

Model d'objectes del document

Xavier Garcia Rodríguez

Desenvolupament web en l'entorn client

Índex

Introducció	5
Resultats d'aprenentatge	7
1 Programació amb el DOM (Document Object Model)	9
1.1 Estructura del DOM. Interfícies principals	9
1.1.1 Tipus de models DOM	10
1.1.2 Estructura del DOM	11
1.1.3 Adaptacions de codi per diferents navegadors	11
1.2 Interfícies del DOM	13
1.3 Interficie 'Node'	13
1.3.1 Informació bàsica dels nodes: 'nodeType' i 'nodeName'	14
1.3.2 Contingut textual: 'textContent'	15
1.3.3 Relacions entre nodes: 'parentNode', 'firstChild', 'lastChild', 'previousSibling' i 'nextSibling'	15
1.3.4 Obtenció de nodes descendents: 'childNodes' i 'hasChildNodes'	16
1.3.5 Inserció de nodes: 'appendChild' i 'insertBefore'	17
1.3.6 Eliminació i substitució de nodes: 'removeChild' i 'replaceChild'	18
1.3.7 Copia de nodes: 'cloneNode'	20
1.4 Interficie 'Document'	20
1.4.1 Creació de nodes: 'createTextNode' i 'createElement'	21
1.4.2 Cerca d'elements simple: 'getElementsByClassName', 'getElementsByTagName' i 'getElementById'	22
1.4.3 Cerca d'elements amb selectors ('querySelector' i 'querySelectorAll')	24
1.4.4 L'extensió 'HTMLDocument'	25
1.4.5 Altres propietats: 'body', 'documentElement' i 'forms'	26
1.5 La interfície 'Element'	27
1.5.1 Informació bàsica dels elements: 'tagName', 'className', 'classList' i 'id'	27
1.5.2 Obtenció dels descendents com HTML: 'innerHTML'	31
1.5.3 Atributs: 'attributes', 'getAttribute', 'removeAttribute', 'setAttribute'	32
1.5.4 Modificar estils CSS: 'style'	33
1.5.5 Relacions entre elements: 'previousElementSibling', 'nextElementSibling', 'firstElementChild' i 'lastElementChild'	36
1.5.6 Cerca d'elements descendents simple: 'getElementsByClassName', 'getElementsByTagName'	37
1.5.7 Cerca d'elements descendents amb selectors: 'querySelector', 'querySelectorAll'	38
1.6 Integració de la detecció d'events amb el DOM	39
1.6.1 Afegir detecció d'esdeveniments: 'addEventListener'	40
1.6.2 Eliminar detecció d'esdeveniments: 'removeEventListener'	42
1.7 Cas pràctic: generador de factures	46

2 Programació amb el DOM i la biblioteca jQuery	55
2.1 Introducció a jQuery	55
2.1.1 Carregar la biblioteca jQuery	56
2.1.2 Carregar jQuery a CodePen	58
2.1.3 Primers passos amb jQuery	58
2.1.4 L'objecte 'jQuery'	60
2.2 Selectors	63
2.2.1 Selectors CSS	64
2.2.2 Selectors propis de jQuery: seleccions especials	68
2.2.3 Selectors propis de jQuery: formularis	71
2.2.4 Cercar entre els elements seleccionats: 'filter' i 'find'	74
2.3 Crear i manipular elements	76
2.3.1 Crear nous elements	76
2.3.2 Manipulació de classes: 'addClass', 'removeClass' i 'toggleClass'	79
2.3.3 Modificar continguts: 'val', 'html' i 'text'	80
2.3.4 Modificar estils: 'css'	81
2.3.5 Manipular atributs: 'attr' i 'prop'	82
2.4 Manipular el DOM	85
2.5 Utilitzar Events amb jQuery	87
2.6 Crear connectors per jQuery	90
2.7 Model-vista-controlador (MVC)	95
2.7.1 Angular	97
2.7.2 Ember	98
2.7.3 React	98
2.7.4 Components Web	99

Introducció

Avui dia és molt difícil trobar una pàgina o aplicació web que no requereixi modificar, afegir o eliminar alguns dels seus elements per respondre a les accions dels usuaris. Tant pot ser un menú desplegable, un diàleg d'entrada de dades o un panell que mostra el contingut d'un carretó electrònic: en tots els casos cal manipular aquests elements. Aquestes modificacions es fan a través del model d'objectes del document (DOM), i es poden fer treballant directament amb les seves interfícies o fent servir alguna biblioteca com jQuery.

En aquesta unitat, “Model d'objectes del document”, s'expliquen les característiques principals del DOM, les principals interfícies que el componen i com s'hi ha de treballar per modificar el contingut de qualsevol pàgina o aplicació web. L'assimilació dels continguts d'aquesta unitat és fonamental, atès que es faran servir en major o menor grau en el desenvolupament de totes les vostres aplicacions a la banda del client.

A l'apartat **“Programació amb el DOM (Document Object Model)“** trobareu una introducció a les interfícies del DOM, més concretament a la utilització de les interfícies principals: Node, Document i Element. Cal destacar que no només es tracta la creació i eliminació, sinó també la modificació i especialment la cerca d'elements, tant a partir del document com entre els elements descendents d'un node concret. Per altra banda. També es tractarà la detecció d'*events* lligada a aquests elements.

A l'apartat **“Programació amb el DOM i la biblioteca jQuery”** s'introduceix la biblioteca jQuery, que destaca especialment per la facilitat amb la qual es poden manipular conjunts d'elements del DOM i assignar-hi o eliminar-ne *events*. Com que és palesa la importància de cercar elements i conjunts d'elements, es recalcarà la utilització de selectors (tant de CSS com de propis) i com refinar aquestes seleccions. Finalment es farà una introducció al patró model-vista-vontrolador i als entorns de treball (*frameworks*) i les biblioteques més populars que l'implementen, juntament amb els components web, una component experimental que forma part de l'especificació del W3C.

Com que els continguts d'aquesta unitat són aplicables a la majoria de les aplicacions web, podeu aplicar aquests coneixements a la vostra feina (si us dediqueu al desenvolupament web) o als vostres projectes personals. Com més practiqueu, més facil serà assimilar aquests continguts.

Per assolir els objectius d'aquesta unitat cal que proveu tots els exemples i que hi feu modificacions per entendre com funcionen; també cal que feu totes les activitats proposades. En cas de dubte, primer es recomana consultar a internet les múltiples fonts d'informació (Mozilla Developer Network, StackOverflow, lloc oficial de jQuery...) i preguntar al fòrum de l'assignatura, ja que així us podran ajudar els vostres companys o el professor.

Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

1. 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 amb el DOM (Document Object Model)

La manipulació del **model d'objectes del document** –més conegut per les seves sigles en anglès: DOM (Document Object Model)– és fonamental per al desenvolupament d'aplicacions web, perquè sense aquesta capacitat no és possible alterar la visualització de les pàgines dinàmicament.

Internament, els navegadors treballen amb els documents web com si es tractés d'un arbre, i és a partir d'aquest arbre que es pot modificar la representació de la pàgina, tant afegint nous elements (paràgrafs, capçaleres, taules...) com modificant els atributs dels nodes que ja es troben al document o eliminant-los.

A més a més, és possible cercar tant elements concrets com llistes d'elements fent servir diferents propietats i mètodes segons les vostres necessitats: a través de relacions (el primer element, el pròxim element...), cercant segons el tipus d'element, el seu identificador o fent una cerca més complexa gràcies als selectors de CSS.

A banda de manipular aquests elements, també és possible accedir-hi per fer operacions de consulta, com per exemple per extreure informació dels atributs o del text contingut.

1.1 Estructura del DOM. Interfícies principals

El model d'objectes del document és una interfície que facilita treballar amb documents **HTML, XML i SVG**. És a dir, independentment del llenguatge de programació que es faci servir, el nom dels mètodes i paràmetres per manipular-lo són idèntics, tant si es tracta de PHP, com de JavaScript o de Python, per exemple.

Aquesta interfície defineix els mètodes i propietats que permeten accedir i manipular el document com si es tractés d'un arbre de nodes, en el qual l'arrel és el node **document**, que pot contenir qualsevol quantitat de nodes fills, i les fulles són els nodes que no tinguin cap descendent (generalment el text dels elements HTML).

Tot i que el més habitual és accedir al DOM a través de JavaScript, el DOM no forma part de l'especificació de JavaScript, sinó que es troba especificat pel W3C com una sèrie d'**interfícies independents** de la plataforma i el llenguatge.

Per tant, els mètodes i les propietats que formen part d'aquestes interfícies no són propis de JavaScript, sinó que es troben **disponibles a qualsevol llenguatge** que ofereixi la capacitat de manipular el DOM, com per exemple Python, tant directament com a través de biblioteques.

Especificació del DOM segons el W3C

Podeu trobar l'especificació del DOM del W3C (que és el consorci internacional que treballa per desenvolupar els estàndards utilitzats per internet) en l'enllaç següent:
www.w3.org/DOM.

Al llarg dels anys s'han produït molts canvis a l'especificació, i ha evolucionat i s'ha simplificat. Algunes de les interfícies antigues han estat declarades obsoletes i no s'han de fer servir perquè els navegadors moderns poden deixar d'admetre-les.

És important tenir-ho en compte a l'hora de consultar materials de referència, ja que és fàcil trobar informació desactualitzada. És recomanable consultar directament la documentació del web Mozilla Developer Network (www.developer.mozilla.org) o l' ‘especificació viva’ (*DOM Living Standard*; www.dom.spec.whatwg.org) per aclarir els dubtes que pugueu tenir.

WHATWG
El Web Hypertext Application Technology Working Group (WHATWG) és una comunitat fundada per integrants d'Apple, la fundació Mozilla i Opera. En trobareu més informació en l'enllaç següent: www.goo.gl/AFxeCb.

Actualment hi ha dos grups treballant en les especificacions per al web, el W3C i el WHATWG. Encara que tots dos van col·laborar temporalment fins al 2012, van deixar de fer-ho a causa del fet que els objectius de WHATWG (a favor dels estàndards vius) i del W3C (especificacions estàtiques) eren contraris. Actualment el W3C actualitza la seva versió de l'especificació a partir d'*instantànies* de l'especificació viva, centrant-se en la correcció d'errors.

Les especificacions proporcionades pel W3C sobre el DOM estan dividides en nivells i cadascun d'aquests nivells ho està, al seu torn, en diferents especificacions. D'aquesta manera es poden conèixer les capacitats quant a manipulació del DOM dels diferents navegadors. Per exemple, el *DOM nivell 1* és admès per pràcticament tots els navegadors, en canvi, durant l'any 2016 l'especificació *DOM3 carregar i guardar* encara no l'admetia cap navegador.

Per aquesta raó, trobareu que a la documentació sobre diferents webs API s'indica el nivell de DOM necessari per utilitzar-les, ja que no funcionaran en navegadors que no admetin aquest nivell.

1.1.1 Tipus de models DOM

En desenvolupar aplicacions web es treballa amb la interfície **HTMLDocument**, que s'aplica als elements HTML, però l'especificació del DOM consta de més interfícies que permeten manipular diferents tipus de documents. Són les següents:

- **Interfície HTMLDocument** (documents HTML): és una extensió de la interfície *Document* i afegeix l'accés a característiques pròpies dels navegadors, com són la finestra (*window*) o les pestanyes.
- **Interfície XMLDocument** (documents XML): aquesta interfície es troba en fase experimental i a finals de l'any 2016 encara no l'admet cap navegador.
- **Interfícies SVG**: es tracta d'un conjunt d'interfícies que permeten manipular directament des de JavaScript els elements d'una imatge vectorial de tipus SVG, per afegir formes, canviar colors... No es fa servir gaire perquè és molt complexa.

Les interfícies SVG

Podeu trobar un exemple d'ús de les interfícies SVG juntament amb JavaScript en l'enllaç següent: [goo.gl/gz6Lg6](http://www.goo.gl/gz6Lg6).

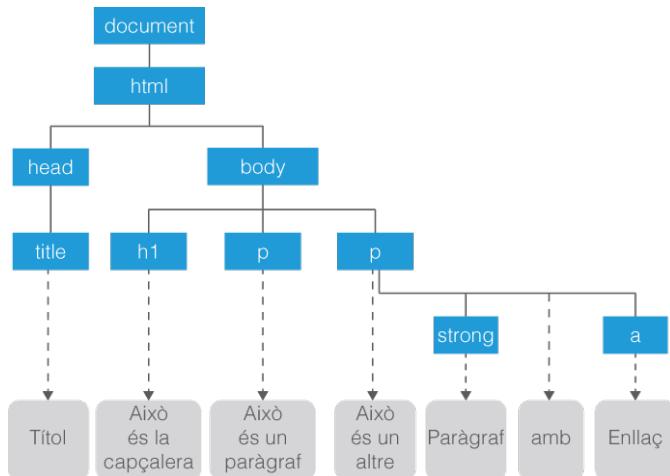
1.1.2 Estructura del DOM

Per entendre més fàcilment com s'estructura el DOM fixeu-vos en el següent codi HTML i contrasteu-lo amb la figura 1.1:

```

1 <html>
2   <head>
3     <title>Títol</title>
4   </head>
5   <body>
6     <h1>Això és la capçalera</h1>
7     <p>Això és un paràgraf</p>
8     <p>Això és un altre <strong>paràgraf</strong> amb <a>enllaç</a></p>
9   </body>
10 </html>
```

FIGURA 1.1. Estructura d'arbre corresponent a un document HTML



Com es pot apreciar, l'arrel de l'arbre és el node **document**, tot i que no forma part del codi HTML. A continuació trobem el node **html**, que conté els nodes **head** i **body**; aquests contenen altres nodes: **title**, **h1** i **p**. Tots aquests nodes són de tipus **Element**. En canvi, l'últim element de cada branca (les fulles) són de tipus **Text** i tenen una consideració diferent.

Pareu atenció a l'últim paràgraf del document: en aquest cas el text inclou els elements **strong** i **a**, però aquests elements no pengen del text sinó que pengen del node **p**. Per tant, es pot concloure que **un element sempre és descendent d'un altre element** i mai d'un text.

1.1.3 Adaptacions de codi per diferents navegadors

S'ha de tenir en compte que encara que s'amplien les especificacions de les interfícies i dels llenguatges de programació contínuament, els navegadors antics no les admeten.

IE 6 al mercat xinès

A l'agost del 2012, Internet Explorer 6 encara era el segon navegador més utilitzat a la Xina (22.41% dels usuaris).

Actualment, tots els navegadors s'actualitzen automàticament de forma predeterminada. Per aquesta raó, es parla de Google Chrome o de Firefox sense indicar-ne la versió. En canvi, els navegadors més antics s'havien d'actualitzar manualment i en alguns països l'ús de navegadors obsolets (especialment Internet Explorer) ha estat molt estès fins fa pocs anys.

Afortunadament, cada vegada hi ha menys usuaris que facin servir navegadors obsolets, i molts grups han deixat fer-los compatibles perquè consideren que mentre hi hagi compatibilitat amb aquests navegadors obsolets, la gent continuarà fent-los servir i això perjudica tant els desenvolupadors com els usuaris amb navegadors actualitzats.

En cas de requerir compatibilitat amb aquests navegadors, al mateix temps que s'aprofiten les noves tecnologies, teniu a la vostra disposició dues opcions:

- **Biblioteques:** fer servir una biblioteca que implementi les característiques que necessiteu i que doni compatibilitat amb navegadors antics (per exemple, la branca 1.12 de jQuery). En aquest cas, sempre que sigui possible, es recomana fer servir la biblioteca: les biblioteques implementen el comportament correcte per a tots els navegadors sense haver de modificar el vostre codi.
- **Polyfill:** consisteix a afegir, juntament amb el vostre codi, una implementació pròpia de la característica que necessiteu i que no és disponible en tots els navegadors antics. Per exemple, abans que HTML5 fos disponible per a tots els navegadors moderns, era possible fer servir *polyfills* per implementar algunes de les noves característiques del llenguatge sense haver de preocupar-se pels navegadors dels usuaris.

Al web de Mozilla Developer Network es pot trobar el *polyfill* per implementar moltes de les característiques d'HTML5.

Implementació 'polyfill' per a JSON

Es pot trobar la implementació completa del *polyfill* per a l'objecte JSON en l'enllaç següent: goo.gl/p4nm6l.

Per exemple, si voleu fer una implementació per treballar amb JSON en navegadors antics, faríeu servir un codi similar al següent:

```
1 if (!window.JSON) {  
2     window.JSON = {  
3         parse: function(text) { // Codi per retornar un objecte a partir del text},  
4         stringify: function(obj) { // Codi per retornar una cadena de text a partir  
5             de l'objecte}  
6     }  
6 }
```

Cal destacar que la implementació pròpia dels navegadors d'aquestes funcions és molt més eficient que els *polyfills*. Per aquesta raó, és molt important comprovar primer si el navegador ja les implementa i, si no és així, cal afegir-les. Per fer-ho només cal comprovar si es troben definides a l'objecte o al prototipus, segons el cas.

Tenint en compte la complexitat afegida d'utilitzar *polyfills* i les seves limitacions, és més recomanable fer servir biblioteques que ja implementin aquests *polyfills* o incloguin les funcionalitats que requeriu en lloc de fer la vostra pròpia implementació.

1.2 Interfícies del DOM

L'especificació del DOM està dividida en múltiples interfícies que determinen les possibles accions que es poden realitzar amb cada element; per exemple, **per conèixer la classe o classes aplicades a un element, caldrà utilitzar la interfície Element**; en canvi, **per consultar o modificar el valor d'un atribut de l'element, s'utilitza la interfície Attr**.

A continuació podeu trobar una llista de les interfícies més destacables per a la manipulació de documents HTML:

- **Node**: molts dels elements del DOM hereten d'aquest tipus i, per tant, ofereixen aquestes funcionalitats. Aquesta interfície permet consultar i manipular els nodes, com per exemple navegar a través dels nodes fills, pares i contigus, cercar nodes, afegir nodes nous o eliminar-los.
- **Document** (herència de Node): aquesta interfície és la que s'aplica a l'arrel del document. Permet, entre altres coses, conèixer l'element actiu, manipular les galetes (més conegeudes com a *cookies*) i obtenir informació global del document.
- **Element** (herència de Node): s'aplica a tots els elements del document, és a dir, a les etiquetes que apareixen com a body, h1 o p. A partir d'aquesta interfície es pot obtenir una llista d'atributs, la classe de l'element, el seu id i afegir observadors per escoltar diferents *events*.
- **Attr**: aquesta interfície permet consultar i modificar els atributs d'un element. En versions anteriors a DOM4 la interfície Attr derivava de Node, però no es recomana fer servir aquestes funcionalitats perquè no hi hagi discrepàncies amb els navegadors més actuals.
- **Event**: aquesta interfície és implementada per altres interfícies de l'especificació i conté tots els mètodes comuns per representar un esdeveniment.

Cal destacar que existeix una interfície per a **text**, però no es fa servir gaire. El seu objectiu principal és representar el text contingut en un **element** o **attr** (atribut).

1.3 Interfície 'Node'

D'entre totes les interfícies la més important és Node, ja que Document i Element deriven d'aquesta. És a dir, totes les propietats i mètodes de Node són accessibles tant per a Document com per a Element.

Cal destacar que tot i que això implica que Document i Element inclouen la funcionalitat per navegar entre nodes, no s'acostuma a fer-les servir. En el cas

Llistat d'interfícies

Podeu trobar una llista de totes les interfícies DOM en l'enllaç següent: goo.gl/X3rGDj.

Propietats de 'Node'

Podeu trobar informació més detallada sobre la interfície Node en l'enllaç següent: [www.http://goo.gl/RmwM4h](http://goo.gl/RmwM4h).

de la primera interfície s'acostuma a cercar directament l'element que interessa (a través del tipus d'element, el seu identificador o la seva classe), mentre que el segon, a més de disposar de la capacitat de cerca, inclou propietats específiques per navegar entre els nodes de tipus *element*, filtrant-ne, així, la resta (per exemple, els de tipus *text*).

1.3.1 Informació bàsica dels nodes: 'nodeType' i 'nodeName'

La propietat `nodeType` permet determinar el **tipus d'un node**. Conté un valor numèric corresponent a les pseudoconstants (recordeu que realment a JavaScript no existeixen les constants), que podeu veure a la taula taula 1.1.

TAULA 1.1. Correspondència de valors de la propietat `nodeType`

Pseudoconstant	Valor
ELEMENT_NODE	1
TEXT_NODE	3
PROCESSING_INSTRUCTION_NODE	7
COMMENT_NODE	8
DOCUMENT_NODE	9
DOCUMENT_TYPE_NODE	10
DOCUMENT_FRAGMENT_NODE	11

Per exemple, per comprovar el valor de la propietat `nodeType` de `document` només cal mostrar-la per la consola:

```
1 console.log(document.nodeType);
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/PGoGwP?editors=0012.

Com es pot apreciar, el valor retornat és 9, ja que és el valor associat a `DOCUMENT_NODE`.

En canvi, si es consulta el valor de la propietat d'un element, el valor associat serà 1, com correspon a `ELEMENT_NODE`:

```
1 <div>
2   <p id="primer">Primer paràgraf</p>
3 </div>
4
5 <script>
6 var primer = document.getElementById('primer');
7 console.log(primer.nodeType);
8 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/QKWKNg?editors=1012.

`getElementById` és un mètode de la interfície `document` que permet obtenir un element per al seu id.

Per altra banda, la propietat `nodeName` permet saber el **nom del node**, i això permet distingir-los entre d'altres. Aquest nom no correspon necessàriament amb el de l'etiqueta que crea el node, com es pot comprovar en l'exemple següent:

```

1 <div>
2   <p id="primer">Primer paràgraf</p>
3 </div>
4
5 <script>
6   var primer = document.getElementById('primer');
7   console.log(document.nodeName);
8   console.log(primer.nodeName);
9 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/EgxgjZ?editors=1012.

Fixeu-vos que el nom del node document és #document, mentre que el de l'element p és P.

1.3.2 Contingut textual: 'textContent'

Aquesta propietat retorna el contingut textual:

```

1 <p id="primer">Primer paràgraf</p>
2
3 <script>
4   var primer = document.getElementById('primer');
5   console.log(primer.textContent);
6 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/GjRjOq?editors=1012.

1.3.3 Relacions entre nodes: 'parentNode', 'firstChild', 'lastChild', 'previousSibling' i 'nextSibling'

La propietat parentNode permet accedir al pare del node o retorna null si no existeix (per exemple, si el node no s'ha afegit al document).

```

1 <div id="contenidor">
2   <p id="primer">Primer paràgraf</p>
3 </div>
4
5 <script>
6   var primer = document.getElementById('primer');
7   console.log(primer.parentNode.nodeName);
8 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/oBXRzP.

Com es pot apreciar, el nom del node retornat és DIV, el corresponent al node pare. Aquest comportament permet recórrer l'arbre ascendentment; per exemple, es podria accedir al pare del pare: primer.parentNode.parentNode.nodeName.

Les propietats `firstChild` i `lastChild` permeten accedir al primer i a l'últim fill d'un node, com es pot comprovar en l'exemple següent:

```

1 <div id="contenedor"><p id="primer">Primer paràgraf</p><p id="segon">Segon parà
2   graf</p><p id="tercer">Tercer paràgraf</p></div>
3
4 <script>
5   var contenido = document.getElementById('contenedor');
6   console.log(contenido.firstChild.textContent);
7   console.log(contenido.lastChild.textContent);
8 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/VKwKbw?editors=1012.

Els **salts de línia** són interpretats com a nodes de tipus text i, per consegüent, en cas d'afegir salts de línia, tant el `firstChild` com el `lastChild` serien nodes de tipus text.

En el cas de `previousSibling` i `nextSibling` es produeix el mateix comportament, i per aquesta raó cal tenir en compte els salts de línies i tabulacions quan es tracta amb aquests mètodes.

Per altra banda, `previousSibling` i `nextSibling` permeten accedir als nodes **germans**, anterior i posterior respectivament, d'un mateix node:

```

1 <div id="contenedor"><p id="primer">Primer paràgraf</p><p id="segon">Segon parà
2   graf</p><p id="tercer">Tercer paràgraf</p>
3 </div>
4
5 <script>
6   var segon = document.getElementById('segon');
7   console.log(segon.previousSibling.textContent);
8   console.log(segon.nextSibling.textContent);
9 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/mAdAxE?editors=1012.

En executar aquest exemple es mostrerà correctament el contingut textual dels nodes anterior i posterior, és a dir, Primer paràgraf i Tercer paràgraf.

1.3.4 Obtenció de nodes descendents: 'childNodes' i 'hasChildNodes'

Per altra banda, la propietat `childNodes` ens permet accedir a la llista de nodes continguts que es pot tractar com un *array* i conèixer la quantitat de nodes a través de la propietat `length`. Per altra banda, si només es vol saber si conté altres nodes o no, es pot invocar el mètode `hasChildNodes`, que retornarà `true` si en conté o `false` en cas contrari.

Com que la llista de nodes es pot tractar com un *array*, es pot recórrer normalment:

```

1 <div id="contenidor"></div>
2   <p id="primer">Primer paràgraf</p>
3   <p id="segon">Segon paràgraf</p>
4   <p id="tercer">Tercer paràgraf</p>
5 </div>
6
7 <script>
8   var contenidor = document.getElementById('contenidor');
9
10  console.log('El contenidor conté altres nodes: ' + contenidor.hasChildNodes()
11    );
12  console.log('Conté i ' + contenidor.childNodes.length + ' nodes');
13
14  for (var i = 0; i < contenidor.childNodes.length; i++) {
15    console.log('Trobat un node de tipus ' + contenidor.childNodes[i].nodeType
16      + ': ' + contenidor.childNodes[i].nodeName);
17  }
</script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/vXYXbX?editors=1012.

Fixeu-vos que en aquest cas s'han fet servir la indentació habitual, i en recórrer la llista de nodes s'han trobat quatre nodes de tipus text, corresponents als nodes generats pels salts de línia i les indentacions.

1.3.5 Inserció de nodes: 'appendChild' i 'insertBefore'

Aquests dos mètodes permeten afegir un node com a fill d'un altre. La diferència és que `appendChild` afegeix el node al final de la llista de fills, mentre que amb `insertBefore` el node s'afegeix abans del node de referència passat com a argument.

El mètode `insertBefore` requereix que es passin dos arguments, el node que s'ha d'afegeir i el node de referència o `null`, obligatoriàment.

```

1 <div id="contenidor"></div>
2
3 <script>
4   var contenidor = document.getElementById('contenidor');
5
6   contenidor.appendChild(document.createElement('p'));
7   contenidor.appendChild(document.createElement('div'));
8   contenidor.insertBefore(document.createElement('h1'), contenidor.firstChild);
9
10  console.log('El contenidor conté altres nodes: ' + contenidor.hasChildNodes()
11    );
12  console.log('Conté ' + contenidor.childNodes.length + ' nodes');
13
14  for (var i = 0; i < contenidor.childNodes.length; i++) {
15    console.log('Trobat un node de tipus ' + contenidor.childNodes[i].nodeType
16      + ': ' + contenidor.childNodes[i].nodeName);
17  }
</script>
```

El mètode `createElement` forma part de la interfície `Element`.

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/WGNooa?editors=1012.

Com es pot apreciar en aquest exemple, els nodes afegits amb `appendChild` s'han afegit en ordre, mentre que el node afegit amb `insertBefore` s'ha afegit davant del primer fill del contenidor i, per tant, ha quedat en primera posició.

S'ha de tenir en compte que **en el cas d'invocar aquests mètodes passant com a paràmetre un node que ja es trobi al document, en lloc d'afegir-se, el node es mourà**, és a dir, s'eliminarà de la seva posició anterior i s'afegirà a la nova:

```

1 <div id="contenedor1">
2   <h1 id="titol1">Aquest és el títol 1<h1>
3 </div>
4 <div id="contenedor2">
5   <h1 id="titol2">Aquest és el títol 2<h1>
6 </div>
7
8 <script>
9   var contenedor2 = document.getElementById('contenedor2');
10  var titol1 = document.getElementById('titol1');
11  contenedor2.appendChild(titol1);
12 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/ALBOjL?editors=1010.

Com es pot apreciar, una vegada s'afegeix el `titol1` al `contenedor2` s'elimina automàticament del `contenedor1`.

1.3.6 Eliminació i substitució de nodes: '`removeChild`' i '`replaceChild`'

El mètode `removeChild` permet eliminar un node fill. S'ha de tenir en compte que per poder eliminar **un node s'ha d'accendir al pare d'aquest, per tant, cal recordar que la propietat `parentNode` hi dona accés**, tal com es pot veure a l'exemple següent:

```

1 <div id="contenedor1">
2   <h1 id="titol1">Aquest és el títol 1<h1>
3 </div>
4 <div id="contenedor2">
5   <h1 id="titol2">Aquest és el títol 2<h1>
6 </div>
7
8 <script>
9   var titol1 = document.getElementById('titol1');
10  titol1.parentNode.removeChild(titol1);
11 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/rrNWpX?editors=1010.

Fixeu-vos que el `titol1` ha desaparegut i només es mostra el `titol2`. El `contenedor1` (el node pare) continua existint, però ara es troba buit.

Tot i que el node ja no forma part del contenidor, es pot conservar una referència (per exemple, la retornada pel mètode `removeChild`) i afegir-lo a un altre node:

```

1 <div id="contenedor1">
2   <h1 id="titol1">Aquest és el títol 1<h1>
3 </div>
4 <div id="contenedor2">
5   <h1 id="titol2">Aquest és el títol 2<h1>
6 </div>
7
8 <script>
9   var contenedor2 = document.getElementById('contenedor2');
10  var titol1 = document.getElementById('titol1');
11
12  var refNode = titol1.parentNode.removeChild(titol1);
13  contenedor2.appendChild(refNode);
14 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/yaLVmN?editors=1010.

Com es pot apreciar, ara el `contenedor2` conté tots dos títols. Cal tenir en compte que en aquest cas concret no caldria guardar la referència perquè la variable `titol1` ja la manté. Si reemplaçau el codi JavaScript pel següent, el resultat és idèntic:

```

1 var contenedor2 = document.getElementById('contenedor2');
2 var titol1 = document.getElementById('titol1');
3
4 titol1.parentNode.removeChild(titol1);
5 contenedor2.appendChild(titol1);
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/WGNRZV?editors=1010.

Per altra banda, `replaceChild` permet **reemplaçar un node per un altre**. Per exemple, es poden intercanviar els continguts dels dos contenidors de l'exemple anterior:

```

1 <div id="contenedor1">
2   <h1 id="titol1">Aquest és el títol 1<h1>
3 </div>
4 <div id="contenedor2">
5   <h1 id="titol2">Aquest és el títol 2<h1>
6 </div>
7
8 <script>
9   var contenedor1 = document.getElementById('contenedor1');
10  var contenedor2 = document.getElementById('contenedor2');
11
12  var titol1 = document.getElementById('titol1');
13  var titol2 = document.getElementById('titol2');
14
15  var refNode = contenedor1.replaceChild(titol2, titol1);
16  contenedor2.appendChild(refNode);
17 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/qRRzLj.

En primer lloc s'obté una referència als contenidors i els nodes, seguidament es fa el reemplaça de `titol1` per `titol2`, i finalment s'afegeix el `titol1` al segon contenidor.

Com que `replaceChild` retorna una referència al node reemplaçat, aquesta referència es pot fer servir per afegir aquest node al `contenidor2`. Cal tenir en compte que no és possible invocar el mètode `replaceChild` per fer l'intercanvi al `contenidor2` perquè aquest ja no conté cap node que pugui ser reemplaçat.

1.3.7 Copia de nodes: 'cloneNode'

Per assegurar la compatibilitat amb navegadors que no admelin el DOM4 és recomanable passar `true` com argument per fer còpies profundes.

Aquest mètode permet clonar un node, de manera que es genera un nou node idèntic. Es pot passar `true` com a paràmetre, si volem que es faci una còpia profunda del node (és a dir, que cloni també tots els seus nodes descendents); o `false`, en cas contrari.

En el següent exemple es pot veure com es clona la capçalera de tipus `h1` i s'afegeix cinc vegades:

```

1 <div id="contenidor"><h1>Això és un títol<h1></div>
2
3 <script>
4   var contenidor = document.getElementById('contenidor');
5
6   for (var i=0; i<5; i++) {
7     var clonedNode = contenidor.firstChild.cloneNode(true);
8     console.log(clonedNode);
9     contenidor.appendChild(clonedNode);
10  }
11 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/GjRApw?editors=1010.

En cas de passar `false` com a argument, s'haurien afegit cinc elements de tipus `h1`, però sense cap contingut, ja que el node amb el contingut textual `Això és un títol` no s'hauria clonat.

1.4 Interfície 'Document'

Mètodes i propietats de 'Document'

Podeu trobar més informació sobre la interfície `Document` en l'enllaç següent: www.goo.gl/RmwM4h.

Aquesta interfície és herència de `Node` i d'`EventTarget` (aquesta darrera permet la utilització d'`events`), així doncs, és possible manipular el document com si es tractés d'un node i gestionar `events`.

Les propietats que ofereix aquesta interfície no són gaire utilitzades, en canvi, és fonamental conèixer els seus mètodes, perquè permeten cercar i crear nous nodes. En aquests materials només es tractarà un subconjunt d'aquests mètodes, els més utilitzats, ja que molts d'aquests tenen aplicacions molt concretes i no cal conèixer-los.

1.4.1 Creació de nodes: 'createTextNode' i 'createElement'

Aquests mètodes permeten crear nodes de text i elements, respectivament. Com que totes dues implementen la interfície Node, es poden afegir al document, eliminar-los, reemplaçar-los o clonar-los:

```

1 <h1 id="titol1">Aquest és el primer node de text.</h1>
2
3 <script>
4   var titol1 = document.getElementById('titol1');
5   var nouNodeDeText = document.createTextNode('I aquest és el segon, afegit din
6     àmicament');
7   titol1.appendChild(nouNodeDeText);
</script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/BLamyb?editors=1010.

Com es pot apreciar, el codi HTML només inclou un fragment de text, però s'afegeix el segon dinàmicament en crear un nou node i afegint-lo al contingut (l'element h1).

Per afegir-lo al principi del contingut en lloc del final només cal substituir l'última línia per:

```
1 titol1.insertBefore(nouNodeDeText, titol1.firstChild);
```

El mètode `createElement` també crea nodes, però en aquest cas són de tipus element i, per consegüent, implementen la interfície 'Element'. Això permet crear elements d'HTML, com es pot apreciar en l'exemple següent:

```

1 <div id="contenidor"></div>
2
3 <script>
4   var contenidor = document.getElementById('contenidor');
5
6   var nouElement = document.createElement('h1');
7   var nouText = document.createTextNode('Això és la capçalera')
8
9   nouElement.appendChild(nouText);
10  contenidor.appendChild(nouElement);
11 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/pEodaE?editors=1010.

L'argument que es passa en invocar el mètode és una cadena de text amb l'etiqueta (*tag*), és a dir, si l'argument és h1 es crearan les etiquetes HTML<h1></h1>. Aquesta etiqueta és important perquè també permet fer cerques a l'arbre de tots els nodes amb la mateixa etiqueta, per exemple: cercar tots els títols principals (h1), tots els enllaços (a)...

Fixeu-vos que no cal afegir els nous nodes al document immediatament, es pot crear una branca a la memòria (en aquest cas el node referenciat per `nouElement`, que conté el node de text) i, una vegada afegits tots els nodes, afegir-la a l'arbre.

Aquesta és la millor manera de fer-ho, perquè si s'afegeixen els nodes al document d'un en un, el navegador renderitzarà l'arbre una vegada per cada node. En canvi, si s'afegeix tota la branca, només ha de renderitzar l'arbre de nou un cop.

1.4.2 Cerca d'elements simple: 'getElementsByClassName', 'getElementsByTagName' i 'getElementById'

Aquests mètodes permeten fer cerques d'elements al document. Cal recalcar que es tracta d'elements (implementen la interfície Elements) i no de qualsevol node. És a dir, es poden cercar tots els paràgrafs d'una pàgina però no els nodes de text que contingui.

L'atribut `id` de qualsevol element d'un document sempre ha de ser únic.

Els mètodes `getElementsByClassName` i `getElementsByTagName` retornen una llista de nodes de tipus element, mentre que `getElementById` retorna sempre un únic node de tipus element.

Cadascun realitza la cerca segons un aspecte diferent:

- **`getElementsByClassName`**: nodes de tipus element que continguin la classe passada com a argument.
- **`getElementsByTagName`**: nodes de tipus element amb l'etiqueta passada com a argument; per exemple: `p`, per obtenir una llista de paràgrafs, o `h1` per obtenir totes les capçaleres de primer nivell.
- **`getElementById`**: node de tipus element amb l'atribut `id` que coincideix amb el valor passat com a argument. Permet accedir de forma més directa a qualsevol element, però requereix afegir, abans, l'atribut `id` als elements que s'han de manipular.

```

1 <h1>Primera capçalera</h1>
2 <ul>
3   <li>Primer element de la primera llista</li>
4   <li>Segon element de la primera llista</li>
5   <li>Tercer element de la primera llista</li>
6 </ul>
7 <h1 class="secundari">Segona capçalera</h1>
8 <p id="contingut" class="secundari" paragraf>Un paràgraf</p>
9 <ul>
10  <li>Primer element de la segona llista</li>
11  <li>Segon element de la segona llista</li>
12  <li>Tercer element de la segona llista</li>
13 </ul>
14
15 <script>
16   var elementsSecundaris = document.getElementsByClassName('secundari');
17
18   console.log('Contingut textual dels elements amb classe="secundari":');
19   for (var i = 0; i < elementsSecundaris.length; i++) {
20     console.log(elementsSecundaris[i].textContent);
21   }
22
23   var elementsLlista = document.getElementsByTagName('li');
24   console.log('Contingut textual dels elements de la llista (<li></li>):');
25   for (i = 0; i < elementsLlista.length; i++) {
```

```

26     console.log(elementsLlista[i].textContent);
27 }
28
29 var elementContingut = document.getElementById('contingut');
30 console.log('Contingut textual del element amb id="contingut": ' +
31   elementContingut.textContent);
</script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/NRWwZx?editors=1011.

Fixeu-vos que el mètode `getElementsByClassName` retorna dos elements, ja que comprova totes les classes de cada element, de manera que tant secundari com secundari paràgraf són coincidències vàlides.

També cal destacar que en el cas de `getElementsByTagName` es retornen tots els elements amb l'etiqueta `li` del document, és a dir, inclou tant els elements de la primera llista com els de la segona.

Com és d'esperar, és possible fer modificacions sobre aquests nodes, per exemple per moure'ls o eliminar-los. En cas de voler eliminar-los, es pot presentar un problema inesperat: com que en eliminar el node canvia la seva posició a la llista, es produirà un error en intentar accedir als elements posteriors. Una possible solució seria la següent: fer servir un bucle amb `while` que elimini el primer element de la llista i es repeteixi fins que la longitud de la llista sigui 0.

```

1 <h1>Primera capçalera</h1>
2 <ul>
3   <li>Primer element de la primera llista</li>
4   <li>Segon element de la primera llista</li>
5   <li>Tercer element de la primera llista</li>
6 </ul>
7 <h1 class="secundari">Segona capçalera</h1>
8 <p id="contingut" class="secundari paràgraf">Un paràgraf</p>
9 <ul>
10  <li>Primer element de la segona llista</li>
11  <li>Segon element de la segona llista</li>
12  <li>Tercer element de la segona llista</li>
13 </ul>
14
15 <script>
16   var elementsLlista = document.getElementsByTagName('li');
17
18   while (elementsLlista.length>0) {
19     elementsLlista[0].parentNode.removeChild(elementsLlista[0]);
20   }
21 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/XjWVWE?editors=1011.

1.4.3 Cerca d'elements amb selectors ('querySelector' i 'querySelectorAll')

Selectors CSS

Podeu trobar informació detallada sobre els selectors CSS en l'enllaç següent:
goo.gl/o8ELvX.

Un element és **descendent directe** d'un altre quan el seu pare és aquest element.

Aquests mètodes permeten fer una cerca específica d'elements, passant com a argument una cadena de text amb un selector CSS. La diferència entre tots dos és que **el primer només retorna el primer element coincident**, mentre que **el segon retorna una llista amb totes les coincidències**.

Entre els selectors que es poden utilitzar hi ha:

- #identificador: que contingui id='‘identificador’'.
- .nom_classe: que contingui class='‘nom_classe’'.
- element: que sigui un element amb l'etiqueta element.
- element1 element2: que sigui un element amb l'etiqueta element2 descendant d'un element amb l'etiqueta element1. Cal destacar que **no cal que sigui descendant directe**.
- element1>element2: que sigui un element amb el tag element2 **descendent directe** d'un element amb l'etiqueta element1.
- [type='‘button’']: elements que tinguin l'atribut type i el seu valor sigui exactament button.
- ':first-child' (exemple de pseudoclasse): elements que siguin el primer fill de qualsevol altre element.
- 'element:first-child' (exemple de pseudoclasse): elements que siguin el primer fill de l'element pare.

Cal remarcar que el nombre de possibles selectors i les seves combinacions és molt gran i el seu estudi queda fora de l'abast d'aquest mòdul.

A continuació podeu veure un exemple d'ús del mètode querySelectorAll:

```

1 <h1>Primera capçalera</h1>
2 <ul>
3   <li>Primer element de la primera llista</li>
4   <li>Segon element de la primera llista</li>
5   <li>Tercer element de la primera llista</li>
6 </ul>
7 <h1 class="secundari">Segona capçalera</h1>
8 <p id="contingut" class="secundari paràgraf">Un paràgraf</p>
9 <ul>
10  <li>Primer element de la segona llista</li>
11  <li>Segon element de la segona llista</li>
12  <li>Tercer element de la segona llista</li>
13 </ul>
14
15 <script>
16   console.log('-- Capçaleras h1 amb classe secundari --');
17   var elementsTitols = document.querySelectorAll('h1.secundari');
18   for (var i=0; i<elementsTitols.length; i++) {
19     console.log(elementsTitols[i].textContent);

```

```

20 }
21
22 console.log('— Tots els elements de la primera llista —');
23 var elementsLlista = document.querySelectorAll('li');
24 for (var i=0; i<elementsLlista.length; i++) {
25   console.log(elementsLlista[i].textContent);
26 }
27
28 console.log('— Primers elements de les llistes —');
29 var elementsPrimers = document.querySelectorAll('li:first-child');
30 for (var i=0; i<elementsPrimers.length; i++) {
31   console.log(elementsPrimers[i].textContent);
32 }
33 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/XjWVky?editors=1011.

Com es pot apreciar, en el primer cas se seleccionen tots els elements amb l'etiqueta h1 amb la classe secundari. Així doncs, només troba una coincidència, ja que tot i que hi ha dos elements amb l'etiqueta h1 i dos elements amb la classe secundari, només hi ha un element que compleixi les dues condicions.

falta veure detall

En el segon cas s'han seleccionat tots els elements amb l'etiqueta li, de manera que el seu efecte resultat és idèntic al que retornaria el mètode `document.getElementsByTagName('li');`

A l'últim cas, en canvi, s'ha afegit al selector :first-child, de manera que en lloc de retornar tots els elements de les llistes, només retorna el primer element de cadascuna.

1.4.4 L'extensió 'HTMLDocument'

En el cas dels documents HTML, a banda de totes les propietats i mètodes de la interfície Document, s'aplica l'extensió HTMLDocument, que afegeix algunes propietats i mètodes extres que no es troben quan es treballa amb documents XML o SVG. Entre les més destacables es troben:

- **activeElement**: referència a l'element enfocat.
- **cookie**: cadena de text que permet consultar o modificar les galetes del document.
- **forms** (només lectura): llista d'elements de tipus `form` (formularis).
- **images** (només lectura): llista d'elements de tipus `img` (imatges).
- **getElementsByName(String name)**: retorna una llista d'elements amb el nom (atribut `name`, habitualment utilitzat en formularis) passat com a argument.
- **getSelection()**: retorna el text seleccionat al document.

- **write(String text)**: escriu el text al document.
- **writeln(String text)**: escriu el text al document i afegeix un salt de línia.

1.4.5 Altres propietats: 'body', 'documentElement' i 'forms'

La interfície `document` ofereix també dues propietats que permeten accedir directament als elements `body` i `html`: `body` i `documentElement` respectivament.

L'accés a l'element `body` és especialment útil per afegir la detecció de l'*event load*, ja que això permet detectar quan s'ha acabat de carregar el DOM. En cas contrari, es poden produir errors, per exemple si es volen realitzar modificacions al DOM abans que aquest s'hagi acabat de carregar completament.

```
1 document.body.onload = function() {
2   console.log("DOM carregat");
3 }
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/rrVmgb?editors=0012.

Fixeu-vos que `onload` és una drecera per l'*event load*, així doncs es podria haver fet servir també el mètode `addEventListener` per afegir la detecció de l'*event*.

Per altra banda, l'extensió `HTMLDocument` afegeix `forms` a la propietat, que permet accedir a una llista d'objectes que conté la informació de tots els formularis del document i, al seu torn, cadascun d'aquests formularis permet accedir als seus elements:

```
1 <form id="primer">
2   <input type="text" />
3 </form>
4
5 <form id="segon">
6   <input type="text" />
7 </form>
8
9 <script>
10  console.log('Nombre de formularis al document: ', document.forms.length);
11
12  for (var i = 0; i < document.forms.length; i++) {
13    console.log('id del formulari amb index' + i + ':', document.forms[i].id);
14  }
15 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/WGQERN?editors=1011.

Tot i que en determinats casos pot ser útil accedir a aquesta propietat per recórrer tots els formularis del document, és poc habitual. Per una banda, els documents no acostumen a contenir múltiples formularis i, per altra banda, quan s'ha de treballar amb múltiples formularis s'acostuma a requerir un control més precís (s'obtenen a través del seu `id`).

1.5 La interfície 'Element'

Aquesta interfície, **com** també **Document**, és herència de **Node** i d'**EventTarget**, per tant, permet manipular els elements i gestionar esdeveniments. A més a més, afegeix mètodes per enregistrar i desenregistrar observadors del mateix element.

Mentre que **Document** facilita la creació i manipulació del document que es mostra al navegador, la interfície **Element** permet manipular els elements concrets, manipulant les classes que els afecten, la llista d'atributs i afegint o eliminant detectors d'*events* específics.

Propietats d'"Element"

Podeu trobar més informació sobre la interfície **Element** en l'enllaç següent:
goo.gl/0bCjXA.

1.5.1 Informació bàsica dels elements: '**tagName**', '**className**', '**classList**' i '**id**'

Aquestes propietats permeten conèixer la informació bàsica d'un element. Fixeu-vos que aquesta és la que ens permet fer una cerca simple des del document, a través de la seva etiqueta, de les classes o de l'**id**. Vegem-les detingudament:

- **tagName** (només lectura): nom de l'etiqueta d'aquest element.
- **className**: cadena de caràcters separats per espais que inclou tots els noms de classes que afecten l'element (per exemple, en aplicar un full d'estils CSS). Es pot modificar, per tant, permet afegir noves classes i establir una nova cadena de text com a valor.
- **classList** (només lectura): llista amb el nom de les classes que es pot recórrer com un *array*.
- **id**: identificador únic de l'element dintre del document.

Com es pot apreciar a la llista anterior, no es pot modificar ni el **tagName** ni la llista de classes, però aquesta darrera pot modificar-se amb dos mètodes que aquesta proporciona:

- **add(String classe)**: afegeix a l'element la classe indicada pel paràmetre. Si ja hi era a l'element, no fa res. Té una versió que admet diferents classes, separades per comes, i que fa exactament el mateix, però per a cada classe que rep com a paràmetre.
- **remove(String nomClasse)**: suprimeix de l'element la classe indicada pel paràmetre. Si l'element no té la classe que es demana suprimir, no fa res. Té també una versió que admet diferents classes, separades per comes, i que suprimeix de l'element cadascuna de les classes rebudes com a paràmetre de la mateixa manera que es feia amb una.

Podem veure com utilitzar-les al següent **exemple**:

```

1 <style>
2   .vermell{
3     color:red;
4   }
5
6   .negreta{
7     font-weight:bold;
8   }
9 </style>
10 <html>
11   <p id="para">paràgraf de prova</p>
12 </html>
13 <script>
14 var paragraf=document.getElementById("para");
15
16   paragraf.classList.add("vermell");
17   paragraf.classList.add("negreta");
18
19   alert("Hem afegit els estils vermell i negreta.\n\n Ara eliminarem l'estil
20     vermell.\n\n Prem una tecla per continuar");
21
22   paragraf.classList.remove("vermell");
23
24   alert("Hem suprimit l'estil vermell.\n\n Ara eliminarem també negreta.\n\n
25     Prem una tecla per continuar.");
      paragraf.classList.remove("negreta");
</script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/NwEMmJ.

El següent **exemple treballa amb les propietats tagName i className**:

```

1 <h1 class="principal">Primera capçalera</h1>
2 <p id="contingut" class="principal paragraf" title="Aqui va un paràgraf">Un paràgraf</p>
3
4 <script>
5   var elementH1 = document.getElementsByTagName('h1')[0];
6   var elementP = document.getElementsByTagName('p')[0];
7
8   elementH1.tagName = 'H2';
9
10  console.log('-- Tag dels elements --');
11  console.log(elementH1.tagName);
12  console.log(elementP.tagName);
13
14  console.log('-- Classes dels elements --');
15  console.log(elementH1.className);
16  console.log(elementP.className);
17
18  console.log('-- Modificació de els classes dels elements --');
19  elementH1.className = elementH1.className + ' ampliat';
20  elementP.className = 'canvi per noves classes ';
21
22  console.log('-- Classes dels elements ampliats--');
23  console.log(elementH1.className);
24  console.log(elementP.className);
25
26  console.log('-- Llista de classes del paràgraf --');
27  for (var i = 0; i < elementP.classList.length; i++) {
28    console.log("Classe " + i + ":" + elementP.classList[i]);
29  }
30
31  console.log('-- Identificadors dels elements --');
32  console.log('Capçalera: ' + elementH1.id);
```

```

33 console.log('Paràgraf: ' + elementP.id);
34 elementH1.id = 'actualitzat';
35 elementP.id = elementP.id + ' actualitzat';
36
37 console.log('— Identificadors dels elements actualitzats—');
38 console.log('Capçalera: ' + elementH1.id);
39 console.log('Paràgraf: ' + elementP.id);
40 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/xExYXV?editors=1012.

Fixeu-vos que tant en el cas de `className` com a `id` es pot assignar com a valor una cadena de text, sigui una de nova o concatenant el valor anterior. En tots dos casos el valor per defecte que prenen és una cadena buida, de manera que es poden concatenar sense fer cap consideració especial.

Relació entre "className" i "classList"

El següent exemple mostra com, tal com és d'esperar, en modificar `className` també es modifica `classList`. Concretament, implementa els mètodes `afegirClasse` i `eliminarClasse`, que fan el mateix, respectivament, que `classList.add` i `classList.remove`, però sense utilitzar aquests. En la pràctica, aquest codi només té utilitat si es fa una aplicació per navegadors antics, que no proporcionen la propietat `classList`.

Una opció seria dividir la cadena de caràcters assignada a `className` fent servir el mètode `split`, i seguidament recórrer l'`array` generat per fer l'acció desitjada, però això no és necessari perquè la propietat `classList` ja conté aquest `array`:

```

1  <style>
2      .principal {
3          background-color: grey;
4      }
5
6      .vermell {
7          color: red;
8      }
9
10     .important {
11         font-weight: bold;
12     }
13 </style>
14
15 <h1 class="principal">Primera capçalera</h1>
16 <p id="contingut" class="principal paragraf" title="Aquí hi va un
17     paràgraf">Un paràgraf</p>
18
19 <script>
20     var afegirClasse = function(element, classe) {
21         var trobada = false;
22         for (var i=0; i <element.classList.length; i++) {
23             if (element.classList[i] == classe) {
24                 trobada = true; // Ja es troba a l'element, no cal afegir
25                     -la
26             }
27             if (!trobada) {
28                 element.className += ' ' + classe;
29             }
30         }
31         var eliminarClasse = function(element, classe) {
32             var nouClassName = '';

```

```

33   for (var i=0; i<element.classList.length; i++) {
34     if (element.classList[i] != classe) {
35       nouClassName += element.classList[i] + ' ';
36     }
37   }
38   element.className = nouClassName;
39 }
40
41 var elementH1 = document.getElementsByTagName('h1')[0];
42 var elementP = document.getElementsByTagName('p')[0];
43
44 afegirClasse(elementH1, 'vermell');
45 afegirClasse(elementH1, 'vermell');
46 console.log(elementH1.className); // No es duplica
47
48 eliminarClasse(elementH1, 'principal');
49 </script>

```

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

Com es pot comprovar, la funció `afegirClasse` afegeix la classe i evita possibles repeticions, i la funció `eliminarClasse` elimina la classe sense haver d'editar manualment la cadena de text i crea una nova cadena de text a partir de la llista de classes que no coincideixin amb la que es vol eliminar.

Per simplificar més la utilització d'aquestes funcions es poden afegir al prototipus d'`Element`, d'aquesta manera són accessibles directament per tots els elements. Modifiqueu el codi JavaScript de l'exemple anterior pel següent:

```

1 Element.prototype.afegirClasse = function(classe) {
2   var trobada = false;
3   for (var i = 0; i < this.classList.length; i++) {
4     if (this.classList[i] == classe) {
5       trobada = true; // Ja es troba a l'element, no cal afegir-
6       la
7     }
8     if (!trobada) {
9       this.className += ' ' + classe;
10    }
11  }
12
13 Element.prototype.eliminarClasse = function(classe) {
14   var nouClassName = '';
15   for (var i = 0; i < this.classList.length; i++) {
16     if (this.classList[i] != classe) {
17       nouClassName += this.classList[i] + ' ';
18     }
19   }
20   this.className = nouClassName;
21 }
22
23 var elementH1 = document.getElementsByTagName('h1')[0];
24 var elementP = document.getElementsByTagName('p')[0];
25
26 elementH1.afegirClasse('vermell');
27 elementH1.afegirClasse('vermell');
28 console.log(elementH1.className); // No es duplica
29
30 elementH1.eliminarClasse('principal');

```

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

Fixeu-vos que la funcionalitat és idèntica, però en lloc d'haver d'invocar una funció i passar cada vegada l'element que s'ha de modificar i la classe, ara es

poden invocar directament a partir de l'element concret que es vulgui manipular:
`elementH1.afegirClasse('vermell');`.

1.5.2 Obtenció dels descendents com HTML: 'innerHTML'

Aquesta propietat és molt interessant perquè permet obtenir i establir el codi HTML dels nodes descendents de l'element. És a dir, en lloc de crear una branca d'un arbre i afegir-la com a nodes, és possible assignar a aquesta propietat el codi HTML que correspondria:

```

1 <div id="contingut"></div>
2
3 <script>
4   var contingut = document.getElementById('contingut');
5   contingut.innerHTML = '<h1>Això és una capçalera</h1>';
6 </script>
```

Podeu veure aquest exemple en l'enllaç següent: <http://codepen.io/ioc-daw-m06/pen/VKwdGz?editors=1010>.

Com es pot apreciar, és fàcil de fer servir, però la cadena de codi HTML es pot complicar ràpidament si s'hi han d'afegir múltiples elements:

```

1 <div id="contingut"></div>
2
3 <script>
4   var contingut = document.getElementById('contingut');
5   contingut.innerHTML = '<h1>Això és una capçalera</h1><p>I això un paràgraf
6     amb un <a href="#">enllaç</a>.</p>';
</script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/NRWzJJ?editors=1010.

Fixeu-vos que només s'ha afegit un paràgraf amb un enllaç a continuació de la capçalera, però el codi és molt menys entenedor i és més difícil detectar els errors.

Per altra banda, com que es tracta d'una cadena de text, es pot manipular de la mateixa manera, per exemple: fent servir expressions regulars per fer canvis al text o concatenant valors per generar la cadena i seguidament assignant-la com a propietat:

```

1 <div id="contingut">Carregant dades...</div>
2
3 <script>
4 var estudiants = ['Josep', 'Maria', 'Carles', 'Montserrat'];
5 var codiHtml = '';
6
7 codiHtml += '<ul>';
8
9 for (var i = 0; i < estudiants.length; i++) {
10   codiHtml += '<li>' + estudiants[i] + '</li>';
11 }
12
13 codiHtml += '</ul>';
```

```

15  document.getElementById('contingut');
16  contingut.innerHTML = codiHTML;
17  </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/yaLEdL?editors=1010.

Cal destacar que **tot el contingut de l'element és reemplaçat**, per aquesta raó no es mostra el text “Carregant dades”: és reemplaçat per la llista de noms una vegada s’executa el codi JavaScript.

Per descomptat, **és possible consultar la propietat per obtenir el codi HTML corresponent als descendents del node**:

```

1 <div id="contingut">
2   <ul>
3     <li>Josep</li>
4     <li>Maria</li>
5     <li>Carles</li>
6     <li>Montserrat</li>
7   </ul>
8 </div>
9
10 <script>
11   document.getElementById('contingut');
12   console.log(contingut.innerHTML);
13 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/mAdKNK?editors=1011.

1.5.3 Atributs: ‘attributtes’, ‘getAttribute’, ‘removeAttribute’, ‘setAttribute’

La interfície Elements ofereix una propietat per consultar la llista completa d’atributs i tres mètodes per afegir, actualitzar o eliminar un atribut complet.

La propietat **attributes** (de només lectura) permet **consultar la llista completa d’atributs**. **Els atributs es retornen com un objecte que es pot recórrer com si es tractés d’un array**. Per consultar el nom i el valor de cadascun d’aquests atributs **es pot consultar la propietat name i value respectivament**:

```

1 <h1 id="titol" class="vermell principal">Aquesta és la capçalera</h1>
2
3 <script>
4 var titol = document.getElementById('titol');
5
6 for (var i = 0; i < titol.attributes.length; i++) {
7   console.log(titol.attributes[i].name + ': ' + titol.attributes[i].value);
8 }
9 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/WGNKAV?editors=1011.

Per altra banda, a diferència de la llista de noms de classes, amb els atributs no cal fer una implementació pròpia dels mètodes d'addició i eliminació, ja que la mateixa interfície els inclou:

- **getAttribute**: retorna el valor de l'atribut.
- **removeAttribute**: elimina l'atribut.
- **setAttribute**: estableix el valor de l'atribut.

```

1 <style>
2   .vermell {
3     color:red;
4   }
5 </style>
6
7 <h1 id="titol" class="vermell principal">Aquesta és la capçalera</div>
8
9 <script>
10 var titol = document.getElementById('titol');
11
12 titol.setAttribute('title', 'Això es mostra en posar el cursor a sobre');
13 titol.removeAttribute('class');
14 console.log('L\'identificador és: ' + titol.getAttribute('id'));
15 </script>
```

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

Cal destacar que a HTML5 es va afegir la possibilitat de definir atributs personalitzats; d'aquesta manera és possible, per exemple, emmagatzemar identificadors de productes o tipus especials d'elements:

```
1 <div data-id-producte="42">...</div>
```

Tot i que els navegadors admeten la definició d'atributs amb qualsevol nom, l'especificació requereix que incloguin el prefix `data-` i que no continguin majúscules, per exemple: `data-id-producte`. Aquests nous atributs es poden fer servir després per obtenir el seu valor o per seleccionar els elements fent servir selectors.

1.5.4 Modificar estils CSS: 'style'

A HTML és possible modificar els estils concrets d'un element a través de l'atribut `style`. **Normalment els estils aplicats d'aquesta forma tenen prioritat sobre qualsevol altre que afecti l'element**.

Tot i així, quan es vol tractar amb aquest atribut des de JavaScript no és tan simple com caldria esperar, perquè **s'han de tenir en compte dos aspectes fonamentals**:

- **El valor final de la propietat no és l'indicat a l'atribut, sinó el calculat, que pot estar modificat per altres regles de diferents orígens** (fulls d'estil, navegador...).

Propietat 'style'

Podeu trobar més informació sobre la propietat `style` en l'enllaç següent: goo.gl/BYz2IC.

- En consultar el valor de l'atribut `style` d'un element des de JavaScript, s'obté una col·lecció i no una cadena de text.

Per altra banda, si es vol treballar amb el contingut textual de la propietat es pot fer o bé fent servir el mètode `setAttribute` o modificant la propietat `cssText` de la col·lecció retornada per `style`:

```

1 <p id="paragraf1" style="color: red; font-weight: bold">Paràgraf 1</p>
2 <p id="paragraf2" style="color: red; font-weight: bold">Paràgraf 2</p>
3
4 <script>
5   var paragraf1 = document.getElementById('paragraf1');
6   var paragraf2 = document.getElementById('paragraf2');
7
8   paragraf1.setAttribute('style', 'color:green;');
9   paragraf2.style.cssText = 'color: blue';
10
11  console.log(paragraf1.style.cssText);
12  console.log(paragraf2.style.cssText);
13 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/zKKNwa?editors=1011.

Com es pot apreciar, tant fent servir el mètode `setAttribute` com assignant el valor directament a la propietat `cssText`, el valor de la propietat és substituït; per aquesta raó cap dels dos textos es mostra en negreta.

És a dir, en cas de voler modificar alguna de les propietats d'estil CSS s'hauria de crear una nova cadena de text amb el contingut correcte i reemplaçar el valor de la propietat `style.cssText`.

Una manera més adequada de treballar amb els estils concrets és fer-ho a través de la col·lecció. Fixeu-vos en l'exemple següent, partint d'un element HTML que conté dos estils (canvi de color a vermell i font gruixuda), s'elimina el color, es modifica el gruix de la font i s'afegeix un nou estil per augmentar-ne la mida. Per facilitar la reutilització i fer-lo més entenedor s'ha aplicat el disseny descendant i s'ha creat una funció per a cada acció:

```

1 <p id="paragraf" style="color: red; font-weight: bold">Paràgraf</p>
2
3 <script>
4   var mostrarEstils = function (element) {
5     var estil = element.style;
6     var css = window.getComputedStyle(element, null);
7
8     for (var i=0; i<estil.length; i++) {
9       console.log (estil[i] + ':' +css[estil[i]] + ";");
10    }
11  }
12
13 var afegirEstil = function (element, clau, valor) {
14   element.style[clau] = valor;
15 }
16
17 var eliminarEstil = function (element, clau) {
18   element.style[clau]=null;
19 }
20
21 var actualitzarEstil = function (element, clau, valor) {
```

```

22     afegirEstil(element, clau, valor);
23 }
24
25 var paragraf = document.getElementById('paragraf');
26
27 console.log("— Estil original —");
28 mostrarEstils(paragraf);
29 afegirEstil(paragraf, 'font-size', '30px');
30 eliminarEstil(paragraf, 'color');
31 actualitzarEstil(paragraf, 'font-weight', 'lighter');
32
33 console.log("— Estil modificat —");
34 mostrarEstils(paragraf);
35 </script>

```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-daw-m06/pen/dprOEx?editors=1011.

El primer que us cridarà l'atenció és la complexitat que té mostrar els estils, ja que s'ha d'invocar el mètode `window.getComputedStyle` per obtenir un objecte amb la informació de tots els valors calculats per l'objecte.

S'ha de tenir en compte que no només s'apliquen els estils de la propietat, sinó que s'apliquen també els propis del navegador i els dels fulls d'estil carregats. És possible obtenir només els valors aplicats al propi estil, però aquests valors no són finals: pot ser que una altra regla CSS ho hagi sobreescrit, per exemple, si s'ha aplicat el modificador `!important` a una regla que l'afecti.

Una vegada s'ha obtingut la llista de propietats calculades, es recorren els noms dels estils establerts a la propietat `style` com si es tractés d'un `array`, ja que la propietat `style` és una col·lecció que té una propietat `length`, i cada element es troba referenciat per un enter que es fa servir com a índice.

D'aquesta manera, combinant els valors calculats amb els estils aplicats a l'element, es pot mostrar una llista dels valors reals aplicats a l'element.

Alternativament, imitant el comportament de la biblioteca jQuery, és possible modificar la interfície `Element` per afegir al seu prototip els mètodes `css` i `mostrarEstils`, de manera que tots els elements tinguin accés a aquesta funcionalitat:

```

1 <p id="paragraf" style="color: red; font-weight: bold">Paràgraf</p>
2
3 <script>
4     Element.prototype.mostrarEstils = function() {
5         var estil = this.style;
6         for (var i = 0; i < estil.length; i++) {
7             console.log(estil[i] + ':' + this.css(estil[i]) + ";");
8         }
9     }
10
11     Element.prototype._mostrarEstil = function(clau) {
12         var css = window.getComputedStyle(this, null);
13         return css[clau];
14     }
15
16     Element.prototype.css = function(clau, valor) {
17         this.style[clau] = valor;
18
19         return this._mostrarEstil(clau);
20     }
21

```

```

22  var paragraf = document.getElementById('paragraf');
23
24  console.log("— Estil original —");
25  paragraf.mostrarEstils();
26
27  paragraf.css('font-size', '30px');
28  paragraf.css('color', '');
29  paragraf.css('font-weight', 'lighter');
30
31  console.log("— Estil modificat —");
32  paragraf.mostrarEstils();
33 </script>

```

Podeu veure aquest exemple en el següent enllaç: codepen.io/ioc-dawm06/pen/VKaPNP?editors=1011.

Com es pot apreciar, a partir d'aquesta implementació es molt fàcil accedir i modificar els estils d'un node, tant si és per afegir-hi un estil nou, modificar-ne algun o eliminar-lo.

1.5.5 Relacions entre elements: '`previousElementSibling`', '`nextElementSibling`', '`firstElementChild`' i '`lastElementChild`'

Tot i que la interfície Element deriva de Node i, consegüentment, disposa de les propietats `previousSibling` i `nextSibling`, aquestes **no retornen els elements, sinó els nodes**. És a dir, inclouen tot tipus de nodes, com per exemple els nodes de tipus text que es generen en afegir un salt de línia o un tabulador.

En canvi, gràcies a `previousElementSibling` i `nextElementSibling` (totes dues propietats són de només lectura), es poden consultar els nodes anterior i posterior directament:

```

1  <ul>
2      <li>Primer element de la llista</li>
3      <li id="central">Segon element de la llista</li>
4      <li>Tercer element de la llista</li>
5  </ul>
6
7  <script>
8      var central = document.getElementById('central');
9
10     console.log('El contingut de l\'element anterior és: ', central.
11                     previousElementSibling.textContent);
12
13     console.log('El contingut de l\'element següent és: ', central.
14                     nextElementSibling.textContent);
15 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/rrVydz?editors=1011.

Com es pot comprovar, a la consola es mostra correctament el contingut textual del primer i el tercer element, ignorant els nodes de tipus text.

De la mateixa manera, es pot accedir al primer i l'últim element a través de les propietats `firstElementChild` i `lastElementChild`:

```

1 <ul id="llista">
2   <li>Primer element de la llista</li>
3   <li>Segon element de la llista</li>
4   <li>Tercer element de la llista</li>
5 </ul>
6
7 <script>
8   var central = document.getElementById('llista');
9
10  console.log('El contingut del primer element és: ', central.firstChild
11    .textContent);
12
13  console.log('El contingut del darrer element és: ', central.lastElementChild.
14    .textContent);
15 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/xEGqWa?editors=1011.

1.5.6 Cerca d'elements descendents simple: 'getElementsByClassName', 'getElementsByTagName'

La funcionalitat dels mètodes `getElementsByClassName` i `getElementsByTagName` és idèntica a la que proporciona la interfície Document amb la peculiaritat que la cerca **només es fa entre els descendents del mateix element**.

Cal destacar que no s'inclou un mètode `getElementById`, ja que els identificadors són únics i no és rellevant si és o no descendant d'un element concret, com es pot comprovar en l'exemple següent:

```

1 <ul>
2   <li>Primer element de la llista</li>
3   <li id="central">
4     <ul>
5       <li>Primer element de la subllista</li>
6       <li>Segon element de la subllista</li>
7     </ul>
8   </li>
9   <li>Tercer element de la llista</li>
10 </ul>
11
12 <script>
13   var central = document.getElementById('central');
14   var elementsSubLlista = central.getElementsByTagName('li');
15
16   for (var i=0; i<elementsSubLlista.length; i++) {
17     console.log(elementsSubLlista[i].textContent);
18   }
19 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/RGwBxE?editors=1011.

Cal destacar que tot i que inicialment s'invoca `getElementById` per obtenir la referència a l'element amb identificador `central`, a continuació se cerquen els

elements a partir d'aquest i no pas de document. Així doncs, es limita la cerca als seus descendents i el resultat obtingut són els elements de la subllista.

1.5.7 Cerca d'elements descendents amb selector: 'querySelector', 'querySelectorAll'

Igual que els mètodes `getElementsByName` i `getElementsByClassName`, la funcionalitat d'aquests és molt similar a la dels mètodes amb el mateix nom de la interfície Document, però en aquest cas la cerca també es limita als elements descendents del mateix element.

En l'exemple següent podeu comprovar com es fa una selecció a partir d'un element, de manera que només retorna la llista d'elements descendents i no pas totes les coincidències del document:

```

1  <ul>
2      <li>Primer element de la llista</li>
3      <li id="central">
4          <ul data-quantitat="0">
5              <li data-quantitat="0">Primer element de la subllista</li>
6              <li data-quantitat="19">Segon element de la subllista</li>
7              <li data-quantitat="0">Tercer element de la subllista</li>
8          </ul>
9      </li>
10     <li data-quantitat="0">Tercer element de la llista</li>
11 </ul>
12
13 <script>
14 var central = document.getElementById('central');
15 var elementsSubLlista = central.querySelectorAll('li[data-quantitat="0"]');
16
17 for (var i = 0; i < elementsSubLlista.length; i++) {
18     console.log(elementsSubLlista[i].textContent + ' (' + elementsSubLlista[i].getAttribute('data-quantitat') + ')');
19 }
</script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/gwOjBN?editors=1011.

Fixeu-vos que s'ha fet servir un atribut propi, `data-quantitat`, i que el selector `li['data-quantitat='0']` ha filtrat els resultats de manera que només es mostren a la consola el primer i tercer element de la subllista, sense incloure ni l'element `ul` ni el segon element, que té com a valor de l'atribut 19. A més a més, com és d'esperar, tampoc no s'hi ha inclòs el tercer element de la llista, ja que no és descendant de l'element `central`.

1.6 Integració de la detecció d'events amb el DOM

La programació d'*events* permet interactuar amb l'aplicació web i afegeix la detecció d'*events* als elements (o al mateix document), de manera que l'aplicació pot reaccionar de diferents maneres, per exemple: afegint files a una taula, eliminant-les, modificant les classes CSS per canviar-ne els colors o aplicar efectes...

S'ha de tenir en compte que quan es dispara un *event* en un element fill, si no s'especifica el contrari, aquest travessa tots els nodes pare, de manera que finalment són rebuts pel node arrel, que és *document*.

Podeu trobar més informació sobre els *events* a la unitat "Programació d'events".

És a dir, si s'implementa la gestió de l'esdeveniment `submit` al document, quan s'envii un formulari (es dispara l'*event submit*) contingut en un element fill, el document detectarà aquest *event*.

A continuació podeu trobar una [llista dels *events* més destacables](#) a l'hora de tractar amb un document o element:

- **blur**: es dispara quan es perd el focus.
- **focus**: es dispara quan l'element rep el focus.
- **click**: es dispara quan es produeix un clic sobre el document.
- **dblclick**: es dispara quan es produeix un doble clic sobre el document.
- **keydown, keypress,keyup**: es disparen quan es detecta que s'ha premut una tecla.
- **load**: es dispara quan es completa la càrrega del document.
- **scroll**: es dispara quan es desplaça verticalment o horitzontalment el document.
- **submit**: es dispara quan s'envia un formulari.
- **input**: es dispara quan es modifica el valor d'un camp d'entrada (per exemple, un quadre de text).
- **change**: es dispara quan es modifica el valor d'un camp d'entrada però, a diferència d'`input`, no es dispara l'*event* per cada canvi produït, només es dispara en determinades circumstàncies, com per exemple en canviar el focus a un altre element.

Això permet, per exemple, controlar quan es fa clic sobre qualsevol element d'una pàgina web, sense haver de gestionar aquest *event* en cadascun dels elements que formen la pàgina.

La interfície ofereix mètodes per "escutar" quan es dispara un *event* (`addEventListener`), per deixar d'escutar-lo (`removeEventListener`) i tot un

Quan es parla d'*events*, el terme 'escutar' (*listening* en anglès) fa referència a la detecció de l'*event* al document o element.

seguit de dreceres en forma de mètodes que permeten realitzar les dues accions per *events* concrets.

La utilització de les dreceres comporta més limitacions, ja que no es pot afegir més d'una funció per a cada *event*, però pot simplificar el codi en casos molt simples (per exemple, per controlar la càrrega de documents o imatges).

1.6.1 Afegir detecció d'esdeveniments: 'addEventListener'

El mètode `addEventListener` permet enregistrar una funció que serà invocada quan es dispari l'*event* passat com a argument a l'element, per exemple quan es faci un doble clic sobre ell o es detecti un canvi en un camp del formulari.

En l'exemple següent podeu veure com s'ha afegit un comptador de caràcters que indica el nombre de caràcters introduït en una àrea de text i que, a més a més, en modifica el color segons els següents paràmetres:

- Menys de 100 caràcters o més de 156: text en vermell, el text és massa curt o massa llarg.
- Menys de 150 caràcters: text en taronja, el text és curt.
- Entre 150 i 156: text en verd, text amb llargària òptima.

```

1 <style>
2   label,
3   small {
4     display: block;
5   }
6
7   .massa-curt-o-llarg {
8     color: red;
9   }
10
11  .curt {
12    color: orange;
13  }
14
15  .correcte {
16    color: green;
17  }
18 </style>
19
20 <div>
21   <label>Introduix el contingut de la descripció meta</label>
22   <textarea name="meta-description" id="meta-description" placeholder="Introduix fins a 156 caràcters" / cols="80" rows="3"></textarea>
23   <small>Nombre de caràcters: <span id="comptador" className="curt">0</span></small>
24 </div>
25
26 <script>
27   var camp = document.getElementById('meta-description'),
28     comptador = document.getElementById('comptador');
29
30   var actualitzarComptador = function() {
31     var llargaria = camp.value.length
32     comptador.textContent = llargaria;

```

```

33 if (llargaria > 156 || llargaria < 100) {
34   comptador.className = 'massa-curt-o-llarg';
35 } else if (llargaria < 150) {
36   comptador.className = 'curt';
37 } else {
38   comptador.className = 'correcte';
39 }
40 }
41
42 camp.addEventListener('input', actualitzarComptador);
43 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/NddQWZ.

Com podeu apreciar, afegir un detector d'*events* és molt senzill, ja que només cal invocar el mètode `addEventListener` passant com a arguments el nom de l'*event* al qual es vol reaccionar i la funció que s'ha d'executar quan es dispara: `camp.addEventListener('input', actualitzarComptador);`

A continuació podeu comprovar, modificant el mateix exemple, la diferència més important entre els *events* `input` i `change`. Mentre que el primer es dispara cada vegada que es modifica el contingut del `textarea`, el segon només es dispara en canviar el focus (per exemple, fent clic en un altre punt del document):

```
1 camp.addEventListener('change', actualitzarComptador);
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/Xjbddo.

Alternativament, es podria fer servir la drecera `oninput`, modificant la línia en què s'afegeix la detecció d'*events* per la següent:

```
1 camp.oninput = actualitzarComptador;
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/jrPWvm.

Aquest format és més concís, però podria provocar errors si es fes servir aquest codi en un projecte més avançat en el qual s'haguessin de fer diferents accions quan es detectessin canvis (per exemple, guardar un esborrany): si s'ha fet servir aquest format en tots dos casos, només s'executarà la funció afegida en darrer lloc.

Per altra banda, les dreceres permeten incrustar el codi JavaScript directament al codi HTML com si es tractés d'un atribut:

```

1 <button onclick="escriureMissatge();">Escriure missatge a la consola</button>
2
3 <button onclick="alert('Alerta!');">Mostra una alerta</button>
4
5 <script>
6   var escriureMissatge=function() {
7     console.log('Aquesta funció ha estat invocada des del botó')
8   }
9 </script>

```

L'etiqueta 'meta-description'

L'etiqueta `meta-description` conté el text que mostren els cercadors a tall de resum quan surt el llistat de pàgines trobades. La llargària màxima recomanada de l'etiqueta per millorar el SEO és de 156 caràcters.

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/vXOKJd?editors=1011.

Com es pot apreciar, es poden invocar tant funcions pròpies com funcions predefinides de JavaScript.

1.6.2 Eliminar detecció d'esdeveniments: 'removeEventListener'

En alguns casos pot interessar-vos eliminar un detector d'*events*, per exemple, per deshabilitar un botó d'enviar formulari si els camps de text són buits o no s'ha passat la validació.

En el següent exemple podeu veure com s'afegeixen i eliminan els detectors dinàmicament, de manera que només es pot clicar sobre el quadre blau:

```

1 <style>
2 div {
3   width: 100px;
4   height: 100px;
5   background-color: grey;
6   float: left;
7   margin: 5px;
8 }
9
10 span {
11   padding: 0 3px;
12 }
13
14 .seleccionat {
15   background-color: green;
16   color:white;
17 }
18
19 .proper {
20   background-color: blue;
21   color:white;
22 }
23 </style>
24
25 <p>El quadre clicable es mostra de color <span class="proper">blau</span> i l'últim quadre clicat de color <span class="seleccionat">verd</span></p>
26 <div id="a"></div>
27 <div id="b"></div>
28 <div id="c"></div>
29
30 <script>
31 var quadreA = document.getElementById('a');
32 var quadreB = document.getElementById('b');
33 var quadreC = document.getElementById('c');
34
35 var seleccionarA = function() {
36   quadreA.className = 'seleccionat';
37   quadreB.className = 'proper';
38   quadreC.className = '';
39   quadreA.removeEventListener('click', seleccionarA);
40   quadreB.addEventListener('click', seleccionarB);
41   console.log(listenerB);
42 }
43
44 var seleccionarB = function() {
45   quadreB.className = 'seleccionat';
46   quadreC.className = 'proper';

```

```

47 quadreA.className = '';
48 quadreB.removeEventListener('click', seleccionarB);
49 quadreC.addEventListener('click', seleccionarC);
50 }
51
52 var seleccionarC = function() {
53     quadreC.className = 'seleccionat';
54     quadreA.className = 'proper';
55     quadreB.className = '';
56     quadreC.removeEventListener('click', seleccionarC);
57     quadreA.addEventListener('click', seleccionarA);
58 }
59
60 var inicialitzar = function() {
61     quadreA.className = 'proper';
62     listenerA = quadreA.addEventListener('click', seleccionarA);
63 }
64
65 inicialitzar();
66 </script>

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/gwpAKx.

Com es pot apreciar, només cal indicar el nom de l'*event* (el seu tipus) i la funció que s'ha d'eliminar. És a dir, en cas que hi hagués múltiples funcions lligades al mateix *event*, només s'eliminaria la indicada.

En cas de fer servir dreceres, per eliminar un detector només cal assignar el valor nul al mètode:

```

1 var quadreA = document.getElementById('a');
2 var quadreB = document.getElementById('b');
3 var quadreC = document.getElementById('c');
4
5 var seleccionarA = function() {
6     quadreA.className = 'seleccionat';
7     quadreB.className = 'proper';
8     quadreC.className = '';
9     quadreA.onclick = null;
10    quadreB.onclick = seleccionarB;
11 }
12
13 var seleccionarB = function() {
14     quadreB.className = 'seleccionat';
15     quadreC.className = 'proper';
16     quadreA.className = '';
17     quadreB.onclick = null;
18     quadreC.onclick = seleccionarC;
19 }
20
21 var seleccionarC = function() {
22     quadreC.className = 'seleccionat';
23     quadreA.className = 'proper';
24     quadreB.className = '';
25     quadreC.onclick = null;
26     quadreA.onclick = seleccionarA;
27 }
28
29 var inicialitzar = function() {
30     quadreA.className = 'proper';
31     quadreA.onclick = seleccionarA;
32 }
33
34 inicialitzar();

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-dawm06/pen/LRVNwR.

Com es pot veure, independentment de si es fan servir dreceres o no, crear una funció per a cada element que cal detectar no és gens pràctic. A l'exemple anterior si és volgués augmentar el nombre de quadres, caldria modificar les tres funcions existents i afegir-n'hi una altra. A més a més, totes les funcions són pràcticament idèntiques. Aquest és un senyal molt clar que cal refactoritzar el codi.

Una opció ésaprofitar les relacions entre nodes: en aquest cas el proper node sempre és el següent, i un cop no n'hi ha més es pot navegar directament al primer fill del node pare:

```

1 <style>
2 div#quadres div {
3   width: 100px;
4   height: 100px;
5   background-color: grey;
6   float: left;
7   margin: 5px;
8 }
9
10 span {
11   padding: 0 3px;
12 }
13
14 #quadres div.seleccionat {
15   background-color: green;
16   color:white;
17 }
18
19 #quadres div.proper {
20   background-color: blue;
21   color:white;
22 }
23 </style>
24
25 <p>El quadre clicable es mostra de color <span class="proper">blau</span> i l'últim quadre clicat de color <span class="seleccionat">verd</span></p>
26 <div id="quadres">
27   <div></div>
28   <div></div>
29   <div></div>
30   <div></div>
31   <div></div>
32   <div></div>
33 </div>
34
35 <script>
36 var quadres = document.getElementById('quadres');
37 var seleccionat;
38 var proper;
39
40 var seleccionar = function() {
41   // S'elimina l'estil i el detector del seleccionat anterior
42   console.log(seleccionat, proper);
43   seleccionat.className = '';
44   seleccionat.removeEventListener('click', seleccionar);
45
46   // S'actualitza el seleccionat al proper quadre
47   seleccionat = proper;
48   seleccionat.className = 'seleccionat';
49
50   // Es determina el proper quadre i s'afegeix el detector
51   proper = seleccionat.nextElementSibling;
52   if (!proper) {

```

```

53     proper = quadres.firstElementChild;
54   }
55   proper.className = 'proper';
56   proper.addEventListener('click', seleccionar);
57 }
58
59 var inicialitzar = function() {
60   seleccionat = quadres.firstElementChild;
61   proper = seleccionat.nextElementSibling;
62   seleccionar();
63 }
64
65 inicialitzar();
66 </script>
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/KgprwE.

Com es podeu veure, s'ha modificat el codi HTML per afegir un element contenidor (quadres); ja no cal fer servir un id per a cada quadre i s'ha refinat el codi CSS perquè els estils només han d'affectar els elements div descendents de quadres.

Ara bé, si no es poden fer servir les relacions entre elements, es pot aprofitar el context en el qual s'invoquen les funcions. S'ha de tenir en compte que dintre de la funció invocada, el seu context és el node al qual s'ha lligat el detector. Així doncs, és possible generalitzar aquestes funcions per fer servir el node com a context.

En l'exemple següent podeu comprovar com es genera aleatoriament quin serà el proper element, només cal reemplaçar el codi JavaScript pel següent:

```

1 var quadres = document.getElementById('quadres');
2 var darrerSeleccionat;
3
4 var seleccionar = function() {
5   var proper;
6
7   // S'elimina l'estil i el detector del seleccionat anterior
8   if (darrerSeleccionat) {
9     darrerSeleccionat.className = '';
10   }
11
12   // S'actualitza l'element actual
13   this.removeEventListener('click', seleccionar);
14   this.className = 'seleccionat';
15   darrerSeleccionat = this;
16
17   // Es determina el proper quadre i s'afegeix el detector
18   proper = seleccionarQuadreAleatori();
19   proper.className = 'proper';
20   proper.addEventListener('click', seleccionar);
21 }
22
23 var seleccionarQuadreAleatori = function(noIncloure) {
24   var maxIndex = quadres.childNodes.length - 1;
25
26   // S'han de descartar el darrer node seleccionat per no repetir i els nodes
27   // de text
28   do {
29     index = Math.floor(Math.random() * (maxIndex + 1));
30     proper = quadres.childNodes[index];
31   } while (proper.nodeName != 'DIV' || proper === darrerSeleccionat);
32
33   return proper;
34 }
```

```

33 }
34
35 seleccionar();

```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/VKLPxQ.

Fixeu-vos que en aquest exemple no cal controlar quin és l'element actual ni el proper globalment, només es guarda la referència al darrer seleccionat per poder esborrar-lo quan es cliqui el següent element i evitar que el mateix element es repeteixi. També s'ha eliminat la funció d'inicialització: no cal inicialitzar cap variable, només cal invokeselector.

Per altra banda, s'ha afegit la funció `seleccionarQuadreAleatori`, que genera un nombre entre 0 i el nombre de nodes descendents de l'element pare menys 1 (recordeu que els índexs dels *arrays* es compten començant per 0) i es van comprovant els nodes fins que se'n troba un que sigui de tipus DIV i no sigui el darrer seleccionat. Heu de tenir en compte que `childNodes` retorna una llista de nodes i, consegüentment, s'inclouen també els nodes amb text que es troben entre els elements: salts de línia i tabulacions.

1.7 Cas pràctic: generador de factures

A continuació trobareu un exemple pràctic, un generador de factures. Hi podeu veure com s'integren les característiques principals de les interfícies del DOM, com pot ser la cerca, la consulta, la creació i l'eliminació d'elements.

Primerament, afegiu el següent codi HTML i CSS en un nou document per crear l'estructura del generador de factures, que està dividit en dues seccions:

- Una àrea d'entrada de dades formada per 3 quadres de text i un botó.
- Una taula que contindrà les línies de la factura i el peu que mostra els totals.

```

1 <style>
2   div {
3     margin-bottom: 15px;
4   }
5
6   label {
7     width: 100px;
8     display: inline-block;
9   }
10
11  input {
12    width: 100px;
13  }
14
15  th {
16    width:136px;
17  }
18
19  td {

```

```

20     text-align:right;
21 }
22
23 td:first-child {
24   text-align:left;
25 }
26
27 tfoot th {
28   text-align:left;
29 }
30
31 tfoot th:last-child {
32   text-align:right;
33 }
34
35 thead, tfoot, tbody, table {
36   border: 1px solid black;
37 }
38
39 thead, tfoot {
40   background-color: grey;
41 }
42
43 tbody tr:hover {
44   background-color:#2897E8;
45 }
46 </style>
47
48 <div>
49   <h3>Introducció de productes a la factura</h3>
50   <label for="producte">Producte:</label>
51   <input type="text" id="producte" />
52   <label for="quantitat">Quantitat:</label>
53   <input type="number" id="quantitat" value="0" />
54   <label for="preu-unitari">Preu unitari:</label>
55   <input type="number" id="preu-unitari" value="0" />
56   <button id="afegir">Afegir</button>
57 </div>
58
59
60 <table rules="groups">
61   <thead>
62     <tr>
63       <th>Producte</th>
64       <th>Quantitat</th>
65       <th>Preu unitari</th>
66       <th>Preu total</th>
67       <th>Accions</th>
68     </tr>
69   </thead>
70   <tfoot>
71     <tr>
72       <th colspan="4">Base imposable</th>
73       <th><span id="base-imposable">0</span>€</th>
74     </tr>
75     <tr>
76       <th colspan="4" data-iva="0.21">IVA 21%</th>
77       <th><span id="iva">0</span>€</th>
78     </tr>
79     <tr>
80       <th colspan="4">Total factura</th>
81       <th><span id="total">0</span>€</th>
82     </tr>
83   </tfoot>
84   <tbody>
85   </tbody>
86 </table>
```

Quant al codi HTML, cal destacar que s'ha afegit la propietat **id** per als següents elements:

- Entrada de text producte per introduir el nom del producte.
- Entrada de text quantitat per introduir la quantitat de productes facturats.
- Entrada de text preu-unitari per introduir el preu unitari del producte.
- Botó afegir per afegir la nova línia a la factura.
- Element base-imposable per mostrar la base imposable total.
- Element iva per mostrar el total calculat per l'IVA.
- Element total per mostrar el total a pagar.

Addicionalment, s'ha afegit l'atribut propi data-iva amb el valor 0.21 de manera que és possible modificar el percentatge d'IVA que s'ha d'aplicar a la factura modificant, directament, aquest atribut. Així doncs, és possible modificar aquest valor per aplicar, per exemple, un 7% d'IVA (0.07).

El codi CSS només s'utilitza per donar estil al document. S'ha aplicat un fons de color gris a la capçalera i al peu de la taula. A més a més, per destacar quina és la fila a la qual afectaran les accions, aquesta canvia de color quan el cursor és a sobre.

El codi JavaScript, tot i que a primera vista pot semblar complicat, és força senzill. No és res més que l'aplicació de les funcionalitats i l'accés a les propietats dels elements i el document.

Primer de tot hi ha la funció `inicialitzar`, que és l'encarregada d'inicialitzar tots els elements necessaris de l'aplicació. Aquesta funció és invocada automàticament quan es completa la càrrega del DOM:

```

1 var inicialitzar = function() {
2   var boto = document.getElementById('afegir');
3   boto.onclick = afegirLinia;
4 }
5
6 // Inicialització de l'aplicació quan es carregui el DOM
7 document.body.onload = inicialitzar;

```

Dins de la funció `inicialitzar` se cerca l'element amb identificador `afegir`, que correspon al botó i assigna a l'*event* `click` la funció `afegirLinia`, fent servir la drecera `onclick`. Seguidament, s'assigna aquesta funció a l'*event* `load` de `document.body`, de manera que aquesta funció s'executa una vegada s'acaba de carregar el DOM.

Fixeu-vos que si en canviu l'ordre, l'aplicació no funcionarà correctament. Primer s'ha de declarar la funció o el valor que s'assignarà a `document.body.onload`, que serà `null`.

La següent funció és `afegirLinia`, que és invocada quan es fa clic al botó `afegir`:

```

1 var afegirLinia = function() {
2   var nomProducte = document.getElementById('producte').value;
3   var quantitatProducte = document.getElementById('quantitat').value;

```

```

4 var preuUnitari = document.getElementById('preu-unitari').value;
5 var totalProducte = quantitatProducte * preuUnitari;
6
7 afegirFilaTaula(nomProducte, quantitatProducte, preuUnitari, totalProducte);
8 recalcularTotal();
9 netejarLinia();
10 }

```

Com es pot apreciar, en la implementació d'aquest programa s'ha aplicat el disseny descendent, de manera que s'ha dividit el codi en múltiples funcions específiques.

En primer lloc, s'obté el valor dels tres camps de text. Com que no cal guardar la referència a l'element, s'accedeix directament a la seva propietat value: document.getElementById('producte').value. Així, doncs, nomProducte contindrà el valor de l'element amb identificador producte. Una manera menys concisa d'implementar-lo seria la següent:

```

1 var elementNomProducte = document.getElementById('producte');
2 var nomProducte = elementNomProducte.value;

```

Una vegada s'han obtingut la quantitatProducte i el preuUnitari, es calcula el preu total i es passen aquests quatre valors a la funció afegirFilaTaula per afegir les dades a la taula. Seguidament s'invoca recalcularTotal (que recalcula la base imposable, l'IVA i l'import total), i finalment netejarLinia, per eliminar les dades de l'entrada de text.

La funció afegirFilaTaula, tot i que és la més llarga de l'aplicació, és molt simple: bàsicament es repeteix el mateix codi per a cada columna. Es podria haver fet servir un *array alternativament*, però per simplificar el codi i fer-lo més entenedor s'ha optat per *duplicar-lo*:

```

1 var afegirFilaTaula = function(nomProducte, quantitatProducte, preuUnitari,
2   totalProducte) {
3   var cosTaula = document.querySelector('tbody');
4
5   var fila = document.createElement('tr');
6
7   var col1 = document.createElement('td');
8   var col2 = document.createElement('td');
9   var col3 = document.createElement('td');
10  var col4 = document.createElement('td');
11  var col5 = document.createElement('td');
12
13  col1.innerHTML = nomProducte + ' (detall)';
14  col2.innerHTML = quantitatProducte;
15  col3.innerHTML = preuUnitari + '€';
16  col4.innerHTML = totalProducte + '€';
17  col5.innerHTML = '(eliminar)';
18
19  col1.addEventListener('click', mostrarDetall);
20  col5.addEventListener('click', eliminarFila);
21
22  fila.appendChild(col1);
23  fila.appendChild(col2);
24  fila.appendChild(col3);
25  fila.appendChild(col4);
26  fila.appendChild(col5);
27
28  cosTaula.appendChild(fila);
}

```

Primer de tot, s'obté la referència al cos de la taula (l'element `tbody`) amb el mètode `document.querySelector`, perquè aquest permet accedir directament a l'element com si fos un selector CSS i sempre retorna un únic element (al contrari de `document.getElementsByTagName`, que retorna un `array`).

A continuació, es creen els nous elements que s'afegiran:

- Una nova fila: `document.createElement('tr')`.
- Cinc noves cel·les, una per a cada columna: `document.createElement('td')`.

Una vegada s'han creat les columnes (recordeu que es tracta de nodes de tipus `element`) s'assigna el contingut corresponent a cadascuna d'elles, i s'estableix la seva propietat `innerHTML`.

Tot seguit, s'afegeix la detecció de l'`event click` a les columnes 1 i 5, de manera que en clicar sobre les columnes s'invoca les funcions `mostrarDetall` o `eliminarFila`, respectivament.

Amb els continguts i la detecció d'`events` afegida, només resta afegir les columnes a la fila (`fila.appendChild(col1)`) i, seguidament, la fila a la taula (`cosTaula.appendChild(fila)`).

El següent mètode, `recalcularTotal`, és l'encarregat de recalcular els totals que es mostren al peu de la taula. Aquest mètode és més complex, perquè es treballa amb selectors, accés als elements descendents i germans i s'accedeix a propietats.

```

1 var recalcularTotal = function() {
2     var files = document.querySelectorAll('tbody tr');
3     var valorBase = 0;
4
5     for (var i=0; i<files.length; i++) {
6         var columnaUltima = files[i].lastElementChild;
7         var columnaPenultima = columnaUltima.previousElementSibling;
8         var valorTotalFila = parseFloat(columnaPenultima.textContent);
9         valorBase += valorTotalFila;
10    }
11
12    var elementBase = document.getElementById('base-imposable');
13    elementBase.innerHTML = valorBase;
14
15    var elementPercentatgeIVA = document.querySelector('[data-iva]');
16    var valorPercentatgeIVA = elementPercentatgeIVA.getAttribute('data-iva');
17    var valorIVA = parseFloat(valorPercentatgeIVA * valorBase);
18
19    var elementIVA = document.getElementById('iva');
20    elementIVA.innerHTML = valorIVA;
21
22    var elementTotal = document.getElementById('total');
23    elementTotal.innerHTML = valorBase + valorIVA;
24 }
```

Fixeu-vos que per obtenir totes les files s'invoca `document.querySelectorAll('tbody tr')`, de manera que s'obtenen totes les files (elements de tipus `tr`) que es trobin dins de l'element `tbody`. És a dir, s'exclouen les files de la capçalera i del peu de la taula. Cal destacar

que a diferència de `querySelector`, que retorna només un únic element, `querySelectorAll` retorna una col·lecció d'elements.

Seguidament es recorre aquesta col·lecció per extreure la informació de cada línia de la factura i calcular-ne el valor base total. Com que la columna que interessa és la cinquena, s'ha optat per accedir a l'últim element descendente de la fila (`files[i].lastElementChild`) i a continuació a l'element anterior (`columnaUltima.previousElementChild`).

Per assegurar que el valor que s'afegeix és un nombre real (és a dir, no s'interpreta com una cadena de text), s'invoca la funció `parseFloat` i aquest valor s'afegeix a la variable `valorBase`. Una vegada acaba l'execució del bucle, la variable `valorBase` correspondrà al total de la base imposable que s'afegeix al document a través de la propietat `innerHTML` de l'element amb l'identificador `base-imposable`.

El següent pas és calcular l'IVA. Per fer-ho primer s'obté la referència a l'element que conté la propietat `data-iva` invocant `document.querySelector('[data-iva]')`. Seguidament, s'invoca `elementPercentatgeIVA.getAttribute('data-iva')` per obtenir el valor d'aquest atribut que indica el percentatge que s'ha d'aplicar. Una vegada s'ha calculat, s'afegeix al document assignant-lo a la propietat `innerHTML` de l'element `iva`.

Amb tots dos valors calculats, només queda sumar-los i afegir-los al document com a contingut de l'element `total`.

La funció `netejarLinia` és molt més simple que l'anterior. De la mateixa manera com s'obtenen els valors a la funció `afegirLinia`, s'estableixen els valors per defecte de les entrades de text: una cadena buida per al producte i 0 per a la quantitat i el preu unitari.

```

1 var netejarLinia = function() {
2   document.getElementById('producte').value = '';
3   document.getElementById('quantitat').value = 0;
4   document.getElementById('preu-unitari').value = 0;
5 }
```

Seguidament es troba el codi que s'invoca en detectar-se l'*event click* sobre la primera i l'última columna. En el primer cas només cal destacar que s'obtenen tots els elements amb l'etiqueta `td` del node pare, i després es recorre aquesta llista per construir el text que es mostrarà com a detall:

```

1 var mostrarDetall = function() {
2   var missatge = 'Detall de la factura:\n';
3   var elementsFila = this.parentNode.getElementsByTagName('td');
4   var etiquetes = ['Producte', 'Quantitat', 'Preu unitari', 'Preu total'];
5
6   for (var i = 0; i < elementsFila.length; i++) {
7     missatge += '\t' + etiquetes[i] + ': ' + elementsFila[i].textContent + '\n'
8     ;
9   }
10
11   alert(missatge);
}
```

- ? Fixeu-vos que es fa servir `this` per accedir a les propietats pròpies de l'element clicat. Tot i que totes les cel·les de la primera columna criden aquesta funció, el context en el qual s'executen sempre és l'element concret on s'ha disparat l'*event*.

Per acabar, hi ha el codi de la funció `eliminarFila`. En primer lloc s'elimina la mateixa fila i per fer-ho s'accedeix al node pare del pare. És a dir, es travessa el node ascendentment fins a l'element `tr` (la fila), seguidament, fins a l'element `tbody` que la conté, i a partir d'aquest, s'elimina. Seguidament, s'invoca la funció `recalcularTotal()` per actualitzar els valors del peu de la taula.

```

1 var eliminarFila = function() {
2   this.parentNode.parentNode.removeChild(this.parentNode);
3   recalcularTotal();
4 }
```

A continuació podeu trobar el [codi JavaScript complet del generador de factures](#):

```

1 var inicialitzar = function() {
2   var boto = document.getElementById('afegir');
3   boto.onclick = afegirLinia;
4 }
5
6 // Inicialització de l'aplicació quan es carregui el DOM
7 document.body.onload = inicialitzar;
8
9 var afegirLinia = function() {
10   var nomProducte = document.getElementById('producte').value;
11   var quantitatProducte = document.getElementById('quantitat').value;
12   var preuUnitari = document.getElementById('preu-unitari').value;
13   var totalProducte = quantitatProducte * preuUnitari;
14
15   afegirFilaTaula(nomProducte, quantitatProducte, preuUnitari, totalProducte);
16   recalcularTotal();
17   netejarLinia();
18 }
19
20 var afegirFilaTaula = function(nomProducte, quantitatProducte, preuUnitari,
21   totalProducte) {
22   var cosTaula = document.querySelector('tbody');
23
24   var fila = document.createElement('tr');
25
26   var col1 = document.createElement('td');
27   var col2 = document.createElement('td');
28   var col3 = document.createElement('td');
29   var col4 = document.createElement('td');
30   var col5 = document.createElement('td');
31
32   col1.innerHTML = nomProducte + ' (detall)';
33   col2.innerHTML = quantitatProducte;
34   col3.innerHTML = preuUnitari + '€';
35   col4.innerHTML = totalProducte + '€';
36   col5.innerHTML = '(eliminar)';
37
38   col1.addEventListener('click', mostrarDetall);
39   col5.addEventListener('click', eliminarFila);
40
41   fila.appendChild(col1);
42   fila.appendChild(col2);
43   fila.appendChild(col3);
44   fila.appendChild(col4);
45   fila.appendChild(col5);
46
47   cosTaula.appendChild(fila);
48 }
```

```
49 var recalcularTotal = function() {
50   var files = document.querySelectorAll('tbody tr');
51   var valorBase = 0;
52
53   for (var i=0; i<files.length; i++) {
54     var columnaUltima = files[i].lastElementChild;
55     var columnaPenultima = columnaUltima.previousElementSibling;
56     var valorTotalFila = parseFloat(columnaPenultima.textContent);
57     valorBase += valorTotalFila;
58   }
59
60   var elementBase = document.getElementById('base-imposable');
61   elementBase.innerHTML = valorBase;
62
63   var elementPercentatgeIVA = document.querySelector('[data-iva]');
64   var valorPercentatgeIVA = elementPercentatgeIVA.getAttribute('data-iva');
65   var valorIVA = parseFloat(valorPercentatgeIVA * valorBase);
66
67   var elementIVA = document.getElementById('iva');
68   elementIVA.innerHTML = valorIVA;
69
70   var elementTotal = document.getElementById('total');
71   elementTotal.innerHTML = valorBase + valorIVA;
72 }
73
74 var netejarLinia = function() {
75   document.getElementById('producte').value = '';
76   document.getElementById('quantitat').value = 0;
77   document.getElementById('preu-unitari').value = 0;
78 }
79
80
81 var mostrarDetall = function() {
82   var missatge = 'Detall de la factura:\n';
83   var elementsFila = this.parentNode.getElementsByTagName('td');
84   var etiquetes = ['Producte', 'Quantitat', 'Preu unitari', 'Preu total'];
85
86   for (var i = 0; i < elementsFila.length; i++) {
87     missatge += '\t' + etiquetes[i] + ': ' + elementsFila[i].textContent + '\n'
88     ;
89   }
90
91   alert(missatge);
92 }
93
94 var eliminarFila = function() {
95   this.parentNode.parentNode.removeChild(this.parentNode);
96   recalcularTotal();
97 }
```

Podeu veure aquest exemple en l'enllaç següent: codepen.io/ioc-daw-m06/pen/XjbooE.

2. Programació amb el DOM i la biblioteca jQuery

La biblioteca jQuery és una de les més populars, ja que inclou tota una sèrie de **funcionalitats** que són bàsiques per a qualsevol aplicació web amb un mínim de complexitat; en concret:

- D'una banda, simplifiquen la realització de tasques com gestionar el DOM, els *events* o les peticions asíncrones.
- De l'altra, evita que hagi de fer servir implementacions alternatives (per a navegadors actuals) per característiques que potser no estan definides de forma estricta a l'especificació del W3C.

Tot i que les versions anteriors de jQuery eren compatibles amb navegadors antics, les noves versions només admeten navegadors actuals.

Concretament, pel que fa a la manipulació del DOM, jQuery ofereix una gran quantitat de mètodes que ajuden a gestionar els elements del DOM i permeten crear-ne de nous o modificar els actuals d'una manera més simple. Cal destacar especialment la potència del seu **sistema de selecció**, que va molt més enllà del que permeten les interfícies del DOM.

A més a més, donada la seva popularitat, és fàcil trobar centenars de **connectors lliures**, així com privatis, per ampliar les seves funcionalitats o per afegir fàcilment nous elements, com poden ser galeries d'imatges, carrusels... És a dir, en molts casos podreu recórrer a components completament desenvolupats i testats per utilitzar-los directament o adaptar-los a les vostres necessitats en lloc de començar des de zero.

2.1 Introducció a jQuery

El primer pas a l'hora d'utilitzar jQuery és **carregar la biblioteca**. La forma de fer-ho variarà segons l'entorn en el qual s'hagi d'utilitzar i les vostres preferències quant al seu allotjament.

Cal recordar que no és segur manipular el DOM abans que acabi de carregar, per aquesta raó jQuery inclou els seus propis sistemes per detectar-ho i permetre que el desenvolupador escriui codi, que començarà a executar una vegada finalitzi la càrrega.

El primer que trobareu en començar a treballar amb jQuery és que cal distingir entre la **funció** jQuery (habitualment substituïda pel símbol \$) i els **objectes** jQuery, que són generats o bé per la mateixa funció o per la crida d'un mètode sobre altres objectes jQuery.

Generalment treballareu amb la funció jQuery per obtenir un objecte que contingui els elements seleccionats, i a partir de llavors les crides a tots els mètodes

les fareu sobre l'objecte que s'ha de manipular, per exemple, cridant el mètode `addClass` per afegir una classe nova als elements.

Cal recordar que tot i que l'estructura dels programes que fan servir jQuery poden semblar molt diferents dels programes en JavaScript, es tracta del mateix llenguatge. Segurament la major part de l'estranyesa es deu a la utilització del símbol \$ com a àlies de la funció jQuery i la utilització d'interfícies fluides.

Gairebé tots els mètodes dels objectes jQuery tenen un comportament fluid, és a dir, retornen el mateix objecte sobre el qual s'han invocat. Així doncs us podeu trobar amb estructures semblants a aquesta:

```

1  $('p')
2   .find('span')
3   .addClass('vermell')
4   .addClass('negreta')
5   .css('font-size', '18px')
6   .css('background-color', 'green');
```

Cal destacar que el mètode `find` retorna un nou objecte jQuery que inclouria només els elements de tipus `span` descendents dels paràgrafs seleccionats inicialment i, per tant, els mètodes següents s'invocarien a partir d'aquest nou objecte i no de l'original.

Fixeu-vos també que només l'última línia inclou el punt i coma; és a dir, aquest codi es podria interpretar com si fos una sola línia:

```

1  $('p').find('span').addClass('vermell').addClass('negreta').css('font-size', '18px').css('background-color', 'green');
```

Com es pot apreciar, el primer cas és més fàcil de llegir i d'entendre. Una altra manera d'escriure el mateix codi seria guardant el valor de la selecció inicial (un objecte jQuery en una variable):

```

1  var $paragraf = $('p');
2  var $span = $paragraf.find('span');
3
4  $span.addClass('vermell');
5  $span.addClass('negreta');
6  $span.css('font-size', '18px');
7  $span.css('background-color', 'green');
```

Tot i que el resultat és el mateix en tots tres casos, quan es treballa amb jQuery, per convenció s'acostuma a utilitzar el format del primer exemple, que genera un codi més concís i més clar.

2.1.1 Carregar la biblioteca jQuery

A l'hora d'incloure la biblioteca jQuery a les vostres aplicacions, el primer pas és carregar la biblioteca. Per fer-ho, les dues opcions més habituals són descarregar la biblioteca i afegir-la juntament amb la resta de fitxers JavaScript de la vostra aplicació o fer que els clients la descarreguin a través d'un CDN.

Xarxes de lliurament de continguts (CDN)

Un CDN és una xarxa de servidors localitzats en diferents punts geogràfics però amb els mateixos continguts, de manera que quan es rep una petició del fitxer aquest és enviat des del servidor més proper fins a l'usuari. D'aquesta manera, s'augmenta la velocitat de la resposta. Podeu trobar més informació sobre les xarxes de lliurament de continguts en l'enllaç següent: www.ca.wikipedia.org/wiki/Xarxa_de_lliurament_de_continguts.

La diferència entre fer servir la versió comprimida o sense comprimir és que la primera ocupa menys espai però és inintel·ligible: com que són fitxers de text pla, no es tracta exactament d'una compressió sinó d'una *minització*. Així doncs, a l'hora de desplegar l'aplicació s'ha de fer servir la versió comprimida, mentre que durant el desenvolupament és recomanable utilitzar la versió sense *minitzar* per poder depurar millor el codi.

Per descarregar la biblioteca jQuery heu de visitar el web següent: www.jquery.com/download/. Entre d'altres descàrregues, hi trobareu la versió més actual comprimida i sense comprimir. Només cal descarregar-ne un dels dos, segons les vostres necessitats.

La **minització** (de l'anglès *minification*) consisteix a eliminar tots els espais i salts de línies i substituir els noms de variables i funcions per altres amb el mínim nombre de caràcters possibles; així, `totalProductes = preUnitari * quantitatProducte;` es convertiria en alguna cosa semblant a `a=b*c.`

Una vegada descarregada la biblioteca podeu afegir-la al vostre programa com qualsevol altre fitxer de codi JavaScript. Suposant que esteu treballant amb la versió 3.1 (aquesta és la versió que es farà servir per als exemples) i que l'heu copiat a un directori anomenat js dintre del vostre projecte, el codi seria el següent:

¹ `<script src="js/biblioteques/jquery-3.1.0.js"></script>`

Per evitar el bloqueig de la pàgina es recomana que tots els fitxers amb codi JavaScript, propi o biblioteques, s'afegeixin al final de la pàgina abans de tancar l'element body, en comptes de fer-ho al principi dintre de l'element head.

Les eines d'automatització per treballar amb JavaScript requereixen tenir instal·lat Node.js i s'utilitzen a través del terminal.

Un avantatge de treballar directament amb la biblioteca és que podeu utilitzar eines d'automatització com Gulp o Grunt per concatenar tots els fitxers JavaScript en un de sol i a continuació minitzar-lo abans de fer el desplegament de l'aplicació. D'aquesta manera es redueix el nombre de fitxers i s'optimitza el temps de descàrrega de la pàgina.

D'altra banda, si feu servir un CDN, no cal descarregar el fitxer: el fitxer es descarregarà per a cada client des del servidor.

¹ `<script src="//code.jquery.com/jquery-3.1.0.js"></script>`

Independentment del sistema que feu servir, sempre s'ha de carregar la biblioteca abans d'utilitzar-la, perquè si ho feu en l'ordre contrari, és molt possible que es produueixi un error quan la vulgueu fer servir.

2.1.2 Carregar jQuery a CodePen

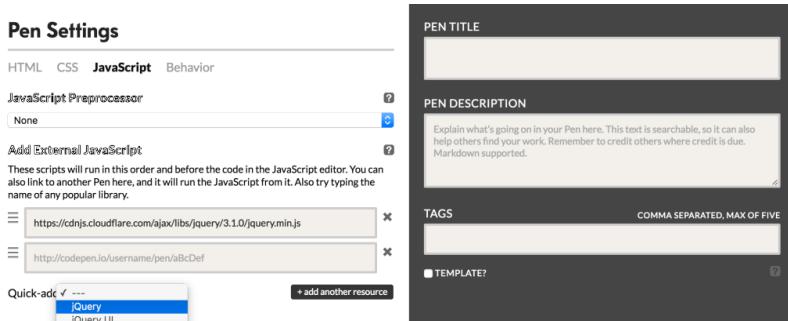
Atès que l'ús de la biblioteca jQuery està tan estès, moltes plataformes d'edició de codi en línia, com CodePen, ofereixen la possibilitat d'incloure'l directament sense haver d'afegir la càrrega manualment.

Fitxers externs a CodePen

A CodePen és possible afegir tant algunes de les biblioteques de JavaScript més conegudes com els vostres propis fitxers externs amb codi JavaScript. A més a més, es poden fer servir preprocessadors com Babel o TypeScript per utilitzar ES6 o TypeScript en lloc de JavaScript.

Per accedir a aquestes opcions en els vostres propis *pens* (nom que rep cada demostració a CodePen) només heu de fer clic al botó *Settings*, a continuació, en el panell de configuració, feu clic a la pestanya *JavaScript* i a la llista desplegable *Quick-add*, que es troba a la part inferior seleccioneu **jQuery**. Podeu veure les opcions de configuració de CodePen a la figura 2.1

FIGURA 2.1. Opcions de configuració de CodePen



Cal destacar que quan s'afegeix la biblioteca d'aquesta manera, no cal afegir el codi per carregar-la perquè es carrega automàticament; per tant, es pot procedir a treballar directament amb la biblioteca. Per aquesta raó, la càrrega de la biblioteca no es mostra en cap dels exemples: es carrega automàticament gràcies a la configuració del *pen*.

2.1.3 Primers passos amb jQuery

A banda d'indicar que s'ha de carregar la biblioteca, us heu d'assegurar que el document (més concretament el DOM) ha carregat completament. En cas contrari, és possible que es produueixin errors. Per aquesta raó la primera línia que trobareu habitualment en el codi que fa servir jQuery és el següent:

```

1 $(document).ready(function() {
2   // Aquí va el nostre codi
3   console.log('Llistos!');
4 });

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/jrbQVZ?editors=0012.

D'aquesta manera us assegureu que és segur començar a treballar amb el document, ja que el DOM estarà completament carregat.

Recordeu que en cas de fer la prova en un fitxer local, heu d'incloure la càrrega de la biblioteca. En primer lloc, haureu d'afegir la càrrega de la biblioteca i, seguidament, el vostre codi d'inicialització. Així doncs, si feu servir el CDN de jQuery, el vostre codi quedaría així:

```
1 <script src="//code.jquery.com/jquery-3.1.0.js"></script>
2
3 <script>
4   $(document).ready(function() {
5     // Aquí va el nostre codi
6     console.log('Llistos!');
7   });
8 </script>
```

L'efecte d'utilitzar el mètode `ready` de jQuery és molt similar al detector de l'`event load` de JavaScript però no és el mateix. **No s'han d'utilitzar mai tots dos sistemes al mateix temps** perquè són incompatibles, en cas de fer servir jQuery, sempre s'ha d'utilitzar el mètode `ready` que facilita la biblioteca.

Fixeu-vos en la primera línia: `$(document).ready()`. Aquesta línia fa dues accions diferenciades: d'una banda, es crida la funció jQuery fent servir la drecera `$`, i de l'altra, sobre l'objecte retornat s'invoca el mètode `ready` passant com a argument una funció, que serà executada una vegada es carregui el DOM.

Atès que a JavaScript les funcions són objectes de primera classe, es poden referenciar fent servir variables i passar-les com a arguments a altres funcions.

És a dir, es podria haver escrit `jQuery(document).ready()` i hauria estat el mateix. Recordeu que el símbol `$` és utilitzable com a nom de variable, tot i que no és recomanable fer-lo servir, perquè algunes biblioteques (com aquesta) el fan servir amb fins específics.

D'altra banda, per convenció, sí que s'acostuma a prefixar el nom de les variables que emmagatzemem objectes de tipus jQuery amb el símbol `$`. Així doncs, si una variable conté un objecte jQuery que conté, al seu torn, una llista de matrícules, és correcte que el seu nom sigui `$matricules`.

La funció que es passa com a argument –i serà cridada en carregar el DOM–, no cal que estigui definida com a funció anònima, es podria haver passat una variable que referencies una funció:

```
1 var inicialitzacio = function() {
2   console.log('Llistos!');
3 }
4
5 $(document).ready(inicialitzacio);
```

Podeu veure aquest exemple en el següent enllaç: www.codepen.io/ioc-daw-m06/pen/zKvrdx?editors=0012.

Fixeu-vos en un detall important: en cas que la funció estigui referenciada per una variable en lloc d'estar declarada directament, s'ha d'assignar la funció a aquesta variable abans d'invocar el mètode `ready`. Si no es fa així, el valor de la variable es passarà com a `null` (a causa del *hoisting*: internament, la declaració de les variables es mou al principi del bloc però sense el valor assignat).

En canvi, si es passa el nom de la funció, l'ordre no importa perquè s'inclou el cos de la funció quan es produeix el *hoisting*:

```

1  $(document).ready(initialitzacio);
2
3  function initialitzacio() {
4      console.log('Llistos!');
5  }

```

Podeu veure aquest exemple en el següent enllaç: www.codepen.io/ioc-dawm06/pen/BLoGdx?editors=0012.

Cal destacar que només cal incloure dins de la funció d'inicialització les crides a les funcions; la declaració, en canvi, es pot realitzar en qualsevol altre lloc (per exemple, en altres fitxers) sempre que us assegureu que l'ordre de declaració i assignació és el correcte.

Per exemple, es podrien invocar múltiples funcions dintre de la funció `initialitzacio` mentre que es tingui la garantia que aquestes han estat invocades una vegada s'ha completat la càrrega del DOM (com succeeix quan s'invoca `initialitzacio`):

```

1  var inicialitzacio = function() {
2      mostrarMissatge('DOM carregat');
3      iniciarAplicacio();
4  }
5
6  var mostrarMissatge = function(missatge) {
7      console.log(missatge);
8  }
9
10 var iniciarAplicacio = function() {
11     console.log("Iniciant l'aplicació...");
12 }
13
14 $(document).ready(initialitzacio);

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/PGPxQR?editors=0012.

2.1.4 L'objecte 'jQuery'

Quan es parla de jQuery cal diferenciar entre la **biblioteca**, la **funció** i els **objectes** de tipus jQuery.

En el primer cas, es fa referència a la biblioteca en conjunt, per exemple: 'carregar jQuery' significa que s'ha de carregar la biblioteca, ja sigui com un fitxer de codi JavaScript o a través d'un CDN.

Recordeu que a JavaScript les funcions també són objectes i, per consegüent, poden contenir propietats i mètodes.

En el segon cas, fa referència a la funció `jQuery`, que pot invocar-se com a `jQuery()` o `$()`. Aquesta funció permet, d'una banda, crear objectes de tipus `jQuery`, i de l'altra, accedir a una sèrie de mètodes genèrics, com per exemple `each`, que permet iterar sobre un *array*.

En el tercer cas, un objecte de tipus `jQuery` fa referència a un objecte retornat per la funció, sigui generat o com a resultat d'una cerca i que hi pot tenir associats un o més elements. En qualsevol cas es tracta sempre com una col·lecció d'elements.

Una peculiaritat dels objectes de tipus `jQuery` és que quan s'invoquen els seus mètodes afecten, generalment, tota la col·lecció. Per exemple, si un objecte conté una col·lecció amb tots els enllaços de la pàgina i s'hi afegeix la detecció de l'*event* `click`, aquesta detecció afectarà tots els elements d'una tacada, no cal recórrer la col·lecció i afegir-lo d'un en un. Fixeu-vos en l'exemple següent:

```

1 <h1>Primer títol</h1>
2 <p>Primer paràgraf</p>
3 <h1>Segon títol</h1>
4 <p>Segon paràgraf</p>
5 <h1>Tercer títol</h1>
6 <p>Tercer paràgraf</p>
7
8 <script>
9   var inicialitzacio = function() {
10     var $paragrafs = $('p');
11     $paragrafs.hide();
12   }
13
14   $(document).ready(inicialitzacio);
15 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/amvxrY?editors=1010.

Com es pot apreciar, dins de la funció `inicialitzacio` s'invoca la funció `jQuery` passant com a paràmetre `p`; això indica que s'ha de fer una cerca de tots els elements de tipus `p`. Així doncs, el retorn de la funció és un nou objecte `jQuery` que conté tots els elements de tipus `p`.

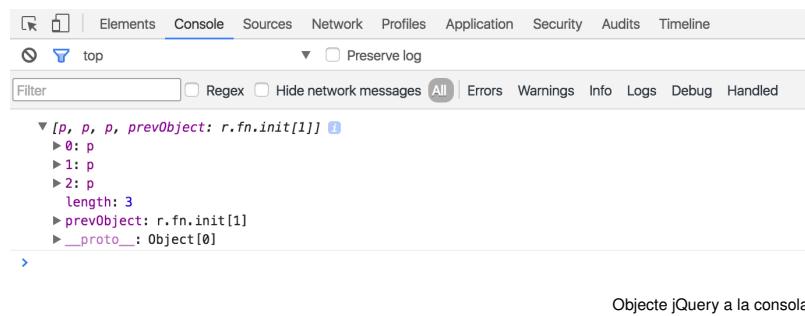
El mètode `hide` dels objectes `jQuery` amaguen tots els objectes continguts, mentre que el mètode `show` els mostra.

Seguidament, es crida al mètode `hide` d'aquest objecte per amagar tots els paràgrafs. Fixeu-vos que no cal iterar sobre els elements de la col·lecció, simplement cridant el mètode ja s'aplica el canvi a tots ells.

Si proveu de mostrar l'objecte `jQuery` per la consola, veureu que té algunes característiques similars als *arrays*, ja que es tracta d'una col·lecció d'elements. Afegiu el codi següent al codi anterior, dins de la funció `inicializacio`:

```
1 console.log($paragrafs);
```

Podeu veure aquest exemple en l'enllaç següent: [codepen.io/ioc-dawm06/pen/rrxvgj?editors=1010](http://www.codepen.io/ioc-dawm06/pen/rrxvgj?editors=1010) i el resultat de la consola a la figura 2.2.

FIGURA 2.2

Per visualitzar el resultat s'ha de fer servir la consola de les eines de desenvolupador del navegador, ja que la consola de CodePen no mostra correctament la informació.

Fixeu-vos que hi ha els tres elements seleccionats amb el seu índex corresponent: '0', '1', '2' i, seguidament, el valor de la propietat `length`. Si desplegueu qualsevol dels objectes niats, veureu tota l'estrucció interna de cadascun d'aquests elements.

Tot i que aquestes propietats i mètodes es gestionen a través de la interfície proporcionada per la biblioteca, és interessant saber com es pot consultar aquesta informació directament al vostre navegador.

En alguns casos necessitareu **accedir directament a algun dels elements seleccionats per l'objecte jQuery**. Es poden obtenir tant tots els elements com un element en particular. En tots dos casos s'accedeix a través del mètode `get`:

- **Si es passa un nombre com a argument**, es **retornarà l'element en aquesta posició**.
- **Si no es passa cap argument**, es **retornarà un array amb tots els elements**.

Alternativament, es pot accedir directament a l'element com si es tractés d'un **array** (per exemple, `$paragrafs[1]`). D'altra banda, **no hi ha cap garantia que l'accés directe a l'índex d'un objecte jQuery continuï funcionant en versions futures**, per consegüent, **es desaconsella fer-lo servir**.

Vegeu a continuació un **exemple** que inclou els tres mètodes d'accés i els mostra per la consola:

```

1  <p>Primer paràgraf</p>
2  <p>Segon paràgraf</p>
3  <p>Tercer paràgraf</p>
4
5  <script>
6      var inicialitzacio = function() {
7          var $paragrafs = $('p');
8          console.log("Element (primer paràgraf):", $paragrafs.get(0));
9          console.log("Element (últim paràgraf):", $paragrafs[2]);
10         console.log("Col·lecció d'elements (tots els paràgrafs):", $paragrafs.get());
11     }
12
13     $(document).ready(inicialitzacio);

```

14 </script>

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/gwAOVv?editors=1010.

Malauradament alguns dels objectes mostrats són massa complexos per a la consola de CodePen, així que haureu de consultar el resultat a la consola de les eines de desenvolupador del vostre navegador.

Com es pot comprovar, tant `$paragrafs.get(0)` com `$paragrafs[2]` retornen un element del DOM, mentre que `paragrafs.get()` retorna un *array* amb tots els elements. Cal destacar que el retorn del mètode `get` (o l'accés directe) són elements del DOM, no objectes jQuery.

Així doncs, **no es poden invocar mètodes de jQuery sobre aquests elements. Per poder manipular-los individualment utilitzant la biblioteca s'hauran de convertir en objectes jQuery, passant-los com argument a la funció jQuery:**

```

1 <style>
2   .vermell {
3     color: red;
4   }
5 </style>
6
7 <p>Primer paràgraf</p>
8 <p>Segon paràgraf</p>
9 <p>Tercer paràgraf</p>
10
11 <script>
12   var inicialitzacio = function() {
13     var $paragrafs = $('p');
14     var $paragraf2 = $($paragrafs.get(1));
15     $paragraf2.addClass('vermell');
16   }
17
18   $(document).ready(inicialitzacio);
19 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/NRNPpj.

Fixeu-vos que primer s'han cercat tots els paràgrafs i, seguidament, s'ha tornat a cridar la funció jQuery passant com a argument el segon element dels seleccionats. El resultat és un nou objecte jQuery que conté només aquest element i, per tant, es pot manipular.

2.2 Selectors

Per generar un objecte jQuery que contingui una llista d'elements ja existents, s'ha d'invocar la funció jQuery passant com a argument un *selector*. Aquest selector pot ser **de diferents tipus**, no està limitat als selectors de CSS com en el cas dels mètodes que formen part de l'especificació del DOM.

Tot i que la funcionalitat és similar a la dels mètodes `querySelector` i `querySelectorAll` de les interfícies del DOM Document i Elements, en comptes d'un element o una col·lecció d'elements, aquest mètode sempre retorna un objecte jQuery. Aquest objecte conté tots els elements que coincideixen amb la selecció. Fins i tot en cas que no hi hagi cap element coincident, es retornarà un objecte però amb longitud 0 (propietat `length`).

Lista de selectors

Podeu trobar una llista completa dels selectors en l'enllaç següent:
www.goo.gl/jn30Ta.

Els tipus de **selectors** disponibles es poden dividir en **dos grans grups**:

- **Selectors CSS:** aquests selectors es corresponen exactament amb els selectors CSS, inclosos els selectors afegits a CSS3.
- **Selectors propis de jQuery:** aquests són selectors específics de jQuery, que funcionen com dreceres per a seleccions molt comunes, com per exemple `:image` –per seleccionar totes les imatges– o `:input` –per seleccionar tots els elements d'entrada de dades (`input`, `textarea`, `select` i `button`)

Cal destacar que mentre que la funció jQuery fa la cerca a tot el document, hi ha un altre mètode que forma part de tots els objectes jQuery i permet cercar, només, entre els elements descendents: és el mètode `find`.

2.2.1 Selectors CSS

Podeu trobar més informació sobre els selectors CSS a l'àpartat "Programació amb el DOM (Document Object Model)" d'aquesta unitat.

El mètode `addClass` de la biblioteca jQuery afegeix la classe passada com a argument als elements seleccionats.

Aquest tipus de selectors són els més utilitzats perquè ofereixen les mateixes possibilitats que CSS i, per tant, la majoria de desenvolupadors web les coneixen.

Per demostrar el funcionament d'aquests selectors s'afegirà una classe CSS que pintarà el fons de l'element de color vermell. Vegeu-ne un primer exemple, en el qual se seleccionen tots els elements amb el *tag p*:

```

1 <style>
2   .vermell {
3     background-color: red
4   }
5 </style>
6
7 <h1>Primer títol</h1>
8 <p>Primer paràgraf</p>
9 <h1>Segon títol</h1>
10 <p>Segon paràgraf</p>
11 <h1>Tercer títol</h1>
12 <p>Tercer paràgraf</p>
13
14 <script>
15   var inicialitzacio = function() {
16     var $paragrafs = $('p');
17     $paragrafs.addClass('vermell');
18   }
19
20   $(document).ready(inicialitzacio);
21 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/LRGVYg.

Com es pot apreciar, el fons de tots els paràgrafs es pinta de color vermell, perquè s'ha afegit la classe `vermell` a cadascun. A més a més, podeu fer servir les eines de desenvolupador per inspeccionar el codi HTML de la pàgina i comprovar que, efectivament, s'ha afegit la classe.

Per descomptat, la selecció no es limita als elements, és possible combinar els selectors per filtrar per classes o identificadors:

```
1 <style>
2   .vermell {
3     background-color: red
4   }
5
6   .verd {
7     background-color: green
8   }
9 </style>
10
11 <div>
12   <h1>Primer títol</h1>
13   <p>Primer paràgraf</p>
14   <h1 class="central">Segon títol</h1>
15   <p class="central">Segon paràgraf</p>
16   <h1>Tercer títol</h1>
17   <p id="tercer">Tercer paràgraf</p>
18 </div>
19 <p class="central">Aquest no canvia de color</p>
20
21 <script>
22 var inicialitzacio = function() {
23   var $central = $('div .central');
24   $central.addClass('vermell');
25
26   var $tercer = $('#tercer');
27   $tercer.addClass('verd');
28 }
29
30 $(document).ready(inicialitzacio);
31 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/ORrVNy.

Fixeu-vos que el selector `div .central` ha seleccionat tots els elements amb la classe `central` descendents d'un element de tipus `div` i, per consegüent, l'últim paràgraf no s'ha inclòs. D'altra banda, com era d'esperar, el fons de l'element amb identificador `tercer` s'ha pintat de color verd.

De la mateixa manera, es poden utilitzar els **selectors d'atribut**; això permet seleccionar elements segons quines siguin les característiques dels seus atributs:

- Si un atribut es troba present: `[atribut]`.
- Si l'atribut conté un valor concret: `[atribut="valor"]`.
- Si el seu valor comença pel valor: `[atribut^="valor"]`.
- Si el seu valor acaba pel valor: `[atribut$="valor"]`.
- Si l'atribut conté el valor: `[atribut*= "valor"]`.

A continuació podeu comprovar el funcionament de dos d'aquests selectors. Primerament se cerquen tots els enllaços a la Wikipedia del document via el seu atribut href, i seguidament se cerquen només els de llengua anglesa (inclouen el subdomini en):

```

1 <style>
2   .vermell {
3     background-color: red
4   }
5
6   .resaltat {
7     font-weight: bold;
8   }
9 </style>
10
11 <div>
12   <h1>Primer títol</h1>
13   <p>Primer paràgraf <a href="https://ca.wikipedia.org/wiki/Aut%C3%B2mat_finit">Enllaç a Automàt finit</a></p>
14   <h1 class="central">Segon títol</h1>
15   <p class="central">Segon paràgraf <a href="https://en.wikipedia.org/wiki/Finite-state_machine">Enllaç a Finite-state machines</a></p>
16   <h1>Tercer títol</h1>
17   <p id="tercer">Tercer paràgraf</p>
18 </div>
19 <p class="central">Aquest no canvia de color <a href="#">Un altre enllaç</a></p>
20
21 <script>
22   var inicialitzacio = function() {
23     var $wikipedia = $('[href*="wikipedia"]');
24     $wikipedia.addClass('vermell');
25
26     var $angles = $('[href^="https://en.wikipedia.org"]');
27     $angles.addClass('resaltat');
28   }
29
30   $(document).ready(inicialitzacio);
31 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/JRGdyx.

Com es pot apreciar, el primer selector (`[href*='wikipedia']`), selecciona només els enllaços que contenen la paraula wikipedia als quals s'afegeix la classe `vermell`, mentre que el segon (`[href ^="https://en.wikipedia.org"]`) selecciona només els que pertanyen al subdomini lligat a la llengua anglesa de la Wikipedia.

D'altra banda, l'enllaç Un altre enllaç no forma part de cap dels dos objectes jQuery, ja que encara que conté l'atribut href el seu valor no compleix les condicions de cap dels dos selectors.

Cal destacar que l'element que enllaça amb la pàgina anglesa forma part de les col·leccions d'elements de tots dos objectes (referenciat per `$wikipedia` i `$angles`), de manera que qualsevol mètode cridat sobre un o altre l'affectarà.

Per exemple, si afegiu el codi següent al final de la funció d'inicialització, s'amagaran tots dos enllaços:

```
1 $wikipedia.hide();
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/ozbXQv.

En canvi, si es crida el mateix mètode sobre \$angles, s'amagarà només l'enllaç a la pàgina anglesa:

```
1 $angles.hide();
```

Podeu veure aquest exemple en el següent enllaç: www.codepen.io/ioc-dawm06/pen/VKZLqY.

Així doncs, es pot concloure que l'element que enllaça amb la pàgina anglesa forma part de la col·lecció d'elements seleccionats per tots dos objectes jQuery.

També es poden fer servir pseudoclases com `first-child` o `last-child`, com es pot comprovar en l'exemple següent:

```
1 <style>
2   .vermell {
3     background-color: red;
4   }
5
6   .verd {
7     background-color: green;
8   }
9 </style>
10
11 <ul>
12   <li>Primer element</li>
13   <li>Segon element</li>
14   <li>
15     <ul>
16       <li>Primer subelement</li>
17       <li>Segon subelement</li>
18       <li>Últim subelement</li>
19     </ul>
20   </li>
21   <li>Últim element</li>
22 </ul>
23
24 <script>
25   var inicialitzacio = function() {
26     var $primers = $('li:first-child');
27     $primers.addClass('vermell');
28
29     var $ultims = $('li:last-child');
30     $ultims.addClass('verd');
31   }
32
33   $(document).ready(inicialitzacio);
34 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/GjoJZj.

Cal destacar que, tot i que no s'han mostrat exemples complexos, jQuery admet tot tipus de combinacions, incloent-hi la combinació amb els seus propis selectors.

2.2.2 Selectors propis de jQuery: seleccions especials

Llista de selectors de jQuery

Podeu trobar una llista completa dels selectors propis de jQuery en l'enllaç següent: www.goo.gl/zzvN1g.

L'ús de selectors a jQuery no està limitat als selectors de CSS, ja que disposa d'un joc propi de selectors que simplifiquen encara més la selecció dels elements.

Entre les dreceres genèriques que s'apliquen a tots els elements, la biblioteca disposa de les següents:

- **:first i :last**: selecciona només el primer element o només l'últim, respectivament, de la selecció. Cal distingir entre la funcionalitat d'aquestes dreceres i la dels selectors de CSS `first-child` i `last-child`, ja que en aquest cas es tracta del primer o últim element que formaria part de la selecció, i no pas dels descendents concrets d'algun altre.
- **:header**: selecciona totes les capçaleres, per exemple `h1`, `h2`, etc.
- **:even i :odd**: selecciona els elements parells i els senars, respectivament.

Vegeu a continuació un exemple amb els selectors `first` i `last` que us ajudarà a entendre la diferència amb els selectors de CSS `first-child` i `last-child`:

```

1 <style>
2   .vermell {
3     color: red;
4   }
5
6   .verd {
7     color: green;
8   }
9
10  .negreta {
11    font-weight: bold;
12  }
13
14  .cursiva {
15    font-style: italic;
16  }
17 </style>
18
19 <ul>
20   <li>Primer element</li>
21   <li>Segon element</li>
22   <li>
23     <ul>
24       <li>Primer sub-element</li>
25       <li>Segon sub-element</li>
26       <li>Últim sub-element</li>
27     </ul>
28   </li>
29   <li>Últim element</li>
30 </ul>
31
32 <script>
33   var inicialitzacio = function() {
34     var $primers = $('li:first-child');
35     $primers.addClass('vermell');
36
37     var $ultims = $('li:last-child');
38     $ultims.addClass('verd');
39

```

```

40 var $primer = $('li:first');
41 $primer.addClass('negreta');
42
43 var $ultim = $('li:last');
44 $ultim.addClass('cursiva');
45 }
46
47 $(document).ready(initialitzacio);
48 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/PGkjJO.

Com es pot apreciar, mentre que els selectors CSS han seleccionat els elements que són primer i últim element respecte al seu element pare, els selectores propis només han seleccionat el primer i l'últim element de la selecció, que en aquest cas correspon a tots els elements amb l'etiqueta li.

A continuació podeu veure un **exemple d'utilització del selector header**. Fixeu-vos que és possible tant seleccionar totes les capçaleres com combinar-lo amb un altre selector CSS per obtenir una selecció més concreta:

```

1 <style>
2   .vermell {
3     color: red;
4   }
5
6   .cursiva {
7     font-style: italic;
8   }
9 </style>
10
11 <h1>Títol 1</h1>
12 <p>Paràgraf 1</p>
13 <div>
14   <h2>Títol 2</h2>
15   <p>Paràgraf 2</p>
16   <p>Paràgraf 3</p>
17   <h3>Títol 3</h3>
18   <p>Paràgraf 4</p>
19 </div>
20
21 <script>
22   var initialitzacio = function() {
23     var $titols = $('#:header');
24     $titols.addClass('vermell');
25
26     var $titolsEnContenidor = $('div :header');
27     $titolsEnContenidor.addClass('cursiva')
28   }
29
30   $(document).ready(initialitzacio);
31 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/kkPwvd.

Els selectors odd i even serveixen per afegir efectes o comportaments diferents d'elements alterns, per exemple les files o columnes d'una taula. En aquest exemple s'han fet servir diferents colors per mostrar-ne el comportament, però cal tenir en compte que si només necessiteu canviar el format, el més recomanable

és fer-ho directament a través de fulls d'estil i no pas modificant el document dinàmicament com s'ha fet aquí:

```

1  <style>
2      table {
3          border-collapse: collapse;
4      }
5
6      td {
7          text-align: center;
8          padding: 2px;
9      }
10
11     th {
12         padding: 5px;
13     }
14     .blau {
15         background-color: #1b95e0;
16     }
17
18     .gris {
19         background-color: gray;
20     }
21
22     .negreta {
23         font-weight: bold;
24     }
25 </style>
26
27 <table>
28     <tr>
29         <th>Columna 1</th>
30         <th>Columna 2</th>
31         <th>Columna 3</th>
32         <th>Columna 4</th>
33     </tr>
34     <tr>
35         <td>Fila 1</td>
36         <td>Fila 1</td>
37         <td>Fila 1</td>
38         <td>Fila 1</td>
39     </tr>
40     <tr>
41         <td>Fila 2</td>
42         <td>Fila 2</td>
43         <td>Fila 2</td>
44         <td>Fila 2</td>
45     </tr>
46     <tr>
47         <td>Fila 3</td>
48         <td>Fila 3</td>
49         <td>Fila 3</td>
50         <td>Fila 3</td>
51     </tr>
52     <tr>
53         <td>Fila 4</td>
54         <td>Fila 4</td>
55         <td>Fila 4</td>
56         <td>Fila 4</td>
57     </tr>
58 </table>
59 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/wzrZXX.

Fixeu-vos que tot i que odd correspon als elements senars, el primer element marcat com a senar no és la primera fila ni la primera columna. Això es deu

al fet que els elements comencen a comptar-se a partir de 0 i, per tant, la primera fila i columna corresponen a even (parell).

2.2.3 Selectors propis de jQuery: formularis

Un altre tipus de selectors que també ofereix la biblioteca faciliten treballar amb formularis:

- **:input**: selecciona tots els elements de tipus input, textarea, select i button.
- **:text**: selecciona tots els elements input de tipus text.
- **:checkbox**: selecciona tots els elements amb tipus checkbox.
- **:radio**: selecciona tots els elements de tipus radio.
- **:button**: selecciona tots els botons i elements amb tipus button.

El mètode `val` dels objectes jQuery permet consultar o establir la propietat `value` d'un element.

```

1 <style>
2   .tabular {
3     margin-left: 25px;
4   }
5
6   div {
7     margin: 0 auto;
8     width: 300px;
9     border: 1px solid black;
10  }
11
12  .caselles {
13    float: right;
14  }
15 </style>
16
17 <div>
18   <h1>Formulari</h1>
19   <input type="text" name="nom" />Nom<br> <input type="text" name="cognom" />
20     Cognom<br>
21   <h2>Descripció</h2>
22   <textarea name="descripcio"></textarea><br> <input type="checkbox" name="
23     acceptar" />Acceptar condicions<br>
24   <input type="radio" name="color" value="blau" checked>blau<br>
25   <input type="radio" name="color" value="vermell">vermell
26 </div>
27
28 <script>
29   var inicialitzacio = function() {
30     var $texts = $('#:text');
31     $texts.val(12345678);
32
33     var $input = $('#:input');
34     $input.addClass('tabular');
35
36     var $radio = $('#:radio');
37     $radio.addClass('caselles');
38
39     var $checkbox = $('#:checkbox');
40     $checkbox.addClass('caselles');
41   }

```

```

40
41     $(document).ready(initialitzacio);
42 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/ZpQJZW.

En aquest exemple es modifica –via codi– el valor dels camps de text (invocant el mètode `val`), s'afegeix un marge a l'esquerra de tots els elements englobats pel selector `:input` (`input` i `textarea`) i, finalment, es converteixen les caselles i els botons de selecció en flotants a la dreta afegint la classe `caselles`.

Fixeu-vos que per afegir la classe `caselles` tant a les caselles com als botons de ràdio s'ha hagut de fer individualment. Afortunadament **jQuery ofereix el mètode `add`, que permet concatenar seleccions**. Canvieu el codi JavaScript de l'exemple anterior pel següent per comprovar-ho:

```

1 var initialitzacio = function() {
2     var $texts = $(':text');
3     $texts.val(12345678);
4
5     var $input = $(':input');
6     $input.addClass('tabular');
7
8     var $caselles = $(':radio').add(':checkbox');
9     $caselles.addClass('caselles');
10 }
11
12 $(document).ready(initialitzacio);

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/EZZqqp.

Com es pot apreciar, l'objecte jQuery referenciat per `$caselles` ara conté tant la selecció de `:radio` com la selecció de `:checkbox` i, consegüentment, quan s'invoca el mètode `addClass`, aquest afecta tots dos tipus d'elements.

Però es pot simplificar encara més. Els objectes jQuery tenen una interfície fluida, és a dir, molts dels seus mètodes retornen sempre el mateix objecte jQuery, de manera que es poden invocar tots els mètodes un darrere de l'altre. A més a més, no cal fer cap operació addicional amb l'objecte, ni tan sols cal emmagatzemar la referència. Vegeu una implementació més concisa del mateix exemple; substituïu el codi JavaScript pel següent:

```

1 var initialitzacio = function() {
2     $(':text').val(12345678);
3     $(':input').addClass('tabular');
4
5     $(':radio')
6         .add(':checkbox')
7         .addClass('caselles');
8 }
9
10 $(document).ready(initialitzacio);

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/EgPwPE.

Fixeu-vos que no s'ha fet servir cap variable, s'han invocat els mètodes necessaris directament. En el cas del selector per a les caselles, s'ha posat la invocació a cada mètode en una línia diferent, en aquest cas només hi ha dues invocacions i es podria haver escrit tot en la mateixa línia, però en casos més complexos la visibilitat millora molt si es fa d'aquesta manera.

Cal destacar els dos elements següents –especialment interessants–, ja que cap dels dos es pot substituir per selectors CSS, ni requereixen que sigui descendant directe:

- **:has(element)**: selecciona els elements que continguin com a descendents **element**.
- **:contains(text)**: selecciona els elements que continguin el **text**.

Vegeu com funciona en l'exemple següent:

```

1 <style>
2   .vermell {
3     color: red;
4   }
5
6   .negreta {
7     font-weight: bold;
8   }
9
10  span {
11    font-style: italic;
12  }
13 </style>
14
15 <div>Que deu ésser feta al cavaller. Volgué anar havent dolor e contricció de.
16   La Comtessa e als servidors la. Ésser atesa sens mitjà de virtuts Los
17   cavallers. Actes frescs de nostres dies.</div>
18 <p> D'or ab les armes sues e de la Comtessa. <a href="#">Lo virtuós Comte</a>
19   en edat avançada de cincuenta-cinc. Als servidors la sua partida En la fè
20   rtil. Ho pres ab molta impaciència.</p>
21 <p> Romans: d'Escipiò d'Anibal de Pompeu d'Octovià. Longitud de molts dies E
22   com entre los altres insignes.</p>
23 <p> Comte en edat avançada de cincuenta-cinc anys. Fama d'aquell no deu
24   preterir per longitud de. Havia rei o fill de rei.</p>
25 <p> Experiència mostra la debilitat de la nostra memòria sotsmetent fàcilment.
26   Cavaller pare de cavalleria lo comte Guillem. De <span><a href="#">Marc
27   Antoni</a></span> e de molts altres.</p>
28 <script>
29   var inicialitzacio = function() {
30     $('p:has(a)').addClass('vermell');
31     $('p:contains(Comte)').addClass('negreta');
32   }
33
34   $(document).ready(inicialitzacio);
35 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/zKAEwa.

Primerament, se seleccionen només els elements de tipus paràgraf (p) que continguin un element de tipus a i, seguidament, s'afegeix la classe vermell, de manera que el text d'aquests elements passa a ser vermel·l. Fixeu-vos que en l'últim

paràgraf l'enllaç no és un descendant directe però, tal com s'espera, se selecciona correctament.

A continuació, se cerquen tots els paràgrafs que continguin la paraula Comte. Com que s'ha indicat expressament que només s'ha de cercar als paràgrafs, la primera línia no se'n veu afectada, ja que es tracta d'un element de tipus div. Com en el cas anterior, no és necessari que sigui descendant directe, i per això s'aplica correctament a la quarta línia.

En tots dos casos es pot passar el paràmetre amb cometes o sense; així doncs, es podrien substituir pel codi següent:

```
1 $(‘p:has(“a”)’).addClass(‘vermell’);
2   $(‘p:contains(“Comte”)’).addClass(‘negreta’);
```

S'ha de tenir en compte que aquests selectors no són tan eficients com els selectors CSS. Tot i així, en la majoria dels casos, aquesta pèrdua d'eficiència és inapreciable.

El mètode 'filter'

Podeu trobar més informació sobre el mètode `filter` en l'enllaç següent: www.api.jquery.com/filter/.

En els casos en què l'eficiència sigui crítica es recomana descompondre la selecció en dues parts. Primerament, s'invoca la funció jQuery amb el selector CSS (que està optimitzat) i sobre l'objecte retornat s'invoca el mètode `filter`, passant com a argument els selectors propis, que actuaran com a filtre. Podeu comprovar-ho canviant el codi JavaScript de l'exemple anterior pel següent:

```
1 var inicialitzacio = function() {
2   $('p').filter(':has("a")').addClass('vermell');
3   $('p').filter(':contains("Comte")').addClass('negreta');
4 }
5
6 $(document).ready(inicialitzacio);
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/AMDLRX.

Com es pot apreciar, el comportament és el mateix però internament el càlcul és més eficient perquè en lloc de fer la selecció a partir de tots els elements de la pàgina, es realitza sobre un petit subconjunt, el dels paràgrafs.

2.2.4 Cercar entre els elements seleccionats: 'filter' i 'find'

Quan s'utilitza la funció jQuery la selecció es fa a partir de l'arrel del document; en moltes situacions el que interessa és obtenir una selecció a partir d'un objecte jQuery ja existent. Per portar a terme aquesta acció els objectes jQuery disposen de dos mètodes: `filter` i `find`.

La **diferència entre els mètodes filter i find** és que **el primer aplica el selector només entre els elements que formen part de la selecció actual;** mentre que **el segon aplica el selector als elements descendents.**

Tots dos mètodes retornen un nou objecte jQuery amb el resultat de la selecció.

Vegeu una demostració d'aquesta diferència en l'exemple següent:

```

1 <style>
2   .filtrat {
3     color: red;
4   }
5
6   .cercat {
7     font-weight: bold;
8   }
9 </style>
10
11 <ul class="principal">
12   <li>Element 1</li>
13   <ul>Subllista
14     <li>Element 1.1</li>
15     <li>Element 1.2</li>
16   </ul>
17
18   <li>Element 2</li>
19 </ul>
20 <ul class="principal">
21   <li>Element 1</li>
22   <ul>Subllista
23     <li>Element 1.1</li>
24     <li>Element 1.2</li>
25   </ul>
26
27   <li>Element 3</li>
28 </ul>
29
30 <script>
31   var inicialitzacio = function() {
32     var $llista = $('ul');
33
34     $filtrat = $llista.filter('li');
35     $filtrat.addClass('filtrat');
36
37     $cercat = $llista.find('li');
38     $cercat.addClass('cercat');
39
40     console.log('Elements a la llista:', $llista.length);
41     console.log('Elements \'li\' filtrats:', $filtrat.length);
42     console.log('Elements \'li\' cercats:', $cercat.length);
43   }
44
45   $(document).ready(inicialitzacio);
46 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/KgVRBd?editors=1111.

Fixeu-vos en els elements seleccionats per a cadascun dels objectes jQuery:

- **\$llista:** 4 elements seleccionats, corresponents als 4 elements ul del document.
- **\$filtrat:** 0 elements seleccionats. Encara que pot sorprendre, cal tenir en compte que a \$llista no es troba cap element de tipus li; així doncs, el resultat és correcte.
- **\$cercat:** 8 elements seleccionats. En aquest cas, se cerca entre cadascun dels elements descendents d'ul i selecciona els elements de tipus li que contenen.

Cal destacar que aquests mètodes són disponibles a tots els objectes jQuery i, per consegüent, no és necessari que els elements es trobin afegits al DOM. És a dir, es poden crear nous elements lligats a objectes jQuery i fer operacions de cerca sobre ells:

```

1 var inicialitzacio = function() {
2   $nousElements = $('

# Títol 1</h1><p>Paràgraf 1</p><p>Paràgraf 2</p><h2>Tí 3 tol 2</h2><p>Paràgraf 3</p>'); 4 5 $titols = $nousElements.filter(':header'); 6 $paragrafs = $nousElements.filter('p'); 7 8 console.log('Nombre de títols:', $titols.length); 9 console.log('Nombre de paràgrafs:', $paragrafs.length); 10 11 $(document).ready(inicialitzacio);


```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/qabYQx?editors=0011.

Com podeu veure en aquest exemple, tot i que els nodes generats per la funció jQuery només es troben a la memòria, és possible invocar el mètode `filter` per cercar les capçaleres i els paràgrafs.

Quan a la funció jQuery es passa una cadena de codi HTML com a argument, es crea un nou objecte que conté com a selecció els elements corresponents a aquest codi.

2.3 Crear i manipular elements

Un dels punts forts de jQuery és la facilitat amb la qual es poden crear nous elements i modificar-los. De la mateixa manera que les interfícies del DOM permeten crear branques i manipular-les abans d'afegir-les al document, jQuery permet treballar amb aquests objectes mentre es troben a la memòria.

D'aquesta manera, és possible generar un objecte jQuery a partir d'un fragment de codi HTML (per exemple, enviat des d'un servidor), fer modificacions als seus atributs o classes, clonar-lo, afegir-lo a un altre objecte jQuery i, finalment, afegir aquesta nova branca a l'arbre que forma el document.

Cal destacar que, al contrari de quan es treballa directament amb les interfícies del DOM, jQuery permet afegir nous estils i atributs d'una forma molt intuitiva a través dels seus propis mètodes.

2.3.1 Crear nous elements

La creació de nous elements amb jQuery és molt simple, només cal **passar com a paràmetre de la funció la cadena de codi HTML i la biblioteca retornarà un nou objecte jQuery que contindrà com a selecció aquests elements generats correctament**.

El mètode `append` permet afegir els elements continguts en un objecte jQuery a un altre objecte jQuery.

Per exemple, per crear un nou paràgraf només cal passar el seu codi corresponent:

```

1 var inicialitzacio = function() {
2   var $paragraf = $('

Paràgraf generat dinàmicament</p>');
3   $(document.body).append($paragraf);
4 }
5
6 $(document).ready(inicialitzacio);


```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/JEWPow.

Com es pot apreciar, primer s'ha creat un nou objecte jQuery referenciat per `$paragraf`, i seguidament s'ha seleccionat l'element body (obtingut a través de la propietat del document) i s'hi ha afegit aquest nou element.

Vegeu-ne a continuació un exemple una mica més complex, però que està fet aplicant la mateixa mecànica:

```

1 var inicialitzacio = function() {
2   var $taula = $('

| nom   | cognom   |
|-------|----------|
| Josep | Campmany |
| Maria | Torres   |

');
3   $(document.body).append($taula);
4 }
5
6 $(document).ready(inicialitzacio);

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/bwEKZE.

Tot i que s'ha fet servir el mateix sistema, aquest cop la codificació és molt menys entenedora. En cas d'haver d'afegir-hi més files, el codi s'aniria fent cada vegada més complicat i més difícil de depurar. Atès que aquest paràmetre no és més que una cadena de text, podem confeccionar-la pas a pas.

Una possible solució seria aplicar el disseny descendant, creant tot un seguit de funcions que permetin confeccionar el codi per generar la taula:

```

1 var generarTaula = function(alumnes) {
2   var html = '<table>' + generarCapcalera();
3
4   for (var i = 0; i < alumnes.length; i++) {
5     html += generarFila(alumnes[i]);
6   }
7
8   html += '</table>';
9   return html;
10 }
11
12 var generarCapcalera = function() {
13   var capcalera = '<tr>';
14   capcalera += '<th>nom</th>';
15   capcalera += '<th>cognom</th>';
16   capcalera += '</tr>';
17   return capcalera;
18 }
19
20 var generarFila = function(alumne) {
21   var fila = '<tr>';
22   fila += '<td>' + alumne.nom + '</td>';
23   fila += '<td>' + alumne.cognom + '</td>';
24   fila += '</tr>';

```

```

25     return fila;
26 }
27
28 var inicialitzacio = function() {
29     var alumnes = [
30         {nom: 'Josep', cognom: 'Campmany'},
31         {nom: 'Maria', cognom: 'Torres'},
32         {nom: 'Alex', cognom: 'Puig'},
33         {nom: 'Ana', cognom: 'Perez'}
34     ];
35
36     var html = generarTaula(alumnes);
37     var $taula = $(html);
38     $(document.body).append($taula);
39 }
40
41 $(document).ready(inicialitzacio);

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/ALEdkJ?editors=0010.

Com podeu veure, la llargària del codi ha augmentat considerablement, però la complexitat s'ha reduït i s'ha millorat l'escalabilitat. Per exemple, si en lloc de quatre alumnes en fossin quaranta, el codi seria el mateix modificant només el diccionari de dades. A més a més, el codi de cada funció és molt clar, de manera que és molt fàcil modificar-lo.

En primer lloc s'ha afegit un diccionari de dades que conté les dades dels alumnes, d'aquesta manera es podria canviar l'origen d'aquestes dades, per exemple, carregant-les d'un fitxer extern sense haver de modificar la resta del programa.

Dintre de la funció d'inicialització es genera el codi HTML cridant la funció `generarTaula` i passant-hi com a argument el diccionari de dades. Dintre d'aquesta funció es genera una cadena de text amb el codi per generar una taula, al qual s'afegeix el codi de la capçalera cridant la funció `generarCapcalera`.

Seguidament es recorren tots els elements del diccionari de dades afegint el codi per cada fila cridant la funció `generarFila` amb les dades de cada alumne. Finalment es tanca l'etiqueta de la taula i es retorna el codi complet, que és convertit en un objecte jQuery i afegit al cos del document.

Aquesta és només una de les tècniques que ofereix jQuery per compondre una branca d'elements; en cada cas s'ha de valorar quin és el sistema que millor s'adapta a la vostra aplicació.

Un altre sistema per generar nous objectes jQuery és passar com a argument elements del DOM:

```

1 <style>
2     .vermell {
3         color: red;
4     }
5
6     .negreta {
7         font-weight: bold;
8     }
9 </style>
10
11 <p>Primer paràgraf</p>

```

```

12 <p id="segon">Segon paràgraf</p>
13 <p>Tercer paràgraf</p>
14
15 <script>
16   var inicialitzacio = function() {
17     var paragrafs = document.getElementsByTagName('p');
18     var paragraf2 = document.getElementById('segon');
19
20     var $paragrafs = $(paragrafs);
21     var $paragraf2 = $(paragraf2);
22     $paragrafs.addClass('vermell');
23     $paragraf2.addClass('negreta');
24   }
25
26   $(document).ready(inicialitzacio);
27 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/rrZaYO.

Com es pot apreciar, primer de tot s'han obtingut els nodes fent servir els mètodes de les interfícies del DOM (una col·lecció de nodes en el primer cas, i un únic node en el segon). A continuació, s'han generat els dos objectes jQuery passant els elements com a argument, i finalment, s'ha afegit la classe corresponent a cadascun: `vermell` a tots els paràgrafs i `negreta` al segon.

2.3.2 Manipulació de classes: '`addClass`', '`removeClass`' i '`toggleClass`'

A banda de l'addició de classes als elements, jQuery ofereix mètodes per eliminar-les i per activar-les/desactivar-les. Tots tres casos són fàcils de fer servir: només cal invocar el mètode sobre l'objecte jQuery passant com a argument el nom de la classe:

```

1 <style>
2   .vermell {
3     color: red;
4   }
5
6   .verd {
7     color: green;
8   }
9
10  .negreta {
11    font-weight: bold;
12  }
13 </style>
14
15 <p class="negreta">Paràgraf 1: originalment en negreta</p>
16 <p class="vermell">Paràgraf 2: originalment vermell</p>
17
18 <script>
19   var inicialitzacio = function() {
20     var $paragrafs = $('p');
21
22     $paragrafs.removeClass('vermell');
23     $paragrafs.addClass('verd');
24     $paragrafs.toggleClass('negreta');
25   }

```

```

26
27     $(document).ready(initialitzacio);
28 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/QKyBdA.

Com es pot apreciar no es produeix cap error en eliminar la classe vermella de tots els paràgrafs, tot i que el primer paràgraf no conté aquesta classe. Com és d'esperar, quan s'afegeix la classe verd, s'afegeix correctament a tots dos paràgrafs.

Cal parar especial atenció al comportament d'invocar `toggleClass`: es determina si s'ha d'activar o desactivar segons l'estat concret de cada element, és a dir, en el primer cas estava activat i s'elimina, mentre que en el segon cas s'afegeix.

2.3.3 Modificar continguts: 'val', 'html' i 'text'

A l'hora de modificar el contingut d'un element s'ha de distingir entre modificar el seu valor (aplicable a elements de formularis), el contingut textual o el codi HTML (quan conté altres elements).

Per consultar i modificar aquests continguts, jQuery ofereix els mètodes `val` per als elements de formularis, `html` com a equivalent a la propietat `innerHTML` i `text` per manipular els continguts textuais.

El funcionament és molt simple: aquests mètodes s'invoquen a partir de l'objecte jQuery que contingui els elements que es volen modificar. Si es vol consultar el valor, s'han d'invocar sense passar cap argument; en canvi, si es volen modificar, s'ha de passar com a argument el valor, codi o text per substituir.

```

1  <label>Nom: </label>
2  <input type="text" value="Pere" />
3  <p>Paràgraf 1</p>
4  <p>Paràgraf 2</p>
5
6  <script>
7      var initialitzacio = function() {
8          var $text = $('input');
9          var $paragraf1 = $('p:first');
10         var $paragraf2 = $('p:last');
11
12         console.log($text.val());
13         $text.val('Maria');
14
15         console.log($paragraf1.html());
16         $paragraf1.html("<ul><li>Element 1</li><li>Element 2</li></ul>");
17
18         console.log($paragraf2.text());
19         $paragraf2.text("El contingut textual d'aquest paràgraf ha estat modificat"
20                         );
21
22         $(document).ready(initialitzacio);
23 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/qakBLk.

Fixeu-vos que el resultat mostrat per la consola es correspon amb el valor original, mentre que al document es mostren els continguts actualitzats: Maria com a valor del quadre de text, una llista de dos elements com a contingut del primer paràgraf, i una frase diferent com a contingut del segon paràgraf.

2.3.4 Modificar estils: 'css'

A banda de treballar amb classes, la biblioteca jQuery també permet manipular directament els estils CSS d'un element de forma molt més simple que si ho heu de fer directament amb les interfícies del DOM.

Per afegir, modificar o eliminar un estil només cal invocar el mètode `css` de l'objecte jQuery que contingui els elements que cal modificar:

- **Afegir estils:** es passa com a argument el nom de la propietat CSS i el valor que s'ha d'assignar.
- **Eliminar estils:** es passa com a argument el nom de la propietat CSS i una cadena buida com a valor.
- **Consultar el valor d'una propietat CSS:** es passa com a argument el nom de la propietat CSS.

A continuació podeu veure un exemple en què es consulten les propietats CSS d'un element, se n'afegeixen de noves i se n'eliminen:

```
1 <p id="paragraf" style="color: red; font-weight: bold">Paràgraf</p>
2
3 <script>
4   var mostrarEstils = function (element) {
5     var estil = element.style;
6     var $element = $(element)
7     for (var i=0; i<estil.length; i++) {
8       console.log (estil[i] + ':' + $element.css(estil[i]) + ";");
9     }
10  }
11
12  var inicialitzacio = function() {
13   var paragraf = document.getElementById('paragraf');
14   var $paragraf = $(paragraf);
15
16   console.log('— Estil original —');
17   mostrarEstils(paragraf);
18
19   $paragraf.css('font-size', '30px');
20   $paragraf.css('color', '');
21   $paragraf.css('font-weight', 'lighter');
22
23   console.log('— Estil modificat —');
24   mostrarEstils(paragraf);
25  }
26
27  $(document).ready(inicialitzacio);
```

28 `</script>`

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/VKaPNP?editors=1011.

Com es pot apreciar, jQuery no ofereix una solució simple per mostrar els estils. En lloc d'utilitzar el mètode `getComputedStyle` s'ha cridat el mètode `css` per a cadascuna de les propietats, que retorna el mateix resultat.

Fixeu-vos que tant per afegir mètodes com per actualitzar només cal passar el nom i el valor de la propietat, mentre que per eliminar-la s'ha de passar una cadena buida.

2.3.5 Manipular atributs: 'attr' i 'prop'

La biblioteca jQuery ofereix dos mètodes diferents per tractar amb atributs:

- El mètode `attr`, que serveix per consultar i manipular els atributs en general.
- El mètode `prop`, que serveix per **consultar i modificar propietats del DOM com `checked`, `selected` o `disabled`**.

Una diferència important entre `attr` i `prop` és que si es fa servir el primer per obtenir el valor d'una propietat com `checked`, el valor retornat és una cadena de text; en canvi, amb el segon mètode el valor de retorn és un booleà, que correspon a l'estat de l'element (marcat o no marcat).

S'ha de tenir en compte que en cas que l'objecte jQuery tingui seleccionats múltiples elements, aquests mètodes retornen el valor del primer element.

```

1  <label id="1"><input type="checkbox" checked="checked"/>Casella A</label>
2  <label id="2"><input type="checkbox" />Casella B</label>
3
4  <script>
5      var $etiquetes = $('label');
6      var $etiqueta1 = $('#1');
7      var $etiqueta2 = $('#2');
8      var $caselles = $('#checkbox');
9
10     console.log('Atribut id de les etiquetes:', $etiquetes.attr('id'));
11
12     console.log('Canviant id per "A"...');
13     $etiquetes.attr('id', 'A');
14
15     console.log('Atribut id de la etiqueta1:', $etiqueta1.attr('id'));
16     console.log('Atribut id de la etiqueta2:', $etiqueta2.attr('id'));
17
18     console.log('Es troben les caselles marcades?', $caselles.prop('checked'));
19
20     console.log('Es troben les caselles marcades?', $caselles.attr('checked'));
21
22     console.log('Desmarquem totes les caselles');
23     $caselles.prop('checked', '');
24 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/GjjmwZ?editors=1011.

Com es pot apreciar, tant el mètode `attr` com el mètode `prop` retornen només un valor, els corresponents al primer element seleccionat. D'altra banda, en canviar l'`id` per A, aquest s'aplica a totes les etiquetes.

Fixeu-vos com el valor retornat per `prop` és diferent d'`attr` quan es passa `checked` com a argument. En el primer cas retorna un booleà, mentre que en el segon retorna el valor. Com en el cas de les etiquetes, el valor retornat en tots dos casos és el corresponent al primer element de la selecció, i en canviar el valor de la propietat, aquest canvi s'aplica a totes les caselles.

Així doncs, si es vol obtenir el valor de cadascun o es volen manipular els elements individualment s'ha d'iterar sobre ells, per exemple fent servir el mètode `each` dels objectes jQuery:

```

1  <label id="1"><input type="checkbox" checked="checked"/>Casella A</label>
2  <label id="2"><input type="checkbox" />Casella B</label>
3
4 <script>
5   var $etiquetes = $('label');
6   var $caselles = $('#checkbox');
7
8   var mostrarId = function ($items) {
9     $items.each(function(index) {
10       console.log('Atribut id de l\'etiqueta ' + index + ':', this.id);
11     });
12   }
13
14   var afegirTextAId = function ($items, text) {
15     $items.each(function() {
16       this.id += text;
17     });
18   }
19
20   var mostrarMarcada = function ($items) {
21     $items.each(function(index) {
22       console.log('Casella ' + index + ' marcada?', this.checked);
23     });
24   }
25
26   mostrarId($etiquetes);
27   console.log('Afegint text als Ids...');
28   afegirTextAId($etiquetes, 'casella');
29   mostrarId($etiquetes);
30
31   console.log('Estat de les caselles');
32   mostrarMarcada($caselles);
33 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/Kggqam?editors=1011.

La biblioteca ofereix **dos mètodes** `each` amb funcions diferents. Si es crida a partir de la funció jQuery (`jQuery.each()`), aquest iterarà sobre l'*array* o la col·lecció passada com a argument; en canvi, si es crida aquest mètode a un objecte jQuery, el mètode `each` iterarà sobre cadascun dels elements seleccionats.

Recordeu que cada element és el context en el que es realitza la acció; ni executa la acció (això ho fa la funció) ni és el destinatari (encara que es pot modificar l'element).

Gràcies al mètode `each` dels objectes jQuery, és molt fàcil iterar sobre tots els elements. Es passa al mètode `each` una funció com argument i aquesta serà invocada una vegada per cada element, fent servir com a context de la funció l'element corresponent. Opcionalment es pot afegir un paràmetre a la funció, que rebrà l'índex de l'element (0 pel primer, 1 pel segon, 2 pel tercer...).

Cal recalcar que el context d'execució de la funció és l'element seleccionat, per consegüent, `this` fa referència a aquest element, que és un node del DOM. És a dir, **no és un objecte jQuery**.

A continuació podeu trobar un exemple en què es modifica l'identificador dels elements de tipus `label` i es mostra per la consola del navegador si les caselles es troben o no marcades. En aquest exemple s'ha accedit directament a les seves propietats per simplificar, però és possible convertir aquests nodes en objectes jQuery:

```

1 <label id="1"><input type="checkbox" checked="checked"/>Casella A</label>
2 <label id="2"><input type="checkbox" />Casella B</label>
3
4 <script>
5   var $etiquetes = $('label');
6   var $caselles = $(':checkbox');
7
8   var mostrarId = function($items) {
9     $items.each(function(index) {
10       var $element = $(this);
11
12       console.log('Atribut id de l\'etiqueta ' + index + ':', $element.attr('id'));
13     });
14   }
15
16   var afegirTextAId = function($items, text) {
17     $items.each(function() {
18       var $element = $(this);
19       $element.attr('id', $element.attr('id') + text);
20     });
21   }
22
23   var mostrarMarcada = function($items) {
24     $items.each(function(index) {
25       var $element = $(this);
26       console.log('Casella ' + index + ' marcada?', $element.prop('checked'));
27     });
28   }
29
30   mostrarId($etiquetes);
31   console.log('Afegint text als Ids...');
32   afegirTextAId($etiquetes, 'casella');
33   mostrarId($etiquetes);
34
35   console.log('Estat de les caselles');
36   mostrarMarcada($caselles);
37 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/rrrwdJ?editors=1011.

Recordeu que una de les maneres de crear un objecte jQuery és passar un element directament a la funció; així doncs, es passa l'element que serveix de context a la funció (`this`) i es genera un nou objecte jQuery a partir del qual es poden cridar els mètodes pertinents.

Fixeu-vos que en aquest cas concret la implementació s'ha complicat més i l'eficiència és menor que a l'exemple anterior, però pot haver-hi situacions en les quals sigui més pràctic treballar dintre de la funció amb objectes jQuery. S'ha de valorar cada cas i fer servir una implementació o altra segons les vostres necessitats.

2.4 Manipular el DOM

La biblioteca jQuery destaca tant per la facilitat amb la qual es poden seleccionar elements com per la facilitat per crear-ne nous. Concretament, **per generar nous elements**, ofereix les **següents opcions**, segons el tipus de paràmetre passat:

- **cadena de text que sembli HTML**: `$('<p>')` o `$('<p id='paragraf2'>Paràgraf amb èmfasi</p>')`.
- **Node HTML existent** (clona un element ja existent node): `$(element)`.

```

1 <div id="contenidor"></div>
2
3 <script>
4   var $contenidor = $('#contenidor');
5
6   var $nouParagraf = $('<p>');
7   $nouParagraf.html('Paràgraf normal');
8
9   var $nouParagrafEmfasi = $('<p id="paragraf2"><em>Paràgraf amb èmfasi</em></p
10  >');
11
12  $contenidor.append($nouParagraf);
13  $contenidor.append($nouParagrafEmfasi);
14
15  var paragraf3= document.createElement('p');
16  paragraf3.innerHTML = 'Paràgraf per clonar';
17
18  var $paragrafClonat = $(paragraf3);
19  $contenidor.append($paragrafClonat);
</script>
```

Podeu veure aquest exemple l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/ALLaAm?editors=1010.

En tots dos casos el retorn de la funció és un objecte jQuery que embolcalla el nou element (o elements). S'ha de tenir en compte que aquests nous elements encara no formen part del document, sinó que només es troben a la memòria.

Per afegir-los al DOM, la biblioteca ofereix diversos mètodes. Aquests mètodes s'han de cridar a partir d'un objecte que serà el contenidor dels elements, passant com a paràmetre el nou element que s'ha d'afegir:

- **prepend**: afegeix l'element al principi de la llista de descendents directes del contenidor.

- **append**: afegeix l'element al final de la llista descendents directes del contenidor.
- **before**: afegeix l'element al mateix nivell que el contenidor, però davant seu.
- **after**: afegeix l'element al mateix nivell que el contenidor, però darrere seu.

És a dir, els mètodes **prepend i append** col·loquen el nou element **dins del contenidor**, mentre que els mètodes **after i before** el col·loquen **fora** d'aquest, davant i darrere respectivament.

```

1  <ul>
2    <li>Element 1</li>
3    <li>Element 2</li>
4    <li>Element 3</li>
5  </ul>
6
7  <script>
8    var $llista = $('#ul');
9
10   var $nouItem1 = $('- Element append</li>');
11   $llista.append($nouItem1);
12
13   var $nouItem2 = $('- Element prepend</li>');
14   $llista.prepend($nouItem2);
15
16   var $nouItem3 = $('
- Element before</li>');
17   $llista.before($nouItem3);
18
19   var $nouItem4 = $('
- Element after</li>');
20   $llista.after($nouItem4);
21 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/vXXjpx?editors=1010.

Fixeu-vos que una vegada s'ha generat l'element, encara que no s'hagi afegit al document ja es troba a la memòria. I com que es tracta d'un objecte jQuery, es poden invocar pràcticament tots els seus mètodes, per exemple per afegir classes:

```

1  <style>
2    .vermell {
3      color: red;
4    }
5  </style>
6
7  <div id="contenidor"></div>
8
9  <script>
10   var $contenidor = $('#contenidor');
11   var $paragraf = $('

Paragraf <span>creat</span> i <span>manipulat</span> a la memòria</p>');
12
13   $paragraf.find('span').addClass('vermell');
14
15   $contenidor.append($paragraf);
16 </script>


```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/wzzjxj.

Com podeu veure, s'afegeix la classe **vermell** als elements de tipus **span** abans d'afegir-lo al document; d'aquesta manera, l'operació s'executa correctament.

Quant a l'**eliminació d'elements**, jQuery ofereix dos mètodes: **`remove`** i **`empty`**. **El primer elimina tots els elements seleccionats** per l'objecte jQuery a partir del qual s'invoca, mentre que **el segon elimina els descendents d'aquests nodes** (és a dir, **`els buida`**):

```

1 <ul id="llista1">
2   <li>Element A</li>
3   <li>Element B</li>
4   <li>Element C</li>
5 </ul>
6
7 <ul id="llista2">
8   <li>Element 1</li>
9   <li>Element 2</li>
10  <li>Element 3</li>
11 </ul>
12
13 <script>
14   var $llista1 = $('#llista1');
15   var $llista2 = $('#llista2');
16
17   $llista1.remove();
18   $llista2.empty();
19
20   console.log("Existeix l'element amb id llista1:", document.getElementById(
21     'llista1'));
22
23   console.log("Existeix l'element amb id llista2:", document.getElementById(
24     'llista2'));
25 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/VKKxNW?editors=1011.

Fixeu-vos que mentre que la primera llista ha deixat d'existir, la segona continua a la pàgina (tot i que no es veu perquè és buida) però s'han eliminat tots els seus descendents.

2.5 Utilitzar **Events amb jQuery**

Entre els avantatges d'utilitzar jQuery a l'hora d'afegir la detecció d'esdeveniments cal destacar-ne dos:

- Es pot assignar la **detecció de múltiples esdeveniments amb una sola línia**.
- **La detecció d'esdeveniments s'aplica a tots els elements seleccionats**.

Per afegir un detector, s'ha de generar primer un objecte jQuery: se seleccionen els elements als quals voleu afegir-lo, i seguidament s'invoca el mètode `on`, passant com a arguments una cadena amb el nom dels elements que cal detectar i la funció que es farà servir de *callback* (funció que serà cridada internament).

Cal que recordeu que tot i que als exemples s'acostuma a fer servir funcions anònimes, es pot passar qualsevol tipus de funció i, per tant, podria ser una funció amb nom definida en altre punt de l'aplicació o una variable que referenciés una funció o un mètode.

```

1 <ul id="llista1">
2   <li>Element A</li>
3   <li>Element B</li>
4   <li>Element C</li>
5 </ul>
6
7 <ul id="llista2">
8   <li>Element 1</li>
9   <li>Element 2</li>
10  <li>Element 3</li>
11 </ul>
12
13 <script>
14   var $llista1 = $('#llista1 li');
15   var $llista2 = $('#llista2 li');
16
17   $llista1.on('click mouseenter mouseout', function () {
18     $(this).toggleClass('vermell');
19   });
20
21   $llista2 dblclick(function () {
22     $(this).remove();
23   });
24 </script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/gwwKmB.

Com es pot apreciar, s'ha afegit la detecció per als *events* click, mouseenter i mouseout als elements de la primera llista, de manera que aquests es posen de color vermell quan hi entra el cursor o quan es fa clic sobre ells, mentre que en sortir es fa l'acció inversa.

D'altra banda, a la segona llista s'ha afegit la detecció de l'*event* dblclick, que es dispara en fer doble clic sobre algun dels elements, i elimina aquest element de la llista en disparar-se.

Mètode remove() al DOM

Existeix un mètode remove a l'especificació del DOM que es pot cridar directament sobre l'element, però no és compatible amb navegadors antics, particularment amb versions d'Internet Explorer anteriors a Edge.

Dreceres d'events

Podeu trobar una llista completa de les dreceres d'*events* en l'enllaç següent: goo.gl/BJMZh.

Cal destacar que el context de la funció (*this*) fa referència a l'element en el qual s'ha detectat l'esdeveniment; així doncs, a la primera llista es genera un objecte jQuery a partir d'aquest i s'activa o desactiva la classe vermell. De la mateixa manera, en el segon cas, per evitar incompatibilitats, es genera l'objecte i s'invoca el mètode remove per eliminar-lo.

Fixeu-vos que en el segon cas, en lloc d'invocar el mètode on i passar una cadena de text amb el nom de l'*event*, s'ha invocat dblclick. Això és possible perquè jQuery ofereix una llarga llista de mètodes que poden invocar-se com dreceres (click, dblclick, load, etc.), és a dir, pot invocar-se aquest mètode passant com a únic argument la funció que serà invocada en detectar-se l'esdeveniment. Com és d'esperar, quan es fan servir dreceres hi ha una limitació important: només es pot detectar un únic *event*.

En cas que necessiteu eliminar la detecció d'esdeveniments només cal fer servir el mètode off:

```

1 <style>
2   div {
3     width: 100px;
4     height: 100px;
5     float: left;
6     background-color: grey;
7     margin: 1px;
8   }
9   .vermell {
10    background-color:red;
11  }
12 </style>
13
14 <h1>Selecciona un quadre</h1>
15 <div></div>
16 <div></div>
17 <div></div>
18 <div></div>
19 <div></div>
20
21 <script>
22   $quadres = $('div');
23
24   $quadres.click(function() {
25     $(this).addClass('vermell');
26     $quadres.off();
27     $('h1').html('Quadre seleccionat');
28   });
29 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/amBKLM.

Com es pot apreciar, una vegada es clica qualsevol dels quadres, el quadre en qüestió es mostra de color vermell (s'aplica la classe `vermell`), i s'elimina la detecció d'esdeveniments de tots els quadres invocant el mètode `off` sense passar-hi cap argument.

Una altra opció que ofereix el mètode `off` és eliminar només alguns dels *events* de la detecció, com es pot comprovar en l'exemple següent:

```

1 <style>
2   .vermell {
3     color:red;
4   }
5 </style>
6
7 <ul>
8   <li>Element A</li>
9   <li>Element B</li>
10  <li>Element C</li>
11 </ul>
12
13 <script>
14   var $elements = $('li');
15
16   $elements.on('click mouseenter mouseout', function () {
17     $(this).toggleClass('vermell');
18   });
19
20   $elements.on('click', function(function() {
21     $(this).off('mouseenter mouseout');
22   })
23 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/LRRBzq.

Fixeu-vos que **s'han afegit dos detectors diferents per a l'*event* click** i tots dos es criden en fer clic sobre qualsevol dels elements. El primer s'encarrega de canviar el color de la classe, mentre que el segon elimina la detecció d'*events* per a mouseenter i mouseout. És a dir, una vegada es fa clic sobre un element aquest només respondrà a l'*event* click.

Alternativament es podria haver fet servir la drecera click en lloc del mètode on per al segon detector; el resultat hauria estat el mateix:

```
1 $elements.click( function() {
2   $(this).off('mouseenter mouseout');
3 }
```

2.6 Crear connectors per jQuery

En el context de la biblioteca jQuery, un connector (*plug-in*, en anglès) és un **component que afegeix noves funcionalitats a la biblioteca**. És a dir, una vegada tenim el connector carregat, aquest s'afegeix automàticament a la biblioteca i és accessible per a tots els objectes jQuery.

El gestor de paquets **npm** està basat en Node.js i per fer-lo servir s'ha de tenir instal·lat.

Node.js és una tecnologia que permet utilitzar JavaScript al servidor.

Les galeries d'imatges o els carrusells d'anuncis o productes són exemples habituals de l'ús d'aquests connectors, però es poden trobar tot tipus de connectors en diferents repositoris, a més a més dels que podeu crear vosaltres mateixos.

Un dels espais web on es poden trobar aquests repositoris és al lloc web d'npm (www.npmjs.com). Aquí es pot localitzar fàcilment tot tipus de programari en JavaScript, tant per als navegadors com per al servidor (per utilitzar amb Node.js).

Fixeu-vos que només una part d'aquests paquets són connectors de jQuery; per localitzar-los **es recomana fer servir aquest enllaç**: www.goo.gl/Lc9c6Z, que filtra directament els components que inclouen la paraula jquery-plugin. Si proveu d'utilitzar el cercador del mateix lloc, veureu que el resultat és diferent.

Per descomptat, l'ús d'aquest gestor de paquets és opcional i podeu utilitzar els vostres propis connectors directament o descarregar-los de qualsevol altre repositori de tercers sense cap problema. Però en aquest últim cas és preferible fer servir una font fiable per evitar trobar-vos que s'està manipulant la vostra aplicació de forma malintencionada.

A continuació veureu com es pot crear el vostre propi connector per a jQuery, de manera que pugui ser utilitzat per tots els components de la vostra aplicació sense haver de fer-lo global.

En primer lloc, cal tenir clar quin serà el comportament que ha de tenir. En aquest cas s'ha decidit que les característiques del connector seran les següents:

- L'objectiu del connector és facilitar l'addició (i eliminació) d'elements a una o més llistes.
- S'afegiran dos botons (+ i -) i un títol que es mostrarà a la part superior de la llista.
- En clicar el botó + s'afegirà un quadre de text en l'última posició de la llista que indicarà que s'ha d'escriure un text.
- Quan es perdi el focus, si es troba algun valor, s'establirà aquest com a text de la llista.
- El quadre de text s'eliminarà o es reemplaçarà en qualsevol cas.
- Si s'ha entrat un valor, s'afegirà automàticament un nou quadre de text al final de la llista.
- Sempre que s'afegeixi un nou quadre, aquest rebrà el focus.
- Si es clica el botó -, s'eliminarà l'últim element de la llista.

Fent una primera implementació d'aquesta funcionalitat es pot obtenir un codi similar al següent:

```

1 <style>
2   ul {
3     list-style-type: none;
4     padding: 0;
5     width: 200px;
6     float: left;
7     margin: 5px;
8   }
9
10  ul div {
11    border-bottom: 1px gray dotted;
12    padding-bottom: 2px;
13  }
14 </style>
15
16 <ul>
17 </ul>
18 <ul>
19 </ul>
20 <ul>
21 </ul>
22
23 <script>
24   $llistes = $('ul');
25
26   $llistes.each(function() {
27     var $barraEines = $('

Gestor Llistes </div>');
28     var $botoMes = $('+</button>');
29     var $botoMenys = $('-</button>');
30
31     $barraEines.append($botoMes);
32     $barraEines.append($botoMenys);
33
34     var $contenidor = $(this);
35     $contenidor.prepend($barraEines);
36
37     $botoMes.click(function() {
38       var $quadre = $('- <input type="text" placeholder="Escriu aquí..."/></li>');
39
40       $quadre.on('change focusout', function() {


```

```

41         var valor = $quadre.find('input').val();
42
43         if (valor) {
44             $quadre.html(valor);
45             $botoMes.trigger('click');
46         } else {
47             // Si no hi ha cap contingut, s'elimina
48             $quadre.remove();
49         }
50     });
51     $contenidor.append($quadre);
52     $quadre.find('input').focus();
53 });
54
55 $botoMenys.click(function() {
56     $contenidor.find('li:last-child').remove();
57 });
58 });
59 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/qaazvw.

Primerament es fa una selecció de tots els elements de tipus llista, i a continuació s'itera sobre aquests elements per afegir la funcionalitat a totes les llistes trobades.

Es crea un contenidor per a la barra d'eines que inclou el títol Gestor Llistes seguit dels dos botons + i -, als quals se'ls afegeix la detecció de l'*event click*.

En clicar el primer botó es crea un nou element a la llista que inclou un quadre d'entrada de text i que detecta els *events change* i *focusout*, de manera que si aquest canvia o perd el focus és substituït per un text pla –si s'ha entrat algun valor– o l'elimina –en cas contrari–.

Addicionalment, en crear-se l'element de la llista, s'estableix el focus en el quadre d'entrada i això permet introduir dades de forma més fluida. D'altra banda, una vegada es detecta el canvi i s'estableix com a text, es genera automàticament un nou element a la llista per omplir (per simplificar s'ha fet disparant l'*event click* sobre el botó +).

En clicar sobre el botó - s'elimina l'última fila de la llista fins que no en queda cap. No cal fer cap comprovació addicional.

Fixeu-vos que també s'ha afegit el codi CSS per formatar les llistes. Aquest codi s'hauria d'incloure amb el connector, de manera que es concretés més per no interferir en l'estructura de la pàgina on es faria servir el connector, per exemple, afegint-hi alguna classe pròpia. Substituïu les declaracions de \$contenidor i de la \$barraEines respectivament al codi JavaScript per les següents:

```

1 var $contenidor = $(this);
2 $contenidor.addClass('gestor-llistes');
3
4 var $barraEines = $('<div>Gestor Llistes </div>');
5 $barraEines.addClass('barra-eines');

```

I substituïu el codi CSS pel següent:

```

1 ul.gestor-llistes {
2   list-style-type: none;
3   padding: 0;

```

El mètode trigger permet disparar un esdeveniment manualment.

```

4   width: 200px;
5   float: left;
6   margin: 5px;
7 }
8
9 ul.gestor-llistes div.barra-eines {
10    border-bottom: 1px gray dotted;
11    padding-bottom: 2px;
12 }
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/QKKZRz.

Tot i que el comportament és el mateix, ja no hi haurà conflictes quan es faci servir el connector. El següent pas és convertir-lo en un connector de jQuery. Per fer-ho només s'ha d'afegir una funció anomenada *arrodonir* a `$.fn`, i a partir d'aquest moment aquesta funció passarà a estar disponible a tots els objectes jQuery. Reemplaçeu el codi JavaScript pel següent:

```

1 $(document).ready(function() {
2   $('ul').gestionar();
3 });
4
5 $.fn.gestionar = function() {
6
7   this.each(function() {
8     var $contenidor = $(this);
9     $contenidor.addClass('gestor-llistes');
10
11    var $barraEines = $('

Gestor Llistes </div>');
12    $barraEines.addClass('barra-eines');
13
14    var $botoMes = $('+</button>');
15    var $botoMenys = $('-</button>');
16
17    $barraEines.append($botoMes);
18    $barraEines.append($botoMenys);
19
20    $contenidor.prepend($barraEines);
21
22    $botoMes.click(function() {
23      var $quadre = $('- <input type="text" placeholder="Escriu aquí..."/></li>');
24
25      $quadre.on('change focusout', function() {
26        var valor = $quadre.find('input').val();
27
28        if (valor) {
29          $quadre.html(valor);
30          $botoMes.trigger('click');
31        } else {
32          // Si no hi ha cap contingut s'elimina
33          $quadre.remove();
34        }
35      });
36      $contenidor.append($quadre);
37      $quadre.find('input').focus();
38    });
39
40    $botoMenys.click(function() {
41      $contenidor.find('li:last-child').remove();
42    });
43  });
44}


```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-dawm06/pen/pENzom.

Com es pot apreciar, només hi ha tres canvis molt lleugers:

- Es crida el mètode `gestionar` directament sobre l'objecte jQuery que conté la selecció de les llistes.
- S'ha posat tot el codi del connector dins de la funció `$.fn.gestionar`: això és el que possibilita cridar la funció des de qualsevol objecte jQuery.
- En lloc d'iterar sobre `$llistes` es fa sobre `this`, ja que en el context del connector `this` fa referència a l'objecte jQuery des del qual s'ha cridat, és a dir, el que selecciona totes les llistes en el nostre cas.

Per millorar el connector es pot implementar de forma que accepti arguments per flexibilitzar encara més la seva utilitat. Per exemple, per afegir un títol i missatge del quadre de text personalitzat:

```

1 <style>
2   ul.gestor_llistes {
3     list-style-type: none;
4     padding: 0;
5     width: 200px;
6     float: left;
7     margin: 5px;
8   }
9
10  ul.gestor_llistes div.barra-eines {
11    border-bottom: 1px gray dotted;
12    padding-bottom: 2px;
13  }
14 </style>
15
16 <ul id="Nom">
17 </ul>
18 <ul id="Generic">
19 </ul>
20 <ul id="Curs">
21 </ul>
22
23 <script>
24   $(document).ready(function() {
25     $('ul#Nom').gestionar('Noms', 'Introduceix un nom');
26     $('ul#Curs').gestionar('Curs', 'Introduceix un curs');
27     $('ul#Generic').gestionar();
28   });
29
30   $.fn.gestionar = function(titol, placeholder) {
31     if (!titol) {
32       titol = "Gestor de llistes";
33     }
34     if (!placeholder) {
35       placeholder = "Escriu aquí...";
36     }
37
38     this.each(function() {
39       var $contenidor = $(this);
40       $contenidor.addClass('gestor_llistes');
41
42       var $barraEines = $('

' + titol + '

');
43       $barraEines.addClass('barra-eines');
```

```

45     var $botoMes = $('<button>+</button>');
46     var $botoMenys = $('<button>-</button>');
47
48     $barraEines.append($botoMes);
49     $barraEines.append($botoMenys);
50
51     $contenidor.prepend($barraEines);
52
53     $botoMes.click(function() {
54         var $quadre = ('<li><input type="text" placeholder=' + placeholder +
55                     '"></li>');
56
57         $quadre.on('change focusout', function() {
58             var valor = $quadre.find('input').val();
59
60             if (valor) {
61                 $quadre.html(valor);
62                 $botoMes.trigger('click');
63             } else {
64                 // Si no hi ha cap contingut s'elimina
65                 $quadre.remove();
66             }
67         });
68         $contenidor.append($quadre);
69         $quadre.find('input').focus();
70     });
71
72     $botoMenys.click(function() {
73         $contenidor.find('li:last-child').remove();
74     });
75
76 });
77 </script>

```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/PGbYpd.

Només ha calgut afegir els paràmetres a la funció gestionar i utilitzar aquests valors internament: el paràmetre titol per substituir el títol que mostra el connector i el paràmetre placeholder com a atribut placeholder del quadre de text.

Un altre canvi que s'ha fet és afegir un identificador per a cada llista al codi HTML. Això no és cap requisit, s'ha fet així per simplificar la selecció individual i mostrar com cadascuna de les llistes aplica el mateix connector d'una manera personalitzada, gràcies a la parametrització.

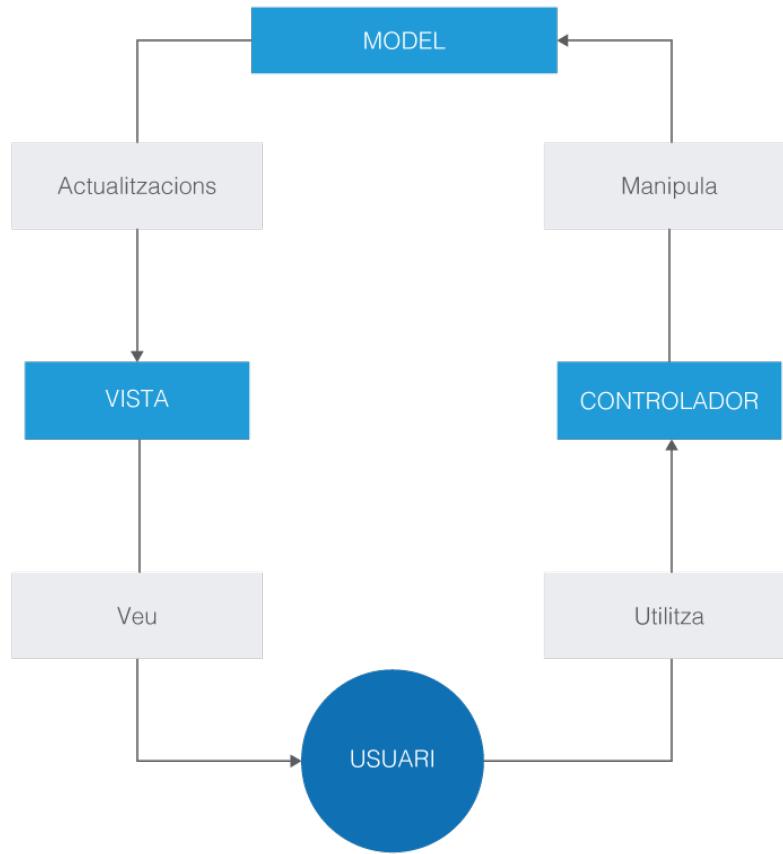
2.7 Model-vista-controlador (MVC)

El model-vista-controlador és un patró de disseny complex utilitzat per implementar interfícies d'usuari, que separa l'aplicació en tres parts, com es pot apreciar a la figura 2.3:

- El **model**: encarregat de gestionar les dades i la lògica de l'aplicació.
- Les **vistes**: representacions de la informació que es mostra a l'usuari. Hi pot haver diferents vistes per a una mateixa informació.

- Els **controladors**: encarregats de gestionar l'entrada de dades des de les vistes i fer les operacions necessàries sobre el model.

FIGURA 2.3. Representació típica dels elements del patró MVC



Fixeu-vos que mentre que la **vista** generalment estarà programada amb HTML, CSS i JavaScript i s'executa en el client (el navegador), el **model** i el **controlador** poden trobar-se en un servidor remot i fer servir un llenguatge diferent de JavaScript, per exemple PHP o Java. Per tant, la implementació d'aquest patró pot requerir la utilització de diferents tecnologies.

Tot i que a cop d'ull sembla senzill, és molt fàcil confondre quin component pertany a cada part, ja que en programació es treballa amb conceptes força abstractes. Vegeu els següents dos exemples d'aplicació d'aquest patró:

Exemple de sistema d'autenticació

Un exemple molt simple seria aplicar aquest patró a un sistema d'autenticació; la divisió de les tres parts seria la següent:

- **Model**: encarregat de comprovar si la informació d'un usuari i la seva contrasenya són correctes, si l'usuari és actiu, etc.
- **Controlador**: rep l'entrada del client (nom d'usuari i contrasenya) i passa aquestes dades al model que retornarà si són correctes o no. En qualsevol cas es retornarà una resposta al model (o es generarà una nova vista) amb aquesta informació (accés concedit o denegat).
- **Vista**: un formulari per autenticar l'usuari amb dos quadres de text (un per al nom d'usuari i un altre per a la contrasenya), i un botó per enviar la petició al controlador.

D'aquesta manera, es podrien afegir nous sistemes d'autenticació sense haver de tocar el **model**, només afegint nous mètodes al controlador (o nous controladors) i les **vistes** necessàries per interactuar amb el sistema. Per exemple, des de la **vista** es podria fer una petició asíncrona al **controlador** i que aquest, com a resposta, retornés un objecte JSON amb la informació de l'intent (èxit o error) i, en conseqüència, la vista mostraria el missatge corresponent.

Un exemple de com es podrien utilitzar diferents vistes a partir d'una mateixa informació seria el següent:

Exemple d'utilització de múltiples vistes

Suposeu que treballieu en una aplicació que gestiona les dades d'un institut i entre aquestes dades es troba la informació de tots els alumnes i els cursos.

Aquesta informació és proporcionada pel **model**, es realitzen les consultes a través del **controlador** i com a resposta es podrien generar diferents **vistes** (segons l'entrada enviada al controlador). Per exemple, es podrien generar les següents vistes:

- Una vista que mostri el llistat de tots els alumnes de l'institut ordenats alfabèticament.
- Una vista que mostri la mateixa informació però agrupada per cursos.
- Una vista que mostri una gràfica amb la informació demogràfica dels alumnes.
- Una vista que mostri un mapa amb xinxetes indicant la localització del domicili de cada alumne.

Cal destacar que aquest és un concepte avançat i no és recomanable que els programadors novells intentin implementar-lo a les seves aplicacions (excepte a les més senzilles). Afortunadament hi ha molts entorns de treball (*frameworks*) que implementen aquest patró i són molt utilitzats en el desenvolupament d'applicacions complexes com Angular o Ember i biblioteques especialitzades com React.

L'avantatge d'utilitzar un entorn de treball és que permet començar a treballar més ràpidament, ja que inclouen tots els elements necessaris per crear una aplicació. En canvi, si es fa servir una biblioteca com React, serà necessari afegir altres biblioteques per afegir certes funcions (o desenvolupar-les vosaltres mateixos).

2.7.1 Angular

Angular (www.angular.io) és un dels entorns de treball més utilitzats i el matenent, principalment, enginyers de Google. Una vegada s'ha afegit ja es pot començar a treballar-hi directament, ja que inclou totes les funcionalitats per desenvolupar aplicacions complexes, sense requerir cap altra biblioteca.

La versió Angular 2 i posteriors són incompatibles amb la versió 1 i, per consegüent, la documentació relativa a versions anteriors està obsoleta i s'ha d'ignorar.

TypeScript

Podeu trobar més informació sobre TypeScript en l'enllaç següent:
www.typescriptlang.org.

Gulp i Grunt

Gulp (www.gulpjs.com) i Grunt (www.gruntjs.com) són eines d'automatització que treballen sota Node.js.

Tot i que es pot programar amb ES5, els desenvolupadors d'Angular recomanen utilitzar TypeScript (una variant de ES6). Així doncs, cal fer servir alguna eina d'automatització com Gulp o Grunt per generar el codi final (que ha d'estar en ES5 per fer que sigui compatible amb els navegadors actuals).

Aquest entorn de treball forma part del que es coneix com a *MEAN stack* (un *stack* és un ‘conjunt de tecnologies’) que consisteix en l’ús de MongoDB (base de dades NoSQL), Express.js (servidor web per a Node.js), Angular a l’entorn client i Node.js com a entorn d’execució al servidor.

Entre les aplicacions webs desenvolupades amb Angular hi ha www.youtube.com, www.freelancer.com, www.telegram.org i www.udemy.com.

Popularitat de les diverses tecnologies

Podeu trobar més informació sobre quines són les tecnologies més populars i quins llocs web les utilitzen en l’enllaç següent:

www.goo.gl/9nkkNN

2.7.2 Ember

Ember (www.emberjs.com) és un altre entorn de treball molt utilitzat, tot i que molt lluny de la popularitat d'Angular. Aquest fa servir un sistema de *convenció sobre configuració*, és a dir, en lloc d'haver de configurar tota l'aplicació, s'han de respectar una sèrie de normes (noms de fitxers, rutes, controladors, etc.) i automàticament són reconeguts.

Un dels inconvenients és que la seva corba d’aprenentatge és més gran que en el cas d'Angular, i un altre és que el sistema de plantilles que utilitza incrusta etiquetes script per tot el document, ja que és així com fa la substitució de codi per continguts de text (tot i que això està previst que canviï en el futur).

Entre les aplicacions webs desenvolupades amb aquest entorn de treball es troben www.twitch.tv, www.digitalocean.com, www.heroku.com i www.sitepoint.com.

2.7.3 React

A diferència d'Angular i Ember, React (facebook.github.io/react) no és un entorn de treball, sinó una biblioteca desenvolupada per Facebook. El seu punt fort és el desenvolupament d’aplicacions d’una sola pàgina (precisament, Facebook el va desenvolupar amb aquest objectiu).

És a dir, en cas de requerir comportaments més complexos (com per exemple l’encaminament de pàgines), s’ha de recórrer a altres biblioteques complementàries. D’altra banda, com que es tracta d’una biblioteca, és molt més fàcil d’integrar en altres projectes o fins i tot combinar-la amb altres entorns de treball.

Entre les aplicacions webs desenvolupades amb aquesta biblioteca hi ha www.facebook.com, www.whatsapp.com, www.imdb.com, www.instagram.com i www.netflix.com.

2.7.4 Components Web

La idea al darrere dels components web és poder crear nous elements de la interfície que siguin reutilitzables i s'integren perfectament al DOM. Aquests components inclouen tant el format com els detectors d'esdeveniments i qualsevol codi necessari per funcionar.

Tot i que els components web actualment no són viables, és important conèixer-ne les característiques i saber com s'han implementat components similars.

Per exemple, seria possible crear un component “FitxaAlumne” que mostrés tota la informació d'un alumne correctament formatada a partir de les seves dades, i que aquest es mostrés al document HTML només afegint-hi l'etiqueta corresponent (similar a <fitxaalumne></fitxaalumne>).

Cal destacar que mentre molts entorns de treball permeten crear elements personalitzats, en inspeccionar el codi es pot comprovar que el que s'ha fet és reemplaçar o embolcallar l'etiqueta original amb tota una sèrie d'elements incrustats per l'entorn de treball per donar format i afegir els detectors d'esdeveniments. En canvi, utilitzant components web només es mostraria l'etiqueta corresponent.

Aquests components consisteixen en l'aplicació de quatre tecnologies diferents; malauradament, no estan disponibles en tots els navegadors d'escriptori i menys encara en els navegadors de dispositius mòbils. Les tecnologies són les següents (totes es consideren experimentals):

- **Custom elements:** capacitat de crear nous elements i etiquetes HTML personalitzats.
- **HTML Templates:** utilització de plantilles directament al codi HTML, que no són mostrades al navegador però es poden afegir via JavaScript.
- **Shadow DOM:** permet encapsular el codi JavaScript i CSS d'un component web per evitar que aquest es barregi amb el de la resta de l'aplicació.
- **HTML Imports:** capacitat d'importar altres fitxers HTML.

Així doncs, per poder utilitzar qualsevol d'aquestes tecnologies és molt probable que hagiu d'activar el mode experimental del vostre navegador i, consegüentment, no es poden utilitzar en aplicacions webs que hagin d'emprar altres usuaris.

Cal destacar que encara que fa anys que s'hi treballa, encara no existeix una definició definitiva i per aquesta raó se n'han fet diferents interpretacions.

Per exemple, Mozilla Firefox no implementarà *HTML Imports* perquè els consideren insegurs. Google va presentar Google I/O 2013 Polymer, la primera versió del seu entorn de treball basat en les tecnologies dels components web. I d'altra

Especificació del W3C dels components web

Podeu trobar tota la informació referent a l'especificació del W3C dels components web al següent enllaç:

www.goo.gl/0OsGNx.

Estat de les tecnologies

Podeu trobar l'estat de les tecnologies utilitzades pels components web als navegadors més populars en l'enllaç següent:

www.goo.gl/5aUF9E.

Google Polymer

Podeu trobar més informació sobre Google Polymer en l'enllaç següent:

www.polymer-project.org/1.0.

banda, els entorns de treball Angular (directrius), Ember (components) i React (components) ofereixen una funcionalitat similar a la dels components web.