versió original de ASCII creada el 1963 admetia 128 caràcters (7 bits) però van crear-se variacions que són conegudes com a *ASCII Estès*, que admeten fins a 256 i inclouen altres caràcters com són les vocals accentuades, la ç...

La codificació Unicode va ser creada a finals del 1987 i suporta una quantitat de caràcters molt més gran, fins a 1.114.112 (0x10FFFF) i es representen afegint-hi el prefix U+ i el codi corresponent en hexadecimal. Aquest codi és compatible amb altres codificacions com ASCII7 o ISO 8859-1.

En realitat tots dos retornen el mateix caràcter, però en el cas de Google Chrome el retorna amb codificació Unicode en lloc de mostrar-lo com a ASCII.

Si ho proveu amb la tecla `control`, veureu que tots dos mostren el mateix:

1. Disparat `keydown` per la tecla amb codi: `Control`
2. Disparat `keyup` per la tecla amb codi: `Control`

En el cas de les tecles no representables com a caràcter no es dispara l'*event* `keypress`.

Normalment els dos casos principals en què ens interessarà capturar els *events* de teclat serà dins dels elements que formen els formularis per determinar si s'està introduint un valor vàlid en una caixa de text, per exemple, i quan tractem amb experiències interactives, com poden ser jocs o *tours* virtuals que reaccionin amb el teclat.

En aquest segon cas, però, ens trobarem amb un problema: si us hi fixeu, quan es deixa polsada una tecla només es repeteix aquesta tecla i, per exemple, seria impossible fer un moviment en diagonal.

D'altra banda, els *events* són disparats en qualsevol moment i no en el punt del joc on els necessitem, és per això que interessa tenir un objecte al qual poder consultar l'estat del botó.

Per aquesta raó s'han de fer servir llibreries específiques o s'ha d'implementar un controlador d'entrada propi que guardi les tecles premudes i pugui ser consultat pel motor del joc o l'aplicació.

#### Evitar que es realitzi l'acció habitual

En alguns casos, amb `Event.preventDefault()` no n'hi ha prou, i pot funcionar fer que la nostra funció retorni el valor `false`, però depèn de l'element i el navegador amb els quals estiguem treballant.

En alguns casos ens pot interessar evitar que es dispari l'*event* que per defecte està associat a una acció; això és possible fent servir el mètode `Event.preventDefault()`. Per exemple, això ens permet aturar l'enviament d'un formulari en prémer el botó d'enviar i, en comptes d'això, realitzar una crida asíncrona.

Es veu més clarament en aquest exemple:

```
1 <script>
2   let element = document.getElementById('test');
3
4   element.addEventListener('keypress', notificaObservador);
5   element.addEventListener('keydown', notificaObservador);
6   element.addEventListener('keyup', notificaObservador);
7
8   function notificaObservador(e) {
```

```

9     console.log("Disparat ", e.type, " per la tecla amb codi: ", e.key || e.
      keyIdentifier);
10     e.preventDefault();
11 }
12 </script>
13 <body id="test">
14     <textarea></textarea>
15 </body>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/EVRdXX?editors=1011](https://codepen.io/ioc-daw-m06/pen/EVRdXX?editors=1011).

Com podeu veure no és possible escriure en el `textarea` i, com que el caràcter no es mostra, tampoc no es dispara l'*event* `keypress`.

### 1.2.4 Altres events del DOM

Hi ha dos subjectes en què ens interessa observar habitualment l'*event* `load` (vegeu la taula 1.6): són l'element `<body>` i l'element `<img>`. El primer cas és més habitual, ja que no és recomanable fer segons quines operacions abans que s'hagi acabat de carregar tot el fitxer HTML, per exemple si volem afegir o eliminar elements del document.

**TAULA 1.6.** Events relacionats amb la càrrega i descàrrega

Tipus	Operació	Descripció
load	onload	Es dispara quan es finalitza la càrrega de la pàgina o d'algun recurs com per exemple una imatge
unload	onunload	Es dispara quan s'està descarregant el document o un recurs
error	onerror	Es dispara si es produeix un error

```

1 <body onload="alert('Document carregat');">

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/XmYxye?editors=1000](https://codepen.io/ioc-daw-m06/pen/XmYxye?editors=1000) (s'executarà cada vegada que canvieu alguna cosa en el panell HTML, ja que el recarrega periòdicament).

L'altre cas en què ens interessa detectar quan es dispara aquest *event* és quan treballem amb imatges, ja que encara que el document s'hagi carregat completament i l'element `<body>` dispari l'*event* `load` no vol dir que les imatges s'hagin carregat, això només indica que el codi per carregar-les és present.

Això ens porta a trobar-nos amb casos en què volem realitzar alguna acció només quan les imatges s'hagin carregat completament, per exemple per reorganitzar-les segons la seva mida o per esperar a inicialitzar un joc en HTML5 perquè fins que no acaben de descarregar les imatges no s'ha de poder iniciar.

Hi ha un altre *event* relacionat directament amb el `load`: si quan es carrega una imatge es produeix un error, no es dispara l'*event* `load` sinó l'*event* `error`, cosa que ens permet reaccionar en conseqüència.

L'exemple següent és força complex: el **gestor de descàrregues** del joc **IOC Puzzle Lite**, que està implementat amb HTML5. No us espanteu, que no cal que entengueu tot el codi ara per ara!, però és interessant que us feu una idea de com funciona, ja que la major part d'informació ja l'hem tractat, i en l'exemple es veu com utilitzar els *events* load i error:

```
1 let DownloadManager = function() {
2   this.successCount = 0;
3   this.errorCount = 0;
4   this.downloadQueue = [];
5   this.cache = {
6     images: {}
7   };
8
9   this.queueDownload = function(imageData) {
10    if (Array.isArray(imageData)) {
11      for (let i = 0; i < imageData.length; i++) {
12        this.downloadQueue.push(imageData[i]);
13      }
14    } else {
15      this.downloadQueue.push(imageData);
16    }
17  };
18
19  this.downloadAll = function(callback, args) {
20    for (let i = 0; i < this.downloadQueue.length; i++) {
21      let path = this.downloadQueue[i].path,
22          id = this.downloadQueue[i].id,
23          img = new Image();
24
25      img.addEventListener("load", function() {
26        this.successCount += 1;
27
28        if (this.isDone()) {
29          callback(args);
30        }
31      }.bind(this), false);
32
33      img.addEventListener("error", function() {
34        this.errorCount += 1;
35
36        if (this.isDone()) {
37          callback(args);
38        }
39      }.bind(this), false);
40
41      img.src = path;
42      this.cache.images[id] = img;
43    }
44  };
45
46  this.isDone = function() {
47    return (this.downloadQueue.length == this.successCount + this.errorCount);
48  };
49
50  this.getImage = function(id) {
51    return this.cache.images[id];
52  };
53 };
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/zvamep?editors=0010](https://codepen.io/ioc-daw-m06/pen/zvamep?editors=0010) (però com que només és una classe aïllada no fa res per si mateixa).

Es tracta d'una classe JavaScript que pot ser instanciada amb `new` d'aquesta manera: `let downloadManager = new DownloadManager();`



Una vegada instanciada, ens exposa les següents operacions:

- **queueDownload(*imageData*)**: afegeix una imatge (o *array* d'imatges) a la cua per ser descarregada. *imageData* conté la informació de la imatge (amb la *id* que li volem assignar per recuperar-la després) i la ruta, per exemple: `{id: "piece-0", path: "img/puzzle-piece-0.png"}`.
- **downloadAll(*callback*, *args*)**: inicia la descàrrega de tots els elements afegits a la cua i, quan acaba, crida la funció passada com a *callback* i *args* com a arguments. Això permet passar de la fase de descàrrega de recursos a la d'iniciar el joc.
- **isDone()**: aquest es fa servir internament; retorna "cert" si ha finalitzat la descàrrega de totes les imatges.
- **getImage(*id*)**: aquest mètode ens retorna la imatge amb l'identificador passat com a argument. Aquesta imatge la recupera del *cache* (memòria cau), on les hem anat descarregant. D'aquesta manera, encara que hi hagi múltiples imatges iguals no cal descarregar-les més d'una vegada.

Fer servir aquesta classe és molt fàcil:

1. Creem una nova instància: `let donwloadManager = new DownloadManager();`
2. Hi afegim els fitxers que vulguem descarregar: `donwloadManager.queueDownload([ {id: 'cotxe', patch: 'imgs/imatge1.png'}, {id: 'tren', patch: 'imgs/imatge2.png'} ] );`
3. Iniciem la descàrrega indicant la funció que s'ha d'invocar quan s'hagin descarregat totes les imatges: `donwloadManager.downloadAll(startGame)`. El segon paràmetre és opcional, pot ser qualsevol cosa que necessitem, per exemple, el nivell.
4. Per accedir a les imatges descarregades només hem d'invocar `downloadManager.getImage('tren')` i obtindrem la imatge, que podrem afegir a la pàgina o a un element de tipus `<canvas>`.

Ara veiem com funciona internament:

- Quan cridem `queueDownload()` la informació que li passem es guarda en un *array*.
- Quan cridem `downloadAll()` es recorre aquest *array*, crea un objecte de classe *Image* (element d'*HTML5*) per a cada element de l'*array* i hi afegeix dos observadors: un per a l'*event* `load` i un altre per a l'*event* `error`.
- Per cada imatge que es descarrega amb èxit o error s'augmenta en 1 el comptador intern d'èxits o errors, segons correspongui.

- Quan la suma de descàrregues amb èxit o error és igual al nombre d'elements a l'*array* d'imatges, sabem que la descàrrega de tots els elements ha finalitzat, així que passem a invocar la funció *callback*, que podria ser, per exemple, la càrrega del nivell o l'inici del joc.

Quan implementem un gestor de descàrregues no s'ha d'oblidar gestionar els errors, ja que si només controléssim els casos d'èxit i una imatge donés error, el joc es bloquejaria perquè mai arribaria a finalitzar les descàrregues.

Això ens porta a un altre problema comú al treballar amb *events* i *callbacks*: el context en què s'executen les funcions no és el mateix en què es van originar; així que, si hem de fer referència a aquest context amb *this*, ens trobem amb problemes. Serà necessari trobar altres solucions, com guardar el context en una altra variable o utilitzar el mètode *bind()* comú a tots els objectes de JavaScript.

### Context d'origen ('this') i context d'execució ('that')

Quan treballem amb *events* i *callbacks*, pot ocórrer que el context en què s'executen les funcions no sigui el mateix en què es van originar. En aquest cas, si hem de fer referència a aquest context amb *this*, ens trobarem amb problemes.

Per exemple, en el cas de la detecció d'*events* el context d'execució serà l'element en que s'ha disparat l'*event* i no pas la funció (o objecte) en el que es va afegir el detector. És a dir, si s'afegeix un detector per l'*event* *load* d'una imatge, el context de la funció que serà invocada quan la imatge acabi de carregar serà l'element imatge que l'ha disparat.

```
1 function ClasseA() {
2   // Aquí this fa referència a la instància de ClasseA
3   let img = new Image();
4   img.addEventListener("load", function () {
5     // En aquest exemple this fa referència a la imatge creada
6   });
7 }
8
9 function ClasseB() {
10  // Aquí this fa referència a la instància de ClasseB
11  let img = new Image();
12  img.addEventListener("load", function () {
13    // En aquest exemple this fa referència a la instància de ClasseB
14  }).bind(this);
15 }
```

Una altra solució habitual per a aquest problema és crear una variable en què guardem la referència a *this*. Aquesta variable acostuma a anomenar-se *that*:

```
1 function ClasseC() {
2   let that = this;
3   // Aquí this i that fan referència a la instància de ClasseB
4   let img = new Image();
5   img.addEventListener("load", function () {
6     // En aquest exemple this fa referència a la imatge creada
7     // Però that conserva la referència a la instància de ClasseB, així que
8     // podem fer-la servir si dins d'aquesta funció ens cal accedir a
9     // ClasseB.
10  });
11 }
```

En altres casos és precisament aquest el comportament que volem: que la referència a `this` sigui a l'objecte on s'està executant la funció, per exemple, per modificar algun atribut d'aquest objecte.

## L'event 'unload'

L'event `unload` té la característica que és disparat quan es descarrega la pàgina, així que és una mica més difícil veure'l en funcionament.

Una particularitat és que **no es poden mostrar alertes** quan es vol reaccionar a aquest *event*, per tant, farem servir els missatges per la consola (que pot visualitzar-se activant les eines de desenvolupador). Això, però, també té un inconvenient: generalment, en recarregar una pàgina s'esborra el *log* de la consola, així que hem de marcar la casella '*preserve log*' (o l'opció de configuració equivalent segons el navegador) a les eines per a desenvolupadors. Vegem-ho:

```
1 <script>
2   let element = document.getElementById('test');
3
4   element.onunload = () => console.log("descarregant el document");
5 </script>
6 <body id="test"></body>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/LprXVM?editors=1011](https://codepen.io/ioc-daw-m06/pen/LprXVM?editors=1011) (Recordeu que heu d'activar l'opció per preservar el *log* o s'esborrarà abans de carregar la pàgina següent).

Per disparar l'event només heu de recarregar la pàgina o modificar el codi HTML a l'entorn `codepen.io` (provoca que només es recarregui la visualització del resultat).

Una vegada s'ha disparat l'event `unload`, **el procés és irreversible**; per exemple, no es pot mostrar un missatge d'avís per indicar que no s'han guardat els canvis. Per altra banda, és possible mostrar un avís abans de recarregar la pàgina observant l'event `beforeunload`.

A la taula 1.7 veureu una llista dels *events* relacionats amb els formularis.

TAULA 1.7. Events relacionats amb els formularis

Tipus	Operació	Descripció
select	onselect	Es dispara quan se selecciona un text
change	onchange	Es dispara quan ha canviat el valor del subjecte i aquest ha perdut el focus
submit	onsubmit	Es dispara quan es pressiona un botó per enviar un formulari
reset	onreset	Es dispara quan es pressiona un botó de reinicialitzar un formulari

S'ha de tenir en compte que l'*event* *change* no es dispara en fer canvis a l'element, sinó **quan perd el focus**. És a dir, mentre escrivim text en un *textarea*, aquest *event* no es dispara, només es dispararà quan canviem a un altre element; per exemple, quan cliquem en un botó o polsem la tecla de tabulació per canviar a l'element següent.

Un fet que haurem de tenir en compte és que quan es dispara l'*event* *submit* serà el moment que podrem fer canvis en la informació que s'enviarà en el formulari, ja sigui afegint més dades, validant les que hi ha o cancel·lant l'enviament si les dades no són correctes.

### Els events 'focus', 'blur' i 'resize'

Tal com mostra la taula 1.8, els *events* *focus* i *blur* són complementaris.

TAULA 1.8. Altres events destacables del DOM

Tipus	Operació	Descripció
focus	onfocus	Es dispara quan un element rep el <i>focus</i>
blur	onblur	Es dispara quan un element perd el <i>focus</i>
resize	onresize	Es dispara quan es canvia la mida de l'element

Quan un element rep el *focus*, just abans l'element que el tenia dispara l'*event* *blur* i a continuació l'element que l'ha rebut dispara l'*event* *focus*; la seqüència serà la següent:

1. Cliquem un element que no té el focus
2. L'element que té el *focus* dispara l'*event* *blur*
3. L'element que hem clicat rep el focus i dispara l'*event* *focus*

Es pot veure en aquest exemple amb dos quadres de text:

#### Com es rep el focus?

Les dues maneres més habituals per les quals un element rep el *focus* són: perquè s'ha clicat l'element o perquè s'ha premut la tecla de tabulació. Amb la tecla de tabulació es van recorrent tots els elements de la pàgina capaços de rebre el *focus*.

```

1 <script>
2   let text1 = document.getElementById('test1');
3   let text2 = document.getElementById('test2');
4
5   text1.onfocus = e => console.log(e.type, "El primer quadre de text té el
6     focus");
7
8   text2.onfocus = e => console.log(e.type, "El segon quadre de text té el focus
9     ");
10
11  text1.onblur = e => console.log(e.type, "El primer quadre de text ha perdut
12    el focus");
13
14  text2.onblur = e => console.log(e.type, "El segon quadre de text ha perdut el
15    focus");
16 </script>
17 <input type="text" id="test1" />
18 <input type="text" id="test2" />

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/LprqNz?editors=1011](https://codepen.io/ioc-daw-m06/pen/LprqNz?editors=1011).

Hi ha dos quadres de text, inicialment cap dels dos elements té el focus, així que no es dispara l'*event* focus de cap d'ells; i si anem alternant entre l'un i l'altre veurem a la consola:

1. focus El primer quadre de text té el focus
2. blur El primer quadre de text ha perdut el focus
3. focus El segon quadre de text té el focus
4. blur El segon quadre de text ha perdut el focus
5. etc.

Si cliquem a l'espai en blanc de la pàgina, es pot veure que es llença un últim blur quan el quadre de text perd el focus.

### Els objectes window i document

Els elements `window` i `document` no són propis de JavaScript sinó que els proporciona el navegador. L'objecte `document` representa el document HTML i l'objecte `window`, la finestra on es mostra.

A més a més, `document` és part de l'objecte `window` i és possible accedir-hi també com a `window.document` (encara que no té sentit fer-ho perquè podem accedir-hi directament com a `document`).

Només hi ha un element al qual podem registrar-nos com a observadors per l'*event* `resize`, i ser notificats d'alguna cosa, es tracta de l'element `window`, que representa la finestra del navegador on es mostra la pàgina.

Vegem-ho en aquest exemple:

```
1 <style>
2   div {
3     position: absolute;
4     top: 0;
5     bottom: 0;
6     left: 0;
7     right: 0;
8     background-color: red
9   }
10 </style>
11 <script>
12   window.onresize = e => {
13     let width = window.innerWidth;
14     let height = window.innerHeight;
15     console.log(e.type, "nova mida: " + width + "px x " + height + "px");
16   };
17
18   let element = document.getElementById('test');
19
20   element.onresize = e => console.log("S'ha disparat l'esdeveniment resize del
21     <div>!");
22 </script>
23 <div id="test"></div>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/meKvMQ?editors=1111](https://codepen.io/ioc-daw-m06/pen/meKvMQ?editors=1111).

Com deveu haver vist, no obtenim la nova mida a partir de l'*event* sinó que el traïem del mateix objecte `window`. Ho fem així perquè aquesta informació no ens la proporciona l'*event*, així que no hi ha cap altra manera de fer-ho.

No tots els navegadors proporcionen aquesta informació de la mateixa manera, tot i que les versions actuals de Mozilla Firefox i Chrome sí que ho fan. Si volem assegurar-nos que obtenim la mida correcta, hem de fer una implementació més complexa tenint en compte les diferents opcions o recórrer a alguna llibreria com **jQuery**, que ja afegeix la nova mida dins de l'objecte *event*,

També deveu haver notat que tot i que la mida de l'element `div` sí que canvia (perquè ocupa tota la finestra), no es dispara l'*event* `resize`. Recordeu que aquest *event* només es dispara per l'objecte `window`.

No hi ha cap *event* que es dispari quan un element del document canvia de mida; per exemple, si no s'ha definit una mida fixa per a una imatge, quan es descarrega, canviarà la mida que té per la mida real, i aquest canvi no és detectable.

L'*event* `resize` només es dispara quan canvia la mida de la finestra; és a dir, quan redimensionem la finestra o canviem l'amplada de la secció de les eines del desenvolupador, perquè només es considera com a finestra del navegador la secció on es mostra la pàgina web.

Així que, en cas de necessitar fer aquestes comprovacions, l'única manera de fer-ho és **crear un temporitzador** que periòdicament comprovi la mida dels elements que ens interressi i, si ha canviat, reaccioni en conseqüència.

## 1.2.5 Events d'HTML5 i HTML5 media

Exposem aquí una llista d'*events* categoritzats com a HTML5 i HTML5 *media*, i només entrarem en detalls en aquells que presentin alguna peculiaritat. Comencem amb els tipus d'*events* generals que podeu trobar a la taula 1.9.

TAULA 1.9. Events d'HTML5 generals

Tipus	Operació	Descripció
beforeunload	onbeforeunload	Es dispara abans de descarregar la pàgina i ens permet cancel·lar l'operació

L'*event* `beforeunload` ens permet evitar que una pàgina sigui descarregada. Es dispara abans que l'*event* `unload` i, si dins de la funció que és notificada que s'ha disparat aquest *event* retornem un missatge, el mostrarà i demanarà confirmació abans de continuar.

Vegem-ne un primer exemple molt senzill perquè quedi clar:

Trobareu més informació sobre l'*event* `beforeunload` en aquest mateix apartat.

```
1 <script>
2   window.onbeforeunload = e => 'Estàs segur que vols abandonar la pàgina?';
3 </script>
4 <p>Recarrega la pàgina per veure el missatge de confirmació.</p>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/YyvgzM?editors=1010](https://codepen.io/ioc-daw-m06/pen/YyvgzM?editors=1010).

Si proveu de recarregar la pàgina, us mostrarà un missatge demanant confirmació abans de sortir. Tingueu en compte que als navegadors més moderns no es pot configurar el text del missatge i en surt un per defecte.

Segons la versió del navegador, **el missatge pot ser que no es mostri**, ja que el fet de mostrar o no el missatge de confirmació abans de sortir no forma part de l'especificació d'HTML5. Així doncs, hi ha navegadors com Chrome que mostren el missatge, mentre que d'altres com Mozilla Firefox no el mostren.

S'ha de tenir en compte que el valor que es retorna és irrellevant, perquè **sempre que es retorni quelcom** es mostrarà el diàleg, encara que sigui el valor *fals*.

Comprovareu que això és cert amb aquest exemple:

```
1 <script>
2   window.onbeforeunload = e => false;
3 </script>
4 <p>Recarrega la pàgina per veure el missatge de confirmació.</p>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/XmYGbZ?editors=1010](https://codepen.io/ioc-daw-m06/pen/XmYGbZ?editors=1010).

En canvi recordeu que, si no es retorna res, no es mostrarà el diàleg:

```
1 <script>
2   window.onbeforeunload = function(e) {
3     console.log("No es mostra el diàleg");
4   };
5 </script>
6 <p>Recarrega la pàgina, però no veuràs cap diàleg.</p>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/ojyVXJ?editors=1011](https://codepen.io/ioc-daw-m06/pen/ojyVXJ?editors=1011) (Recordeu que heu d'activar l'opció per preservar el *log* o s'esborrarà abans de carregar la pàgina següent).

Això ens dóna molt de joc, perquè ens permet fer diverses coses, com comprovar si s'ha fet alguna modificació quan volem recarregar la pàgina o demanar confirmació només si s'han produït canvis, tal com es pot veure en l'exemple següent:

```
1 <script>
2   let comptador = 0;
3   let element = document.getElementById('test');
4
5   element.onclick = () => {
6     comptador++;
7     console.log("Comptador:", comptador);
8   }
```

```

9
10
11   window.onbeforeunload = e => {
12       if (comptador>0) {
13           return "El comptador ha canviat, segur que vols sortir?";
14       }
15   };
16 </script>
17 <button id="test">Augmenta comptador</button>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/pjKYjP?editors=1010](https://codepen.io/ioc-daw-m06/pen/pjKYjP?editors=1010).

Si proveu de recarregar la pàgina sense prémer el botó, no us demanarà confirmació perquè el comptador serà igual a 0; en canvi, en el moment en què el cliqueu, el comptador augmentarà i sí que us demanarà confirmació per recarregar la pàgina.

Vegem un exemple de quan es dispara l'*event* input (vegeu la taula 1.10) amb un quadre de text:

**TAULA 1.10.** Events d'HTML5 relacionats amb els formularis

Tipus	Operació	Descripció
input	oninput	Es dispara cada vegada que es produeix un canvi a l'element
invalid	oninvalid	Es dispara quan s'intenta enviar un formulari i l'element no compleix amb els requisits que s'han imposat

```

1 <script>
2   let element = document.getElementById('test');
3
4   element.addEventListener('input', e => console.log("S'han produït canvis al
   quadre de text", e));
5 </script>
6 <input type="text" id="test"/>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/gaKEPb?editors=1011](https://codepen.io/ioc-daw-m06/pen/gaKEPb?editors=1011).

Fixeu-vos que cada vegada que premeu una tecla o enganxeu, elimineu o retalleu el text, aquest *event* es dispara.

L'*event* input és molt similar a change, però input es dispara quan es produeix qualsevol canvi, mentre que change només es dispara si el contingut ha canviat en perdre el focus. Així que l'*event* input **es dispararà molt més sovint** que change.

S'ha de tenir en compte que no tots els elements dispatchen els *events* input, només ho fan aquells que tenen la propietat `contenteditable` i aquesta propietat té el valor de *cert*.

L'*event* invalid el veurem detalladament a l'apartat "Programació amb formularis", junt amb la resta d'*events* relacionats amb els formularis.



## HTML5 media

Amb HTML5 es van afegir molts elements per millorar la gestió d'elements multimèdia (àudio i vídeo), i, juntament amb aquests elements, es van afegir també una gran quantitat d'*events* que poden servir-nos per millorar-ne el control; vegeu a la taula 1.11 un petit extracte d'aquests *events*:

TAULA 1.11. Events d'HTML5 media

Tipus	Operació	Descripció
canplay	oncanplay	Es dispara quan és possible començar la reproducció del recurs d'àudio o vídeo
ended	onended	Es dispara quan finalitza la reproducció del recurs
pause	onpause	Es dispara quan es posa en pausa la reproducció
play	onplay	Es dispara quan s'inicia la reproducció del recurs
waiting	onwaiting	Es dispara quan es pausa la reproducció perquè hi ha una falta temporal de dades (per exemple, si la reproducció és en <i>streaming</i> i encara no s'ha acabat de descarregar el recurs sencer)

Com que el tractament d'elements multimèdia queda fora de l'abast d'aquests materials no en veurem cap exemple.

Per acabar, és interessant saber que hi ha una sèrie d'*events* relacionats amb la funcionalitat **Drag&Drop**, però no n'hem afegit detalls ni exemples perquè no està completament implementada de manera homogènia a tots els navegadors.

### Drag&Drop

La funcionalitat Drag&Drop ens permet arrossegar elements per reorganitzar-los a la pàgina, per exemple, per canviar l'ordre d'una sèrie de fotos. És recomanable fer servir llibreries com jQuery si necessitem aquesta funcionalitat, ja que ens permeten aplicar-la amb molt poc esforç.

## 1.2.6 Events de les Web API estàndard

Encara que la llista de les Web API estàndard és força extensa, en aquesta secció ens centrarem a comentar els *events* relacionats amb algunes de les Web API més utilitzades.

Les quatre Web API que veurem són:

- **XMLHttpRequest**: permet fer peticions asíncrones al servidor.
- **Web Storage**: permet emmagatzemar dades de forma persistent al navegador.
- **WebSocket**: permet crear una connexió TCP amb un servidor, per exemple, per crear una aplicació de xat o un joc multijugador.

- **Web Workers:** tot i que els navegadors no permeten fer programes que aprofitin les possibilitats multifil dels ordinadors amb més d'un processador, gràcies a aquesta API sí que és possible treballar amb programació multifil, creant tasques que es realitzaran de forma paral·lela sense bloquejar el fil principal de la interfície.

### Programació multifil

Programació que crea tasques que es realitzaran de forma paral·lela.

## API XMLHttpRequest

Normalment, quan es realitza una petició via **AJAX**, el servidor ens retorna una resposta amb una sèrie de dades. Quan parlem de *descàrrega de dades* ens referim a aquestes dades (vegeu la taula 1.12).

TAULA 1.12. Events de l'API XMLHttpRequest

Tipus	Operació	Descripció
abort	onabort	Es dispara quan es cancel·la la descàrrega de dades
error	onerror	Es dispara quan es produeix un error durant la descàrrega
load	onload	Es dispara quan finalitza la descàrrega amb èxit
loadend	onloadend	Es dispara quan s'atura la descàrrega, sigui quina sigui la raó (error, abort o load)
loadstart	onloadstart	Es dispara quan s'inicia la descàrrega de dades
readystatechange	onreadystatechange	Es dispara quan canvia la propietat <code>readyState</code> . Aquesta propietat és la que ens indica si s'ha completat amb èxit o no la petició.

## API Web Storage

Aquest API permet guardar informació al navegador, de forma semblant a les galetes, però amb una capacitat d'emmagatzemage molt superior (vegeu la taula 1.13). S'ha de tenir en compte que aquest magatzem de dades és gestionat pel navegador i, per consegüent, si obriu la mateixa pàgina amb dos navegadors diferents, cadascun utilitzarà el seu magatzem independent.

TAULA 1.13. Events de l'API Web Storage

Tipus	Operació	Descripció
storage	onstorage	Es dispara quan hi ha canvis al magatzem de dades

Cal destacar que la detecció de canvis en el magatzem de dades no es dispara a la pestanya on es produeixen els canvis (ja que l'aplicació responsable de fer canvis ha de saber, forçosament, què ha canviat), sinó que aquests són detectat per altres pestanyes obertes al mateix navegador.

Una altra peculiaritat és que, per escoltar els *events* del Web Storage, s'ha d'afegir la detecció d'*events* a l'objecte `window` i no pas al magatzem: `window.addEventListener('storage', callback);`.

## API WebSocket

Aquesta API ens permet connectar amb un servidor via TCP. És una opció relativament recent, ja que abans d'HTML5, això no era possible i s'havien de fer servir altres tecnologies, com Flash o Java (vegeu la taula 1.14).

TAULA 1.14. Events de l'API WebSocket

Tipus	Operació	Descripció
open	onopen	Es dispara quan s'estableix una connexió
close	onclose	Es dispara quan es tanca la connexió
message	onmessage	Es dispara quan es rep un missatge a través del WebSocket
error	error	Es dispara quan s'ha tancat la connexió amb perjudici, per exemple, amb pèrdua de dades

### Connexions TCP

TCP és un protocol que permet establir connexions entre aplicacions client i servidor. En podeu trobar més informació al següent enllaç: [www.goo.gl/Ibd0k3](http://www.goo.gl/Ibd0k3).

## API Web Workers

L'API Web Workers només té un *event* propi que podem observar (vegeu la taula 1.15).

TAULA 1.15. Events de l'API Web Workers

Tipus	Operació	Descripció
message	onmessage	Es dispara al <i>web worker</i> quan hi ha un missatge per comunicar a l'aplicació



## 2. Programació amb formularis

Els aspectes més importants que cal conèixer sobre els formularis són: per a què serveixen i com validar que la informació és correcta abans de fer-la servir. Amb l'arribada d'HTML5 els elements que formen l'estructura dels formularis van adquirir una gran quantitat de noves opcions i, d'aquesta manera, van facilitar-ne la validació i van oferir als navegadors l'opció de mostrar quadres d'entrada de dades més adients, com per exemple els selectors de dates.

Cal destacar la importància de la validació de les dades -aquest és un dels principals usos que es va donar a JavaScript inicialment. Gràcies a l'evolució del llenguatge HTML s'ha simplificat molt aquesta tasca, ja que ara és possible incrustar regles de validació directament a les etiquetes HTML. Tot i així, en alguns casos més complexos, com la càrrega de fitxers, continua requerint implementar la validació mitjançant JavaScript.

Una de les tècniques més potents per realitzar validacions complexes, reemplaçaments i extracció de textos és la utilització *expressions regulars*. Aquestes expressions permeten utilitzar patrons complexos per realitzar cerques a les cadenes de text. Gràcies a aquestes és possible simplificar el tractament de la informació guardada en galetes (*cookies* en anglès), ja que aquestes no utilitzen un format fàcil d'utilitzar.

### 2.1 Què és un formulari?

Segurament tots us haureu trobat amb formularis en paper en un moment o altre: es tracta d'un document amb uns textos que us demanen determinada informació, i uns espais en blanc per omplir-los. És a dir, la seva funció és **recollir informació** que més endavant **serà processada** d'una manera o altra.

En el cas que ens interessa el seu fi és el mateix, tot i que la seva forma és molt diferent. Pràcticament totes les pàgines que visitem diàriament tenen formularis per una banda o altra:

- Formularis per realitzar cerques com [www.google.com](http://www.google.com).
- Formularis per identificar-nos en una pàgina com el de [www.ioc.xtec.cat](http://www.ioc.xtec.cat).
- Pàgines amb *quizzes* que, seleccionant unes opcions o les altres, ens mostren diferents resultats.
- Els comentaris que es publiquen a través de Facebook.
- I, per descomptat, la versió digital dels formularis paper d'ús habitual; com el de la declaració de la renda.

El seu funcionament és molt semblant als formularis en paper: ofereixen unes opcions o quadres de text per marcar o omplir; una vegada s'ha omplert aquesta informació, és **recollida, enviada/entregada i processada**.

S'ha de tenir en compte que un formulari només és un mitjà per introduir informació que serà enviada a un servidor o bé serà utilitzada per l'aplicació en JavaScript.

#### Exemples de formularis segons la finalitat

- Un exemple de formulari enviat a un servidor seria el de cerca de Google. Aquest formulari recull les dades necessàries per fer la cerca i, un cop l'usuari prem el botó, s'envien al servidor. Aquest fa la cerca amb les dades i retorna el resultat al navegador del client perquè el mostri a la finestra.
- Un exemple -molt simple- de formulari en JavaScript seria una aplicació amb dos quadres de text, en què s'han d'introduir dos nombres, i un botó que en clicar-lo mostri el resultat de la suma. En aquest formulari, quan l'usuari prem el botó, és el propi navegador qui directament fa el processament de les dades i mostra el resultat que obté.

Tots dos fan servir formularis, però amb objectius diferents.

## 2.2 Estructura d'un formulari

En general, tots els formularis comparteixen una mateixa estructura similar a aquesta:

```
1 <form>
2   Introdueix el teu nom:
3   <input type="text" />
4   <br>
5   Introdueix el teu cognom:
6   <input type="text" />
7   <br>
8   <input type="submit" value="enviar" />
9 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/eJxpVE?editors=1000](https://codepen.io/ioc-daw-m06/pen/eJxpVE?editors=1000).

D'aquesta manera, tots els elements del formulari han d'anar niats dins del bloc format per l'etiqueta `form`, i tots els elements que es troben dins queden lligats a aquest formulari.

Podeu veure que dintre del formulari hi ha un element `input` que es repeteix 3 vegades. Els dos primers són de tipus `text` i això fa que es mostrin com una capsa per escriure text, mentre que el darrer és de tipus `submit` i es mostra com un botó.

Aquest últim element és imprescindible, ja que és el que ens permet enviar la informació del formulari al servidor.



### Serveis Web RESTful

RESTful és una arquitectura de serveis web que es basa a fer servir un mateix URL amb diferents verbs per realitzar diferents accions. Per exemple, suposant que tenim un servei web que permet gestionar l'usuari, podríem tenir els següents URL:

- `www.exemple.cat/usuarios/joan` (GET): retorna les dades de l'usuari joan.
- `www.exemple.cat/usuarios/joan` (PUT): modifica les dades de l'usuari joan.
- `www.exemple.cat/usuarios/joan` (DELETE): elimina l'usuari joan.
- `www.exemple.cat/usuarios/` (POST): crea un usuari nou.
- `www.exemple.cat/usuarios/` (GET): retorna la llista completa d'usuaris.

Tot i que HTML no admet l'ús de tots els verbs des dels formularis, sí que és possible fer crides *AJAX* que els facin servir.

### 2.2.1 Elements d'un formulari

Dintre de les etiquetes `form` podem trobar els mateixos elements que en qualsevol document HTML, per aquesta raó només ens centrarem en els que són específics dels formularis, en la seva estructuració i en l'enviament de dades.

Aquesta és la llista dels elements que s'utilitzen més sovint a l'hora de crear un formulari:

- **form**: element principal que forma un formulari; totes les dades definides en elements que es trobin entre l'etiqueta d'apertura i tancament seran enviats en clicar el botó d'enviament.
- **input**: aquest element permet que l'usuari introdueixi dades, i pot representar-se de moltes maneres diferents: botons, quadres de text, selectors de colors...
- **textarea**: serveix únicament per introduir textos. A diferència de l'`input` de tipus `text`, el `textarea` és multilíneal i s'utilitza generalment per la introducció de textos llargs.
- **button**: aquest element crea un botó clicable que, per defecte, el seu comportament és enviar el formulari. Pot donar problemes en versions de navegadors antigues.
- **label**: per tal de millorar l'estructura semàntica del document és recomanable fer servir aquest element per afegir les etiquetes corresponents a cada element d'entrada de dades.
- **datalist** (HTML5): aquest element ens permet afegir llistes al nostre codi HTML que poden ser utilitzades per altres elements del formulari.



Si volem elaborar llistes desplegable més complexes, haurem de fer servir els elements `select`, `option` i `optgroup`:

Tenim altres elements que es poden considerar opcionals, ja que no afecten com s'envia la informació sió com s'estructuren les opcions:

- **select**: embolcalla tots els elements que formen la llista, i inclou l'atribut que determina el nom del paràmetre amb què s'enviarà la informació a l'atribut `name`.
- **option**: aquest element conté el valor que es passarà en enviar el formulari i el text que es veurà en desplegar la llista que correspon al seu atribut `value`.
- **optgroup**: serveix per agrupar les opcions de la llista, separades per un títol no seleccionable que s'especifica al seu atribut `label`.

Al contrari dels elements que es poden utilitzar individualment, aquestes llistes s'han de crear amb una estructura específica; vegem-ho:

```
1 <form action="test.php" method="GET">
2   <select name="selector">
3     <option value="SEU-A">Barcelona</option>
4     <option value="SEU-B">Tarragona</option>
5     <option value="SEU-C">Lleida</option>
6     <option value="SEU-D">Girona</option>
7   </select>
8   <input type="submit" />
9 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/gPqPdW?editors=1000](https://codepen.io/ioc-daw-m06/pen/gPqPdW?editors=1000).

Com es pot veure en la pestanya de xarxa de les eines del desenvolupador, en fer clic sobre el botó d'enviar s'envia al servidor el paràmetre anomenat `selector` amb el valor de l'opció seleccionada, per exemple `SEU-A` si se selecciona *Barcelona*: `test.php?selector=SEU-A`

La diferència que trobem entre aquest tipus de llista i fer servir un `datalist` amb un element `input` és que aquestes llistes obliguen a seleccionar un dels elements (no es pot escriure el que vulguem) i a més permet afegir grups d'opcions gràcies a l'element `optgroup`.

```
1 <form action="test.php" method="GET">
2   <select name="assignatura">
3     <optgroup label="Primer Curs">
4       <option value="M01">Sistemes operatius</option>
5       <option value="M03">Programació</option>
6     </optgroup>
7     <optgroup label="Segon Curs">
8       <option value="M06">Desenvolupament d'aplicacions en entorn client</option>
9       <option value="M09">Disseny d'interfícies Web</option>
10    </optgroup>
11  </select>
12  <input type="submit" />
13 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/mVvVaW?editors=1000](https://codepen.io/ioc-daw-m06/pen/mVvVaW?editors=1000).

En aquest cas es pot comprovar que la llista desplegable inclou les categories *Primer Curs* i *Segon Curs*, que tenen la funció de separar tots dos grups d'assignatures, i no són seleccionables.

També trobem alguns elements que, si bé no tenen efecte sobre l'enviament de dades, sí que milloren l'estructura semàntica del formulari, ja que agrupen els possibles conjunts de dades indicant clarament de què es tracta, en lloc de fer servir capçaleres i elements div genèrics; com ara fieldset i legend:

- **fieldset:** permet agrupar els elements d'un formulari (vegeu la figura 2.1).

**FIGURA 2.1.** Representació d'un "fieldset"

Dades d'usuari

Nom d'usuari:

Contrasenya:

- **legend:** afegeix un títol dins d'un fieldset.

Vegem-ne el funcionament:

```

1 <style>
2 fieldset {
3     width: 240px;
4 }
5 </style>
6 <fieldset>
7     <legend>Dades d'usuari</legend>
8     <label>Nom d'usuari:
9         <input type="text" name="nom_usuari" /><br>
10    </label>
11    <label>Contrasenya:
12        <input type="password" name="contrasenya" />
13    </label>
14 </fieldset>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/KVJOOB?editors=1000](https://codepen.io/ioc-daw-m06/pen/KVJOOB?editors=1000).

### 2.2.2 Element 'form'

### Atributs de 'form'

Podeu trobar una descripció completa de les propietats de l'element `form` en l'enllaç següent: [www.google.com/p5DRi8](http://www.google.com/p5DRi8).

Tal com passa amb altres elements, si volem poder identificar fàcilment un formulari, hem d'especificar el seu atribut `id`, que ha de ser únic entre els elements de la pàgina. Això ens permet lligar-lo a elements externs a l'estructura del formulari, i accedir de forma fàcil i ràpida a través de JavaScript.

L'element `form` accepta dos mètodes per enviar les dades: `POST` i `GET`. A banda de la intenció que tinguem quan enviem les dades, cal tenir en compte algunes diferències importants entre tots dos; vegem-les:

- **GET**: els valors dels paràmetres es passen al servidor a través de l'URL. El format és l'URL de destí, seguida d'un interrogant (?), afegint els parells de paràmetre-valor units amb un signe d'igual (=), i encadenant els següents paràmetres amb el símbol (&). Per exemple, si tenim els paràmetres nom i cognom, en enviar el formulari l'URL resultant podria ser semblant a `test.php?nom=john&cognom=doe`. A causa de la manera com es passen les dades hi ha una limitació de longitud imposada pels servidors -no per l'especificació-, i **no s'ha de fer servir GET per enviar grans volums de dades o text**.
- **POST**: aquest mètode afegeix els paràmetres dins de la capçalera, de manera que ni és visible a l'URL ni té la mida limitada. Cal tenir en compte que encara que no siguin visibles a l'URL **no estan encriptats ni amagats**: des del panell de xarxa de les eines de desenvolupador dels navegadors es poden veure els paràmetres enviats amb la capçalera.

#### Limitació de la mida de l'URL

Si volem assegurar-nos que les peticions `GET` es fan correctament, hem de tenir en compte que els navegadors més antics no admeten URL de més de 255 bytes; per exemple, Internet Explorer només admet fins a 2.083 caràcters ([www.google.com/search?q=2.083+characters+url+limit&btnG=Search](http://www.google.com/search?q=2.083+characters+url+limit&btnG=Search)).

Es recomana fer servir el mètode `GET` per l'enviament d'un volum de dades baix o si volem que es generi un URL amb els paràmetres. Els URL generats d'aquesta manera ens permeten fer proves més fàcilment o guardar els enllaços per visitar-los posteriorment.

En tots els altres casos, com poden ser enviar fitxers o volums importants de dades o si desitgem que no es vegin a l'URL els paràmetres enviats, farem servir `POST`.

Un altre atribut molt important és el `enctype`. Aquest atribut especifica el tipus de format (*media type* o *MIME type*) que s'envia al servidor:

- **application/x-www-form-urlencoded**: aquest és el **valor per defecte** que s'assigna si no s'especifica res.
- **multipart/form-data**: aquest valor és el que s'ha d'especificar **si volem enviar fitxers** juntament amb el nostre formulari. Amb aquest tipus de format sempre s'ha de fer servir el mètode `POST`.

Així doncs, un formulari amb enviament de fitxers quedaria així:

```
1 <form action="test.php" method="POST" enctype="multipart/form-data" id="form-  
  principal">  
2   <input type="file" />  
3   <input type="submit" />  
4 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-dawm06/pen/LGqxWm?editors=1000](https://codepen.io/ioc-dawm06/pen/LGqxWm?editors=1000).

Amb HTML5 s'han afegit algunes propietats als botons (tant els elements `button` com els `input` de tipus botó) que permeten modificar com s'envia el formulari. Aquestes propietats sobreescriuen el comportament normal en enviar el formulari clicant el botó; són les següents:

- **formaction** especifica on s'envien les dades del formulari.
- **formenctype** especifica el tipus de format per enviar les dades.
- **formmethod** especifica el mètode que es farà servir per enviar les dades.
- **formnovalidate** salta la validació de les dades.

### 2.2.3 Element 'input'

L'element `input` es va enriquir moltíssim amb HTML5. Mentre que altres elements es mantenen pràcticament igual que en versions anteriors, a aquest se li han afegit una gran quantitat de tipus que no són disponibles als navegadors antics o fins i tot en alguns navegadors actuals. Per aquesta raó, cal distingir entre els elements que són compatibles amb versions anteriors i aquells que es van afegir amb HTML5 (podeu comprovar les compatibilitats amb els diferents navegadors a la pàgina <https://caniuse.com/?search=input%20type>).

Fixeu-vos que l'element `input`, al contrari que la majoria dels elements HTML, s'autotanca: `type="text" />`. Com que no ha de contenir res al seu interior **no cal afegir una etiqueta de tancament independent**, igual que passa amb l'element `img`.

En general, quan parlem d'un element de tipus `input`, ens referim a un camp del formulari

### Atributs d'input

Podeu trobar una llista completa dels atributs d'input al següent enllac: [www.goo.gl/jSaZgH](http://www.goo.gl/jSaZgH).

Els atributs comuns d'ús més freqüent i, per tant, compatibles amb versions anteriors, són:

- **id:** ens permet identificar unívocament l'element, de manera que el podem manipular fàcilment des de codi, i ens serveix per enllaçar-lo amb elements `label`.
- **name:** aquest atribut és indispensable per poder enviar dades, ja que el seu valor és el que es fa servir com a nom del paràmetre.
- **value:** valor per defecte o valor actual del camp, segons el seu tipus.
- **disabled** (booleà): quan s'aplica aquest atribut el camp estarà desactivat i es mostrarà amb un estil diferent per destacar-ho. Els valors dels camps amb aquest atribut **no s'envien amb el formulari**.
- **readonly:** similar a `disabled`, però no canvia l'estil de representació i

només funciona amb alguns tipus. A diferència de `disabled`, els valors d'aquests camps **sí que s'envien amb el formulari**.

- **size**: permet establir la mida del control, **en caràcters, si conté text**, o en píxels si es tracta d'altres tipus i treballem amb navegadors antics. En HTML5 aquest atribut només s'aplica a elements que contenen text, així que sempre s'aplica com a nombre de caràcters.
- **maxlength**: en els camps de tipus text com `text`, `email`, `password`, etc. especifica el nombre màxim de caràcters que admet.

Proveu l'exemple següent per veure la diferència entre els camps activats i els desactivats:

```

1 <input type='text' value="aquest text es troba activat" /><br>
2 <input type='text' disabled value="aquest text es troba desactivat" /><br>
3 <label>Casella desactivada
4   <input type='checkbox' disabled />
5 </label><br>
6 <label>Casella activada
7   <input type='checkbox' />
8 </label><br>
9 <label>Casella desactivada i marcada
10  <input type='checkbox' disabled checked />
11 </label><br>
12 <label>Casella activada i marcada
13  <input type='checkbox' checked />
14 </label>

```

#### L'atribut booleà

Un atribut booleà només pot tenir dos valors: `true` i `false` (cert i fals); aquests valors són admesos pels operadors lògics (`&&`, `||` i `!`). En el cas dels formularis, l'existència de l'atribut equival a `true` i l'absència a `false`, independentment del valor assignat.

#### Quan fer servir la propietat 'size'

No és recomanable fer servir la propietat `size`, ja que per modificar l'estil és més apropiat utilitzar CSS. Però si es fa servir juntament amb la propietat `maxlength` s'aconsegueix un quadre de text de la mida exacta.

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/eJxvj?editors=1000](https://codepen.io/ioc-daw-m06/pen/eJxvj?editors=1000).

En el següent exemple, podeu veure la diferència entre un camp amb la mida establerta com a propietat `size` o fent servir CSS, i l'efecte de la propietat `maxlength`:

```

1 <style>
2   label {
3     display: block; /* això evita haver d'afegir un salt de línia */
4   }
5   .mida {
6     width: 5em;
7   }
8 </style>
9 <label>Quadre amb size="5"
10  <input type="text" size="5" />
11 </label>
12 <label>Quadre amb estil CSS width="5em"
13  <input type="text" class="mida" />
14 </label>
15 <label>Quadre amb size="5" i maxlength="5"
16  <input type="text" size="5" maxlength="5" />
17 </label>
18 <label>Quadre amb estil CSS width="5em" i maxlength="5"
19  <input type="text" class="mida" maxlength="5" />
20 </label>
21 <label>Quadre amb size="10" i maxlength="5"
22  <input type="text" size="10" maxlength="5" />
23 </label>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/GozeMe?editors=1000](https://codepen.io/ioc-daw-m06/pen/GozeMe?editors=1000).

Entre els elements que **només estan disponibles per a HTML5** trobem els següents:

- **placeholder**: si s'especifica aquest atribut, es mostra un text quan el camp és buit; generalment s'aprofita per indicar a l'usuari què s'espera que s'introdueixi en aquest camp, per exemple: *introdueix el teu nom*. Aquest atribut només està disponible per als elements que contenen text i no compta com a contingut per al camp, és a dir, si s'envia el formulari i no hem afegit cap text, s'enviarà buit.
- **autofocus** (booleà): només es pot fer servir en un element per document. Fa que aquest element sigui el seleccionat automàticament en carregar la pàgina.
- **form**: permet indicar l'id d'un formulari al qual s'enllaçarà aquest element. Gràcies a aquest atribut és possible afegir un element fora de l'estructura del formulari però que les seves dades s'enviïn juntament amb el formulari amb l'id especificat.
- **autocomplete**: habilita la funció d'autocompletar del navegador per al tipus que s'indiqui, per exemple: email, tel, bday, postal-code, etc.
- **list**: s'utilitza juntament amb l'element `datalist`; afegeix la llista especificada a aquest atribut com a valors per mostrar.
- **required** (booleà): quan hi ha aquest atribut, el camp ha de tenir algun valor abans de poder enviar el formulari. No tots els camps admeten aquest atribut.
- **pattern**: el valor és una expressió regular que es fa servir per validar el contingut del camp. Només s'aplica quan el tipus de camp conté text.

Les expressions regulars es veuran detalladament en la secció "Expressions regulars" d'aquest mateix apartat.

Fixeu-vos com fent servir l'atribut `placeholder` es fa molt més entenedor com s'ha d'omplir el formulari:

```
1 <style>
2   label {
3     display: block;
4   }
5
6   input {
7     width: 20em;
8   }
9 </style>
10 <label>
11   <input type="text" name="dni" placeholder="Introdueix el teu DNI incloent la
12     lletra"/>
13   DNI
14 </label>
15 <label>
16   <input type="email" name="email" placeholder="Introdueix una adreça de correu
17     vàlida"/>
18   Correu electrònic
19 </label>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/zreXBL?editors=1100](https://codepen.io/ioc-daw-m06/pen/zreXBL?editors=1100).

Vegem un exemple de com s'enllaça un element extern a l'estructura del formulari:

```
1 <form id="principal" action="test.php" method="GET">
2   <input type="text" name="element-intern" value="intern" />
3   <input type="submit"/>
4 </form>
5 <input type="text" name="element-extern" value="extern" form="principal" />
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/RrvdMv?editors=1000](https://codepen.io/ioc-daw-m06/pen/RrvdMv?editors=1000).

Si us fixeu en el panell de xarxa de les eines de desenvolupador, veureu que en fer clic sobre el botó “enviar” la petició que s'envia conté els paràmetres `element-intern` i `element-extern`, tot i que el camp corresponent a `element-extern` es troba fora de l'estructura del form.

Vegem, ara, el funcionament d'un element `data-list` juntament amb l'atribut `list` en un altre camp:

```
1 <label>Selecciona una seu:
2   <input list="seus" name="seu" />
3 </label>
4 <datalist id="seus">
5   <option value="Barcelona">
6   <option value="Girona">
7   <option value="Lleida">
8   <option value="Tarragona">
9 </datalist>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/gPqyMa?editors=1000](https://codepen.io/ioc-daw-m06/pen/gPqyMa?editors=1000).

Fixeu-vos que l'element `input` conté com a atribut `list` l'identificador de l'element `datalist`.

### Tipus de l'element 'input': l'atribut 'type'

Encara que es tracti només d'un atribut, les diferències entre seleccionar un tipus o un altre són tan importants que cal tractar-les en una secció a part.

D'una banda, podem definir dos grans grups:

- Els que tracten amb text; com serien el tipus `text`, `password` o `email`.
- Els que afegeixen un component completament diferent; com són una casella de selecció (`checkbox`), un botó de selecció única (`radio`) o un botó per afegir fitxers (`file`).

D'altra banda, amb HTML5 es van afegir una gran quantitat de components amb noves funcionalitats, com el tipus `time` per introduir hores; `date` per introduir dates; `email`, que es tracta com un camp de text però valida automàticament que sigui una adreça de correu vàlida, o `range`, que mostra una barra per seleccionar un valor entre un mínim i un màxim.

En la llista següent, trobareu alguns dels diferents tipus que admeten tots els navegadors:

- **button**: el camp es tracta com un botó sense cap comportament per defecte.
- **submit**: es mostra com un botó però el seu comportament per defecte és enviar el formulari en fer clic.
- **reset**: en aquest cas també es mostra un botó, i el comportament per defecte és restablir tots els camps del formulari.
- **image**: en cas de voler fer servir una imatge personalitzada en lloc d'un botó d'enviament es pot fer servir aquest tipus.
- **text**: quadre de text bàsic per introduir qualsevol tipus de text; es mostra en una sola línia.
- **password**: és com el tipus text, però no mostra els caràcters introduïts per pantalla.
- **hidden**: els camps d'aquest tipus no són visibles a la pàgina, tot i que el seu valor s'enviarà juntament amb el formulari.
- **checkbox**: el camp serà una casella de selecció.
- **radio**: afegeix un botó de selecció única que només permet seleccionar un dels valors; si se'n selecciona un altre, es desselecciona l'anterior. Es considera que **tots els botons de selecció única que tenen la propietat name igual pertanyen al mateix grup**.
- **file**: afegeix un component d'enviament de fitxers que consta d'un botó i una etiqueta que indica el fitxer (o fitxers en HTML5) que s'enviaran juntament amb el formulari.

#### Camps ocults (hidden)

Els camps ocults ens permeten afegir informació que s'ha d'enviar amb el formulari, però que no ha de ser modificada per l'usuari. Per exemple, en un formulari per demanar més informació d'un producte que es trobi en una pàgina cal incloure l'identificador únic del producte com a valor ocult, ja que aquest no ha de ser modificat per l'usuari.

Recordeu que per poder enviar fitxers juntament amb el formulari, cal fer servir el mètode POST i l'encype ha de ser multipart/form-data.

L'estil de l'element de tipus file **no es pot modificar mitjançant CSS**; ja que la implementació d'aquest component no forma part de l'especificació d'HTML i cada fabricant la fa d'una manera diferent. Les solucions per donar format a aquest element impliquen implementar el nostre propi component en JavaScript o fer servir biblioteques externes.

Vegeu, en aquest exemple, l'aspecte de cadascun d'aquests tipus:

```
1 <form enctype="multipart/form-data" method="POST" action="test.php">
2   <input type="button" value="No fa res" /><br>
3   <input type="submit" value="Envia el formulari" /><br>
4   <input type="reset" value="Restableix el formulari" /><br>
5   <input type="text" value="Camp de tipus text" /><br>
6   <input type="password" value="Camp de tipus password" /><br>
7   <input type="hidden" value="Camp ocult" /><br>
8   <label>
```



```

9     <input type="checkbox" />Tipus checkbox<br>
10  </label>
11  <label>
12     <input type="radio" name="exemple"/>Tipus radio 1<br>
13  </label>
14  <label>
15     <input type="radio" name="exemple"/>Tipus radio 2<br>
16  </label>
17  <input type="file" />
18 </form>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/dGaBdj?editors=1000](https://codepen.io/ioc-daw-m06/pen/dGaBdj?editors=1000).

El resultat que obtindreu serà similar al de la figura 2.2.

**FIGURA 2.2.** Representació d'elements 'input' d'HTML

La imatge mostra una representació visual dels elements HTML 'input' definits en el codi. A la part superior, hi ha tres botons: 'No fa res', 'Envia el formulari' i 'Restableix el formulari'. Just a sota, hi ha un camp de text etiquetat 'Camp de tipus text' amb punts de textura. A continuació, hi ha tres elements de selecció única: un casella de selecció (checkbox) etiquetada 'Tipus checkbox', i dos botons de selecció única (radio) etiquetats 'Tipus radio 1' i 'Tipus radio 2'. A la part inferior, hi ha un camp de text etiquetat 'Navega...' i un missatge d'estat: 'No s'ha seleccionat cap fitxer.'

Podeu comprovar que com que els dos botons de selecció única tenen el mateix nom (exemple), només n'hi pot haver un de seleccionat en un moment donat.

Amb HTML5 es van afegir molts més tipus, amb funcionalitats que abans requerien ser programades pel nostre compte en JavaScript o fer servir llibreries externes. Així, s'ha de tenir en compte que no existeixen en navegadors antics o que fins i tot poden no ser compatibles amb alguns dels navegadors actuals. Per aquesta raó, si voleu que funcioni correctament en tot tipus de navegadors, s'ha de proporcionar a l'usuari una alternativa.

#### Incongruències entre l'atribut 'value' i el contingut d'un camp

En alguns casos ens trobarem que hi ha discrepàncies entre l'un i l'altre. Això és a causa del fet que l'especificació d'HTML indica un format que és el que s'utilitza internament, mentre que a la representació del navegador se n'utilitza una altra de basada en la configuració de llenguatge i idioma. Per exemple, ens trobem amb aquests casos quan treballem amb nombres reals i dades.

En la següent llista, trobareu alguns dels diferents tipus que no admeten els navegadors antics:

- **email:** obliga a introduir una adreça de correu vàlida per enviar el formulari.
- **number:** només permet escriure nombres, i per passar la validació cal que sigui un nombre vàlid.
- **url:** és un quadre de text que obliga a introduir una adreça URL vàlida,

Alguns dels nous tipus són versions modificades dels originals, que afegeixen capacitats de validació, com per exemple el tipus `email`.

incloent-hi el protocol.

- **range:** aquest tipus mostra una barra per seleccionar el valor desplaçant el marcador. Es pot configurar fent servir els atributs `max`, `min` i `step` per indicar els valors màxims, mínims i els increments respectivament.
- **date:** serveix per introduir dates, i pot mostrar un selector de dates que ens permet seleccionar-les en lloc d'afegir-les manualment.
- **time:** és similar a l'anterior però per introduir hores i minuts; en aquest cas no es mostra cap component extra per seleccionar-les.
- **color:** permet seleccionar un color de dintre d'una paleta de colors.

Vegeu en aquest exemple l'aspecte de cadascun d'aquests tipus:

```

1 <form>
2   <label>Tipus email:
3     <input type="email" value="test@exemple.cat"/>
4   </label><br>
5   <label>Tipus number:
6     <input type="number" value="42"/>
7   </label><br>
8   <label>Tipus url:
9     <input type="url" value="http://ioc.xtec.cat"/>
10  </label><br>
11  <label>Tipus range:
12    <input type="range" value="42" min="1" max="100" step="2"/>
13  </label><br>
14  <label>Tipus date:
15    <input type="date" value="1963-11-22"/></label><br>
16  <label>Tipus time:
17    <input type="time" value="08:15"/></label><br>
18    <input type="submit" />
19 </form>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/vLboBw?editors=1000](https://codepen.io/ioc-daw-m06/pen/vLboBw?editors=1000).

Fixeu-vos en el camp de tipus `date`: tot i que el valor s'ha afegit amb el format `aaaa-mm-dd` (el corresponent a l'especificació d'HTML), el format que es mostra al navegador és `dd/mm/aaaa`, el corresponent a la nostra llengua. Una cosa semblant passa amb els decimals: en el codi hem de fer servir el punt com a separador, mentre que per introduir les dades es fa servir la coma.

A la figura 2.3 podeu veure un exemple de representació de diferents elements afegits a HTML5.

**FIGURA 2.3.** Representació d'elements input d'HTML5

Tipus email:

Tipus number:

Tipus url:

Tipus range:

Tipus date:

Tipus time:

Tot i que l'element `input` accepta més tipus, en aquests materials ens hem limitat a enumerar els que s'utilitzen més habitualment.

### 2.2.4 Element 'textarea'

Aquest element té un funcionament similar al d'un element `input` de tipus text, però la diferència principal és que aquest és multilíneal, fet que permet escriure extensions més grans de text i incloure salts de línia.

Primerament, veurem algunes diferències de funcionament entre aquest element i l'element `input`. A diferència d'aquest últim, el `textarea` no s'autotanca: s'ha d'afegir l'etiqueta de tancament, i el valor del camp és el que es troba entre l'etiqueta d'obertura i de tancament, per exemple:

```
1 <script>
2   let textarea = document.getElementById('test');
3
4   function mostrarValor() {
5     alert(textarea.value);
6   }
7 </script>
8
9 <textarea id="test">Aquest es el valor del camp</textarea>
10 <br>
11 <button onclick="mostrarValor();">Mostrar valor</button>
```

Fixeu-vos que el valor inicial s'obté del text contingut entre les etiquetes d'apertura i tancament del `textarea`, però una vegada que el modifiqueu, el valor que es mostra en clicar el botó és el que conté el `textarea`.

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/LGoowV?editors=1010](https://codepen.io/ioc-daw-m06/pen/LGoowV?editors=1010).

Vegem-ne ara un altre exemple molt semblant:

```
1 <script>
2   let textarea = document.getElementById('test');
3
4   function mostrarValor() {
5     alert(textarea.value);
6   }
7 </script>
8
9 <textarea id="test" value="Aquest es el valor del camp"></textarea>
10 <br>
11 <button onclick="mostrarValor();">Mostrar valor</button>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/XXwwwQ?editors=1011](https://codepen.io/ioc-daw-m06/pen/XXwwwQ?editors=1011).

En aquest cas hem establert el mateix text que abans com a valor de la propietat del `textarea`, però **tant el `textarea` com el missatge d'alerta mostrat pel**

**navegador són buits.** És a dir, si ho establim com a valor des de la declaració HTML, aquest valor és ignorat.

Però què passa si intentem establir el valor del `textarea` per codi?

```
1 <script>
2   let textarea = document.getElementById('test');
3
4   textarea.value = "Valor establert via codi";
5
6   function mostrarValor() {
7     alert(textarea.value);
8   }
9 </script>
10
11 <textarea id="test"></textarea>
12 <br>
13 <button onclick="mostrarValor();">Mostrar valor</button>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/vLwqNg?editors=1010](https://codepen.io/ioc-daw-m06/pen/vLwqNg?editors=1010).

Com es pot veure, es mostra el text introduït per codi. Per tant, podem concloure que:

- Si necessitem afegir un text per defecte a un `textarea` des de l'HTML, ho hem de fer entre les etiquetes d'apertura i tancament.
- Si ho hem de fer mitjançant codi, ho farem a través de la propietat `value`.

Finalment ens queda un dubte per resoldre: com reacciona el `textarea` en establir el seu contingut amb la propietat `innerHTML`?

```
1 <script>
2   let textarea = document.getElementById('test');
3
4   textarea.innerHTML = "Valor establert via codi amb innerHTML";
5
6   function mostrarValor() {
7     alert(textarea.value);
8   }
9 </script>
10
11 <textarea id="test" value="Aquest és el valor del camp"></textarea>
12 <br>
13 <button onclick="mostrarValor();">Mostrar valor</button>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/xZNoEG?editors=1010](https://codepen.io/ioc-daw-m06/pen/xZNoEG?editors=1010).

L'efecte és similar, però s'ha de tenir en compte una diferència important: si inspeccionem l'element `textarea` amb les eines de desenvolupador, veurem que si establim l'atribut `value` d'aquest element el codi HTML inspeccionat no ha canviat, continua buit. En canvi, si ho fem mitjançant `innerHTML`, en inspeccionar l'element veurem que el valor és entre les etiquetes del `textarea` perquè amb `innerHTML` **es modifica l'estructura del document**.

Tot i que la funció del `textarea` és la mateixa que la del `input` de tipus `text`, té unes diferències de funcionament importants que s'han de tenir en compte.

Podem canviar el contingut d'un `textarea` tant a través de la propietat `value` com de la propietat `innerHTML`, però **es recomana fer servir** `value`, ja que és més coherent amb el tractament de dades, mentre que `innerHTML` en modifica l'estructura.

El `textarea` compta amb moltes menys opcions que l'element `input`, ja que aquest element apleix una funció molt específica. Tot i així s'ha afegit noves propietats a HTML5 i cal diferenciar-les; si més no, per ser conscient de quines propietats no són compatibles amb navegadors antics. Vegem, primer, les propietats de les versions anteriors:

- **id**: idèntic que en el cas de l'element `input`. Ens permet identificar-lo unívocament i enllaçar-lo amb elements `label`.
- **name**: igual que en l'element `input`, aquest és el nom del paràmetre que s'enviarà amb el formulari.
- **value**: aquesta propietat no té cap efecte si l'apliquem com a etiqueta HTML, en canvi, ens permet obtenir el contingut del `textarea` quan hi accedim des de JavaScript.
- **disabled** (booleà): quan s'aplica aquest atribut el contingut no serà modificable, es presentarà amb un estil diferent, i els seus continguts no s'enviaran amb el formulari.
- **readonly** (booleà): similar a l'anterior, però en aquest cas les dades sí que són enviades.
- **cols**: especifica l'amplada de l'àrea de text en caràcters, de manera similar al funcionament de l'atribut `size` amb els camps de text.
- **rows**: nombre de files de text que es veuran. S'ha de tenir en compte que encara que `cols` i `row` tracten amb l'aparença en lloc de l'estructura, no es pot fer servir CSS en aquest cas perquè CSS no treballa amb caràcters.
- **selectionStart**: indica l'índex del primer caràcter seleccionat.
- **selectionEnd**: indica l'índex de l'últim caràcter seleccionat. Això, combinat amb l'anterior propietat, ens permet capturar el valor de la selecció mitjançant JavaScript.

Aquest és un petit exemple de com capturar el text seleccionat en un `textarea`:

```
1 <script>
2   let textarea = document.getElementById('test');
3
4   function mostrarSeleccio() {
5     let seleccio,
6       start,
7       length;
```

#### Mètode `String.substr(start, length)`

El mètode `substr()`, dels objectes de tipus `String`, permet obtenir el fragment de la cadena que va des de la posició indicada pel primer paràmetre fins a aquesta posició més el valor del segon paràmetre. Per exemple: `"hola mon".substr(5,3)` retorna `mon`.

```

8
9     start = textarea.selectionStart;
10    length = textarea.selectionEnd - textarea.selectionStart;
11    seleccio = textarea.value.substr(start, length);
12    alert(seleccio);
13  }
14 </script>
15 <textarea id="test">Contingut de prova</textarea>
16 <br>
17 <button onclick="mostrarSeleccio();">Mostrar seleccio</button>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/XXwLVJ?editors=1010J](https://codepen.io/ioc-daw-m06/pen/XXwLVJ?editors=1010J).

D'altra banda, també podem fer servir JavaScript per fer la selecció d'un text, tal com podeu veure en aquest exemple: permet cercar dins del textarea i seleccionar el primer mot que coincideixi amb la cerca; vegem-ho:

Una altra opció per establir la selecció d'un textarea és fer servir el mètode `setSelectionRange(selectionStart, selectionEnd)`, proporcionat per la interfície de l'element `textarea`.

```

1 <script>
2   let text = document.getElementById('text'),
3       cerca = document.getElementById('mot');
4
5   function cercar() {
6
7     let mot = cerca.value,
8         start = text.value.indexOf(mot);
9
10    if (start > 0) {
11      text.selectionStart = start;
12      text.selectionEnd = start + mot.length;
13      text.focus(); //necessari en alguns navegadors (p.e. a Firefox) perquè
14                    es visualitzi la selecció
15    } else {
16      alert("No s'ha trobat el mot");
17    }
18  }
19 </script>
20 <label>Mot a cercar:
21   <input type="search" id="mot"/>
22 </label><button onclick="cercar();">cercar</button><br>
23 <textarea id="text" readonly cols="150" rows="7">--Ja era passat l'any e lo dia
    , e les festes eren complides de solemnizar, com la magestat del senyor
    rey tramés a preguar a tots los stats que·s volguessen esperar alguns dies
    per ço com la magestat sua volia fer publicar una fraternitat, la qual
    novament havia instituïda, de XXVI cavallers, sens que negú no fos
    reproche. E tots de bon grat foren contents de aturar. E la causa e
    principi de aquesta fraternitat, senyor, és stada aquesta, ab tota veritat
    , segons yo e aquests cavallers que aquí són havem hoïda reïtar per boca
    del senyor rey mateix: com, un dia de solaç que·s fehien moltes dançes e
    lo rey, havent dançat, restà per reposar al cap de la sala, e la reyna
    restà a l'altre cap ab les sues donzelles, e los cavallers dançaven ab les
    dames; e fon sort que una donzella, dançant ab un cavaller, apleguà fins
    en aquella part hon lo rey era e, al voltar que la donzella féu, caygué-li
    la liguacama de la calça, e al parer de tots devia ésser de la sinestra
    cama, e era de çimolça.
24 </textarea>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/BjeggG?editors=1010](https://codepen.io/ioc-daw-m06/pen/BjeggG?editors=1010).

Vegem ara les propietats afegides amb HTML5, molt similars a les que trobem a l'element `input`:

- **placeholder**: com en el cas de l'element `input`, afegeix un text per indicar què s'espera que s'introdueixi en aquest camp. Aquest text no s'envia amb el formulari, i si no s'introdueix res al `textarea`, s'enviarà buit.
- **required**: serveix per validar el camp i ens obliga a introduir-hi quelcom abans de poder enviar el formulari.
- **maxlength**: limita el nombre de caràcters que es poden introduir.
- **minlength**: estableix el nombre mínim de caràcters per poder enviar-lo. **No s'aplica si el camp es deixa buit**; si volem controlar que no sigui buit hem d'afegir també la propietat `required`.
- **form**: igual que en el cas de l'element `input` aquesta propietat permet associar el `textarea` a un formulari en el qual no està inclòs.

En aquest exemple podeu veure aquestes propietats en funcionament:

```
1 <form method="GET" action="test.php" id="principal">
2   <input type="submit" />
3 </form>
4 <textarea name="text" form="principal" minlength="5" maxlength="50" cols="100"
   rows="5" required>Text de prova que passa la validació</textarea>
```

Podeu veure aquest exemple en aquest enllaç: [codepen.io/ioc-daw-m06/pen/adreVV?editors=1010](https://codepen.io/ioc-daw-m06/pen/adreVV?editors=1010); i el resultat a la figura 2.4.

FIGURA 2.4. Formulari amb `textarea`



## 2.2.5 Element 'button'

Aquest element, com es pot deduir pel seu nom, serveix per representar botons; tal com els que es mostren amb l'element `input`, amb els tipus: `button`, `submit`, i `reset`.

En aquest cas el comportament per defecte de l'element és enviar el formulari, igual que fa el tipus `submit`; tot i que es pot especificar el tipus mitjançant les seves propietats, que són les següents:

- **name**: com en altres elements ja vistos, és el nom del paràmetre que s'enviarà amb el formulari.
- **id**: identificador únic habitual de tots els elements HTML.
- **type**: els botons poden ser de tres tipus: `submit`, `reset` i `button`, amb el comportament equivalent als mateixos tipus de l'element `input`.

En cas de fer servir l'element `button` fora de l'estructura de formulari, el comportament per defecte és ignorat.

- **value:** valor del botó que s'afegirà com a valor en enviar el formulari.

Fixeu-vos en aquest exemple: en fer clic al botó, com que no s'ha especificat cap tipus, es realitza l'enviament del formulari i, com podeu veure a la pestanya *Network* de les eines de desenvolupador, s'ha afegit el paràmetre `esborrar` amb el valor `true`:

```
1 <form method="GET" action="test.php">
2   <button name="esborrar" value="true">Esborrar</button>
3 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/KVLOeM?editors=1000](https://codepen.io/ioc-daw-m06/pen/KVLOeM?editors=1000).

En navegadors antics, **el funcionament de l'element** `button` **no sempre és correcte**, per aquesta raó és preferible fer servir l'element `input` amb un tipus de botó.

### 2.2.6 Element 'label'

La finalitat d'aquest element és purament semàntica, ja que no afecta l'enviament de les dades. Serveix per indicar clarament que el text que inclou acompanya un dels elements del formulari.

Aquest element es pot aplicar de dues formes:

- Afegint l'element `input` entre les etiquetes d'apertura i tancament:

```
1 <label>Etiqueta:
2   <input type="text" />
3 </label>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/zrQgmP?editors=1000](https://codepen.io/ioc-daw-m06/pen/zrQgmP?editors=1000).

- Fent servir l'atribut `for` i assignar com a valor l'id del camp pel qual serveix d'etiqueta:

```
1 <label for="camp1">Etiqueta:</label>
2 <input id="camp1" type="text" />
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/xZNvyW?editors=1000](https://codepen.io/ioc-daw-m06/pen/xZNvyW?editors=1000).



### 2.2.7 Element 'datalist'

Aquest element es va afegir amb HTML5 i serveix per definir una llista de valors que després pot ser utilitzada per l'element `input` especificant el nom de la llista a l'atribut `list`.

A diferència de les llistes generades amb l'element `select`, aquestes permeten introduir valors propis, i el valor no queda limitat als valors del `datalist`. Vegem-ne el funcionament:

```
1 <label>Selecciona un navegador de la llista:  
2   <input list="navegadors" name="meuNavegador" />  
3 </label>  
4 <datalist id="navegadors">  
5   <option value="Chrome">  
6   <option value="Firefox">  
7   <option value="Internet Explorer">  
8   <option value="Opera">  
9   <option value="Safari">  
10 </datalist>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/NxVQex?editors=1000](https://codepen.io/ioc-daw-m06/pen/NxVQex?editors=1000).

## 2.3 Modificació d'aparença i comportament

A l'hora de crear un formulari, s'ha de tenir en compte que molt rarament n'hi haurà prou amb l'aparença original. Aquests formularis estaran incrustats en una pàgina o aplicació que tindrà un estil ja definit per l'empresa i, per tant, el formulari haurà de fer servir els mateixos colors i estils.

D'altra banda, també serà habitual fer algun tipus de personalització del comportament, ja sigui fer una validació de dades abans d'enviar el formulari, fer alguns càlculs amb la informació que serà enviada o simplement actualitzar el valor d'alguns camps a partir dels canvis produïts en uns altres.

### 2.3.1 Modificació de l'aparença d'un formulari

Un formulari molt rarament es trobarà de forma independent, sempre serà part d'un altre lloc web o aplicació i, per aquest motiu, la seva aparença ha de concordar amb la resta de l'aplicació.

Aquesta concordança d'estil la farem fent servir CSS, raó per la qual segurament ja tindrem la major part de la feina feta.

És important recordar que, per agrupar els elements del formulari, en lloc de fer servir l'element `div` i etiquetar fent servir text directament o capçaleres, disposem d'elements com ara:

- **fieldset**: per agrupar els continguts en lloc de fer servir *divs*.
- **label**: per etiquetar els camps.
- **legend**: per fer servir com a capçalera dins d'un **fieldset**.
- **placeholder**: propietat per donar pistes als usuaris de com s'ha d'emplenar el formulari.

S'ha de tenir compte que si no mantenim amb un estil idèntic els elements `input` de tipus botó i els elements `button`, l'aparença pot resultar incoherent.

Malauradament, alguns dels elements que es fan servir per als formularis, com els botons i el tipus de camp `file`, no es poden personalitzar, ja que el CSS no s'aplica correctament. La seva implementació depèn del navegador i, si volem poder modificar-ne l'aparença, hem de recórrer a fer servir una biblioteca externa o hem de crear el nostre propi component.

En l'exemple següent, podeu veure com canvia l'aspecte d'un formulari quan s'hi aplica l'estil CSS. Primer, vegem el formulari sense aplicar-hi cap estil:

#### Fulls d'estil CSS

Podeu trobar més informació sobre els fulls d'estil CSS en aquest enllaç: [www.developer.mozilla.org/es/docs/Web/CSS](http://www.developer.mozilla.org/es/docs/Web/CSS).

```

1 <form>
2   <fieldset>
3     <legend>Connectar</legend>
4     <label>Usuari
5       <input type="text" name="usuari" />
6     </label>
7     <label>Contrasenya
8       <input type="password" name="password" />
9     </label>
10
11   <button>Connectar</button>
12 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/YwoPLR?editors=1000](https://codepen.io/ioc-daw-m06/pen/YwoPLR?editors=1000); i el resultat a la figura 2.5.

**FIGURA 2.5.** Formulari sense estils CSS

I ara, vegem el mateix formulari, però afegint-hi el següent codi CSS:

```

1 <style>
2   form {
3     margin: 20px auto;
4     width: 300px;
5   }
6
7   label {
8     text-transform: lowercase;
9     font-weight: bold;
10    font-size: small;
11    color: grey;
12    width: 100%;
13    display: block;
```

```

14     padding: 5px;
15 }
16
17 input {
18     width: 60%;
19     float: right;
20 }
21
22 button {
23     margin-top: 10px;
24     width: 100%;
25     height: 25px;
26 }
27
28 legend {
29     text-transform: uppercase;
30     font-style: italic;
31     background: azure;
32 }
33
34 fieldset {
35     background: azure;
36     border: 1px dotted ;
37 }
38 </style>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/yedyyy?editors=1100](https://codepen.io/ioc-daw-m06/pen/yedyyy?editors=1100); i el resultat a la figura 2.7.

**FIGURA 2.6.** Formulari amb estils CSS

### 2.3.2 Modificació del comportament d'un formulari

Quan treballem amb formularis i amb els elements relacionats (com són les llistes, els quadres de text i els botons), no sempre voldrem fer un enviament d'aquesta informació i, en molts casos, voldrem fer un tractament d'aquest abans d'enviar-la.

Potser hem de fer alguns càlculs amb la informació, formatar-la o validar-la abans de fer l'enviament. En qualsevol d'aquests casos, el que ens interessa és interrompre l'execució de l'enviament, comprovar o processar les dades i, si tot és correcte, continuar amb l'enviament.

A continuació podeu trobar un exemple en el qual es comprova si el nombre introduït és parell en enviar el formulari i, si no ho és, s'interromp l'enviament:

```

1 <script>
2   let principal = document.getElementById('principal'),
3     nombre = document.getElementById('nombre')
4
5   principal.addEventListener('submit', event => {

```

```
6     if (nombre.value % 2 === 0) {
7
8         alert("Correcte, es continua amb l'enviament");
9
10    } else {
11
12        alert("El nombre ha de ser parell");
13        event.preventDefault();
14
15    }
16    });
17 </script>
18 <form action="test.php" method="GET" id="principal">
19     <label>Introdueix un numero parell:<input type="number" name="nombre" id =
        nombre" required/>
20 </label>
21     <input type="submit" />
22 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/GobJzN?editors=1010](https://codepen.io/ioc-daw-m06/pen/GobJzN?editors=1010).

Si us fixeu en el panell *Network* de les eines de desenvolupador, podreu comprovar que, en efecte, si el nombre no és parell, no es produeix l'enviament, perquè `event.preventDefault()` evita el comportament per defecte de l'*event*, que en aquest cas és l'enviament.

D'aquesta manera podem evitar que es produeixi l'enviament del formulari fins que es compleixin totes les condicions necessàries.

Cal recordar que en el cas dels elements `form`, `preventDefault()` és un **mètode clau si estem treballant amb AJAX**, perquè ens permet aturar l'enviament de la informació i invocar, en el seu lloc, les funcions que ens permeten fer la petició asíncrona.

Un altre exemple d'ús interessant és preprocessar la informació que s'enviarà al servidor; vegem-ho:

```
1 <script>
2     let principal = document.getElementById('principal'),
3         nom = document.getElementById('nom'),
4         cognom = document.getElementById('cognom'),
5         nomcomplet = document.getElementById('nomcomplet')
6
7     principal.addEventListener('submit', event => {
8         nomcomplet.value = (nom.value + ' ' + cognom.value).toLowerCase();
9     });
10 </script>
11 <form action="test.php" method="GET" id="principal">
12     <label>Nom:<input type="text" name="nom" id="nom" required/>
13 </label>
14     <label>Cognom:<input type="text" name="cognom" id="cognom" required/>
15 </label>
16     <input type="hidden" name="nomcomplet" id="nomcomplet"/>
17     <input type="submit" />
18 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/jWjPjx?editors=1010](https://codepen.io/ioc-daw-m06/pen/jWjPjx?editors=1010).

En aquest cas hem afegit un camp ocult que originalment no conté res; però, abans d'executar l'enviament, es processa la informació del nom i el cognom per crear el `nomcomplet`, que està compost per la concatenació del nom i el cognom amb totes les lletres en minúscula.

No obstant això, fer servir els botons de tipus `submit` no és l'única manera de provocar l'enviament de dades; també es pot fer manualment, cridant el mètode `submit` del formulari. Per exemple, en fer clic sobre un element personalitzat per emular el funcionament d'un botó:

```
1 <style>
2   #botoFals {
3     display: inline;
4     border: 2px solid black;
5     border-radius: 15px;
6     padding: 5px;
7     font-style: italic;
8     background: azure;
9   }
10 </style>
11 <script>
12   let principal = document.getElementById('principal'),
13       boto = document.getElementById('botoFals');
14
15   boto.addEventListener('click', event => principal.submit());
16
17 </script>
18 <form action="test.php" method="GET" id="principal">
19   <input type="text" name="nom" />
20   <div id="botoFals">
21     Enviar!
22   </div>
23 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/vLqNEX](https://codepen.io/ioc-daw-m06/pen/vLqNEX).

Com es pot veure, hem afegit un *listener* a l'element `div` per detectar quan es clica, i una vegada es dispara l'*event* `click` fem l'enviament del formulari cridant el mètode `submit`.

## 2.4 Validació

Un dels primers usos que es va donar a JavaScript va ser el de validar la informació del navegador abans d'enviar els formularis. Imagineu els problemes que suposava enviar un formulari amb informació errònia o incompleta el 1995, amb una velocitat de connexió de 28,8 kbit/s com a màxim i pagant pel temps de connexió. Aquest formulari s'havia d'enviar al servidor; allà es comprovava si era correcte i, en cas contrari, es retornava al client (el navegador de l'usuari) que havia de corregir-lo i tornar-lo a enviar fins que finalment passava la validació.

Cal remarcar que no es pot confiar la validació de les dades al client, les dades **també han de ser validades al servidor**. Tot i així, el fet de poder validar-les abans de fer l'enviament, ens estalvia temps i consum de dades; temps, perquè

no s'ha de comunicar amb el servidor per realitzar la validació; i consum de transferència en els plans de dades, si ens connectem mitjançant una xarxa de dades com 4G.

**Les dades sempre s'han de validar i sanejar al servidor**, ja que no podem comptar que la informació que ens arribi sigui sempre correcta. Sense aquests controls, seria molt fàcil que un usuari maliciós pogués accedir-hi i manipular o corrompre les dades del servidor.

### 2.4.1 Validació d'HTML5

Amb HTML5 es van afegir moltes millores als elements que formen els formularis. Gairebé tots els elements inclouen propietats que permeten fer validacions simples sobre les dades entrades, sense necessitat de codificar res.

A les especificacions anteriors d'HTML **no s'inclouia la capacitat de validar les dades automàticament**, l'única limitació que es podia afegir era la llargària d'una entrada de text. Aquest fet obligava a fer servir JavaScript per realitzar qualsevol tipus de validacions.

Vegem una llista d'exemples de com aplicar aquestes propietats per afegir la validació als nostres formularis:

- Fer que un camp sigui obligatori; hem d'afegir-hi la propietat `required`:

```
1 <form action="test.php" method="GET">
2   <input type="text" placeholder="Aquest camp no pot ser buit" size="50"
   required>
3   <button>Enviar</button>
4 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/QtyXEdV?editors=1000](https://codepen.io/ioc-daw-m06/pen/QtyXEdV?editors=1000).

- Afegir una validació de correu electrònic, el tipus ha de ser `email`; vegem-ho:

```
1 <form action="test.php" method="GET">
2   <input type="email" placeholder="Aquest camp ha de ser un email correcte"
   size="50" required>
3   <button>Enviar</button>
4 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/vLqKxx?editors=1000](https://codepen.io/ioc-daw-m06/pen/vLqKxx?editors=1000).

- Acceptar només nombres enters:

```
1 <form action="test.php" method="GET">
2   <input type="number" placeholder="Només nombres enters" required>
3   <button>Enviar</button>
4 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/rxElyE?editors=1000](https://codepen.io/ioc-daw-m06/pen/rxElyE?editors=1000).

- Acceptar també nombres reals:
  - Només haurem d'afegir l'atribut `step` amb el valor que representarà la diferència entre un nombre i el següent. Per exemple, si volem només una precisió d'un decimal, hi afegirem `step="0.1"`:

```
1 <form action="test.php" method="GET">
2   <input type="number" placeholder="Accepta nombres reals" required step="0.1">
3   <button>Enviar</button>
4 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/RrzRVV?editors=1000](https://codepen.io/ioc-daw-m06/pen/RrzRVV?editors=1000).

- Establir un valor mínim i màxim (contingut numèric):
  - Continuant amb els exemples numèrics, ens trobem que en molts casos necessitarem establir un valor mínim i un de màxim. Això també es pot fer mitjançant les propietats de l'element `input`, en aquest cas amb `max` i `min`:

```
1 <form action="test.php" method="GET">
2   <input type="number" placeholder="Accepta nombres reals" required step="0.1"
3     min="1" max="42">
4   <button>Enviar</button>
5 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/NxZrjz?editors=1000](https://codepen.io/ioc-daw-m06/pen/NxZrjz?editors=1000).

- Establir un valor mínim i màxim (cadena de text):
  - Seria possible aplicar els valors `max` i `min` a un camp de text? Per exemple, per requerir que una contrasenya tingui com a mínim 5 caràcters? Vegem-ho:

```
1 <form action="test.php" method="GET">
2   <input type="password" placeholder="Entre 5 i 10 caràcters" required min="5"
3     max="10">
4   <button>Enviar</button>
5 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/EPByXa?editors=1000](https://codepen.io/ioc-daw-m06/pen/EPByXa?editors=1000).

- No funciona, els valors de min i max només s'apliquen a continguts numèrics. Si volem treballar amb cadenes de text, hem de fer servir les propietats `minlength` i `maxlength`:

```
1 <form action="test.php" method="GET">
2   <input type="password" placeholder="Entre 5 i 10 caracters" required
      minlength="5" maxlength="10">
3   <button>Enviar</button>
4 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/yedJXw?editors=1000](https://codepen.io/ioc-daw-m06/pen/yedJXw?editors=1000).

### Validació per 'pattern'

La propietat de validació més potent és `pattern`, introduïda a HTML5 i, per consegüent, no disponible en els navegadors més antics. Aquesta propietat ens permet introduir una expressió regular que es farà servir per validar el contingut del camp. Per exemple, si l'entrada de text ha de contenir només una paraula i ha de començar per majúscula, podríem fer-ho així:

Les *expressions regulars* es troben detallades a la secció "Expressions regulars" d'aquest mateix apartat.

```
1 <form action="test.php" method="GET">
2   <input type="text" placeholder="Només una paraula que comenci per majúscula"
      required size=50 pattern="^\b[A-Z]\w*\b$" >
3   <button>Enviar</button>
4 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/YwoWOx?editors=1000](https://codepen.io/ioc-daw-m06/pen/YwoWOx?editors=1000).

Si proveu aquest exemple, veureu que només us deixa enviar el formulari si es compleix la condició indicada per l'expressió regular, que es pot interpretar de la manera següent:

- Començant al principi del text: `^`
- Fins al final del text: `$`
- Ha de contenir només una paraula, que és envoltada per `\b` que marca l'inici i el final
- El primer caràcter ha d'estar comprès entre A i Z, ambdós inclosos: `[A-Z]`
- A continuació pot haver-hi qualsevol quantitat (0 o més) de lletres i números: `\w*`



## 2.4.2 Validació per a JavaScript

Tot i que HTML5 ens permet validar els camps en una gran quantitat de situacions, pot ocórrer que necessitem programar les nostres pròpies validacions; per exemple, en el cas d'haver de donar suport a navegadors antics o quan tractem amb un tipus de validació més complexa.

Detectant quan es dispara l'*event* submit de l'element form, és possible aturar l'enviament d'un formulari i afegir les nostres pròpies validacions. Vegem una variació per simplificar el codi, afegint directament el nom de la funció que serà invocada en detectar-se aquest *event*:

```
1 <form action="test.php" method="GET" onsubmit="return validar();">
2   <input name="dni" id="dni" type="text" minlength="9" maxlength="9" size="9"
3     placeholder="DNI" pattern="\d{8}[a-zA-Z]"/>
4   <button>Enviar</button>
5 </form>
6
7 <script>
8   function validar() {
9     let dni = document.getElementById('dni');
10    return comprovarDni(dni.value);
11  }
12
13  function comprovarDni(dni) {
14    let nombre,
15        lletraIntroduïda,
16        lletraEsperada,
17        correcte = false;
18
19    nombre = dni.substr(0, dni.length - 1); // Extraïem el nombre
20    lletraIntroduïda = dni.substr(dni.length - 1, 1); // Extreiem la lletra
21    nombre = nombre % 23;
22    lletraEsperada = 'TRWAGMYFPDXBNJZSQVHLCKET'; // Possibles valors de la
23      lletra ordenats per la posició corresponent
24    lletraEsperada = lletraEsperada.substring(nombre, nombre + 1);
25
26    if (lletraEsperada === lletraIntroduïda.toUpperCase()) {
27      correcte = true;
28      alert('DNI correcte: ' + dni);
29    } else {
30      alert('DNI incorrecte: ' + dni);
31    }
32
33    return correcte;
34  }
35 </script>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/XXLKGX?editors=1010](https://codepen.io/ioc-daw-m06/pen/XXLKGX?editors=1010).

Com es pot apreciar, a l'element input del bloc HTML s'ha especificat:

- La mida mínima ha de ser 9 caràcters (`minlength`).
- La mida màxima ha de ser 9 caràcters (`maxlength`).
- El valor s'ha de correspondre amb una expressió regular (`pattern`) que requereix 8 nombres `\d{8}` seguits d'una lletra en minúscules o majúscules `[a-zA-Z]`.

Com que la lletra corresponent a cada DNI es pot calcular en funció dels nombres, és possible comprovar que el DNI introduït sigui correcte abans d'enviar el formulari. Per afegir aquesta comprovació només cal afegir el nom de la funció encarregada de la validació a la detecció de l'*event* submit del formulari perquè aquesta funció sigui invocada quan el formulari sigui enviat.

Fixeu-vos que en lloc de simplement invocar la funció validar en disparar-se l'*event* submit, el que hem fet és retornar el valor d'aquesta funció: `onsubmit="return validar();"` . Si aquest valor és `true`, el formulari s'enviarà, i si és `false`, s'aturarà l'enviament. Aquesta és una alternativa a invocar el mètode `preventDefault()` per aturar el comportament per defecte.

#### Validació del DNI

Podeu trobar més informació sobre com es calcula la lletra del DNI en el següent enllaç: [www.goo.gl/VrlzYQ](http://www.goo.gl/VrlzYQ).

Fer la validació de la lletra és més complicat, així que per fer-ho hem de recórrer a JavaScript. En aquest exemple hem invocat la funció `validar()`, cosa que ens permetria centralitzar la validació i afegir més comprovacions, si fos necessari. Només s'ha de vigilar que ha de retornar `true`, si es passen totes les comprovacions, o a `false`, si alguna falla.

Fixeu-vos que la funció `validar()` retorna el valor de la funció `comprovarDni()`, ja que en aquest cas només ha de fer una comprovació. El procés a seguir per determinar si s'ha de retornar cert o fals és el següent:

1. Separem la part de la cadena que correspon als nombres i la que correspon a la lletra.
2. Calculem quina és la lletra que hi hauria de correspondre.
3. Si la lletra introduïda correspon amb l'esperada, llavors és correcta. Fixeu-vos que fem la comprovació del valor en majúscules, així és indiferent si l'hem introduït d'una manera o de l'altra, ja que la lletra esperada és majúscula.
4. Si és correcte, retornarem `true`; si no, retornarem `false`.

Un altre cas que us podeu trobar és haver de validar la mida màxima d'un o més fitxers abans de fer l'enviament al servidor. Aquest cas és molt interessant perquè, en cas contrari, no sabríeu si s'ha superat la mida màxima fins que falli el servidor, cosa que pot provocar llargues esperes abans de conèixer l'error. Vegem-ho:

#### Web API File

La Web API que permet obtenir la informació dels fitxers pertany a HTML5. Podeu trobar-ne més informació a l'enllaç següent: [www.goo.gl/7dtSKT](http://www.goo.gl/7dtSKT).

```

1 <script>
2   let fitxers = document.getElementById('fitxers'),
3     MAX_MIDA_FITXERS = 2048; // Mida màxima en bytes
4
5   function validarFitxers() {
6     let fitxer, midaTotal;
7
8     if (fitxers.files.length > 0) {
9
10      midaTotal = 0;
11
12      for (let i = 0; i < fitxers.files.length; i++) {
13        fitxer = fitxers.files[i];
14        console.log("Fitxer: " + fitxer.name + " te una mida de " + fitxer.size
15          + " bytes");
16        midaTotal += fitxer.size;
17      }
18    }
19  }

```

```
17 console.log("Mida total dels fitxers: ", midaTotal);
18
19
20 if (midaTotal > MAX_MIDA_FITXERS) {
21     fitxers.value = [];
22     alert("La mida màxima dels fitxers és " + MAX_MIDA_FITXERS +
23         " bytes, però els fitxers seleccionats tenen una mida de " +
24         midaTotal + " bytes.");
25 }
26 }
27 };
28 </script>
29 <form action="test.php" method="POST" enctype="multipart/form-data">
30     <input type="file" id="fitxers" name="fitxer[]" multiple="true" onchange="
31         validarFitxers();">
32     <button>Enviar</button>
33 </form>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/obrzEd?editors=1011](https://codepen.io/ioc-daw-m06/pen/obrzEd?editors=1011).

En aquest exemple, en lloc de fer la validació en enviar el formulari, l'hem afegit a l'*event* *change* de l'element *input*. En el cas dels fitxers, l'*event* *change* es produeix sempre tan bon punt es tanca el diàleg de selecció de fitxers, així que és el punt ideal per fer la validació. Tampoc no cal retornar cap valor; si la mida no és correcta, el que fem és buidar la selecció establint l'*array* de fitxers a un *array* buit.

---

MAX\_MIDA\_FITXERS és una variable definida per nosaltres que ens facilita ajustar la mida segons els requeriments, en lloc de fer servir un nombre. La posem en majúscules perquè la seva funcionalitat coincideix amb la de les constants d'altres llenguatges.

---

Com que és un exemple per a HTML5 es pot fer servir l'atribut *multiple* de l'element *input*, cosa que ens permet fer una selecció múltiple de fitxers. Fixeu-vos que en aquest cas l'atribut *name* és un *array*: això permet enviar tota la llista de fitxers seleccionats en lloc d'un de sol.

En el codi de la funció *validarFitxers()*, veureu que primer hem afegit una sèrie de missatges que es mostren per consola per facilitar el seguiment de què està passant. Aquests missatges, però, no són necessaris per al bon funcionament del programa. El procés seria el següent:

1. El primer pas és comprovar si la mida de l'*array* de fitxers *file* (exposat per la Web API de fitxers) conté algun element.
2. Si és així, inicialitzem la mida total a 0 i recorrem l'*array*.
3. Mostrem la informació de cada fitxer per la consola i afegim la mida al total mitjançant la seva propietat *size*.
4. Una vegada els hem recorregut tots, mostrem per la consola la mida total.
5. Procedim a comprovar si la mida és superior al límit (en el nostre cas, 2.048 bytes) i, si és així, buidem el contingut de l'element *input* que conté els fitxers, i mostrem l'alerta.

Les bases per realitzar validacions directament amb JavaScript són molt simples. Això sí, s'ha de tenir sempre a mà una bona pàgina de consulta com [www.developer.mozilla.org](https://www.developer.mozilla.org), sobretot quan treballem amb les Web API, ja que poden oferir-nos possibilitats que van més enllà de l'especificació bàsica d'HTML.

## 2.5 Expressions regulars

Es pot fer servir l'atribut `pattern` dels elements `input` per realitzar validacions, aplicant una expressió regular. En aquesta secció veurem què són exactament les expressions regulars i com s'ha d'utilitzar-les.

### Comprovador d'expressions regulars en línia

En molts casos pot ser interessant comprovar que l'expressió regular que volem aplicar és correcta. Per comprovar-ho existeixen eines en línia com *Regex Pal* ([www.regexpal.com](http://www.regexpal.com)). Aquest web permet introduir una expressió regular i un text, de manera que podem comprovar si la selecció es correspon amb el que s'esperava.

L'ús de l'atribut `pattern` dels elements `input` es tracta a la secció "Validació d'HTML5" d'aquest apartat.

### Expressions regulars per a JavaScript

Es pot trobar més informació sobre totes les opcions que ofereixen les expressions regulars per a JavaScript a l'enllaç següent: [www.google.com/p3Ejc3](http://www.google.com/p3Ejc3).

Una expressió regular és una seqüència de caràcters que defineixen un patró de cerca. Aquest patró es pot utilitzar en operacions del tipus *cerca i reemplaçar*, oferint inclús capacitat de captura de grups per realitzar uns reemplaçaments molt potents.

Es poden trobar processadors d'expressions regulars tant en eines dels sistemes operatius Unix i Linux, com en editors i processadors de textos. A banda de les aplicacions externes, pràcticament tots els llenguatges de programació ofereixen la possibilitat d'utilitzar-les. Així que encara que a primera vista siguin inintel·ligibles, dominar-les ens pot facilitar molt la manipulació de cadenes de text, sigui quin sigui el llenguatge amb el qual treballem.

### Exemple d'usos de les expressions regulars

A banda de validar dades, a les expressions regulars se'ls poden donar molts altres usos, per exemple, els següents:

- Normalitzar textos: eliminar espais en blanc duplicats, salts de línia erronis (la següent línia comença en minúscula), etc.
- Canviar els formats: substituir cometes normals per tipogràfiques, que requereixen conèixer on comença i on acaba l'element.
- Convertir un text d'un tipus en un altre: extreure la informació d'una consulta SQL i convertir-la en una cadena de text en format JSON.
- Realitzar cerques de fitxers amb patrons concrets (utilitats del sistema operatiu Unix/Linux)
- Creació de *parsers* per extreure dades de fitxers amb formats coneguts de text pla (XML, JSON...) i convertir-les en objectes per poder manipular-los.
- Creació d'interprets de nous llenguatges de programació.
- Creació de plantilles on se substituiran valors clau pels valors de les variables que ens interessin.

En el següent exemple podeu veure com es fa servir un patró molt simple per eliminar tots els salts de línia i espais de sobra al text original, i reemplaçar-los per un sol espai:

```
1 <script>
2   let original = document.getElementById('original'),
3   net = document.getElementById('net');
```

```
4
5 function netejar() {
6     let text = original.innerHTML,
7     patro = /\s+/g;
8     net.innerHTML = text.replace(patro, ' ');
9 }
10 </script>
11 <pre id="original">>E en aquell
12 punt
13 se
14 féu deslligar
15 la çimolsa, que no la volgué més portar, ab molta malenconia que
16     li restà, emperò no·n
17 féu demostració alguna.
18 </pre>
19 <button onclick="netejar()">Netejar</button>
20 <pre id="net"></pre>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/qbzrQY?editors=1010](https://codepen.io/ioc-daw-m06/pen/qbzrQY?editors=1010).

Tot i que la majoria dels *metacaràcters* es poden trobar en totes les implementacions, en cas de dubte, és recomanable fer servir una eina com **RegexPal** ([www.regexpal.com](http://www.regexpal.com)) per comprovar que la nostra expressió funciona com esperem, o bé consultar la documentació especial. Per exemple, no tots els processadors d'expressions regulars ofereixen la capacitat de fer *backtracing*.

En aquests materials ens centrarem en la implementació de les expressions regulars per a JavaScript, tot i que no farem una anàlisi en profunditat de totes les seves opcions.

En JavaScript es poden crear expressions regulars com a representació literal:

```
1 let patro = /ioc/;
```

També es poden crear expressions regulars com a objecte RegExp, fent servir el constructor i passant l'expressió com a cadena de text:

```
1 let patro = new RegExp("ioc");
```

Es prefereix l'**ús de la creació literal** quan coneixem el patró a priori; mentre que l'**ús de l'objecte** és el mètode preferit quan l'expressió és desconeguda inicialment, per exemple, quan es crea a partir de les dades entrades per l'usuari.

Totes dues formes admeten opcions addicionals (*flags*), que s'afegeixen de la següent manera en la representació literal:

```
1 let patro = /ioc/flags;
```

I d'aquesta si fem servir el objecte:

---

Un *metacaràcter* és un caràcter o conjunt de caràcters que tenen un significat especial a l'expressió regular; per exemple `^` que coincideix la primera posició del text o `.` que coincideix amb qualsevol caràcter.

---

---

Les barres inicial i final d'una expressió regular no formen part del patró sino que són els delimitadors de l'expressió regular.

---

```
1 let patro = new RegExp("ioc", "flags");
```

Els **flags** formen part de l'expressió regular, i no poden afegir-se ni eliminar-se després de la seva creació. A JavaScript es poden utilitzar els següents:

- **g**: *Global*, cerca tots els resultats a la cadena de text; si no s'afegeix, es para la cerca quan troba la primera coincidència.
- **i**: *Case-insensitive*, no distingeix entre majúscules i minúscules.
- **m**: *Multi-line*, el metacaràcter de final del text coincideix amb el de salt de línia, per exemple, el següent patró coincideix amb el punt i a part de tots els paràgrafs d'un text: `/\.$/gm`

El caràcter `\` és un caràcter d'escapament; això vol dir que altera la interpretació del següent caràcter:

- Si es tracta d'un caràcter amb un significat especial, com `.`, es tractarà com un caràcter normal.
- Si es tracta d'un caràcter normal, i és interpretable com a especial com `b`, s'interpretarà segons la seva funció especial, en aquest cas com el delimitador de paraula `\b`.

Si no es fa servir cap metacaràcter de tipus quantificador, cada caràcter del patró només se cerca una vegada. Per exemple, `Adéu` valida `Adéu`, però no `AAAdéu`, el patró correcte per al segon cas seria `A+déu`.

Podeu trobar els metacaràcters més utilitzats a la taula 2.1.

**TAULA 2.1.** Taula de metacaràcters per a expressions regulars

Metac.	Descripció	Patró exemple	Vàlid	No vàlid
<code>^</code>	Coincideix només amb el començament de la cadena	<code>^a</code>	avió	campana
<code>\$</code>	Coincideix amb el final de la cadena	<code>a\$</code>	Samarreta	avió
<code>*</code>	Cerca el caràcter o grup precedent 0 o més vegades, equival a <code>{0,}</code>	<code>c*</code>	camió	avió
<code>+</code>	Cerca el caràcter o grup precedent 1 o més vegades, equival a <code>{1,}</code>	<code>c+</code>	colecció	camió
<code>?</code>	Aquest caràcter pot tenir diferents significats segons on es trobi. En situacions normals cerca el caràcter precedent 0 o 1 vegada, i equival a <code>{0,1}</code>	<code>bici?</code>	<b>bici, bic</b>	col
<code>.</code>	Coincideix amb tots els caràcters excepte salt de línia	<code>bi.i</code>	<b>bici, bili</b>	billi
<code>{n}</code>	Repeteix la cerca del caràcter o grup precedent exactament <i>n</i> vegades	<code>o{2}</code>	<b>oo, ooo</b>	o
<code>{n,m}</code>	Repeteix la cerca del caràcter o grup precedent entre <i>n</i> i <i>m</i> vegades	<code>o{2,4}</code>	<b>oo, ooo, oooo, ooooo</b>	o

TAULA 2.1 (continuació)

Metac.	Descripció	Patró exemple	Vàlid	No vàlid
( i )	Obren i tanquen un grup, el fragment contingut dins dels parèntesis es tracta com un bloc i pot ser capturat.	o(na){2}	<b>onana</b> , <b>onananana</b> , <b>onanaonana</b>	ona
[ i ]	Creem un joc de caràcters, el valor d'aquesta posició pot ser qualsevol dels que en formen part. Dins del joc de caràcters el guió - serveix per indicar un rang de caràcters, per exemple [A-Z] forma un joc de caràcters que inclou totes les lletres des de la A majúscula fins a la Z majúscula	coordenada[XYZ]	<b>coordenadaX</b> , <b>coordenadaY</b> , <b>coordenadaZ</b>	coordenadaB
[ i ] començant per ^	Si el joc de caràcters comença per ^ nega el joc de caràcters	coordenada[^XYZ]	<b>coordenadaB</b>	coordenadaY, coordenadaZ
\b	Coincideix amb el principi o final de la paraula	\bBarcelona	Som a <b>Barcelona</b>	AutoBacerlona
\d	Coincideix amb un dígit, equival a [0-9]	\d\d\d	<b>123</b> , <b>1234</b>	12
\D	L'invers a \d, equival a [^0-9]	\D\D\D	<b>abc</b>	123
\w	Coincideix amb un caràcter alfanumèric anglès, inclosa la barra baixa. Equival a [A-Za-z-0-9_]. <b>no inclou caràcters d'accentuació, ni propis d'altres llengües com la ç</b>	\w+	<b>pala_72</b>	.?!
\W	L'invers a \w, equival a [^A-Za-z-0-9_]	\W	42%	pala_72
\s	Coincideix amb els caràcters d'espai en blanc, incloent-hi tabulacions i salts de línia	\w\s\w	sense sortida	-
\S	L'invers a \s, coincideix amb tot el que no siguin espais en blanc	\w\S\w	<b>cotxe</b>	c o t x e

En negreta es mostra la coincidència del patró; cal fixar-se que sovint només és parcial.

## 2.5.1 Grups de captura

L'ús de parèntesis ens serveix per agrupar seleccions, que poden ser extreïdes o reemplaçades, dins del patró. A aquests grups se'ls coneix com a **grups de captura**.

Vegem-ne un exemple fent servir grups de captura. Cerquem totes les ocurrences de lo i afegirem les etiquetes <b> i </b> a cadascuna per ressaltar el canvi:

```

1 <script>
2   let original = document.getElementById('original'),
3     nou = document.getElementById('nou');
4
5   function canviar() {
6     let text = original.innerHTML,
```

```

7      patro = /(\\b[\\L]o\\b)+/g;
8      nou.innerHTML = text.replace(patro, '<b>$1</b>');
9  }
10 </script>
11 <p id="original">»Aprés diré a la senyoria vostra les cerimònies que s'an de
    fer en la capella. Ara diré los cavallers qui foren elets. Primerament, lo
    rey elegí XXV cavallers e, ab lo rey foren XXVI. Lo rey fon lo primer qui
    jurà de servir totes les ordinacions en los capítols contengudes e que no
    fos cavaller negú qui demanàs aquest orde que·l pogués haver. Tirant fon
    elet lo primer de tots los altres cavallers, per ço com fon lo millor de
    tots los cavallers. Aprés fon elet lo príncep de Gales, lo duch de
    Betafort, lo duch de Lencastre, lo duch d'Atçètera, lo marquès de Sófolch,
    lo marquès de Sant Jordi, lo marquès de Belpuig, Johan de Varoych, gran
    conestable, lo comte de Nortabar, lo comte de Salasberi, lo comte d'
    Estafort, lo comte de Vilamur, lo comte de les Marches Negres, lo comte de
    la Joyosa Guarda, lo senyor d'Escala Rompuda, lo senyor de Puigvert, lo
    senyor de Terranova, miçer Johan Stuart, miçer Albert de Riuçech. Aquests
    foren del regne. Los strangers foren: lo duch de Berri, lo duch d'Anjou,
    lo comte de Flandes. Foren tots en nombre de XXVI cavallers.
12 </p>
13 <button onclick="canviar()">Canviar</button>
14 <p id="nou"></p>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/jWjmYL?editors=1010](https://codepen.io/ioc-daw-m06/pen/jWjmYL?editors=1010).

Com es pot apreciar, en el segon paràmetre de la invocació a `replace`, l'expressió `$1` indica que en aquest lloc anirà la primera -i en aquest cas única- cadena capturada. Si capturéssim una segona cadena, podríem referir-nos-hi amb `$2`, a una tercera amb `$3`, etc.

Fixeu-vos que el patró utilitzat ha estat `(\\b[\\L]o\\b)+`; això és així pels motius següents:

- Volem capturar els grups que continguin el patró envoltat pels parèntesis: `( i )`.
- Només volem fer el reemplaçament si n'hi ha 1 o més: `+`.
- Ha de ser la paraula completa, no ho volem substituir en els casos què formi part d'una altra paraula, com pot ser **los** o **millor**, així que ho hem delimitat al principi i final de les paraules amb `\\b`.
- Volem fer el canvi tant en el cas que la `l` sigui majúscula com si no, per això afegim la selecció `: [\\L]`.

A partir d'aquest codi, només canviant el patró, podem veure com és molt fàcil afegir més elements a la selecció; per exemple, si volem que la substitució afecti “la”, “les”, “lo” i “los”, només hem de tenir en compte els elements comuns i les variacions; el patró seria el següent: `(\\b[\\L] [aoe]s?\\b)`. Per tant:

- Hem afegit un joc de selecció de caràcters nou: `[aoe]`.
- Hem afegit una `s` opcional: `s?`.

A continuació podeu trobar un exemple que cerca totes les coincidències de “la”, “las”, “le”, “les”, “lo” i “los” tant amb “l” minúscula com majúscula i les



reemplaça per la mateixa coincidència però afegint les etiquetes per mostrar-les en negreta (element `<b>`). És a dir, si es troba la coincidència “lo” ho substituirà per `<b>lo</b>`. Vegem-ho:

```

1 <p id="original">»Aprés diré a la senyoria vostra les cerimònies que s'an de
  fer en la capella. Ara diré los cavallers qui foren elets. Primerament, lo
  rey elegí XXV cavallers e, ab lo rey foren XXVI. Lo rey fon lo primer qui
  jurà de servir totes les ordinacions en los capítols contengudes e que no
  fos cavaller negú qui demanàs aquest orde que·l pogués haver. Tirant fon
  elet lo primer de tots los altres cavallers, per ço com fon lo millor de
  tots los cavallers. Aprés fon elet lo príncep de Gales, lo duch de
  Betafort, lo duch de Lencastre, lo duch d'Atçètera, lo marquès de Sófolch,
  lo marquès de Sant Jordi, lo marquès de Belpuig, Johan de Varoych, gran
  conestable, lo comte de Nortabar, lo comte de Salasberi, lo comte d'
  Estafort, lo comte de Vilamur, lo comte de les Marches Negres, lo comte de
  la Joyosa Guarda, lo senyor d'Escala Rompuda, lo senyor de Puigvert, lo
  senyor de Terranova, miçer Johan Stuart, miçer Albert de Riugech. Aquests
  foren del regne. Los strangers foren: lo duch de Berrí, lo duch d'Anjou,
  lo comte de Flandes. Foren tots en nombre de XXVI cavallers.
2 </p>
3 <button onClick="canviar()">Canviar</button>
4 <p id="nou"></p>
5
6 <script>
7   let original = document.getElementById('original'),
8     nou = document.getElementById('nou');
9
10  function canviar() {
11    let text = original.innerHTML,
12      patro = /(\\b[LL][aoe]?\\b)/g;
13    nou.innerHTML = text.replace(patro, '<b>$1</b>');
14  }
15 </script>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/YwoQKq?editors=1010](https://codepen.io/ioc-daw-m06/pen/YwoQKq?editors=1010).

La resta del patró és idèntic, en canvi, ara s'aplica a moltes més combinacions.

Com podeu veure, encara que a primera vista les expressions regulars poden semblar intel·ligibles, si se'n divideixen els components, són més fàcils d'entendre. Ara bé, hi ha excepcions, com per exemple en el cas de les substitucions, que es poden fer servir per localitzar en altres idiomes diferents de l'anglès.

## 2.5.2 Altres mètodes d'utilització de les expressions regulars ('String' i 'RegExp')

Cal destacar que la utilització de patrons d'expressions regulars no es troba limitada als objectes de tipus `RegExp`, sinó que són utilitzats també per altres objectes. Per exemple, els objectes de tipus `String` ofereixen dos mètodes que accepten expressions regulars com a paràmetres, el que permet fer reemplaçaments i divisions de cadenes complexes.

Vegem a continuació alguns mètodes dels objectes de tipus `String` i els objectes de tipus `RegExp` que permeten realitzar diferents operacions amb expressions regulars:

### Captura de caràcters accentuats

El metacaràcter `\w` no captura ni la `ç` ni els caràcters accentuats, per tant, si volem seleccionar tots aquests caràcters, el patró equivalent a `\w` és:

[A-Za-z-çàêéíïóôüçÀÊËÏÎÔÕÜŮ]

- `exec(cadena)`: mètode de `RegExp` que executa una cerca sobre el paràmetre i retorna un *array* amb la informació de la cadena trobada (posició 0) i, si n'hi ha, grups de captura (resta de posicions); si no troba la cadena, retorna `null`.
- `test(cadena)`: mètode de `RegExp` que cerca una coincidència a cadena i retorna `true` si la troba o `false` en cas contrari.
- `match(regex)`: mètode de `String` amb funcions similars a `exec`.
- `search(regex)`: mètode de `String` que cerca a la cadena de text el patró passat com argument. En cas de trobar-se alguna coincidència retorna la posició (dintre de la cadena) on comença la coincidència o el valor -1 si no s'ha trobat. En cas de trobar-se múltiples coincidències només es té en compte la primera d'aquestes i la resta són ignorades.
- `replace(regex, cadena)`: mètode de `String` que executa la cerca en base a la expressió regular indicada al primer paràmetre i la reemplaça per la cadena de reemplaçament especificada al segon paràmetre (que pot contenir referències als grups de captura).
- **`split(regex)`**: mètode de `String` que divideix una cadena en fragments i els col·loca com a *array*. L'expressió regular identifica els separadors dels diferents fragments.

Tant `replace` com `split` poden treballar directament amb cadenes, no és obligatori fer servir expressions regulars.

Vegem amb un exemple com funcionen tots aquests mètodes; obriu la consola de CodePen o les eines de desenvolupador per veure'n la informació extreta:

```
1 <script>
2   let original = document.getElementById('original'),
3     nou = document.getElementById('nou');
4
5   function canviar() {
6     let text = original.innerHTML;
7
8     provaTest(text);
9     provaSearch(text);
10    provaExec(text);
11    provaMatch(text);
12    provaSplit(text);
13
14    nou.innerHTML = provaReplace(text);
15  }
16
17  function provaTest(text) {
18    let patro = /(cavallers)|(rey)/g;
19    console.log("——RegExp.test()——");
20    console.log("Es troba la paraula cavallers o rey?", patro.test(text));
21  }
22
23  function provaSearch(text) {
24    let patro = /(cavallers)|(rey)/g;
25    console.log("——String.search()——");
26    console.log("Es troba la paraula cavaller o cavallers, i si és així en quina posició?", text.search(patro));
27  }
28
29  function provaExec(text) {
30    let patro = /((cavallers)|(rey))+/g;
31    console.log("——RegExp.exec()——");
```

```

32     console.log("Quines coincidències de la paraula cavallers o rey s'han
        trobat?", patro.exec(text));
33 }
34
35 function provaMatch(text) {
36     let patro = /((cavallers)|(rey))+/g;
37     console.log("——String.match()——");
38     console.log("Quines coincidències de la paraula cavallers o rey s'han
        trobat?", text.match(patro));
39 }
40
41 function provaSplit(text) {
42     let patro = /[[:\.\.]]? /g;
43     console.log("Array amb tots els mots del text:", text.split(patro));
44 }
45
46 function provaReplace(text) {
47     let patro = /([A-Za-z_çàèèííóòüüÇÀÉÈÍÍÓÓÜÜ]+)([\.:\. ]?\s)([A-Za-z_çàèèííóòü
        üÇÀÉÈÍÍÓÓÜÜ]+)/gm;
48
49     return text.replace(patro, '$3$2$1');
50 }
51 </script>
52 <p id="original">Aprés diré a la senyoria vostra les cerimònies que s'an de fer
    en la capella. Ara diré los cavallers qui foren elets. Primerament, lo
    rey elegí XXV cavallers e, ab lo rey foren XXVI. Lo rey fon lo primer qui
    jurà de servir totes les ordinacions en los capítols contengudes e que no
    fos cavaller negú qui demanàs aquest orde que·l pogués haver. Tirant fon
    elet lo primer de tots los altres cavallers, per ço com fon lo millor de
    tots los cavallers. Aprés fon elet lo príncep de Gales, lo duch de
    Betafort, lo duch de Lencastre, lo duch d'Atçètera, lo marquès de Sófolch,
    lo marquès de Sant Jordi, lo marquès de Belpuig, Johan de Varoych, gran
    conestable, lo comte de Nortabar, lo comte de Salasberi, lo comte d'
    Estafort, lo comte de Vilamur, lo comte de les Marches Negres, lo comte de
    la Joyosa Guarda, lo senyor d'Escala Rompuda, lo senyor de Puigvert, lo
    senyor de Terranova, miçer Johan Stuart, miçer Albert de Riuçech. Aquests
    foren del regne. Los strangers foren: lo duch de Berrí, lo duch d'Anjou,
    lo comte de Flandes. Foren tots en nombre de XXVI cavallers.
53 </p>
54 <button onclick="canviar()">Canviar</button>
55 <p id="nou"></p>

```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/RrzgYL?editors=1011](https://codepen.io/ioc-daw-m06/pen/RrzgYL?editors=1011).

Fixeu-vos que fent servir el mateix patró, el mètode `RegExp.exec()` **no retorna els valors esperats**; en canvi, el mètode `String.match()` ens retorna un *array* amb tots els elements, com és d'esperar.

Com es pot apreciar, s'ha fet servir el mètode `String.replace()` amb un patró una mica més complex per capturar correctament totes les paraules. Com a segon paràmetre s'ha utilitzat la cadena de reemplaçament `$3$2$1`, de manera que es posa primer el tercer grup capturat (`$3`), a continuació el segon grup (`$2`) i finalment el primer (`$1`). Així doncs, la primera coincidència del patró serien els grups *Aprés*, i *diré*; per tant, en reemplaçar el resultat és *diré Aprés*.

## 2.6 Utilització de galetes i Web Storage

Com que *HTTP* és un protocol sense estat, cada vegada que es demana una pàgina al servidor, el servidor no “ens recorda”. Això, com podeu imaginar, suposa un greu inconvenient, ja que no és possible, per exemple, canviar de pàgina sense perdre tota la informació, com pot ser:

- Estar autenticats a un lloc web
- La cistella de la compra en un negoci de venda per internet
- Les preferències seleccionades en visitar un lloc web

Tampoc no es pot accedir als següents serveis:

- Fer el seguiment de les pàgines visitades per generar analítiques
- Mostrar vídeos (Youtube)
- Mostrar mapes (Google Maps)
- Mostrar anuncis publicitaris
- Serveis de xat

La solució que es va donar a aquest problema va ser la invenció de les galetes que són uns petits fragments d'informació que el navegador guarda en fitxers i s'envien juntament amb la capçalera al servidor; de manera que es pot simular que el servidor ens recorda, tot que en realitat sempre es tracta cada petició com si fos nova.

### Mètode alternatiu a l'ús de galetes per autenticar-se

En cas de no tenir activada l'opció d'acceptar galetes, el que es fa habitualment per mantenir sessions *obertes* al servidor és passar la informació d'autenticació com a paràmetre d'una petició GET; és a dir, es codifica al mateix *URL*.

Com que aquestes galetes poden ser llegides i escrites des de JavaScript, es poden utilitzar per a altres propòsits que no estan relacionats amb el servidor; per exemple, per guardar les preferències d'un usuari en visitar una pàgina. S'ha de tenir en compte que la mida màxima per a cada galeta és de 4.095 bytes.

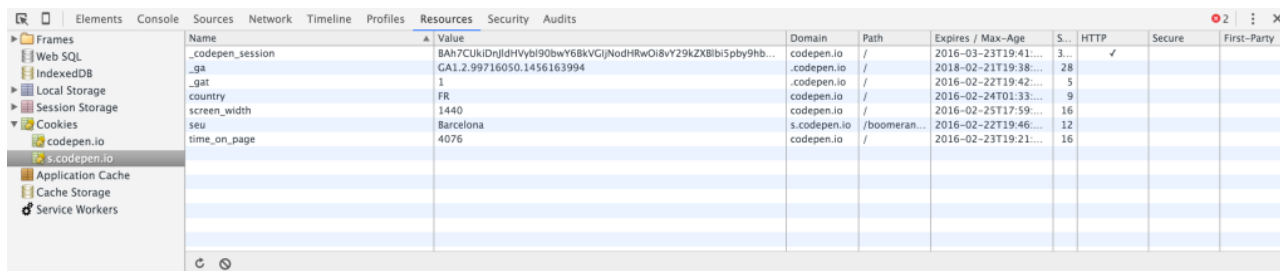
Amb HTML5 es va afegir una nova Web API per emmagatzemar informació al navegador anomenada Web Storage que, a diferència de les galetes, no s'envien mai al servidor.

Així doncs, si necessitem compartir la informació guardada amb el servidor, farem servir galetes, però, si s'hi ha d'accedir només localment, es recomana fer servir Web Storage, ja que accepta fins a 5 MB per domini i s'estalvia amplada de banda perquè no s'envia aquesta informació al servidor (al contrari del que passa amb les galetes).

Si es desactiva l'ús de galetes al navegador, queden inhabilitats tant el Web Storage com les galetes.

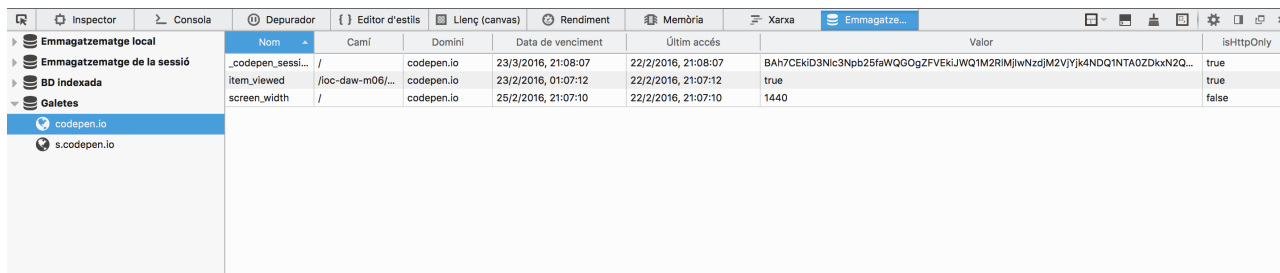
Les galetes poden visualitzar-se des de les eines de desenvolupador dels navegadors, però la seva localització canvia segons el navegador amb el qual treballem. A Google Chrome podeu trobar-les a la pestanya **Resources**, en el desplegable **Cookies** del panell esquerre, tal com es veu en la imatge figura 2.7. En canvi, si feu servir Mozilla Firefox, les podeu trobar a la pestanya **Emmagatzematge**, en el desplegable **Galetes** del panell esquerre, tal com es veu en la imatge figura 2.8.

FIGURA 2.7. Visualització de galetes a Google Chrome



Name	Value	Domain	Path	Expires / Max-Age	S...	HTTP	Secure	First-Party
_codepen_session	BAH7CUkiDnjdHvYbI90bWY6BkVCjNodHRwOi8vY29kZXBlbi5pby9hb...	codepen.io	/	2016-03-23T19:41:...	3...			
_ga	GA1.2.99716050.1456163994	codepen.io	/	2018-02-22T19:38:...	28			
_gat	1	codepen.io	/	2016-02-22T19:42:...	5			
country	FR	codepen.io	/	2016-02-24T01:33:...	9			
screen_width	1440	codepen.io	/	2016-02-25T17:59:...	16			
seu	Barcelona	s.codepen.io	/boomeran...	2016-02-22T19:46:...	12			
time_on_page	4076	codepen.io	/	2016-02-23T19:21:...	16			

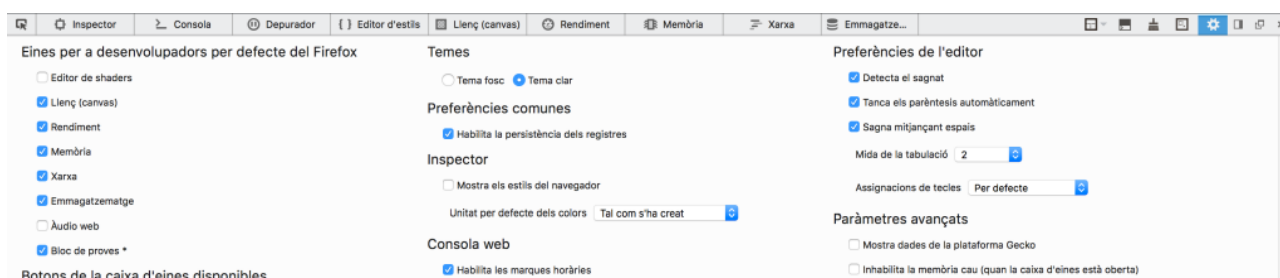
FIGURA 2.8. Visualització de galetes a Mozilla Firefox



Nom	Camí	Domini	Data de venciment	Últim accés	Valor	isHttpOnly
_codepen_sessi...	/	codepen.io	23/3/2016, 21:08:07	22/2/2016, 21:08:07	BAH7CEkiD3Nic3Npb25faWQGOgZFVEkiJWQ1M2RlMjIyYjYkNDQ1NTA0ZDkxN2Q...	true
item_viewed	/loc-daw-m06/...	codepen.io	23/2/2016, 01:07:12	22/2/2016, 21:07:12	true	true
screen_width	/	codepen.io	25/2/2016, 21:07:10	22/2/2016, 21:07:10	1440	false

En cas que no us aparegui aquesta pestanya a Mozilla Firefox, la podeu habilitar a les opcions de les eines de desenvolupador (símbol de la roda dentada a la dreta), marcant la casella **Emmagatzematge** (vegeu la figura 2.9).

FIGURA 2.9. Visualització d'opcions a Mozilla Firefox



Eines per a desenvolupadors per defecte del Firefox	Temes	Preferències de l'editor
<input type="checkbox"/> Editor de shaders <input checked="" type="checkbox"/> Llenç (canvas) <input checked="" type="checkbox"/> Rendiment <input checked="" type="checkbox"/> Memòria <input checked="" type="checkbox"/> Xarxa <input checked="" type="checkbox"/> Emmagatzematge <input type="checkbox"/> Àudio web <input checked="" type="checkbox"/> Bloc de proves *	<input type="radio"/> Tema fosc <input checked="" type="radio"/> Tema clar <b>Preferències comunes</b> <input checked="" type="checkbox"/> Habilita la persistència dels registres <b>Inspector</b> <input type="checkbox"/> Mostra els estils del navegador Unitat per defecte dels colors: Tal com s'ha creat <b>Consola web</b> <input checked="" type="checkbox"/> Habilita les marques horàries	<b>Preferències de l'editor</b> <input checked="" type="checkbox"/> Detecta el sagnat <input checked="" type="checkbox"/> Tanca els parèntesis automàticament <input checked="" type="checkbox"/> Sagna mitjançant espais Mida de la tabulació: 2 Assignacions de tecles: Per defecte <b>Paràmetres avançats</b> <input type="checkbox"/> Mostra dades de la plataforma Gecko <input type="checkbox"/> Inhabilita la memòria cau (quan la caixa d'eines està oberta)

La informació sobre els **Web Storage** es troba a la mateixa pestanya, però a la secció corresponent: **Local Storage** o **Session Storage**.

## 2.6.1 Llei de cookies

Abans d'endinsar-nos en la gestió de galetes hem de parlar de la normativa que en regula l'ús, ja que si no s'informa adequadament l'usuari que les fem servir i amb quina finalitat, podem ser sancionats. En aquests materials no tractarem el tema en

profunditat, però és important conèixer-les i saber que sempre que treballem amb galetes haurem d'aplicar aquesta llei. Recomanem la lectura de la **Guia sobre les normes d'ús de les galetes**.

#### Guia sobre les normes d'ús de les galetes

Podeu trobar la guia oficial de l'Agencia española de protección de datos amb tota la informació sobre la legislació de galetes a l'enllaç següent: [www.goo.gl/akFmH](http://www.goo.gl/akFmH).

#### Exemple de sanció econòmica per vulnerar la llei de galetes

En aquest enllaç podeu trobar informació sobre el primer cas de **sancions econòmiques per vulnerar la llei de galetes**: [www.goo.gl/Ej7bFQ](http://www.goo.gl/Ej7bFQ).

Si us hi fixeu, no se n'està fent cap ús maliciós, totes les galetes que es fan servir tenen una finalitat legítima. Pertanyen a components que fan funcionar la pàgina correctament, o a publicitat, o bé a serveis externs, com Google Maps, YouTube... Tot i així, com que el text amb la política de galetes no està ben definit, les empreses involucrades van ser sancionades.

Les sancions per aquests motius poden arribar fins als 30.000€.

A l'hora de treballar amb galetes s'ha de tenir molt present la legislació vigent, que ens obliga a informar l'usuari que el nostre lloc web fa servir galetes i amb quins fins, i amb un enllaç que expliqui com pot desactivar-les al seu navegador.

A la pràctica és molt difícil trobar un lloc web que no faci servir galetes, ja que són imprescindibles fins i tot per generar analítiques. Per aquesta raó podeu donar per descomptat que haureu d'informar l'usuari que el vostre lloc web fa servir galetes i respectar la normativa.

#### Declinació de l'ús de galetes

Com que per guardar la informació sobre l'acceptació de les galetes cal fer servir galetes, si un usuari en declina l'ús, no es pot guardar aquesta preferència.

Haurà de declinar-la a cada pàgina que visiti o resignar-se a veure el missatge.

S'ha de tenir en compte també que si les dades guardades en les galetes són de caràcter personal, s'ha d'aplicar també la Llei orgànica de protecció de dades (LOPD).

#### Llei orgànica de protecció de dades (LOPD)

Podeu trobar més informació sobre aquesta llei al següent enllaç: [www.goo.gl/Fmpuzv](http://www.goo.gl/Fmpuzv) i [www.goo.gl/zHqQIY](http://www.goo.gl/zHqQIY).

## 2.6.2 Utilització de les galetes

El funcionament de les galetes o *cookies* és molt senzill, però utilitzar-les no ho és tant. S'hi accedeix a través de la propietat `document.cookie`, tant per recuperar-les com per guardar-les. Podeu veure-ho en el següent exemple:

```
1 <script>
2   // Afegim una nova galeta
3   document.cookie="seu=Barcelona";
4   document.cookie="paf=1";
5
6   // Recuperem totes les galetes
7   let galetes = document.cookie;
8
9   // Les mostrem a la consola
10  console.log(galetes);
11 </script>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/YwovaK?editors=0011](https://codepen.io/ioc-daw-m06/pen/YwovaK?editors=0011).

Si comproveu la consola, veureu el contingut de la vostra galeta. En cas de provar-lo en CodePen, veureu també les galetes afegides per CodePen separades per un ;

unes de les altres: “seu=Barcelona; paf=1; grid\_layout=list; \_gat=1; \_ga=GA1.2.197952496.1453793789”

El primer que veieu en aquest exemple és com es guarda una informació a una galeta; sempre ha de ser amb el següent format: `clau=valor`, i si existeixen més galetes, estaran separades amb un `;`.

Segurament us ha estranyat aquest fragment de codi:

```
1 document.cookie="seu=Barcelona";
2 document.cookie="paf=1";
```

Es podria esperar que en sobreescrivre el valor de la propietat `document.cookie`, aquest valor fos “paf=1” i el valor `seu=Barcelona` fos eliminat. Això no passa, perquè aquesta propietat té uns *setter* i *getter* natius que en modifiquen el comportament i, per tant, **no és el mateix el que s’escriu i el que es llegeix**.

Provem ara de guardar múltiples galetes:

```
1 <script>
2 // Afegim una nova galeta
3 document.cookie="seu=Barcelona";
4 document.cookie="seu=Tarragona";
5 document.cookie="seu=Lleida";
6 document.cookie="seu=Girona";
7
8 // Recuperem totes les galetes
9 let galetes = document.cookie;
10
11 // Les mostrem a la consola
12 console.log(galetes);
13 </script>
```

Es coneixen com a *setter* les funcions (o mètodes) encarregades d'establir un valor a una propietat i *getter* les que els recuperen.

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/BjgVqp?editors=0011](https://codepen.io/ioc-daw-m06/pen/BjgVqp?editors=0011).

Com podeu apreciar, només s’ha guardat el valor `seu=Girona`. En realitat s’han guardat tots, però els següents els han sobreescrit, ja que només hi pot haver un valor associat a cada clau.

Una possible solució seria guardar les dades com un *array*, però **les galetes només poden guardar cadenes de text**, i per tant, si necessitem un comportament més avançat, l’hem d’implementar nosaltres mateixos o fer servir alguna llibreria externa.

Després d’entendre com guardar i recuperar les galetes, se’ns presenta un problema nou: com ho fem per recuperar només la galeta que ens interessa i no totes? Malauradament ja heu vist tot el suport proporcionat per l’especificació, és a dir: com afegir una galeta, com sobreescrivre’n el valor i com recuperar-les totes. Si volem recuperar-ne només una, hem d’implementar la nostra pròpia solució, per exemple:

```
1 <script>
```

```
2 // Afegim una nova galeta
3 document.cookie = "seu=Barcelona";
4 document.cookie = "paf=1";
5
6 // Extraiem la galeta corresponent a la seu
7 let patro = new RegExp("(?:seu)=(.+)?(?=:|$)", "g"),
8     seu = patro.exec(document.cookie)[1];
9
10 // La mostrem a la consola
11 console.log(seu);
12 </script>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/Ywovoy?editors=0011](https://codepen.io/ioc-daw-m06/pen/Ywovoy?editors=0011).

El primer que fem és definir un patró que ens capturarà el valor de la galeta amb el nom seu, i a continuació l'executem. El resultat d'executar l'expressió regular és un *array* que conté a la primera posició (índex 0) el text coincident complet i a les següents posicions el contingut dels grups de captura ignorant el primer, ja que té el format de grup no capturable: (?:). És a dir, a l'índex 1 es troba el contingut del grup que es troba a continuació del signe igual, ja que és el primer grup de captura vàlid i, per consegüent, es correspon amb el valor de la seu.

Tot això és força enrevessat, així que, per simplificar-ho, podem crear una funció que ens facilitarà la tasca, com en aquest exemple:

```
1 <script>
2 // Afegim una nova galeta
3 document.cookie = "seu=Barcelona";
4 document.cookie = "paf=1";
5
6 // Recuperem les galetes
7 let paf = getCookie('paf'),
8     seu = getCookie('seu');
9
10 // Mostrem les dades per consola
11 console.log("Totes les galetes:", document.cookie);
12 console.log("Seu:", seu);
13 console.log("PAF:", paf);
14
15 function getCookie(key) {
16     let patro = new RegExp("(?:"+key + ")=(.+)?(?:=:|$)", "g");
17     return patro.exec(document.cookie)[1];
18 }
19 </script>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/ZQdjKP?editors=0011](https://codepen.io/ioc-daw-m06/pen/ZQdjKP?editors=0011).

Com podeu comprovar, en fer servir una funció que encapsuli la lògica per recuperar una galeta, se'n simplifica molt la utilització; només hem d'invocar `getCookie()` i passar el nom de la clau que volem recuperar.

A banda d'aquest mètode -més simple- per guardar una galeta, les galetes admeten altres opcions per limitar-ne l'ús i la caducitat; són les següents:

- **path**: per exemple `/`, `/actualitat`. Si no s'especifica a la galeta, s'hi podrà accedir des de totes les pàgines del lloc web.



- **domain:** per exemple `exemple.com`, `subdomini.exemple.com`. Si s'especifica el domini principal, també es podrà llegir des dels subdominis; si no s'especifica, només es podrà llegir des del domini principal.
- **max-age:** durada màxima en segons fins que la galeta caduqui.
- **expires:** data en format GMT en què expirarà la galeta. Si no s'especifica, la galeta caduca quan finalitza la sessió (generalment quan es tanca la pestanya concreta o el navegador)
- **secure:** la galeta només es podrà transmetre sobre protocols segurs com HTTPS. Aquest tipus de galetes no necessita valor. Totes les galetes enviades a través de HTTPS són automàticament segures.

Tot i que existeix una opció **secure**, **les galetes són inherentment insegures**, i no heu de fer-les servir mai per guardar informació sensible ni confidencial.

Cada opció i el seu valor anirà separat per un punt i coma (;) i un espai en blanc, i es separaren els parells *opció* i *valor* per un signe igual (=).

Aprofitant que ja coneixem totes les opcions possibles, implementarem la funció `setCookie()`:

```
1 <script>
2 function setCookie(key, value, options) {
3     let cookie = encodeURIComponent(key) + '=' + encodeURIComponent(value);
4
5     if (options) {
6         for (option in options) {
7             if (option === 'secure') {
8                 cookie += ';' + encodeURIComponent(option);
9             } else {
10                 cookie += ';' + encodeURIComponent(option) + '=' + encodeURIComponent(
11                     options[option]);
12             }
13         }
14     }
15     document.cookie = cookie;
16 }
17
18 function getCookie(key) {
19     let patro = new RegExp("(?:" + key + ")=([^;]|$)", "g");
20     return patro.exec(document.cookie)[1];
21 }
22
23 // Afegim una nova galeta
24 setCookie('seu', 'Barcelona', {
25     'max-age': 300,
26     path: '/',
27     domain: 'codepen.io'
28 });
29
30 // Mostrem les dades per consola
31 console.log("Totes les galetes:", document.cookie);
32 console.log("Seu:", getCookie('seu'));
33 </script>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/zrVeGp?editors=0011](https://codepen.io/ioc-daw-m06/pen/zrVeGp?editors=0011).

Fixeu-vos que, per passar les opcions de les galetes, hem fet servir un objecte literal de JavaScript, d'aquesta manera el codi queda molt més net i compacte. La propietat `max-age`, en incloure el signe `-`, s'ha de posar entre cometes.

Si proveu de passar el paràmetre `secure` no us funcionarà perquè requereix que la connexió es faci sobre HTTPS, i ni l'exemple en local ni en CodePen corren sobre aquest protocol.

Tot i que l'especificació no requereix fer la codificació completa dels continguts de la galeta (només els espais, `,` i `;` han de ser codificats) és més simple codificar tots els valors i les claus fent servir la funció `encodeURIComponent()`. D'aquesta manera ens assegurem que els valors guardats seran correctes.

### Temps i durada de les galetes

Tot i que la nostra funció per guardar galetes rutlli i contempli totes les opcions disponibles, encara podem tenir problemes amb un apartat: establir la data en la qual volem que expiri.

L'especificació ens diu que el format ha de ser el següent: `Wdy, DD-Mon-YYYY HH:MM:SS GMT` com per exemple: `Sat, 02 May 2009 23:38:25 GMT`. Però si fem servir l'objecte `Date` de JavaScript el que ens retorna serà semblant a això:

```
1 Date.now();  
2 // 1456167356375
```

L'objecte `Date` ens retorna la data en mil·lisegons. Afortunadament, disposa de mètodes per poder modificar aquesta data; vegem-ne alguns exemples:

```
1 <script>  
2   let ara = new Date(),  
3       mes,  
4       proximMes,  
5       nadal;  
6  
7   console.log("La data d'avui en UTC", ara.toUTCString());  
8  
9   mes = ara.getMonth();  
10  console.log("Quin és el mes actual?", mes);  
11  
12  proximMes = (mes + 1) % 12;  
13  ara.setMonth(proximMes);  
14  console.log("Quina és la data corresponent al pròxim mes?", ara.toUTCString()  
15           );  
16  
17  nadal = new Date(2016, 11, 25);  
18  console.log("Quin dia és Nadal el 2016?", nadal.toUTCString());  
19  
20  console.log("Desfasament horari per la nostra localització (en minuts):",  
21           nadal.getTimezoneOffset()  
22  );  
23 </script>
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/mVZoqm?editors=0011](https://codepen.io/ioc-daw-m06/pen/mVZoqm?editors=0011).

El resultat d'invocar el mètode `toUTCString()` produeix una data exactament amb el format que necessitem, com aquesta: `Tue, 22 Mar 2016 19:08:01 GMT`. Fixeu-vos que, en l'últim exemple, creem una data en concret, però el resultat pot ser inesperat, ja que:

- Gener correspon al mes 0, i per tant desembre és el mes 11.
- La data mostrada serà aquesta: `Sat, 24 Dec 2016 23:00:00 GMT`
- El desfasament horari serà de -60 minuts, això explica per què la data mostrada sembla errònia per una hora. Quan es crea l'objecte `Date` es fa servir l'ús horari del client, que en el nostre cas és `GMT+1`, i en convertir-la en `GMT` es descompta aquesta hora.

Treballar amb dates afegeix una complicació extra a la creació de galetes. Per aquesta raó és més recomanable fer servir l'opció `max-age`, que ens permet especificar l'edat màxima que pot assolir la galeta, expressada en segons. Així, per exemple, podem crear-ne una que expiri en un mes així:

```
1  setCookie('seu', 'Barcelona', {  
2    'max-age': 60*60*24*30  
3  });
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/bEPZZx?editors=0011](https://codepen.io/ioc-daw-m06/pen/bEPZZx?editors=0011).

Per tant, podem fer que les galetes persisteixin en el temps, encara que es finalitzi la sessió del navegador. Ara bé, com podem esborrar-les? La manera de fer-ho és molt simple, només hem de guardar una galeta amb la mateixa clau i amb l'opció `max-age=-1` (o amb l'opció `expires` amb una data ja passada). D'aquesta manera la galeta expira immediatament:

```
1  setCookie('seu', 'Barcelona', {  
2    'max-age': -1  
3  });
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-daw-m06/pen/vLqMEp?editors=0011](https://codepen.io/ioc-daw-m06/pen/vLqMEp?editors=0011).

### 2.6.3 Utilització de Web Storage

Aquesta Web API permet emmagatzemar dades en l'ordinador client amb més possibilitats que les galetes. Distingeix dues maneres de realitzar aquest emmagatzematge:

- `LocalStorage`: les dades no expiren mai i, per tant, es requereix esborrat manual o per programa

- **SessionStorage**: les dades es guarden només per una sessió i, en tancar la finestra del navegador s'esborra.

A banda de l'anterior, el funcionament és idèntic a tots dos tipus de Web Storage.

### Web Storage

Podeu trobar l'especificació del *Living Standard* al següent enllaç: [www.goo.gl/JVCEHI](http://www.goo.gl/JVCEHI). I informació sobre el funcionament detallat en aquest: [www.goo.gl/mPwJxl](http://www.goo.gl/mPwJxl).

El **LocalStorage** comparteix la informació entre totes les pàgines del mateix domini, en canvi, el **SessionStorage** és diferent per a cada finestra; de manera que si tenim tres finestres obertes amb la mateixa pàgina, la informació del **SessionStorage** serà diferent per a cadascuna, mentre que la del **LocalStorage** serà la mateixa.

A diferència de l'ús de galetes, aquesta Web API ofereix una interfície molt clara per interactuar amb el magatzem de dades:

- **Storage.setItem(key, value)**: per guardar un valor amb la clau especificada.
- **Storage.getItem(key)**: per recuperar el valor corresponent a la clau.
- **Storage.removeItem(key)**: per eliminar l'element corresponent a la clau del magatzem.
- **Storage.clear()**: elimina totes les claus del magatzem.

Vegem com funcionen tots aquests mètodes:

```
1 <script>
2   localStorage.setItem('seus', ['Barcelona', 'Girona', 'Lleida', 'Tarragona']);
3   localStorage.setItem('seu seleccionada', 'Barcelona');
4   localStorage.setItem('paf', 1);
5   localStorage.setItem('confirmat', false);
6
7
8   console.log("Comprovem que s'han creat:"); console.log("seus:", localStorage.
9     getItem('seus'));
10  console.log("seleccionada:", localStorage.getItem('seu seleccionada'));
11  console.log("paf:", localStorage.getItem('paf'));
12  console.log("confirmat:", localStorage.getItem('confirmat'));
13
14  console.log("Eliminem la seu seleccionada.")
15  localStorage.removeItem('seu seleccionada');
16
17  console.log("Comprovem el valor de la seu seleccionada:", localStorage.
18    getItem('seu seleccionada'));
19
20  console.log("Eliminem tots els valors");
21  localStorage.clear();
22
23  console.log("Comprovem que s'han eliminat:")
24  console.log("seus:", localStorage.getItem('seus'));
25  console.log("seleccionada:", localStorage.getItem('seu seleccionada'));
26  console.log("paf:", localStorage.getItem('paf'));
27  console.log("confirmat:", localStorage.getItem('confirmat'));
28 </script>
```

Per fer servir el **SessionStorage** en lloc del **LocalStorage**, només hem de canviar `localStorage` per `sessionStorage`.

Com podeu veure, a diferència de les galetes, els valors emmagatzemats poden ser de diferents tipus: enters, cadenes, *booleans* i fins i tot *arrays* (només caldrà fer servir `String.split()` per recuperar l'*array*).

No cal crear cap objecte ni configurar res per accedir als magatzems de dades, són accessibles directament a través dels objectes globals `localStorage` i `sessionStorage` que ens proporciona el navegador, i la informació es guarda automàticament associada al domini on s'ha creat.

La persistència de les dades es basa en el tipus de magatzem: si fem servir el *LocalStorage* aquestes dades persistiran fins que les esborrem nosaltres mateixos via codi o quan l'usuari esborri les dades de navegació; en canvi, si fem servir el *SessionStorage*, totes les dades s'esborraran tan bon punt es tanqui la finestra del navegador.