El marc de treball **jQuery** és una biblioteca de **JavaScript**, creada inicialment per **John Resig**, que permet simplificar la manera d'interactuar amb els documents **HTML**, manipular l'arbre **DOM**, manejar esdeveniments, desenvolupar animacions i afegir interacció amb la tècnica **AJAX** a pàgines web. Va ser presentada el **14 de gener de 2006** al **BarCamp NYC**. **jQuery** és la biblioteca de **JavaScript** més utilitzada.

jQuery és programari lliure i de codi obert, té un doble llicenciament sota la **Llicència MIT** i la **Llicència Pública General de GNU v2**, permetent el seu ús en projectes lliures i privats. **jQuery**, igual que altres biblioteques, ofereix una sèrie de funcionalitats basades en **JavaScript** que d'altra manera requeririen de molt més codi, és a dir, amb les funcions pròpies d'aquesta biblioteca s'aconsegueixen grans resultats en menys temps i espai.

Poseeix un nombre molt gran de plugins, cada any creix, i alguns d'ells estàn registrats a **jQuery**. Podeu veure aquests últims en les mateixes pàgines del projecte, provar-los i baixar-los. Posiblement, a dia d'avui, és encara més popular gràcies a la multitut de plugins de qualitat que disposa.

Amb aquesta pràctica no pretenem cubrir tot el jQuery, per això ja existeixen llibres força bons, simplement domen un repàs a allo que ens sembla més interesant.

jQuery consisteix en un únic fitxer **JavaScript** que conté les funcionalitats comunes de **DOM**, esdeveniments, efectes i **AJAX**.

En el moment d'escriure aquesta pràctica tenim la versió **v1.11.2** de la **branca 1**, i la versió **v2.1.3** de la **branca 2**. Recordeu que la **branca 2**, no es apta pel navegador **IE** de **Microsoft** en les seves versions menors de la **9**.

La característica principal de la biblioteca és que permet canviar el contingut d'una pàgina web sense necessitat de

recarregar-la, mitjançant la manipulació de l'arbre **DOM** i peticions **AJAX**. Per a això utilitza les funcions **`$()`** o **`jQuery()`**.

Les característiques d'aquesta biblioteca son:

- Selecció d'elements DOM.
- Interactivitat i modificacions de l'arbre DOM, incloent suport per a CSS 1-3 i un plugin bàsic de XPath.
- Esdeveniments.
- Manipulació del full d'estils CSS.
- Efectes i animacions.
- Animacions personalitzades.
- AJAX.
- Suporta extensions (plugins).
- Utilitats diverses com obtenir informació del navegador, operar amb objectes i vectors, funcions per rutines comunes, etc.
- Compatible amb els navegadors Mozilla Firefox 2.0+, Internet Explorer 6+, Safari 3+, Opera 10.6+ i Google Chrome 8.5+.

Abans d'utilitzar **`jQuery`** hauriem de tenir els coneixements de:

- HTML i Full d'estils en cascada (CSS).
- Editors de text i navegadors Web
- Un Javascript bàsic.

En el vostre cas, teniu els coneixements més que suficients per treballar amb aquest framework, i a més, gaudir-ne.

En el **`mòdul 9`** d'aquest cicle ja varem veure, en les activitats del llibre, la biblioteca **`jQuery`**. Així que no ens caldrà gaires explicacions.

La llibreria jQuery pot realitzar nombroses tasques amb el seu accés fàcil d'usar per al scripting, incloent:

- **Accedint als elements** utilitzant la sintaxi CSS, de manera que no es necessita aprendre a utilitzar la totalitat del codi JavaScript referent al DOM per accedir a qualsevol part del document HTML.
- **Fer canvis en el document** modificant l'aparença o el contingut dins dels elements, molt més intuïtiu que en JavaScript. Per exemple, jQuery utilitza el mètode append() per afegir contingut a la final d'un element, és fàcil de recordar quan es necessita per realitzar la tasca.

- **Creació d'efectes** amb nombrosos mètodes per a la creació d'animacions per moure, mostrar o ocultar elements.
- **AJAX**: jQuery proporciona mètodes, molt més senzills que Javascript,per recuperar informació des del servidor, i així, alleujar la càrrega d'escriure codi multi-navegador.

A més d'això, **jQuery** té mètodes que li permeten treballar fàcilment amb matrius i altres elements en **JavaScript**. Per exemple, el mètode **jQuery.inArray()**, que es pot utilitzar per facilitar la recerca d'elements d'una matriu.

Podem obtenir la biblioteca **jQuery** baixant-la de la pàgina web oficial "**Download jQuery**", qualsevol de les dues branques, i si la volem comprimida o no.

L'hem de posar en la capçalera del document **HTML**, mitjançant l'element "**<script>**".

```
<script src="jquery-1.11.2.min.js" type="text/javascript"></script>
```

Si no ens la volem baixar, sempre podem utilitzar un **CDN** (**Content Delivery Network**), o en català, una **Xarxa de Lliurament de Continguts**.

Per exemple, el proveidor **MaxCDN**, ens ofereix el codi directament amb:

```
<script src="//code.jquery.com/jquery-1.11.2.min.js"></script>
<script src="//code.jquery.com/jquery-migrate-1.2.1.min.js"></script>
```

El segon escript es un plugin per simplificar la transcició (migració) completa de **jQuery**. El connector o plugin restaura les característiques i comportaments obsolets perquè el codi més antic encara s'executi correctament en **jQuery 1.9** i posteriors.

Altres **CDN** que podem utilitzar:

- Google CDN

jQuery

1.x snippet: `<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>`
2.x snippet: `<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>`
site: jquery.com
versions: 2.1.3, 2.1.1, 2.1.0, 2.0.3, 2.0.2, 2.0.1, 2.0.0, 1.11.2, 1.11.1, 1.11...
note: 2.1.2, 1.2.5 and 1.2.4 are not hosted due to their short and unstable lives in the wild.

jQuery Mobile

snippet:

```
<link rel="stylesheet" href="https://ajax.googleapis.com/ajax/libs/jquerymobile/1.4.3/jquery.mobile.min.css" />
<script src="https://ajax.googleapis.com/ajax/libs/jquerymobile/1.4.3/jquery.mobile.min.js"></script>
```

site: jquerymobile.com
versions: 1.4.3, 1.4.2, 1.4.1, 1.4.0
note: This library depends on jQuery. You must also load jQuery before loading this module.

jQuery UI

snippet:

```
<link rel="stylesheet" href="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.2/themes/smoothness/jquery-ui.css" />
<script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.2/jquery-ui.min.js"></script>
```

site: jqueryui.com
versions: 1.11.2, 1.11.1, 1.11.0, 1.10.4, 1.10.3, 1.10.2, 1.10.1, 1.10.0, 1.9...
note: This library depends on jQuery. You must also load jQuery before loading this module. Version 1.8.3 is not hosted due to its short life, and the alias `1.8.3` actually loads 1.8.4.

- Microsoft CDN
- CDNJS CDN
- jsDelivr CDN

Moltes vegades, pels exercicis, posem l'última versió d'aquesta forma:

```
<script src="https://code.jquery.com/jquery-latest.min.js" type="text/javascript"
></script>
```

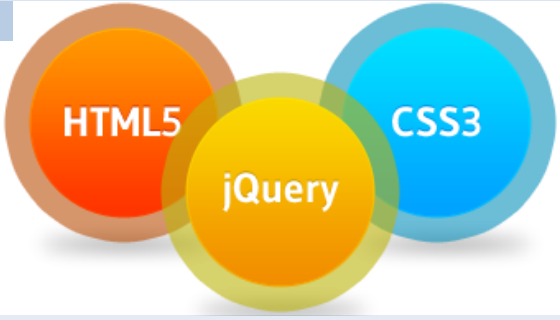
Encara que vosaltres, quan la utilitzeu de forma professional, no ho haurieu de fer. Tingueu en compte que s'actualitza amb certa freqüència, i els vostres codis poden quedar obsolets o no funcionar com vosaltres vareu projectar. Haurieu de posar sempre la versió sobre la que heu treballat el codi.

La forma correcte de treballar amb la biblioteca interna sería, per exemple, utilitzant la forma comprimida:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>El Meu Lloc</title>
		<link href="styles.css" rel="stylesheet">
		<script src="jquery-1.11.2.min.js" type="text/javascript"></script>
		<script src="meucodi.js" type="text/javascript"></script>
	</head>
	<body>
		...
	</body>
</html>
```

O externa, utilitzant el **CDN** de **Google**:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>El Meu Lloc</title>
		<link href="styles.css" rel="stylesheet">
		<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
		<script src="meucodi.js" type="text/javascript"></script>
	</head>
	<body>
		...
	</body>
</html>
```



La Funció $()

La forma d'interactuar amb la pàgina és mitjançant la funció **$()**, un àlies de **jQuery()**, que rep com a paràmetre una expressió **CSS** o el nom d'una etiqueta **HTML** i retorna tots els nodes (elements) que concordin amb l'expressió. Aquesta expressió és denominada selector en la terminologia de **jQuery**.

```
$("#taulaAlumnes"); // Retornarà l'element amb id = "taulaAlumnes"
$(".actiu"); // Retornarà una matriu d'elements amb la class = "actiu"
```

Un cop obtinguts els nodes, se'ls pot aplicar qualsevol de les funcions que facilita la biblioteca.

```
// S'elimina l'estil (amb removeClass ()) i s'aplica un de nou
// (amb addClass ()) a tots els nodes amb class = "actiu"
$(".actiu").removeClass("actiu").addClass("inactiu");
```

Com veieu "**enganxem**" dues funcions mitjançant un punt. Amb la primera eliminem la classe, i amb la segona afegim una nova classe.

O per exemple, efectes gràfics:

```
// Anima tots els components amb class = "actiu"
$(".actiu").slideToggle("slow");
```

Un exemple algo més complex utilitza **jQuery** per trobar en una **<table>** amb un id de "**datagrid**", cada fila **<tr>** parella i li afegim una classe **CSS** anomenada "**parell**" - que es podria utilitzar per alternar el color de fons de cada fila:

```
$("#datagrid tr:nth-child(even)").addClass("parell");
```

Hem de tenir en compte que la majoria de vegades, jQuery, recupera una matriu d'elements, tots els nodes que troba amb el selector, i que es tracte de forma global.

Inici de jQuery

Comunament abans de realitzar qualsevol acció en el document amb **jQuery()**, hem d'esperar que el document estigui llest (carregat). Per a això fem servir **$(document).ready()**, d'aquesta manera:

```
$(document).ready (function () {
    // Aquí van totes les accions del document.
});
```

Això garanteix que el document estigui completament carregat i puguem manipular qualsevol element de la pàgina. Per exemple:

```
$(document).ready(function() {
      $(".quan-a").addClass("especial");
});
```

Afegim la classe "**especial**" a tots els nodes que tinguin la classe "**quan-a**".

Per dur a terme la mateixa tasca, proporcionant una funció definida en el seu lloc, podria utilitzar el següent codi:

```
function ferEspecial() {
      $(".quan-a").addClass("especial");
}
$(document).ready(ferEspecial);
```

La principal diferència entre els dos enfocaments és que una funció anònima s'utilitza més sovint quan el codi de funció no és reutilitzable, mentre que una funció definida s'utilitza quan la funció es tornarà a utilitzar en un moment posterior.

Ús dels selectors CSS

La majoria dels selectors disponibles a la recomanació **CSS** Nivell 3 estan disponibles per al seu ús en **jQuery**. Això fa que **jQuery** molt més fàcil d'utilitzar que **JavaScript** per a aquells que ja estan familiaritzats amb la sintaxi **CSS**.

El lloc oficial de **jQuery**, http://api.jquery.com/category/selectors/, enumera tots els seus selectors recolzats. La majoria d'aquests són selectors **CSS**, però també hi ha un nombre de selectors extesos disponibles a **jQuery** que no formen part de l'especificació **CSS**.

*	[nom-atribut | = 'valor']	[nom-atribut * = 'valor']	[nom-atribut ~ = 'valor']
[nom-atribut $ = 'valor']	[nom-atribut = 'valor']	[nom-atribut ^ = 'valor']	:checked
pare > fill	.nom-classe	ancestre descendent	:disabled
nom-element	:empty	:enabled	:first-child
:first-of-type	:focus	[nom-atribut]	#id
:lang	:last-child	:last-of-type	[nom-atribut= "valor"] [nom-atribut= "valor"]
selector, selector	prev + next	prev ~ siblings	:not()
:nth-child()	:nth-last-child	:nth-of-type()	:only-child
:only-of-type	:root	:target	

Aquí teniu un recull d'ells.

```
var comptaElements = $("*").css("border", "3px solid red").length;
$("a[hreflang|='ca']").css("border", "3px dotted green");
var n = $("input:checked").length;
$("ul.topnav > li").css("border", "3px double red");
$(".mevaClasse").css("border", "3px solid red");
$("form fieldset input").css("backgroundColor", "yellow")
$("input:disabled").val("això és tot");
$("div").css("border","9px solid red");
$("td:empty")
      .text("És buida!")
      .css("background","rgb(255,220,200)");
$("input:enabled").val("això és tot!");
$("div span:first-child")
      .css("text-decoration", "underline")
      .hover(function() {
            $(this).addClass("tanverd");
      }, function() {
            $(this).removeClass("tanverd");
      });
$("span:first-of-type").addClass("fot");
$("tr:first").css("font-style", "italic");
$("span:first").
      text("Trobats " + hiddenElements.length + " elements ocults en total." );
$("div:hidden").show( 3000 );
$("span:last").text("Trobats " + $("input:hidden").length + " inputs ocults." );
$("span:nth-of-type(2)")
      .append("<span> és el segon germà de span</span>")
      .addClass("nth");
```

I molts més, consulteu la pàgina abans recomenada per obtenir tots els detalls de tots els selectors.

L'ús de selectors extesos de jQuery

A més de selectors **CSS** estàndard, **jQuery** ofereix les seves pròpies extensions per tal de proporcionar mètodes addicionals per seleccionar fàcilment els elements.

:animated	[nom-atribut! = 'valor']	:button	:checkbox
:contains()	:eq()	:even	:file
:first	:gt()	:has()	:header
:hidden	:image	:input	:last

:lt()	:odd	:parent	:password
:radio	:reset	:selected	:submit
:text	:visible		

```
$("td:eq(2)").css("color", "red");
$("tr:even").css("background-color", "#bbf");
$("td:gt(4)").css("backgroundColor", "yellow");
$("td:gt(-2)").css("color", "red");
var input = $(":button").addClass("marcat");
$("div:contains('Joan')").css("text-decoration", "underline");
var input = $("form input:checkbox")
      .wrap( "<span></span>" )
      .parent()
      .css({
            background: "yellow",
            border: "3px red solid"
      });
var input = $("input:file").css({
                  background: "yellow",
                  border: "3px red solid"
            });
var input = $("form input:radio")
      .wrap( "<span></span>" )
      .parent()
      .css({
            background: "yellow",
            border: "3px red solid"
      });
var input = $("input:reset").css({
            background: "yellow",
            border: "3px red solid"
      });
$("<input>").is("[type=text]"); // false o true
$("<input>").is(":text"); // false o true
var input = $("form input:text").css({
            background: "yellow",
            border: "3px red solid"
      });
$("div:visible").click(function() {
      $(this).css("background", "yellow");
});
```

Controladors detectors d'esdeveniments

Els esdeveniments són un aspecte fonamental de la creació d'scripts dinàmics i sensibles. En general, es produeix un esdeveniment quan un usuari realitza algun tipus d'acció com un clic del ratolí, prèmer una tecla, o l'enviament de formularis. Alguns esdeveniments també poden ser provocats per cridar mitjançant programació, una tasca de forma automàtica

La llibreria **jQuery** ofereix una sèrie de característiques que fan que el maneig d'esdeveniments sigui una tasca més fàcil que quan s'utilitza JavaScript pur, sobretot quan es tracta de diferències de manipulació de navegador, sobre tot del **IE**.

En lloc de preocupar-nos de gestors d'esdeveniments basats en les capacitats dels diferents navegadors, podem utilitzar una cosa tan simple com fer **clic()** o **keydown()** per gestionar un esdeveniment, i **jQuery** farà la resta del treball per nosaltres.

Esperant el document que estigui llest

Com ja varem dir amb anterioritat, **jQuery** ens ofereix la funció **ready()** de manera que podrem estar segurs que tots els elements del document es carreguen abans d'executar el codi **jQuery** en ells. D'aquesta manera no rebrem error de Javascript en la interacció amb el lloc web o aplicació.

ready() vs. load()

En **JavaScript**, podem veure sovint scripts utilitzant l'esdeveniment de càrrega en l'objecte **window** per determinar quan tots els elements s'han carregat completament. Per exemple, podem veure alguna cosa semblant al següent codi:

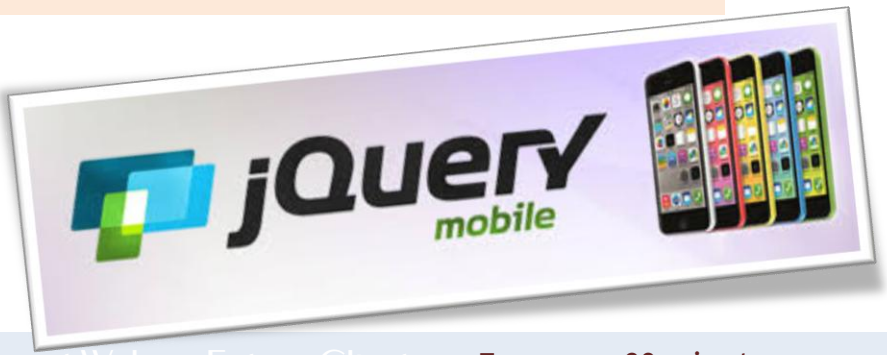
```
window.onload = function() {
      // Codi a executar quan els elements s'hagin carregat
}
```

Alternativament, és possible que vegem una versió més modernitzada, utilitzant la funció **addEventListener()**, com en el següent codi:

```
window.addEventListener("load", function() {
      // Codi a executar quan els elements s'hagin carregat
}, false);
```

Noteu la similitud amb l'ús de **ready()** de **jQuery**, que es mostra en el codi:

```
$(document).ready(function() {
      // Codi a executar quan els elements s'hagin carregat
});
```



Quina és la diferència?

Hi ha una important diferència entre els dos mètodes. El mètode `ready()` ens permet començar a executar l'script tan aviat com tots els elements hagin estat carregats, però no espera a imatges o altres mitjans de comunicació per acabar de carregar-se. L'esdeveniment de càrrega de `JavaScript` espera el document i a tots els mitjans que es carreguin abans d'estar disponible.

Molts scripts es poden executar sense preocupar-se de si les imatges o altres mitjans de comunicació s'han carregat, `ready()` permet donar-nos accés als elements en el document amb més rapidesa que l'esdeveniment de càrrega `load`. Això dóna als usuaris una millor experiència, el que els permet interactuar de forma més ràpida amb el seu lloc web o aplicació.

D'altra banda, quan s'executa un script que funciona amb imatges (com un vídeo), utilitzant `ready()` podria causar resultats inesperats. Si l'script ha de mostrar una imatge que encara no s'ha carregat, podria mostrar una imatge danyada o res en absolut. En casos com aquest, és millor esperar que l'esdeveniment de càrrega `load` es carregui per assegurar-nos que tots els mitjans de comunicació s'hagin carregat abans de ser manipulats.

Per solucionar aquest problema podriem utilitzar el següent codi :

```
$(window).load(function () {
      // Codi a executar quan els elements s'hagin carregat
});
```

Normalment s'utilitza `ready()` per a la majoria dels guions, mentre que el mètode de càrrega `load` s'utilitzarà quan sigui necessari esperar que altres mitjans de comunicació es carreguin.

L'ús de l'argument $ amb ready()

El mètode `ready()` també realitza una tasca més útil per nosaltres. Suposem que estem utilitzant `jQuery`, així com una altra biblioteca JavaScript (com per exemple `prototype`) que utilitza l'identificador `$` en el codi com `jQuery`. En lloc de reemplaçar totes les instàncies de `$` amb `jQuery`, podem cridar `jQuery(document).ready()` i incloure tot el codi `jQuery` dins del mètode `ready()`, enviant `$` com un argument, mireu el següent codi:

```
jQuery(document).ready(function($) {
      // Codi que s'executa. Aquí $ es pot utilitzar com s'espera.
})
```

Això permet que l'altra biblioteca pugui utilitzar l'identificador `$` i ens permet utilitzar-lo amb el nostre codi de `jQuery`. El problema encara pot existir, però.

Quan s'utilitza més d'una biblioteca que utilitza l'identificador `$`, l'últim en el codi `HTML` prendrà el control d'aquest identificador, i el podria fer inutilitzable per a les altres biblioteques. A més d'enviar `$` com argument a `ready()`, `jQuery` proporciona el mètode `noConflict()` per assegurar-se que no es prengui el control de l'identificador `$`.

"Vols dir que jo sóc l'únic que veu un conflicte d'interessos aquí?"

Suposem que tenim el següent codi a la secció de capçalera del document **HTML**:

```
<script type="text/javascript" src="prototype.js"></script>
<script type="text/javascript" src="jquery-1.11.2.min.js"></script>
<script type="text/javascript" src="meucodi.js"></script>
```

La biblioteca prototype, inclosa, controla l'identificador $

El nostre arxiu de JavaScript, que ara no pot usar $ amb les dues biblioteques

La llibreria jQuery, inclosa, pren el control de l'identificador $ des de la biblioteca prototype!

Observem que **prototype** controla inicialment l'identificador **$**, però això canvia ràpidament en la següent línia per la inclusió de **jQuery**. Per arreglar això, podem utilitzar el mètode **noConflict()** de **jQuery**, com en el següent codi:

```
<script type="text/javascript" src="prototype.js"></script>
<script type="text/javascript" src="jquery-1.11.2.min.js"></script>
<script type="text/javascript">
     jQuery.noConflict();
</script>
<script type="text/javascript" src="meucodi.js"></script>
```

Al cridar al mètode noConflict() immediatament després de carregar jQuery li donarà el control de l'identificador $ i el retornarà a la biblioteca prototype

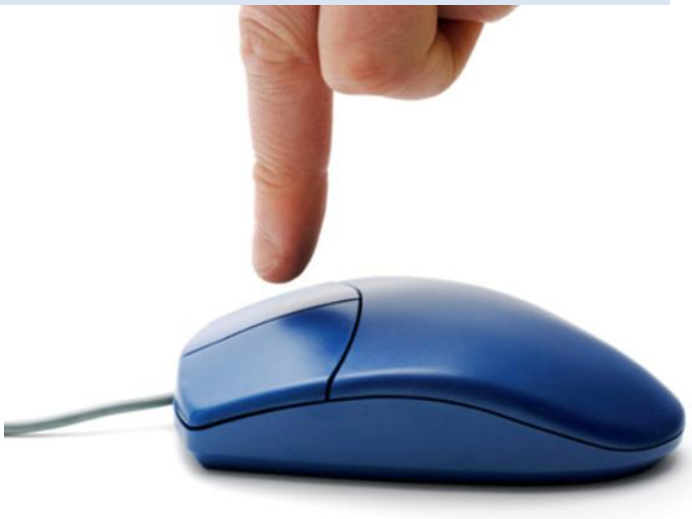
Això permetrà utilitzar **$** per a **prototype**, amb un bonus addicional, encara podrem usar **$** amb el nostre codi **jQuery** enviant **$** com argument al mètode **ready()**, com es va descriure anteriorment!.

NOTA:

Podem cridar al mètode **noConflict()** en el nostre propi arxiu **JavaScript** (*meucodi.js* en aquest cas), si ho preferim; només hem d'assegurar-nos de posar-lo a la primera línia del guió.

Maneig d'un Esdeveniment

La llibreria **jQuery** proporciona diversos mètodes per a la gestió d'esdeveniments. Per als esdeveniments més comuns, **jQuery** té les seves pròpies funcions abreujades, per fer més fàcil la reacció davant d'un esdeveniment d'usuari. Altres esdeveniments es poden afegir utilitzant el mètode **on()**, que discutirem més endavant. En primer lloc, veurem la gestió d'esdeveniments bàsics utilitzant una de les funcions curtes de **jQuery**.



Maneig d'Esdeveniments Bàsics

Suposem que necessitem per dur a terme una acció quan l'usuari faci clic en un element. **jQuery** proporciona el mètode **click()** per manejar aquest esdeveniment. Per exemple, si teníem un element amb un **id** de "**canviar_mida**", podríem utilitzar el següent codi:

```
$("#canviar-mida").click(function() {
      // Codi que s'executa quan es fa clic a l'element
});
```

El mètode **click()** crida a una funció que s'executa quan es produeix l'esdeveniment. El codi dins de la funció reaccionarà a l'esdeveniment clic executant-se.

Aquest script no requerirà de cap mitjans de comunicació per ser carregat, es pot col·locar tot dins de la funció **ready()**, com en el següent codi:

```
$(document).ready(function() {
    $("#canviar-mida").click(function() {
        // Codi que s'executa quan es fa clic a l'element
    });
});
```

La funció ready() envolta tot el codi que s'executa

L'esdeveniment clic es controla amb la funció interna

Utilitzant l'edeveniment clic

Ara que ja hem vist la sintaxi bàsica, es pot aplicar a una pàgina web per que reaccioni davant d'un esdeveniment clic. Poseu el següent codi **HTML** en un arxiu i deseu-lo com a **mida.html**:

```
<!DOCTYPE html>
    <html lang="ca">
    <head>
        <meta charset="utf-8">
        <title>Canviar mida</title>
        <style type="text/css">
            .font-gran { font-size:1.5em; }
        </style>
        <script src="jquery-1.11.2.min.js" type="text/javascript"></script>
        <script src="mida.js" type="text/javascript"></script>
    </head>
    <body>
        <div id="canviar-mida">
            Aquest text és important! Feu clic en aquest text per augmentar-lo, si cal!
        </div>
    </body>
</html>
```

Aquest element tindrà una classe agregada quan es faci clic en ell

Després, poseu el codi següent en un arxiu i deseu-lo com a **mida.js**:

```
$(document).ready(function() {
      $("#canviar-mida").click(function() {
            $("#canviar-mida").addClass("font-gran");
      });
});
```

El mètode click() crida a una funció per manejar esdeveniments de clic a l'element #canviar-mida

Una classe s'afegeix a l'element #canviar-mida quan es fa clic, que en aquest cas serà fer més gran la mida de la font mitjançant la classe .font-gran

Obriu el fitxer mida.html al vostre navegador i feu clic al text. Es farà més gran quan es fa clic, ja que s'agrega una classe, que fa la mida de lletra més gran, en la funció que controla l'esdeveniment.

Si bé aquest guió funciona, no és molt obvi que la mida de la font pot ser ampliada quan fem clic sobre el text. L'usuari hauria de saber que fer clic al text augmentarà la mida de la font. Per millorar la facilitat d'ús i no es perdi polsant a altres llocs, seria útil que quan l'usuari faci clic a un element diferent se li digui a l'usuari el que fa, com per exemple un botó.

Actualitzar el guió de fer clic

Per actualitzar el guió, s'haurà d'alterar tant el codi **HTML** com el codi de **jQuery**. En primer lloc, canviem l'**HTML** dins de l'etiqueta **<body></body>** de la pàgina **mida.html** al següent codi i deseu el fitxer:

```
<div id="canviar-mida">
      Aquest text és important! Feu clic en aquest text per augmentar-lo, si cal!
</div>
<div>
      <form action="alternativa_mida.html">
            <button id="ferMesGran">Ampliació de text</button>
      </form>
</div>
```

S'afegeix un formulari amb un botó etiquetat "**Ampliació de text**", donant a l'usuari una clara acció a prendre per fer més gran el text.

Tingueu en compte que es troba dins d'un formulari, i que l'atribut d'acció del formulari apunta a una pàgina **HTML** alternativa que tindria el text ampliat. Es tracta d'una reserva per a l'accessibilitat en cas que l'usuari no tingui habilitat el **JavaScript**: L'usuari encara serà capaç de veure el text ampliat des del navegador anant a aquesta pàgina alternativa.

> **NOTA:**
>
> Hi ha un nombre de maneres de fer el guió accessible; això només demostra una opció. Altres possibilitats inclouen l'ús d'un guió del costat del servidor per l'acció del formulari, ocultant el botó per als que no tenen **JavaScript** (si la funcionalitat es considera una millora més que una necessitat), i altres opcions.

Per fer que el botó canviï la mida de la font, haurem de modificar l'script de jQuery. Canvieu l'arxiu mida.js utilitzant el codi següent i deseu-lo:

```
$(document).ready(function() {
      $("#fesMesGran").click(function() {
            $("#canviar-mida").addClass("font-gran");
      });
});
```

Obriu el fitxer mida.html al vostre navegador i feu clic al botó. El text canvia de mida, però el navegador va a la pàgina alternativa (potser fins i tot més ràpid del que es produeix el canvi)! Això no és el resultat desitjat, i requerirà un parell de modificacions per tal de solucionar-ho.

Fixant el guió d'actualització de fer clic

L'atribut d'acció especifica que el botó ha d'anar a una segona pàgina HTML (alternativa_mida.html) quan es fa clic. Si bé aquesta és una bona característica per als que no tenen habilitat JavaScript, certament no és el que nosaltres voldríem que faci per als que ho tenen habilitat.

Per solucionar aquest problema, haurem d'enviar l'objecte d'esdeveniment amb la funció de clic com a argument. L'objecte d'esdeveniment que conté la informació sobre l'esdeveniment que es produeix, té un mètode propi anomenat preventDefault() per evitar que l'acció per defecte es produeixi quan es produeix l'esdeveniment especificat.

En primer lloc, necessitem enviar l'objecte d'esdeveniment com un argument a la funció que controlarà l'esdeveniment clic. En JavaScript, simplement posem els arguments entre parèntesis () que segueixen a la paraula clau de la funció, i el mateix succeeix per a jQuery. El següent codi mostra com la funció de crida en el mètode click es pot alterar per passar-ho a l'objecte d'esdeveniment event:

```
$("#fesMesGran").click(function(event) {
```

Observem que l'objecte d'esdeveniment event es col·loca simplement dins dels parèntesis. Com que ja és un objecte definit, no necessita cometes al voltant d'ell com faria un argument de cadena.

El següent que haurem de fer és utilitzar el mètode preventDefault() de l'objecte d'esdeveniment event dins de la funció de gestió de l'esdeveniment click. Això es mostra en aquest codi:

```
$(document).ready(function() {
      $("#fesMesGran").click(function(event) {
            event.preventDefault();
            $("#canviar-mida").addClass("font-gran");
      });
});
```

Aquesta declaració impedeix que es produeixi l'acció per defecte (el que portaria a l'usuari a la pàgina HTML alternativa) i permet a l'usuari romandre a la pàgina actual.

Deseu els canvis i torneu a carregar el fitxer mida.html al vostre navegador. Amb la declaració event.preventDefault(); en el seu lloc, ara serem capaços d'ampliar el text sense sortir de la pàgina actual.

Ara que hem après a manejar un esdeveniment, podem començar a aprendre uns altres esdeveniments jQuery.

Esdeveniments jQuery

La biblioteca **jQuery** ofereix suport pels esdeveniments de **JavaScript** tant a través de mètodes abreujats com mitjançant l'ús del mètode **on()**. Com s'ha vist, els mètodes abreujats són molt útils i són fàcils d'enllaçar amb un esdeveniment d'un element.

Podem aprendre el que cada esdeveniment és i quan passarà mentre un usuari està interactuant amb la nostra pàgina web o aplicació. En primer lloc, hem d'aprendre els esdeveniments que tenen dreceres disponibles; a continuació, aprendrem els esdeveniments restants i com utilitzar el mètode **on()** per enllaçar-lo als elements.

Esdeveniments de ratolí

Els esdeveniments del ratolí ens permeten reaccionar a les accions que l'usuari pot prendre amb el ratolí, com fer clic o moure el ratolí sobre un determinat element de la pàgina. Molts gions fan ús d'un o més d'aquests esdeveniments per interactuar amb els usuaris

Els esdeveniments click i dblclick

Hem estat utilitzant el mètode **click()** per a esdeveniments de clic, de manera que no serà pas cap sorpresa que **dblclick()** fa el mateix per fer l'esdeveniment doble clic del ratolí.

Un doble clic es produeix quan es pressiona el botó del ratolí i s'allibera dues vegades en un període de temps determinat. El temps permès entre els dos clics depèn del sistema, però és típicament un curt període de temps, tal com un mig segon. Els usuaris poden substituir això, en la majoria dels sistemes, a un temps més còmode per a ells (ja sigui més llarg o més curt).

Un exemple del mètode **dblclick()** es mostra en el següent codi:

```
$("#meu-element").dblclick(function() {
      // Codi que s'executa quan es fa clic en #meu-element
});
```

El codi dins de la funció de manipulació s'executarà quan es fa doble clic a l'element.

Llista de mètodes abreujats d'esdevenimants

.blur()	.change()	.click()	.dblclick()
.error()	.focus()	.focusin()	.focusout()
.keydown()	.keypress()	.keyup()	.load()
.mousedown()	.mouseenter()	.mouselesve()	.mouseout()
.mouseover()	.mouseup()	.resize()	.scroll()

.select()	.submit()	.unload()	

Els esdeveniments mousedown i mouseup.

Els esdeveniments **mousedown** i **mouseup** constitueixen una part de l'esdeveniment **click**. Aquests esdeveniments són útils si necessitem capturar una part particular d'un clic del ratolí. Per exemple, **mousedown** es pot utilitzar per ajudar a determinar quan l'usuari ha iniciat l'arrossegament d'un element en lloc de fer clic.

Aquesta és una composició d'un clic del ratolí:

- Es pressiona el botó del ratolí. Es llança l'esdeveniment mousedown.
- Es deixa anar el botó del ratolí. Es llança l'esdeveniment mouseup.
- El botó del ratolí es pressiona i es deixa anar. Es dispara l'esdeveniment click.

Això mostra l'ordre en què es dispara cada esdeveniment, pot ajudar-nos quan necessitem saber quin esdeveniment a reaccionat en diferents situacions. Aquests mètodes abreujats es mostren en el codi:

```
$("#meu-element").mousedown(function() {
      // Codi per executar en mousedown
});
$("#meu-element").mouseup(function() {
      // Codi per executar en mouseup
});
```

Els esdeveniments mouseover i mouseout

Els esdeveniments **mouseover** i **mouseout** es produeixen quan l'usuari mou el ratolí dins un element o es mou fora d'un element. A causa de la forma en què aquests esdeveniments es transmeten a altres elements, en general és preferible utilitzar els esdeveniments **mouseenter** i **mouseleave**.

Els exemples d'aquests mètodes abreujats es mostren en el codi:

```
$("#meu-element").mouseover(function() {
      // Codi per executar en mouseover
});
$("#meu-element").mouseout(function() {
      // Codi per executar en mouseout
});
```

Els esdeveniments mouseenter i mouseleave

Els esdeveniments **mouseenter** i **mouseleave** es produeixen quan l'usuari mou el ratolí dins un element o es mou el ratolí fora d'un element. L'element seleccionat serà sempre un per controlar l'esdeveniment, de manera que

aquests mètodes es recomanen sobre `mouseover` i `mouseout`. El codi per mostrar és exactament igual que l'anterior codi per `mouseover` i `mouseout.`

Una altra abreujament proporcionat per `jQuery` és el mètode `hover()`, que es pot utilitzar per proporcionar els controladors d'esdeveniments per als esdeveniments `mouseenter` i `mouseleave` en un sol mètode. Un exemple es mostra en el següent codi:

```
$("#meu-element").hover(
      function () {
            // Codi per executar en mouseenter
      },
      function () {
            // Codi per executar en mouseleave
      }
);
```

Aquesta funció s'executa quan es produeix l'esdeveniment mouseenter

Aquesta funció s'executa quan es produeix l'esdeveniment mouseleave

`IMPORTANT!`, Noteu la coma entre les dues funcions, que els separa com arguments en el mètode `hover()`.

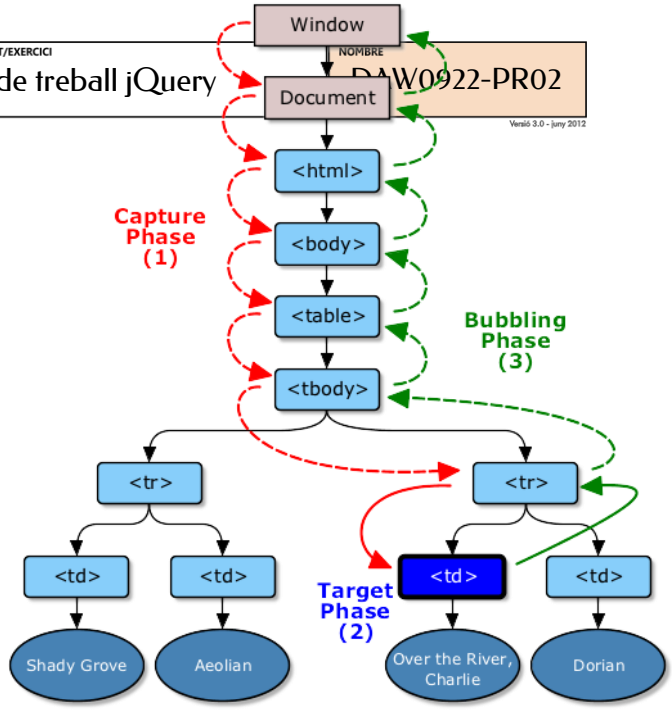
L'esdeveniment mousemove

L'esdeveniment `mousemove` es produeix quan l'usuari té el punter del ratolí sobre un element i el mou. Aquest esdeveniment seguirà passant mentre el punter del ratolí es mogui dins de l'element. Quan el punter del ratolí surt de l'element, l'esdeveniment s'aturarà fins que el punter del ratolí es torni a moure de nou en l'element.

Esdeveniments de teclat

Els esdeveniments de teclat s'activen quan l'usuari prem o deixa anar una tecla en el teclat.

Esdeveniment: Captura i propagació



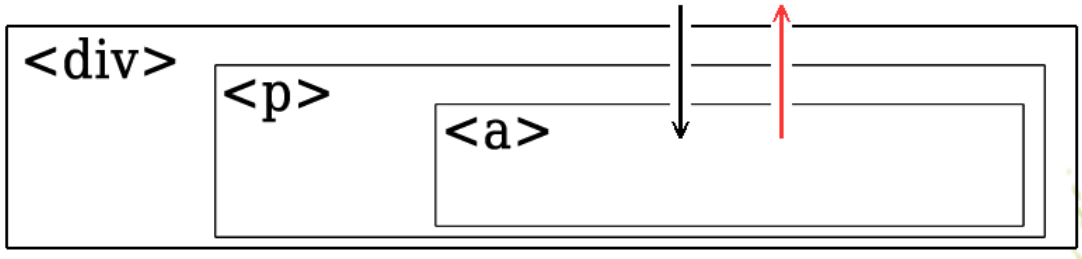
Quan succeeix un esdeveniment, hi ha dues fases: **captura** i **propagació**. Aquests s'utilitzen quan diversos elements poden registrar l'esdeveniment, i és típicament el resultat d'elements imbricats, com en el següent codi (escriviu aquest codi en un arxiu i deseu-lo com **bubble.html**):

```
<!DOCTYPE HTML>
<html lang="ca">

    <head>
        <meta charset="utf-8">
        <title>Exemple de bombolla</title>
        <style>
            #exterior {
                border:1px solid #000;
                background-color:#00FF33; padding:20px; }
            #enmig {
                border:1px solid #000;
                background-color:#0000FF;
                padding:10px; }
            #interior {
                border:1px solid #000;
                background-color:#FFFFFF;
                padding:5px; }
        </style>
        <script src="jquery-1.11.2.min.js" type="text/javascript"></script>
        <script src="bubble.js" type="text/javascript"></script>
    <body>
        <div id="exterior">
            <p id="enmig">
                <a id="interior" href="page.html">Mostrar Missatge</a>
            </p>
        </div>
        <div id="text"></div>
    </body>
</html>
```

Cadascun d'aquests elements pot gestionar un esdeveniment com mouseout.

En un cas com aquest, els elements **<a>**, **<p>** i ** ** poden registrar un esdeveniment comú, com ara **mouseout**. L'últim element ** (#text)** no es veu afectat si un esdeveniment s'afegeix a qualsevol dels altres elements, ja que no està dins de la imbricació.

En la fase de captura, un esdeveniment serà enviat a l'element menys específic que pot gestionar i després ho transmès a cada nivell fins que arribi a l'element més específic que pot manejar (en el nostre cas va des de ** ** a **<p>** fins a **<a>**). En la fase de propagació, un esdeveniment serà enviat a l'element més específic que pot manejar i després passarà a cada nivell fins que arribi a l'element menys específic que pot gestionar (en el nostre cas va des **<a>** a **<p>** fins a ** **).



JavaScript permet especificar una fase o l'altre per controlar els esdeveniments, però els navegadors (especialment els més vells) potser no segueixi això, en lloc d'això utilitza la fase predeterminada d'aquest navegador..

Per ajudar a eliminar la confusió, jQuery sempre utilitza la fase de propagació en registrar els esdeveniments, el que fa que el maneig d'esdevenimants es faci a través dels navegadors.

Guió d'un mouseout

L'altre problema es produeix quan s'utilitzen mètodes com `mouseover()` o `mouseout()`, que no aturen la formació de continuada de propagacions. Això vol dir que si es registra un esdeveniment `mouseout` en l'element `<div>` del codi d'exemple, la propagació farà que l'esdeveniment es quedi registrat primer en l'element `<a>`, ja que és l'element més específic que pot gestionar l'esdeveniment. A continuació, registra el mateix controlador d'esdeveniments per a l'element `<p>` i després a l'element `<div>`, pel qual va ser pensat originalment l'esdeveniment.

Per veure el que passa, col·loqui el següent codi jQuery en un arxiu i guardar-lo com `bubble.js`:

```
$(document).ready(function() {
	$("#exterior").mouseout(function() {
		$("#text").append("Mouseout!<br>");
	});
});
```

Obriu el fitxer `bubble.html` al navegador i moveu el ratolí dins i fora dels tres elements (que són de color i envoltats per bores per ajudar a visualitzar el que està succeint).

Us adonareu que cada vegada que es mou el punter del ratolí fora de qualsevol dels tres elements, s'afegeix un nou missatge a l'últim element `<div>(#text)`. Això probablement no és la funcionalitat que estàvem buscant. El més probable és que preferim que això només passi quan el punter del ratolí surti de l'element de la intenció original, que era l'element `<div>(#exterior)`.

> NOTA:
>
> La funció append() de jQuery afegeix contingut a un element després de qualsevol contingut que ja existent. Aprendrem més sobre aquest mètode a mesura que avançem.

L'obtenció de la funcionalitat desitjada amb mouseleave

Afortunadament, jQuery proporciona el mètode `mouseleave()`, que no continua amb la propagació, sinó que utilitza l'esdeveniment només en l'element desitjat. Per exemple, actualitzeu l'arxiu `bubble.js` arxiu per utilitzar el codi següent i deseu-lo.

```
$(document).ready(function() {
	$("#exterior").mouseleave(function() {
		$("#text").append("Mouseout!<br>");
	});
});
```

Actualitzeu la pàgina `bubble.html` al vostre navegador i tracteu de moure el ratolí fora dels tres elements del `<div>` de nou. Aquesta vegada, el missatge només ha d'aparèixer quan es mou el punter del ratolí fora de l'etiqueta `<div>` element exterior.

Utilitzar el mètode on() per controlar els esdeveniments

El mètode `on()` és relativament nou en `jQuery`. Aquest mètode va ser introduït en `jQuery 1.7` per reemplaçar a altres mètodes de controladors d'esdeveniments d'unió als elements: `bind()`, `delegate()`, i `live()`.

És molt comú veure a aquests utilitzats en guions, especialment els escrits per a versions de `jQuery` abans de `1.7`, de manera que haurem d'estar familiaritzats amb ells si necessitem actualitzar o depurar un script que les utilitza.

El mètode bind()

El mètode `bind()`, vincular, s'utilitza per enllaçar un controlador d'esdeveniments a un element que existeix en l'actualitat. Això vol dir que qualsevol element usant `bind()` ha d'existir, aquest mètode no porta el compte de la possibilitat que l'element s'agregi, més tard, mitjançant programació, una mica com ho fa `delegate()`.

Un exemple de l'ús del mètode `bind()` es mostra en el següent codi:

```
$("#meu-element").bind("click", function() {
      // Codi que s'executa quan es fa clic a #meu-element
});
```

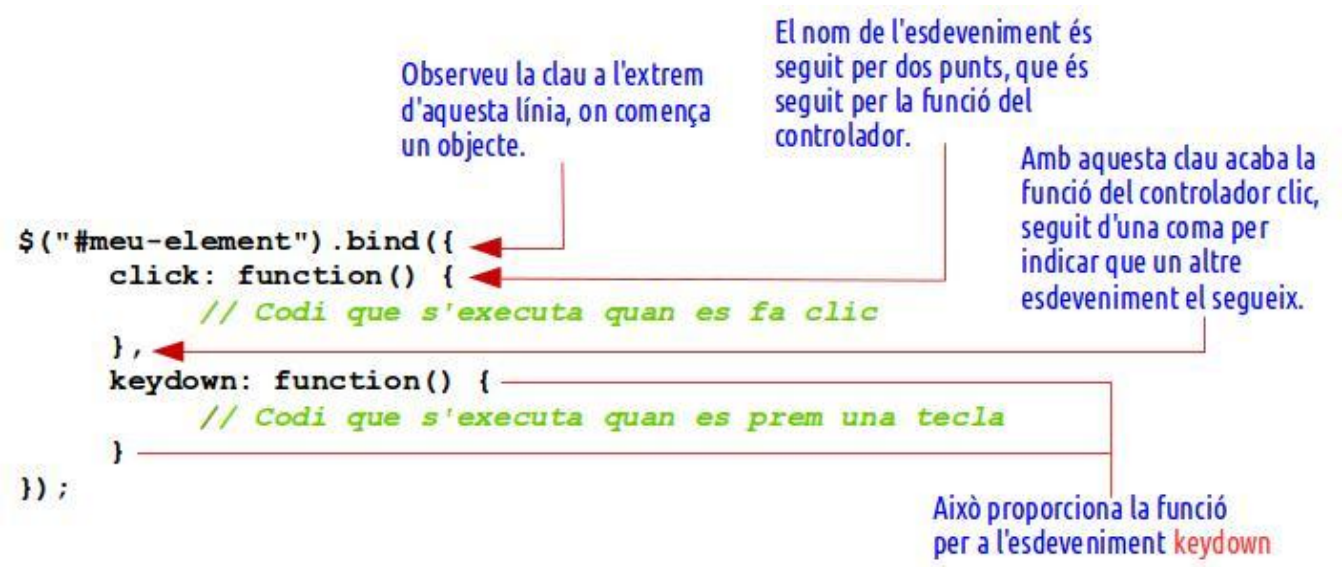
Mentre que el mètode abreujat `click()` pren només la funció de maneig com a argument, el mètode `bind()` pren dos arguments: el nom de l'esdeveniment i la funció per controlar l'esdeveniment.

El podeu personalitzar encara més utilitzant la mateixa funció de controlador per dos esdeveniments diferents, com en el següent codi:

```
$("#meu-element").bind("click keydown", function() {
      // Codi que s'executa quan es fa clic o es prem una tecla a #meu-element
});
```

Aquest codi executaria la funció quan es fa clic al botó del ratolí sobre l'element o quan l'usuari prem una tecla mentre és a dins d'aquest element.

En **jQuery 1.4** i posteriors, podem utilitzar la notació objecte per unir o vincular a diversos esdeveniments que utilitzen diferents funcions de maneig a un element. El següent codi mostra un exemple d'això:



En **JavaScript**, estructures com les funcions i els objectes es poden passar com a arguments. Aquesta estructura passa un objecte usant la notació literal d'objectes, que li permet definir una llista de propietats i valors. Aquí, les propietats són noms d'esdeveniments i els valors són funcions del controlador.

Finalment, si voleu eliminar un controlador d'esdeveniments d'un element, potdem utilitzar el mètode de desvinculació **unbind()**. Si volem desenllaçar (desvincular) tots els esdeveniments d'un element, només ens cal cridar al mètode **unbind()** en l'element, com en el següent codi:

```
$("#meu-element").unbind();
```

Si voleu eliminar un tipus d'esdeveniment, però mantenir els altres, podeu especificar el tipus d'esdeveniment per eliminar passant-lo com a argument, com en el següent codi:

```
$("#meu-element").unbind("click");
```

Això eliminarà tots els controladors d'esdeveniments **click** de l'element.

El mètode delegate()

El mètode **delegate()** ens permet enllaçar un controlador d'esdeveniments a qualsevol element, encara que no existeixi fins a més tard a causa de ser afegit a través de la programació. Utilitza un selector per obtenir els elements pares, i després, un altre selector, es passa pels elements secundaris que existeixen o s'afegiran.

En general, això succederia amb una taula, on potser s'hagi de afegir files més endavant sobre la base de la informació subministrada per l'usuari.

El següent codi mostra un exemple del mètode de **delegate()**:

```
$("#meva-taula").delegate("td", "click", function() {
      // Codi a executar quan es fa clic a un element <td>
});
```

Això assignarà la funció del controlador per reaccionar a tots els esdeveniments de **clic** a tots els elements **<td>** dins de la taula, si ja existeixen o s'afegeixen al document posteriorment.

Podem treure els controladors definits usant **delegat()**, utilitzant el mètode **undelegate()**. Això funciona de manera similar **unbind()**, com es mostra en el següent codi:

```
$("#meva-taula").undelegate(); // Desvincular tots els controladors d'esdeveniments
```

O un esdevenimernt específic

```
$("#meva-taula").undelegate("click");     // Desvincular tots els controladors
                                          // d'esdeveniments click
```

Aquest es crida al mètode delegate perquè delega la gestió d'esdeveniments d'element especificat en l'element seleccionat utilitzant la propagació d'esdeveniments. Considereu el següent codi:

```
$("#meva-taula tr").delegate("td", "click", function() {
      // Codi a executar quan es fa clic a un element <td>
});
```

Aquí, el **<td>** que es passa com a primer argument, assigna el maneig de l'esdeveniment **clic** a l'etiqueta **<tr>** que l'envolta. Tots els elements **<td>** dins de la selecció (el **#meva-taula <tr>** en aquest cas) delegarà l'esdeveniment **clic** a l'etiqueta **<tr>** que l'envolta.

En delegar l'esdeveniment al seu element exterior, tots els elements **<td>** poden utilitzar la funció de maneig per a l'esdeveniment **clic**, fins i tot si s'afegeixen al document posteriorment.

El mètode live()

El mètode **live()** funciona com el mètode **delegate()**, però utilitza un mètode més lent vinculant esdeveniments en l'element de document. La documentació de l'**API** de **jQuery** recomana l'ús **on()** o **delegate()** en el seu lloc.

El mètode on()

El mètode **on()** combina els mètodes **bind()** i **delegate()**, que ens permet utilitzar un únic mètode si voleu utilitzar els elements actuals i /o futurs. La diferència està en si un argument és present per dir-li a **jQuery** per delegar el controlador a un element en particular.

L'ús del mètode `on()` per enllaçar un controlador d'esdeveniments: Podeu utilitzar el mètode `on()` de la mateixa manera com ho varem fer amb el mètode `bind()`, simplement cridant-la com en el següent codi:

```
$("#meu-element").on("click", function() {
      // Codi que s'executa quan es fa clic a #meu-element
});
```

En la mateixa manera que `bind()`, podem utilitzar-lo per enllaçar diversos esdeveniments a elements, com en el següent codi:

```
$("#meu-element").on({
      click: function() {
            // Codi que s'executa quan fem clic a #meu-element
      },
      keydown: function() {
            // Codi que s'executa quan es prem una tecla a #meu-element
      }
});
```

L'ús del mètode `on()` per delegar un gestor d'esdeveniments: Per utilitzar el mètode `on()` per a la delegació d'esdeveniments, només hem d'afegir un argument al mètode, com en el següent codi:

```
$("#meva-taula tr").on("click", "td", function() {
      // Codi a executar quan es fa clic a un element <td>
});
```

En aquest cas, es passa un argument després del nom de l'esdeveniment, el qual determina quins elements seran delegats en l'esdeveniment per a l'element seleccionat (s). En aquest cas, qualsevol element `<td>` delegarà l'esdeveniment `clic` per l'element `<tr>` que l'envolta.

L'ús del mètode `off()` per desenllaçar (`unbind` o `undelegate`) un gestor d'esdeveniments: Igual que amb els mètodes `unbind()` i `undelegate()`, podem utilitzar `off()` per eliminar un controlador d'esdeveniments d'un element afegexit amb el gestor usant `on()`.

Per eliminar tots els controladors d'esdeveniments d'un element, podem utilitzar el següent codi:

```
$("#meu-element").off();
```

Si voleu eliminar un tipus d'esdeveniment, podem utilitzar el següent codi:

```
$("#meu-element").off("click");
```

Finalment, si volem eliminar un esdeveniment delegat, podem utilitzar el següent codi:

```
$("#meva-taula tr").off("click", "td");
```

En aquest exemple eliminarà l'esdeveniment `clic` delegat dels elements `<td>` als seus voltants dels elements `<tr>` dins `#meva-taula`.

Altres esdeveniments

A més dels esdeveniments disponibles amb els mètodes abreujats, hi ha altres esdeveniments que es poden cridar usant el mètode `on()` i el nom de l'esdeveniment.

contextmenu	copy	cut	mousewheel
paste	reset		

Aquests són els esdeveniments de `JavaScript` per a la qual `jQuery` no té cap funció abreujada definida. Aquests esdeveniments tendeixen a ser utilitzats amb menys freqüència que els altres, però es poden utilitzar en la majoria dels navegadors moderns, proporcionant el nom de l'esdeveniment en el mètode `on()`.

Per exemple, per assignar un controlador d'esdeveniments per a l'esdeveniment de `reset`, podríem utilitzar el següent codi:

```
$("#meu-formulari").on("reset", function() {
      // Codi que s'executa quan #meu-formalari es restableix (reset)
});
```

L'esdeveniment `reset` està lligat a un element `<form>`, aquest codi d'exemple s'executaria quan el formulari amb un id de `#meu-formulari` sigui restablert per l'usuari.

Afegeix controladors d'esdeveniments

Aquest projecte ens permet practicar l'ús de mètodes d'esdeveniment `jQuery` afegint un controlador d'esdeveniments utilitzant un mètode abreujat i un altre utilitzant el mètode `on()`.

Poseu el següent codi `HTML` a vostre editor i deseu el fitxer com `esdeveniments.html`:

```
<!DOCTYPE html>
<html>
      <head>
            <meta charset="utf-8">
            <title>Projecte d'esdeveniments</title>
            <style type="text/css">
                  .font-negreta { font-weight:bold; }
            </style>
            <script src="jquery-1.11.2.min.js" type="text/javascript"></script>
            <script src="esdeveniments.js" type="text/javascript"></script>
      </head>
```

```
      <body>
            <div id="imp-text">
                  Aquest text és important! Feu clic al botó per posar-lo en
negreta, si cal!
            </div>
            <div>
                  <form action="negreta.html">
                        <button id="fer-negreta">Fer el text en Negreta</button>
                  </form>
            </div>
            <div id="missatge"></div>
            <div>
                  <form action="missatge.html">
                        <button id="mostrar-msg">Mostrar Missatge</button>
                  </form>
            </div>
      </body>
</html>
```

A l'arxiu **esdeveniments.js**, afegiu codi **jQuery** que afegirà la classe font-negreta al primer element ** ** quan es faci **clic** al primer element **<button>**. Utilitzeu el mètode abreujat **click()**. Assegureu-vos d'evitar l'acció per defecte que es produeixi.

Afegir codi **jQuery** que s'annexarà el text "Hola, espero que t'hagi agradat el text important!" a l'element **div #missatge** quan es faci **clic** al **<button> #mostrar-msg** . Utilitzeu el mètode **on()**. Assegureu-vos d'evitar l'acció per defecte que es produeixi.

Deseu el fitxer. Quan estigui complet, que ha de ser similar aquest codi:

```
$(document).ready(function() {
      $("#fer-negreta").click(function(event) {
            event.preventDefault();
            $("#imp-text").addClass("font-negreta");
      });
      $("#mostrar-msg").on("click", function(event) {
            event.preventDefault();
            $("#missatge").append("Hola, espero que t'hagi agradat el text important!");
      });
});
```

Obriu el fitxer **esdeveniments.html** en el vostre navegador web. Feu **clic** al primer botó i el text important s'ha de convertir en negreta. Feu **clic** al segon botó i el missatge "Hola, espero que t'hagi agradat el text important!", s'ha d'inserir en el **div** de sobre del botó.

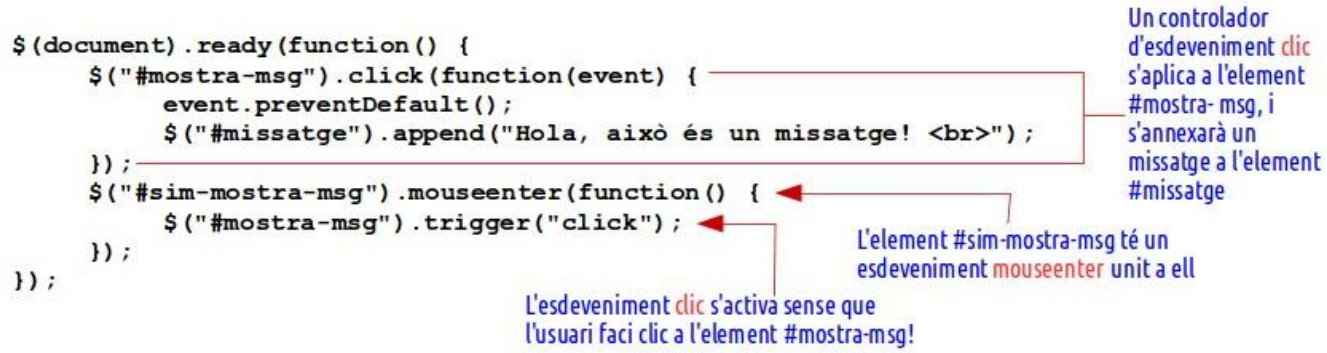
Activació d'esdeveniments

A vegades és útil ser capaç de simular un esdeveniment que succeeix sense que l'usuari hagi de realitzar l'acció.

jQuery proporciona el mètode **trigger()** que ens permet activar un esdeveniment sense que l'usuari realitzi l'esdeveniment real. Un exemple d'aquest mètode es mostra en aquest codi:

```
$("#meu-element").trigger("click");
```

Això farà que l'esdeveniment **clic** es dispari sobre l'element **#meu-element**. S'executarà qualsevol controlador d'esdeveniments per a l'esdeveniment **clic** que s'uneixi a l'element. Per exemple, consideri el següent codi (deseu-lo com **trigger.js**):



Un controlador d'esdeveniments clic s'assigna a l'element **#mostra-msg**, que mostrarà un missatge. Un controlador d'esdeveniments **mouseenter** s'aplica a un altre element (**#sim-mostra-msg**), que utilitza el mètode **trigger()** per activar l'esdeveniment **clic** a l'element **#mostra-msg**.

Utilitzant el següent codi **HTML** amb el codi **jQuery** li permetrà veure això en acció. Deseu el fitxer **HTML** com **trigger.html** i l'arxiu de **JavaScript** com **trigger.js**.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>Example</title>
		<style type="text/css">
			#sim-mostra-msg { border:1px solid #000; width:50%; }
		</style>
		<script src="jquery-1.11.2.min.js" type="text/javascript"></script>
		<script src="trigger.js" type="text/javascript"></script>
	</head>
	<body>
		<div id="sim-mostra-msg">
			Moure el punter del ratolí aquí per evitar fer clic!
		</div>
		<div id="missatge"></div>
```

```
            <div>
                  <form action="missatge.html">
                        <button id="mostra-msg">Mostra Missatge</button>
                  </form>
            </div>
      </body>
</html>
```

Obriu el fitxer trigger.html en el vostre navegador web. Moveu el punter del ratolí sobre l'element que limita el `<div>` o feu `clic` al botó per a mostrar el missatge. Cada vegada que realitzi qualsevol d'aquestes accions, el missatge s'afegirà de nou a l'element `#missatge`.

Treballar amb estils

Els Fulls d'Estil en Cascada (CSS) jugen un paper important en el disseny de les pàgines web, i jQuery ens permet treballar amb la seva sintaxi de moltes maneres per satisfer els nostres objectius de scripting. Ja ha vist com la funció `$()` ens permet seleccionar elements mitjançant selectors CSS. A més d'això, jQuery ofereix una sèrie de mètodes que ens donen la capacitat d'obtenir els valors de les propietats CSS o per alterar aquestes propietats.

Regles CSS

Com s'ha vist anteriorment, CSS ens permet seleccionar els elements mitjançant l'ús de noms d'elements, classes, identificadors, i altres tipus de selectors. Quan es treballa amb codi CSS, es defineix el selector i després es defineix una regla dins d'un conjunt de claus (`{}`). Una regla es compon d'un o més parells de valors de propietat i, cada parell separats per un punt i coma. Per exemple, el següent codi mostra com una definició típica full d'estil es veu en el codi CSS:

```
selector { propietat: valor; propietat: valor; }
```

El mètode css

La llibreria jQuery ofereix el mètode `css()`, que permet obtenir valors o realitzar modificacions sobre la marxa. Com es recordarà, l'ús de `addClass()` requereix que la classe a afegir ja existeixi al codi CSS. Aquest no és el cas amb el mètode `css()`; que ens permet realitzar canvis en les propietats específiques sense la necessitat de definir quins són els valors que estaran en el codi CSS per endavant.

El mètode `css()`, igual que els altres mètodes jQuery CSS, ens permet cridar-la de dues maneres diferents. Una manera per obtindra els valors i una altra d'establir els valors. En el cas del mètode `css()`, cridant amb un string o un argument literal de matriu obtindrà els valors, i cridant amb dos arguments de sèrie o amb un objecte literal, establirà els valors.

Aconseguir valors

Si simplement volem obtenir el valor d'una propietat, podem cridar al mètode `css()` amb el nom de la propietat com argument, com es mostra en el següent codi:

```
var eColor = $("#meu-element").css("color");
```

En aquest exemple, el mètode retornarà el valor de la propietat de color per l'element amb un `id` de `meu-element`. Per tal de fer ús del valor, el resultat retornat s'assigna a una variable de `JavaScript` anomenada `eColor`.

Per utilitzar el valor, es pot afegir a un element perquè es mostri a la pàgina, com en el següent codi:

> **NOTA:**
>
> En obtenir els valors de propietat, no és capaç d'utilitzar les propietats resumides de **css**. Per exemple, si necessitem obtenir el farciment en tots els costats d'un element, s'haurà d'utilitzar les quatre propietats de farciment (padding-top, padding-right, padding-bottom, i padding-left) en lloc de la propietat padding abreujada.

```
var eColor = $("#meu-element").css("color");   // El valor de la propietat de color
                                               // per #meu-element se li assigna a
                                               // una variable anomenada eColor.

$("#altre-element").append(eColor);// El valor s'escriu a la pàgina afegint el text
                                   // a l'element #altre-element. Tingueu en compte
                                   // que eColor és un nom de variable, per tant no
                                   // necessita estar dins de les cometes.
```

Aquest codi tant assigna un valor a una variable de `JavaScript` com utilitza el valor de la variable. Quan s'utilitza el valor d'una variable, no cal tancar-la entre cometes.

PRECAUCIÓ: Els diferents navegadors poden tornar diferents cadenes per al valor d'una propietat. Per exemple, el valor de color per al negre podria ser retornat com **`#000000`**, **`rgb(0,0,0)`**, o d'una altra cadena. Un plugin com **jQuery Color** pot ajudar a obtenir un valor consistent si cal comparar els colors.

Si necessitem obtenir múltiples valors, podem utilitzar un literal de matriu com a argument, com en el següent codi:

```
var eColor = $("#meu-element").css(["color", "background-color"]);
```

Això retornarà una matriu en lloc d'una cadena. En aquest cas, `eColor` se li assignarà una matriu, que li permet accedir al valor de `color` utilitzant `eColor[0]` i el valor de `background-color` utilitzant `eColor[1]`.

> **NOTA:**
>
> Si la selecció en la funció `$()` coincideix amb més d'un element, només el valor de la propietat serà retornat per al primer element coincident.

Col·locar valors

La funció `css()` també ens permet col·locar els valors de les propietats dels elements seleccionats.

Això pot ser útil quan no tenim accés al codi `CSS` o simplement necessitem canviar ràpidament un o més valors de la propietat sobre la marxa.

Establir un sol valor de la propietat

Si desitjem establir un únic valor de la propietat, hem d'enviar, el mètode `css()`, dos arguments: el nom de la propietat i el valor que l'establim. Per exemple, per establir la propietat `color` a `#FF0000`, utilitzariem el següent codi:

```
$("#meu-element").css("color", "#FF0000");
```

En establir un color, es pot utilitzar qualsevol valor vàlid CSS. En el cas del color vermell, valors com `#FF0000`, `#F00`, `red`, `rgb(100%, 0%, 0%)`, i `rgb(255,0,0)` són formes vàlides d'establir un valor en vermell .

A més, els noms de propietat amb guions poden ser escrits en CSS o la sintaxi de JavaScript. Per exemple, la propietat `background-color` en CSS es representa com `backgroundColor` en JavaScript (s'elimina el guió i la següent lletra apareix en majúscules). Aquest mateix patró es manté per a totes les propietats amb guió: `margin-left` per `marginLeft`, `border-right-width` per `borderRightWidth`, i així successivament.

Si utilitzem la sintaxi de JavaScript, les cometes són opcionals al voltant del nom de la propietat. Com a resultat, cada línia de jQuery en el codi següent seria establir el color de fons de l'element a vermell:

```
$("#meu-element").css("background-color", "#FF0000"); // Sintaxi CSS.
$("#meu-element").css("backgroundColor", "#FF0000");  // Sintaxi Javascript
                                                      // amb cometes.
$("#meu-element").css(backgroundColor, "#FF0000");    // Sintaxi Javascript
                                                      // sense cometes.
```

Per ser coherent, el millor és triar una convenció i fer-la servir durant tot el nostre codi. Això farà que sigui més fàcil d'editar o depurar més tard si cal. En aquesta pràctica, fem servir la sintaxi CSS per a noms de propietat, però no dubteu a utilitzar un dels altres si us fan sentir més còmode.

> **NOTA:**
>
> Si la selecció en la funció `$()` coincideix amb més d'un element, la propietat especificada s'establirà per a tots els elements coincidents en la selecció.

Establir diversos valors de propietats

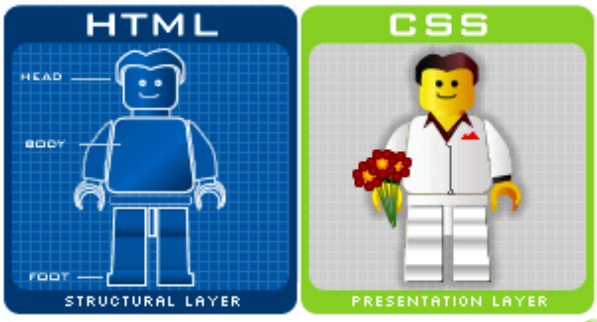
Si voleu configurar diversos valors de propietats a la vegada, podeu enviar un sol argument per al mètode `css()` en la forma d'un objecte literal. Un objecte literal tanca noms i valors de propietat entre claus (`{}`).

El format general per a la notació literal d'objectes té el següent codi:

```
{ propietat: valor, propietat: valor }
```

Observeu que cada parella de `propietat/valor` està separada per `una coma`. Cada propietat i el seu valor corresponent estan separats per `dos punts`. Tot el conjunt de parells de `propietat/valor` es tanca entre `claus`.

Per utilitzar això amb el mètode `css()`, només hem d'enviar un literal d'objecte com a argument.



Suposeu que voleu canviar la propietat `color` a blanc i la propietat `background-color` a negre. Podriam utilitzar el següent codi:

```
$("#meu-element").css({"color": "#FFFFFF", "background-color": "#000000"});
```

Per facilitar la lectura (especialment quan la llista s'allarga), és possible que vulgueu col·locar cada parella de `propietat/valor` en la seva pròpia línia, com en el següent codi:

```
$("#my-element").css({                  // La clau d'obertura és en aquesta línia.
      "color": "#FFFFFF",               // Primer parell propietat/valor.
      "background-color": "#000000"     // Segon parell propietat/valor.
});                                     // La clau final de tancament del mètode css().
```

L'ús de valors relatius

En el `jQuery 1.6` i superior ens permet utilitzar valors començant amb `+=` o `-=` per establir els valors relatius (establint el valor incremental superior o inferior al seu valor actual) per a les propietats que es basen en valors numèrics, com `width`, `height`, `font-size`, i així successivament.

Els valors que es passen amb `+=` o `-=` seran interpretats per `jQuery` en píxels. Com que no sempre es pot saber el valor actual en píxels d'amplada, alçada i altres propietats (l'element pot tenir el seu ample definit amb un percentatge o un altre tipus de valor), això ens permet incrementar el valor del píxel, en lloc d'endevinar si es passa un valor de píxels en concret, fent un valor de propietat més gran o més petit.

Per exemple, suposeu que teniu el següent codi `HTML` (deseu-lo com `expand.html`):

```
<!DOCTYPE html>
<html>
      <head>
            <meta charset="utf-8">
            <title>Expand</title>
            <style type="text/css">
                  #expandible { width:20%; border:solid 1px #000; }
            </style>
            <script src="jquery-1.11.2.min.js" type="text/javascript"></script>
            <script src="expand.js" type="text/javascript"></script>
      </head>
      <body>
            <div id="expandible">
                Aquest div es pot engrandir! Feu clic al botó de sota per fer-lo més ample!
            </div>
            <div>
                  <form action="expandid.html">
                        <button id="engrandir">Fer-lo Gran</button>
                  </form>
            </div>
      </body>
</html>
```

Com hem aprendrés, es pot assignar un esdeveniment clic per a l'element de botó i després realitzar-lo. El següent codi mostra com es pot utilitzar **css()** per ampliar l'element ** #expandible** amb **50** píxels cada vegada que es fa clic al botó (deseu-lo com **expand.js**):

```
$(document).ready(function() {
      $("#engrandir").click(function(event) {
            event.preventDefault();
            $("#expandible").css("width", "+=50em");
      });
});
```

Obriu el fitxer **expand.html** al vostre navegador i feu **clic** al botó. Cada vegada que feu **clic**, l'amplada de l'element ** ** augmentarà, donant al text més espai per expandir-se en la mateixa línia.

Podem fer això amb múltiples valors de la propietat també. Com es recordarà, podem establir múltiples valors de les propietats utilitzant notació literal d'objectes. Per tant, si volem establir el farciment a l'esquerra per ampliar juntament amb l'ample, es pot editar el fitxer **JavaScript** per utilitzar el següent codi:

```
$(document).ready(function() {
      $("#engrandir").click(function(event) {
            event.preventDefault();
            $("#expandible").css({
                  "width", "+=50em",
                  "padding-left": "+=10"
            });
      });
});
```

Dues propietats diferents s'incrementen utilitzant notació literal d'objectes.

Deseu el fitxer **expand.js** i torneu a carregar **expand.html** al vostre navegador. En fer clic al botó, l'element ** ** s'ampliarà i hi haurà un encoixinat addicional a l'esquerra dels continguts.

L'ús de funcions per establir els valors

En el **jQuery 1.4** i vesions superiors ens permet establir el valor de retorn d'una funció com el valor de la propietat. Això pot ser útil si es necessita realitzar els passos addicionals o càlculs per determinar quin ha de ser el valor de la propietat.

Un exemple es mostra en aquest codi:

```
$("#meu-element").css("width", function() {     // La funció comença aquí
      var nouAmple;                              // Es defineix una variable
      // Realitzar càlculs ...                   // Càlculs per alterar el valor de la variable
      return nouAmple;                           // Es retorna el valor de la variable, que s'establirà
});                                              // com el valor de la propietat en el mètode css().
```

En aquest cas, es defineix una variable anomenada **nouAmple**, es realitzen els càlculs, i es retorna el valor de la variable **nouAmple**, que fixarà el valor de la propietat d'ample per a l'element seleccionat. Això pot ser una forma pràctica de calcular o determinar un nou valor per a una propietat.

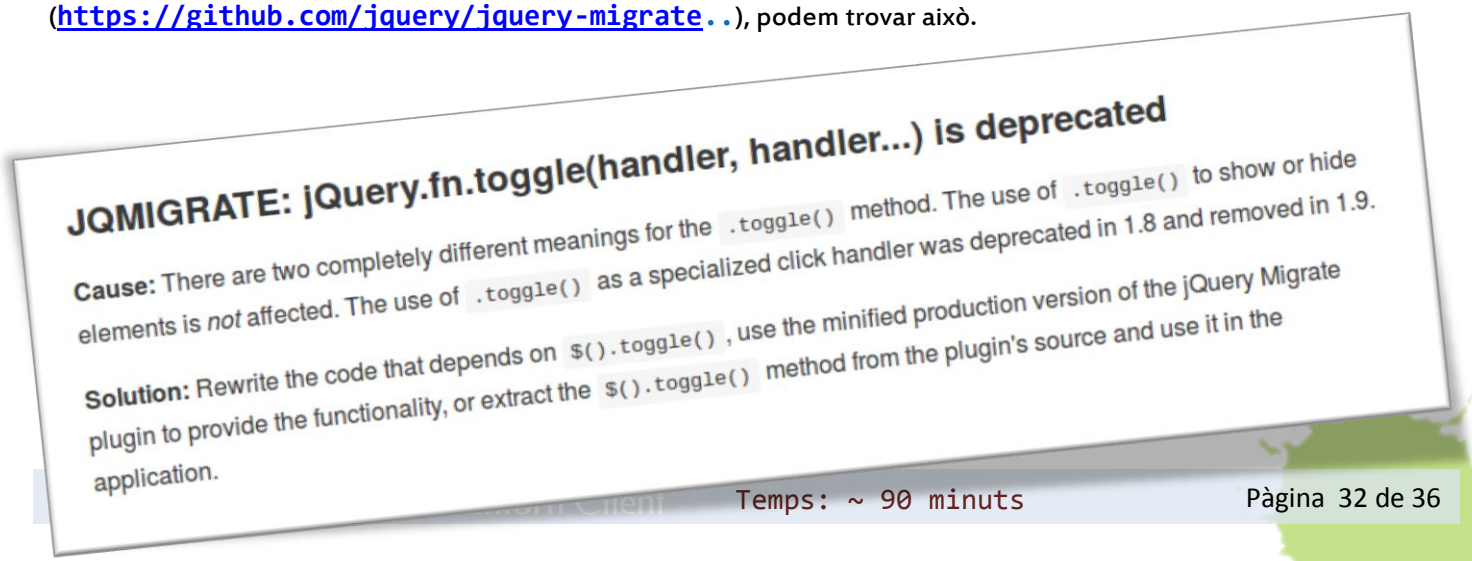
I finalment.

Es evident que aquesta pràctica no és un llibre detallat de **jQuery**. Ens deixem la majoria de mètodes que podem trobar en aquest marc de treball i que vosaltres hauríeu de seguir esbrinant per aconseguir un grau de destresa major. Coses com ara:

- Els mètodes de classe.
- Els mètodes de mida i posició.
- Animacions i efectes.
- L'objecte d'esdeveniments en jQuery.
- Propietats i mètodes d'esdeveniments.
- Mètodes d'elements de formulari.
- Mètodes d'esdeveniments de formulari.
- Propietats i mètodes per AJAX.
- Utilització i creació de connectors (plugins).

Esperem que completeu, seriosament, el vostre aprenentatge amb llibres i llocs web, així com la consulta permanent de la pàgina web de documentació de l'**API** de **jQuery**. En l'aula virtual del **Moodle**, en els recursos de la **UF**, podeu trobar el llibre gratuït, traduït al castellà anomenat "**Fundamentos de jQuery**" de **Rebecca Murphey**, així com una adreça d'una acadèmia per aprendre **jQuery** des de zero de forma gratuïta **on-line**.

Per últim comentar-vos sobre un mètode que ha sigut declarat en desús de la biblioteca actual, bé més concretament des de **jQuery versió 1.9**. Es tracte del mètode **toggle()**, d'una part d'ell, que pot fer que vells guions no funcionin com haurien. Encara que això té diferents solucions. Una, com hem dit al principi de la pràctica, es la utilització de la llibreria **jQuery-migrate,** per casos com aquest. Consultant la pàgina del projecte (**https://github.com/jquery/jquery-migrate..**), podem trobar això.



Aquí teniu la traducció:

JQMIGRATE: jQuery.fn.toggle (controlador, manipulador ...) està en desús

Causa: Hi ha dos significats completament diferents per al mètode **`.toggle()`**. L'ús de **`.toggle()`** per mostrar o ocultar els elements no es veu afectada. L'ús de **`.toggle()`** com un controlador de clic especialitzat està desfasada en 1.8 i es retira en 1.9.

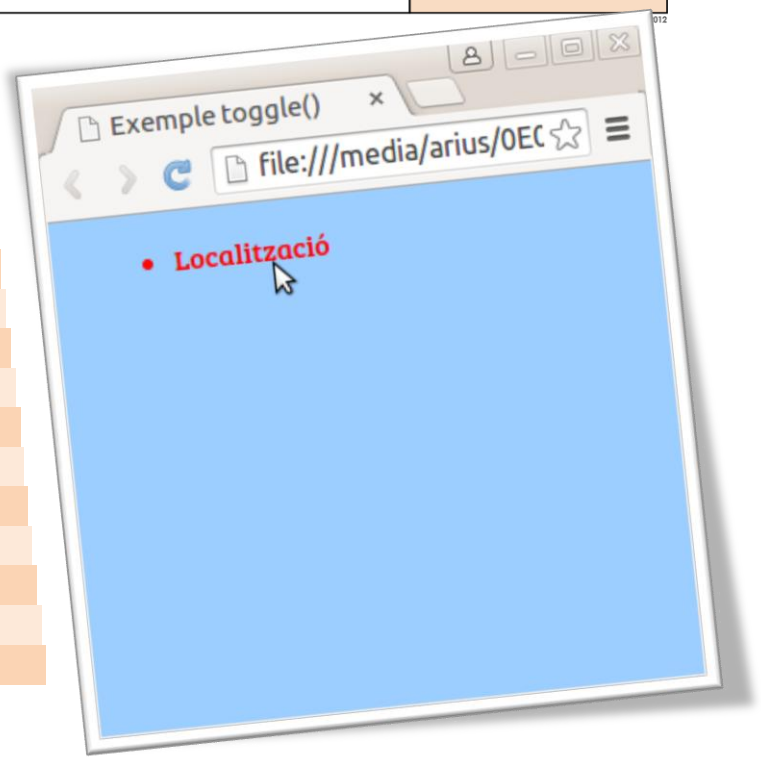
Solució: Torneu a escriure el codi que depengui de **`$().toggle()`**, utilitzeu la versió de producció mini del plugin **jQuery Migrate** per proporcionar la funcionalitat, o extreure el mètode **`$().toggle()`** des del codi font del plugin i utilitzar-lo en l'aplicació.

És comú trobar-nos problemes com aquest. Anem a veure un cas.

```
<!DOCTYPE html>
<html lang=”ca”>
      <head>
            <meta charset="utf-8">
            <title>Exemple toggle()</title>
            <link href='http://fonts.googleapis.com/css?family=Bree+Serif'
rel='stylesheet' type='text/css'>
            <style type="text/css">
                  body { background: #99CCFF; font-family: 'Bree Serif', serif; }
                  ul { margin-left: 20px; color: blue; }
                  li { cursor: default; }
                  span { display: none; background: #CCCCFF; }
            </style>
            <script src="http://code.jquery.com/jquery-1.11.2.min.js"
type="text/javascript"></script>
            <script src="activitat-4.js" type="text/javascript"></script>
      </head>
      <body>
            <ul>
                  <li id=”local”>Localització</li>
                  <span id=”op_loc”>
                  <ul>
                        <li>Mapa</li>
                        <li>Adreça</li
                        <li>Telèfons</li>
                  </ul>
                  </span>
            </ul>
      </body>
</html>
```

Aquest codi, abreujat, trobat en un llibre és un clar exponent d'aquest mètode. Si utilitzem la solució donada en el llibre, aquest no funciona. Realment funciona bé, però amb el **`jQuery versió 1.6.4`**, utilitzada originalment en el llibre.

Aquí teniu el codi proporcionat en el llibre i que funciona bé amb la versió antiga, però no amb una versió moderna. Comproveu-lo




```
$(document).ready(function() {
	$("li").hover(function() {
		$(this).css("color", "red");
		}, function(){
		$(this).css("color", "blue");
	});
	$("li#local").toggle(function(){
		$("span#op_loc").show();
		}, function(){
		$("span#op_loc").hide();
	});
});
```

Per solucionar-ho podem afegir, en el codi HTML, després de carregar la biblioteca jQuery, la següent línia:

```
<script src="http://code.jquery.com/jquery-migrate-1.2.1.min.js"></script>
```

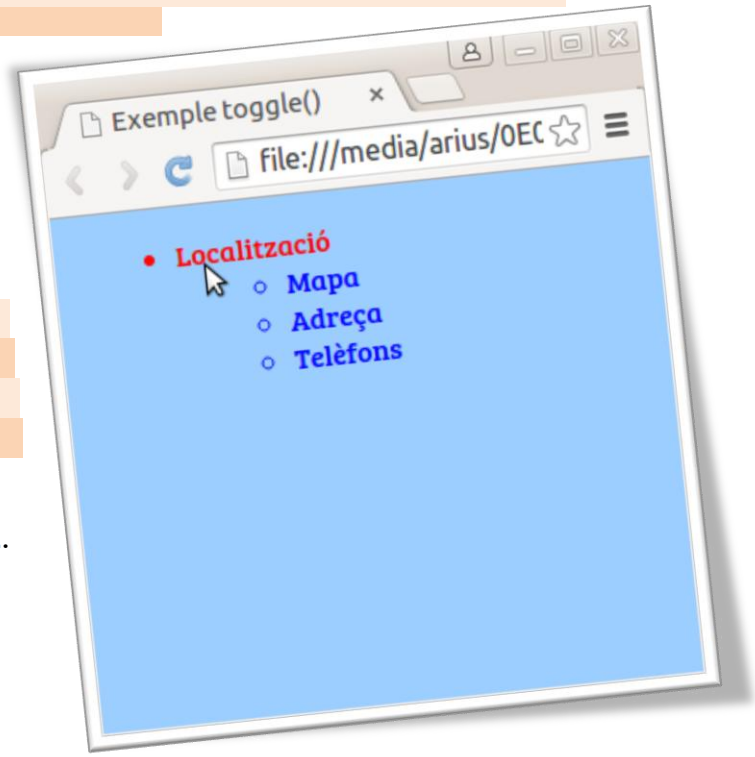
Ara el programa funcionarà adequadament quan fem clic sobre "Localització", ens obrirà (mostrarà) el menú (els li ocults per el). Comproveu-lo!.

Una altre possibilitat fora afegir el codi referent a la funció toggle() existent en la biblioteca jQuery-Migrate, i estalviar-nos la incorporació de la biblioteca sencera.

Per exemple afegint el següent codi, en el codi javascript, ens queda de la següent manera:

```
// Incorporació de la funció toggle() de la biblioteca jQuery-Migrate
jQuery.fn.toggle = function( fn, fn2 ) {

	// Don't mess with animation or css toggles
	if ( !jQuery.isFunction( fn ) || !jQuery.isFunction( fn2 ) ) {
		return oldToggle.apply( this, arguments );
	}
	// Save reference to arguments for access in closure
	var args = arguments,
		guid = fn.guid || jQuery.guid++,
		i = 0,
		toggler = function( event ) {
			// Figure out which function to execute
			var lastToggle = ( jQuery._data( this, "lastToggle" + fn.guid ) || 0 ) % i;
			jQuery._data( this, "lastToggle" + fn.guid, lastToggle + 1 );

			// Make sure that clicks stop
			event.preventDefault();

			// and execute the function
			return args[ lastToggle ].apply( this, arguments ) || false;
		};
```

```
        // link all the functions, so any of them can unbind this click handler
        toggler.guid = guid;
        while ( i < args.length ) {
                args[ i++ ].guid = guid;
        }

        return this.click( toggler );
};
```



I segueix amb el codi que hem fet anteriorment.

```
// Inici del codi de l'exercici
$(document).ready(function() {
        .....
});
```

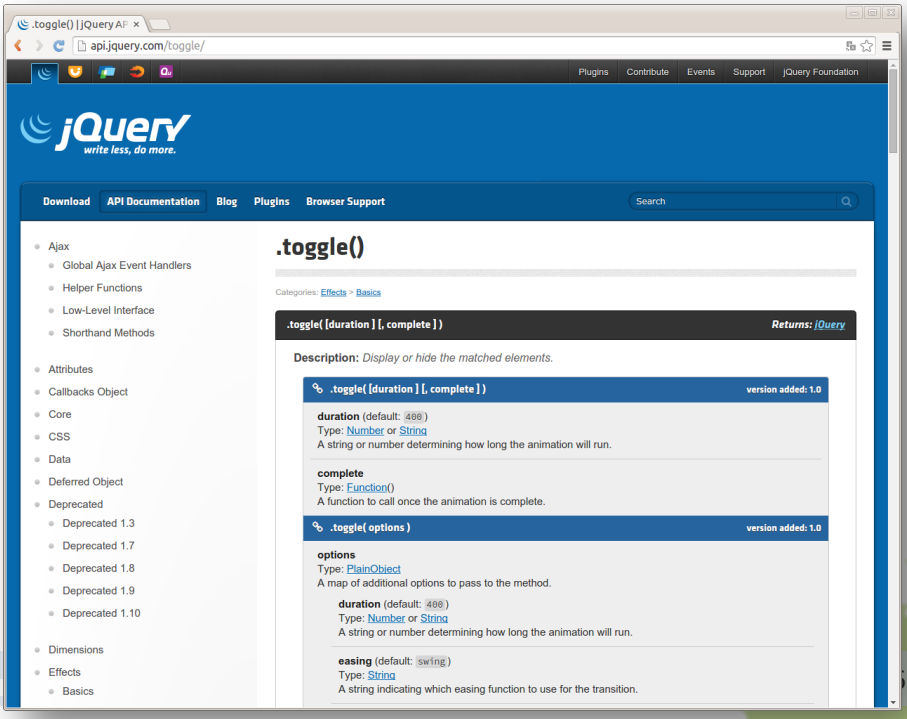
Podeu comprovar que funciona perfectament, recordeu-vos de treure la biblioteca **jQuery-Migrate** de la pàgina **HTML**. Verifiqueu la seva funcionalitat.

Encara que podríem estalviar-nos tot aquest embolic i reprogramar el codi per que s'adapti a les noves exigències sense afegir cap biblioteca o funció extra.

Utilitzant el que ara sabem que si funciona de la funció **toggle**, recordeu-vos que han eliminat només l'esdeveniment **click** de la funció, podríem refer-la de la següent manera:

```
$(document).ready(function() {
        $("li").hover(function() {
                $(this).css("color", "red");
                }, function(){
                $(this).css("color", "blue");
        });
        $("li#local").click(function(){
                $("span#op_loc").toggle();
        });
});
```

Aneu a la pàgina, actual, de l'**API** de **jQuery** sobre aquesta funció per comprendre el que hem fet. Recordeu que ara ja no caldrà afegir ni la biblioteca, ni la funció continguda a **jQuery-Migrate**.



ACTIVITATS i EXERCICIS:

Heu de llegir aquesta pràctica i fer els exemples utilitzant el vostre l'editor preferit.

PREMISSES:

1. Llegir-vos completament la pràctica.
2. Feu els exemples utilitzant el vostre editor preferit

ACTIVITATS i EXERCICIS:

1. Feu els exemples de la pràctica. Podeu utilitzar, si voleu, un editor en línia.
2. Heu de resoldre els exercicis proposats que trobareu dins de la carpeta **/exercicis** de l'arxiu comprimit, aquests exercicis son autoexplicatius i heu de confeccionar el codi adient incrustant-lo al lloc establert. Fixeu-vos en el primer exercici:

```
<!doctype html>
<html lang="ca">
    <meta charset="UTF-8">
    <head>
        <title>01-Primera prova amb jQuery</title>
        <script type = "text/javascript" src = "jquery-2.1.3.min.js"> </script>
        <script type = "text/javascript">
            // Poseu aquí el codi pertinent


        </script>
    </head>
    <!--    Hem de canviar el color de text que es troba en l'element "body"
            per que el mostri de color blau.
            Consell: utilitzeu el mètode css -->
    <body>
        Hola Món
    </body>
 </html>

```

És imprescindible que utilitzeu la pàgina web de l'API de jQuery per fer aquests exercicis. En la pàgina trobareu la informació necessària per arribar a completar-los amb èxit.

LLIURAMENT DE LA PRÀCTICA:

Comprimiu tots els exemples de la pràctica i la carpeta /exercicis amb els exercicis completats , en un arxiu de format **7z**, anomeneu-lo "**DAW0922-PR02-vostrenom.7z**" i lliureu-lo a l'aula virtual.

A on "vostrenom" correspon al vostre primer cognom. Exemple: Si el vostre nom fos Pere Beltran, l'arxiu el podríeu anomenar "DAW0922-PR02-beltran.7z" o també "DAW0922-PR02-pbeltran.7z".