DOM

Introdución

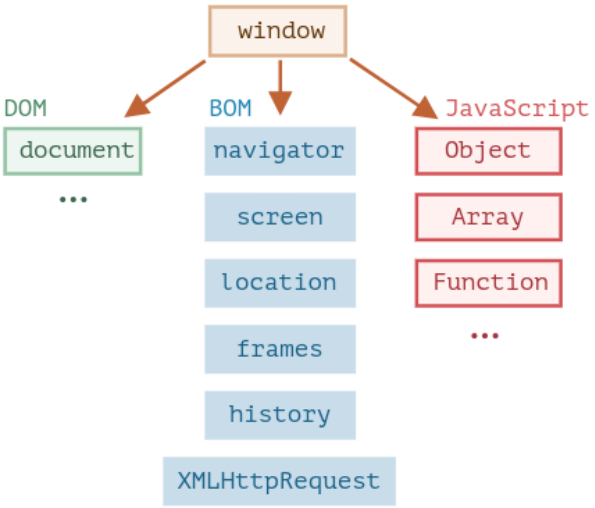
A linguaxe JavaScript foi creada inicialmente para os navegadores web. Hoxe en día pode usarse en diferentes contornas: un navegador, un servidor web ou outro host. Cada unha destas contornas proporciona os seus propios obxectos e funcións adicionais ao núcleo da linguaxe.

Os navegadores son pezas de software formadas por diferentes partes, moitas das cales non poden ser manipuladas dende JavaScript por razóns de seguridade. A seguinte imaxe representa as partes principais dun navegador web:

- **Window**: representa unha ventá que contén un documento DOM. En JavaScript faise referencia a ela usando o obxecto Window. Este obxecto pode usarse, por exemplo, para manipular o documento cargado.
- **Navigator**: representa o estado e identificación do navegador e pode usarse para obter información relativa a el. En JavaScript é representado polo obxecto Navigator.
- **Document**: representa a páxina actual cargada na ventá do navegador. En JavaScript é representado polo obxecto document. Este obxecto pode usarse para manipular a información no HTML e CSS que compón o documento.

Tanto document como navigator están accesibles como propiedades do obxecto global Window.

A continuación móstrase unha representación xerárquica dos elementos de JavaScript cando este se executa nun navegador web.



Observar que hai un obxecto raíz chamado **Window** que é un **obxecto global** que representa a "ventá do navegador" e proporciona métodos para controlala.

O **DOM** (*Document Object Model*) representa todo o contido da páxina,

sendo **document** a raíz da árbore DOM.

O **BOM** (*Browser Object Model*) é o modelo de obxectos do navegador. Proporciona obxectos como **navigator**, que permite acceder a información sobre o navegador (Navigator.userAgent) e os idiomas coñecidos pola persoa usuaria (Navigator.languages), ou o obxecto **location** para acceder á URL actual e cambiala.

DOM

O **DOM** (*Document Object Model*) é unha API[1] para representar e interactuar con calquera documento **HTML** ou **XML**.

O **DOM** (*Document Object Model*) tamén é o modelo de documento que se carga no navegador e que se representa usando unha **estrutura en forma de árbore de nodos**, onde cada nodo representa unha parte do documento (pode ser un elemento, unha cadea de texto ou un comentario). Para cada páxina cargada nunha pestana do navegador créase un DOM.

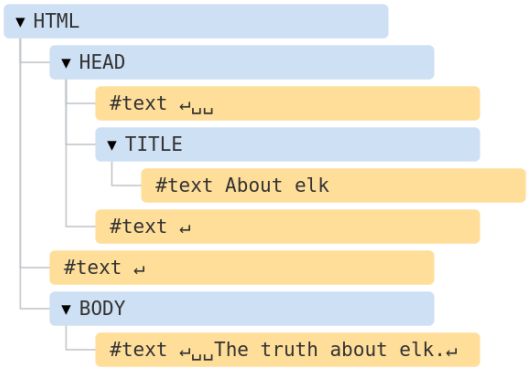
O DOM é unha das APIs máis usadas, pois permite executar código no navegador para acceder e interactuar con calquera nodo do documento. É posible acceder a cada nodo e modificalo, eliminalo ou engadir novos nodos permitindo cambiar dinamicamente a páxina. Cada nodo da árbore é un obxecto coas súas propiedades e métodos.

A raíz da árbore DOM é **document** e deste nodo colgan o resto de elementos HTML. As etiquetas aniñadas son "**fillas**" da etiqueta que as contén.

Todas as propiedades, métodos e eventos dispoñibles para manipular e crear páxinas web están organizados en obxectos: **document**, Node, Element, HTMLTableElement, etc. Todo son obxectos.

O seguinte exemplo mostra unha páxina HTML e o seu DOM.

```
<!DOCTYPE HTML>
<html>
<head>
  <title>About elk</title>
</head>
<body>
  The truth about elk.
</body>
</html>
```



Live DOM Viewer, permite visualizar o DOM correspondente a un código HTML.

[1] Unha API (*Application Programming Interface*) é un conxunto de características e regras dun programa (como unha interface) que describen a forma de interactuar con el.

As etiquetas HTML soen xerar dous nodos: a propia etiqueta e o seu contido.

Os espazos e novas liñas son caracteres totalmente válidos, ao igual que as letras e os díxitos, polo que tamén se converten en nodos de texto e forman parte do DOM.

NOTA: os espazos ao inicio/final da cadea e os nodos de texto que só conteñen espazos habitualmente están ocultos nas **ferramentas de desenvolvemento do navegador**. Tales espazos xeralmente non afectan á forma na que o documento é mostrado.

Cando o navegador encontra **HTML mal escrito** corríxeo automaticamente ao crear o DOM. Por exemplo, se nunha páxina faltan as etiquetas html, head ou body, o navegador engádeas ao crear o DOM. Ás táboas tamén se lles engade a etiqueta tbody se non aparece no código HTML.

A árbore DOM está formada por diferentes tipos de nodos, aínda que os máis habituais son:

- O nodo Document: representa o punto de entrada do DOM
- O nodo Element: representa nodos de elementos (etiquetas HTML).
- O nodo Text: representa nodos de texto.
- O nodo Comment: representa nodos de comentarios.

Recorrendo o DOM

Todas as operacións no DOM comezan co obxecto document, que é a raíz da árbore. O obxecto document ten diferentes propiedades e métodos para traballar cos nodos do DOM:

- document.documentElement: devolve o elemento raíz de document, é dicir, o elemento **<html>** para documentos HTML.
- document.body: devolve o elemento **<body>** de document.

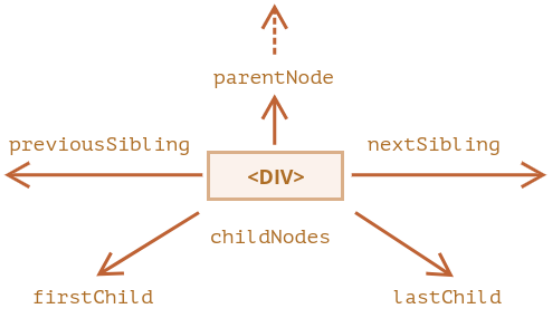
  NOTA: se un script está dentro de <head> e intenta acceder a document.body, este aínda non existe, polo que devolve **null**.

  NOTA: repasar estratexias de carga de scripts.

- document.head: devolve o elemento **<head>** de document.

Dado que o DOM é unha estrutura en forma de árbore xerárquica, hai mecanismos de acceso aos nodos fillos e aos descendentes (inclúe todos os descendentes).

A seguinte imaxe indica como navegar polos nodos do DOM:



Node.firstChild	devolve o Nodo que representa o primeiro fillo directo do nodo, ou null se non ten fillos.
Node.lastChild	devolve o último fillo directo do nodo, ou null se non ten fillos.
Node.nextSibling	devolve o nodo que está a continuación do especificado de entre os childNodes do pai ou null se é o último fillo.
Node.previousSibling	devolve o nodo que está antes do especificado de entre os childNodes do pai ou null se é o primeiro da lista.
Node.childNodes	devolve unha NodeList viva de nodos fillo.
Node.parentNode	devolve o ancestro do nodo especificado. Dado que document é a raíz da árbore, non ten ancestros, polo que document.parentNode devolve null.
Node.hasChildNodes()	devolve un booleano indicando se o nodo ten fillos ou non.

Os nodos fillos (**childNodes**): son os fillos directos, é dicir, os descendentes inmediatos. Por exemplo, <head> e <body> son fillos directos de <html>. Pode accederse aos nodos fillos usando a propiedade Node.childNodes, que devolve un NodeList de nodos fillos directos, incluíndo elementos, texto e comentarios.

NodeList é unha colección **viva** de nodos (obxectos). Dise que a colección está viva porque o seu contido actualízase cada vez que se engade ou elimina un elemento do DOM. Sen embargo, non se poden engadir nin eliminar elementos da colección directamente, para facelo haberá que utilizar os métodos apropiados de modificación do DOM.

Aínda que NodeList non é un Array, é posible iterar sobre a colección usando un bucle **for…of** ou **forEach()**.

NOTA: non usar **for…in** para iterar sobre coleccións como NodeList. O bucle for…in itera sobre as propiedades enumerables, e as coleccións teñen propiedades adicionais sobre as que non interesa iterar.

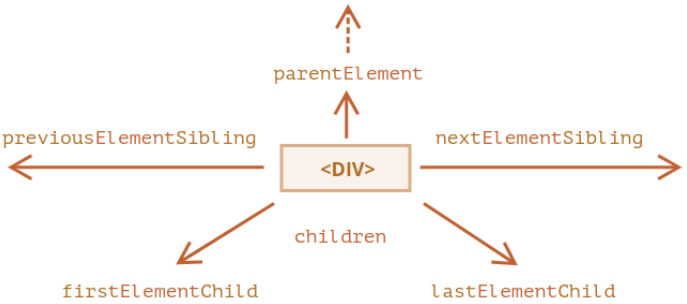
O seguinte script mostra os nodos fillos de body dun documento:

```
for (const node of document.body.childNodes) {
  console.log(node);
}
```

NOTA: **console.log** mostra a árbore DOM do elemento e **console.dir** mostra o elemento como un obxecto do DOM, o que resulta útil para explorar as súas propiedades.

As propiedades de navegación de Node refírense a todos os nodos, é dicir, inclúen nodos de elementos, texto e comentarios. Sen embargo, algunhas veces quérese traballar só cos nodos que representan **etiquetas**, polo que haberá que usar métodos de navegación de Element.

A seguinte imaxe amosa como navegar polos nodos que son elementos:



Element.firstElementChild	devolve o primeiro elemento fillo dun elemento, ou null.
Element.lastElementChild	devolve o último elemento fillo dun elemento ou null.
Element.nextElementSibling	devolve o seguinte elemento.
Element.previousElementSibling	obtén o elemento anterior.
Element.children	devolve unha colección viva que contén todos os fillos que son elementos.
Node.parentElement	devolve o elemento pai do nodo especificado, ou null se non ten pai ou se non é un elemento

NOTA: pode accederse aos elementos das coleccións (HTMLCollection) usando a propiedade ou o índice:

```
let fillos = document.body.children;

console.log(document.body.childNodes);
console.log(fillos[0]);
```

```
console.log(fillos.item(0));
// considerando que hai un elemento con id="identificador"
console.log(fillos['identificador']);

// lonxitude da colección
console.log(fillos.length);
```

A propiedade Node.parentElement devolve o elemento pai do nodo especificado, ou null se non ten pai ou se non é un elemento. ¿Pode o elemento pai dun elemento non ser un elemento?

```
console.log(document.documentElement.parentNode);
console.log(document.documentElement.parentElement);
```

A propiedade parentElement devolve o "elemento pai", mentres que parentNode devolve "calquera nodo" pai.

Recorrer táboas

Hai certos elementos do DOM que teñen propiedades específicas para recorrelos, por exemplo, as táboas:

Propiedades dispoñibles para o elemento táboa (HTMLTableElement):

- HTMLTableElement.tHead: devolve o primeiro <thead> da táboa.
- HTMLTableElement.tFoot: devolve o primeiro <tfoot> da táboa.
- HTMLTableElement.tBodies: devolve unha colección viva dos <tbody> da táboa.
- HTMLTableElement.rows: devolve unha colección viva das filas da táboa, incluíndo as filas de <thead>, <tfoot> e <tbody>.

  Tamén é posible acceder ás filas de tHead, tFoot e tBodies usando a propiedade rows, por exemplo, **table.tHead.rows**.

Propiedades dispoñibles para as filas (HTMLTableRowElement):

- **HTMLTableRowElement.cells**: devolve unha colección viva de celas da fila.
- **HTMLTableRowElement.rowIndex**: devolve a posición da fila no conxunto da táboa.
- **HTMLTableRowElement.sectionRowIndex**: devolve a posición da fila dentro da sección thead/tbody/tfoot.

Propiedades dispoñibles para as celas (HTMLTableCellElement):

- **HTMLTableCellElement.cellIndex**: devolve a posición da cela na coleción de <tr>

Exercicios:

1. Dado o seguinte código HTML:

```
<html>
<body>
  <div>Users:</div>
  <ul>
    <li>John</li>
    <li>Pete</li>
  </ul>
</body>
</html>
```

   Indica, polo menos, unha forma de acceder aos seguintes nodos:
   - o nodo
   - o nodo
   - o segundo

2. Dado un elemento calquera dunha árbore DOM
   a. ¿É certo que **elemento.lastChild.nextSibling** é sempre **null**?
   b. ¿É certo que **elemento.children[0].previousSibling** é sempre **null**?

3. Escribe o código para pintar as celas diagonais dunha táboa de vermello.

   Investiga en internet como cambiar a cor de fondo dunha cela mediante JavaScript

   A táboa debería quedar similar a esta:

1:1	2:1	3:1	4:1	5:1
1:2	2:2	3:2	4:2	5:2
1:3	2:3	3:3	4:3	5:3
1:4	2:4	3:4	4:4	5:4
1:5	2:5	3:5	4:5	5:5

Acceso a nodos

No apartado anterior víronse diferentes formas de recorrer a árbore DOM. Tamén existen utilidades para buscar e acceder directamente a un nodo concreto.

- document.getElementById(id): devolve o elemento co id especificado no parámetro. Dado que os IDs son únicos, é unha forma útil de acceder a un elemento específico rapidamente.

- document.querySelector(selector): devolve o primeiro elemento do documento co selector especificado. O selector pasado como parámetro debe ser un selector CSS válido.

  Element.querySelector(selector): devolve o primeiro elemento descendente do elemento dende o que se invoca o método e que coincide co selector especificado como parámetro.

- document.querySelectorAll(selector): devolve un NodeList estático (colección de nodos) de elementos que coinciden co selector especificado.
- document.getElementsByTagName(tagName): devolve unha colección viva de elementos coa etiqueta indicada como parámetro. Este método pode ser invocado no obxecto document ou en calquera elemento.
- document.getElementsByClassName(): devolve unha colección viva de todos os elementos fillos co nome de clase especificada. Este método pode ser invocado no obxecto document ou en calquera outro elemento.
- document.getElementsByName(name): devolve un NodeList de elementos que teñan o atributo “name” co valor especificado como parámetro.

Exercicios:

1. Descarga o código fonte 02-accesoNodos01.html e indica, polo menos, unha forma de acceder aos seguintes nodos:

   - A táboa con id="age-table".
   - Todos os elementos label dentro da táboa (debería haber 3).
   - O primeiro td da táboa.
   - O form con name="search".
   - O primeiro input do formulario anterior.
   - O último input do formulario do exercicio anterior.

2. Descarga o código fonte 02-accesoNodos02.html e indica, polo menos, unha forma de acceder aos seguintes nodos:

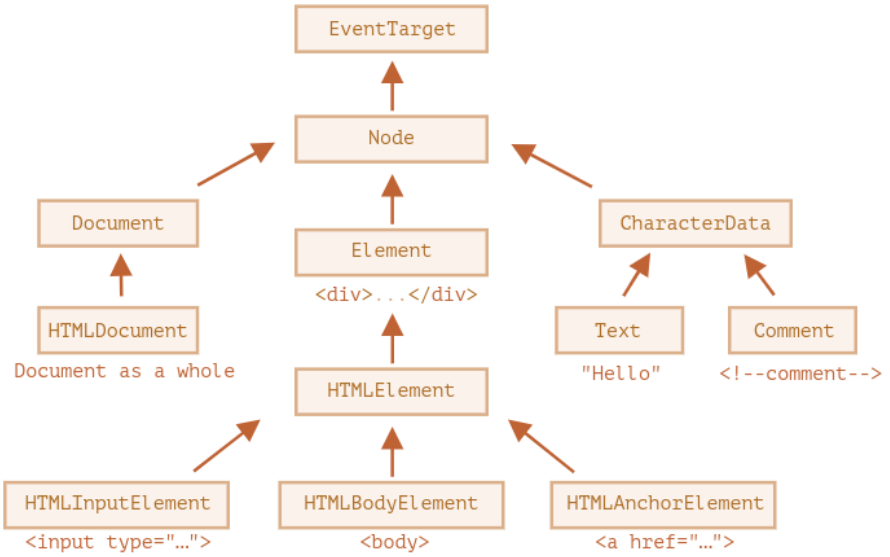
   - O elemento con id “input2”.
   - A colección de parágrafos
   - Todos os parágrafos dentro do div con id “lipsum”.
   - O formulario
   - Todos os inputs
   - Só os inputs con nome “sexo”.
   - Os items da lista con clase “important”.

Propiedades dun nodo

Dado que existen diferentes tipos de nodos, implementados por obxectos diferentes en JavaScript, cada un terá propiedades e características diferentes. Así, por exemplo, unha

etiqueta <a> terá propiedades relacionadas coa ligazón e un <input> terá propiedades relacionadas coa caixa de entrada.

A seguinte imaxe mostra a xerarquía obxectos de Nodos, onde a raíz da árbore é EventTarget.



A estrutura xerárquica anterior está formada por diferentes clases de obxectos:

- EventTarget: clase “abstracta” que é a raíz da árbore. Non se crean obxectos desta clase senón que se utiliza de base por todos os nodos para soportar os “eventos”.
- Node: clase abstracta que serve de base para os nodos DOM. Proporciona funcionalidade para navegar polos nodos do DOM (firstChild, lastChild, parentNode, …). As clases que herdan dela, herdan estas funcionalidades.
- Document: é o documento. O obxecto document serve como punto de entrada do DOM.
- CharacterData: clase abstracta herdada por:
    - Text: clase correspondente ao texto dentro dos elementos, por exemplo “Hello” en Hello
    - Comment: clase para os comentarios.
- Element: clase base para elementos do DOM. Proporciona métodos de navegación (firstElementChild, lastElementChild, parentElement, …) e métodos para buscar, como querySelector.

  O DOM pode usarse para documentos HTML, XML e SVG, polo que Element serve de base para clases máis específicas como SVGElement, XMLElement e HTMLElement.

- HTMLElement: clase base para todos os elementos HTML. Algunhas das súas subclases son: HTMLInputElement, HTMLBodyElement, HTMLAnchorElement, …

Hai moitos outros elementos con clases específicas, aínda que non todos a teñen. Por exemplo, o elemento non ten clase específica, polo que deriva directamente de HTMLElement.

O conxunto completo de propiedades e métodos dun nodo obtense como resultado da herdanza de varias clases.

A clase dun nodo está almacenada na propiedade **constructor.name**:

console.log(document.body.constructor.name);

As principales propiedades dun nodo son:

- Element.innerHTML: permite ler/establecer o HTML contido no elemento.

```
// se elemento é o nodo “<p>Esta páxina é <strong>simple</strong></p>
console.log(elemento.innterHTML); // Esta páxina é <strong>simple</strong>
```

- Node.textContent: devolve só texto, sen etiquetas. O textContent é o contido de todo o que hai entre a etiqueta de apertura e a de cerre do elemento, incluíndo o contido de todos os descendentes. É dicir, devolve unha concatenación do textContent de todos os descendentes, excluíndo comentarios. As etiquetas descendentes non se inclúen na saída.

```
// elemento é o nodo “<p>Esta páxina é <strong>simple</strong></p>
console.log(elemento.textContent); // Esta páxina é simple
```

- HTMLElement.innerText: representa o texto dun nodo e os seus descendentes. Aproxímase ao texto que se obtería se seleccionásemos co rato un texto e se copiase no portapapeis.

  NOTA: consultar as diferencias entre textContent e innerText.

- .**value**: devolve o valor da propiedade “value” dun <input> (HTMLInputElement). Como os <inputs> non teñen etiqueta de cerre, non se pode usar .innerHTML nin .textContent.

```
<input type="text" id="elem" value="value">
```

- HTMLAnchorElement.href: devolve a cadea que contén a URL dunha etiqueta
- Element.id: devolve o valor do atributo “id” dun elemento

Os nodos teñen outras moitas propiedades. Ademais, **a maioría dos atributos HTML estándar convértense automaticamente en propiedades do obxecto DOM correspondente**. Por exemplo, se a etiqueta é <body id=”page”>, entón o obxecto DOM ten unha propiedade **body.id**=”page”.

Todos os atributos, tanto os que son estándar como os que non o son, están accesibles usando os seguintes métodos:

- Element.hasAttribute(name): comproba se un elemento ten o atributo especificado.
- Element.getAttribute(name): obtén o valor do atributo especificado.
- Element.setAttribute(name, valor): establece o valor do atributo.
- Element.removeAttribute(nombre): elimina o atributo.
- Element.attributes: devolve unha colección viva de pares chave/valor de todos os atributos rexistrados.

Os atributos non estándar, creados polas persoas desenvolvedoras, tamén son accesibles usando os métodos anteriores.

```
<div class="order" order-state="cancelado">...</div>
```

NOTA: usar atributos non estándar pode dar lugar a problemas se no futuro HTML decide estandarizar ese atributo. Para evitar este problema, existen os atributos data-*.

Todos os atributos que comecen por “data-” están reservados para o seu uso polas persoas desenvolvedoras e están dispoñibles na propiedade **HTMLElement.dataset.** Así, se un elemento ten un atributo chamado “data-about”, estará dispoñible dende a propiedade elemento.dataset.about:

```
<div id="user" data-id="1234567890" data-user="carinaanand" data-date-of-birth>...</div>

…
const el = document.querySelector('#user');

// el.id === 'user'
// el.dataset.id === '1234567890'
// el.dataset.user === 'carinaanand'
// os atributos de varias palabras (data-date-of-birth) convértense en camelCase:
// el.dataset.dateOfBirth === ''
```

Exercicios:

1. Descarga o código fonte 03-propiedadesNodo01.html e indica, polo menos, unha forma de acceder ao seguinte contido:
    - O innerHTML da etiqueta “Escolle sexo”:
    - O textContent da etiqueta anterior
    - O valor do primeiro input de sexo
    - O valor do sexo que estea seleccionado.
    - O texto de cada un dos elementos
    - Indica cantos elementos <li> hai.
    - Indica o valor do atributo data-widget-name

2. Descarga o código fonte 03-propiedadesNodo02.html e indica:
    - O número de ligazóns da páxina.
    - A dirección da penúltima ligazón.
    - O número de ligazóns que apuntan a http://proba
    - O número de ligazóns do terceiro parágrafo.
    - Fai que as ligazóns que apuntan a http://proba teñan o texto de cor laranxa.

Modificando o documento

É posible crear novos elementos e modificar o contido existente da páxina para crear dinamismo.

Para crear novos elementos utilízanse os métodos:

- Document.createElement(etiqueta): crea un elemento HTML do tipo especificado no parámetro.

```
const newDiv = document.createElement("div");
newDiv.innerHTML = "<strong>Ola</strong>";
```

- Document.createTextNode(data): crea un nodo de texto.

```
const newtext = document.createTextNode(text);
```

Métodos para inserir elementos na árbore DOM:

- Element.append(paramametros): insire un conxunto de nodos ou cadeas despois do último fillo de Element. As cadeas son inseridas como nodos de texto.

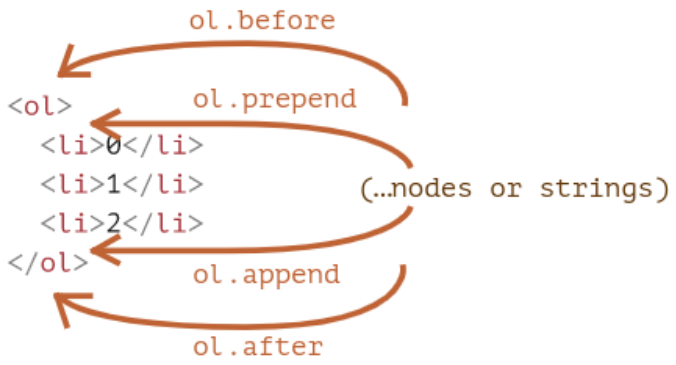
```
let div = document.createElement("div");
let p = document.createElement("p");
div.append(p);
div.append("Some text")

console.log(div.childNodes) // NodeList [ p, text ]
```

- Element.prepend(parametros): insire un conxunto de nodos ou cadeas antes do primeiro fillo de Element. As cadeas son inseridas como nodos de texto.
- Element.before(parametros): insire un conxunto de nodos ou cadeas na lista de fillos do pai de Element, xusto antes deste Element. As cadeas son inseridas como nodos de texto.
- Element.after(parametros): insire un conxunto de nodos ou cadeas na lista de fillos do pai de Element, xusto despois de Element. As cadeas son inseridas como nodos de texto.

- Element.replaceWith(parametros): substitúe o Elemento por un conxunto de nodos ou cadeas. As cadeas son inseridas como nodos de texto.

A continuación pode verse visualmente os puntos de inserción dos métodos anteriores:



Exemplo:

<ol id="ol">
  0
  1
  2
 	
	let ol = document.getElementById("ol");
	// insire "before" antes de
	ol.before("before");
	// insire "after" despois de
	ol.after("after");
	
	let liFirst = document.createElement("li");
	liFirst.innerHTML = "prepend";
	// insire liFirst ao principio de
	ol.prepend(liFirst);
	
	let liLast = document.createElement("li");
	liLast.innerHTML = "append";
	// insire liLast ao final de
	ol.append(liLast);
		before
		<ol id="ol">
		** - prepend **
		  0
		  1
		  2
		** - append **
		
		after

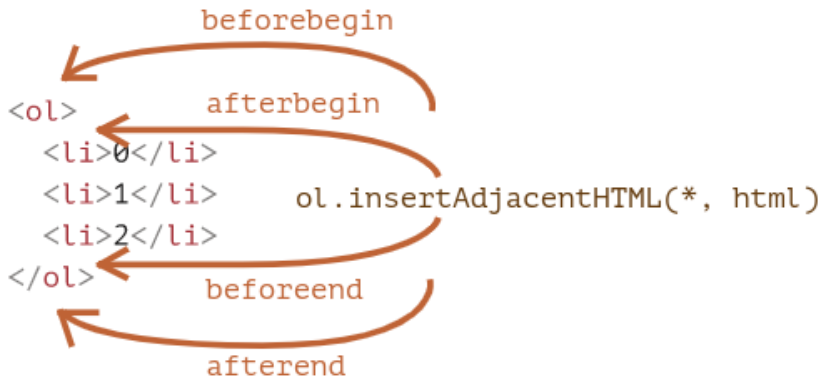
Tamén é posible inserir HTML:

- Element.insertAdjacentHTML(position, html): parsea o código HTML e insire os nodos resultado na árbore DOM na posición indicada.

  O parámetro posición debe ser unha das seguintes cadeas:

  - “beforebegin”: antes do elemento.
  - “afterbegin”: dentro do elemento, antes do primeiro fillo.
  - “beforeend”: dentro do elemento, despois do último fillo.
  - “afterend”: despois do elemento.

A continuación poden verse visualmente, as popsicións onde se inserirá o HTML.



Esta imaxe é similar á anterior. Os puntos de inserción son os mesmos, aínda que neste caso estase inserindo HTML.

Existen versións equivalentes deste método para inserir texto (Element.insertAdjacentText()) e elementos (Element.insertAdjacentElement()).

Para eliminar elementos do DOM pode usarse o método Element.remove().

Se o que se quere é mover un nodo dun lugar a outro do DOM, non hai necesidade de eliminalo. Todos os métodos de inserción quitan automaticamente o nodo do lugar vello.

```
<div id="first">Primero</div>
<div id="second">Segundo</div>
<script>
  // non hai necesidade de chamar a "remove"
  second.after(first); // toma #second e despois insire #first
</script>
```

Os métodos vistos ata agora son métodos modernos e flexibles, xa que permiten engadir múltiples elementos á vez. Antigamente usábanse outros métodos que se van listar aquí para poder entender scripts antigos:

- Node.appendChild(child): engade un nodo ao final da lista de fillos do nodo especificado.
- Node.insertBefore(newNode, referenceNode): insire newNodo como fillo de Node e antes do nodo especificado como referencia. Exemplo:

  parentElem.insertBefore(newNode, nextSibling)

- Node.replaceChild(newChild, oldChild): substitúe un nodo fillo do pai especificado.
- Node.removeChild(child): elimina un fillo da árbore DOM.

Clonado de nodos

É posible clonar nodos utilizando o método Node.cloneNode(). Este método devolve un duplicado do nodo dende o que se invoca.

Pódese especificar un parámetro para indicar se tamén se clona a subárbore. Se o parámetro é true clónase toda a subárbore e se é falso só se clona o nodo.

NOTA: cloneNode pode dar lugar a **IDs duplicados** no DOM. Se un nodo ten id, haberá que modificar o valor na copia.

Exercicios:

1. Imaxinar que a variable **elemento** fai referencia a un elemento do DOM e text é unha variable con unha cadea de texto que inclúe etiquetas HTML. ¿Cales dos seguintes comandos farán exactamente o mesmo?:
    a. elemento.append(document.createTextNode(text));
    b. elemento.innerHTML = text;
    c. elemento.textContent = text;
2. Dada unha lista <ol> con varios elementos <li>, crea o código necesario para eliminar os <li> da lista.
3. Dado o seguinte código, ¿por que segue aparecendo o texto despois de borrar a táboa?

<table id="taboa"> Texto <tr> <td>Test</td> </tr> </table>	let taboa = document.getElementById("taboa"); taboa.remove();

4. Crea un documento HTML que conteña un elemento <ul>. Dende JavaScript crea 4 elementos <li> e engádeos á lista <ul>, de tal forma que sexan visibles no navegador.
5. Escribe o código JavaScript para inserir “
2
3” entre os dous <li> seguintes:

```
<ul id="listaULExercicio5">
  <li id="one">1</li>
  <li id="two">4</li>
</ul>
```

6. Dado un obxecto como o seguinte:

```
let arbore = {
  "Fish": {
    "trout": {},
    "salmon": {}
  },

  "Tree": {
    "Huge": {
      "sequoia": {},
      "oak": {}
    },
    "Flowering": {
      "apple tree": {},
      "magnolia": {}
    }
  }
};
```

- Fish
  - trout
  - salmon
- Tree
  - Huge
    - sequoia
    - oak
  - Flowering
    - apple tree
    - magnolia

Crea unha función **createTree(data)** que devolva unha lista ul/li coma a da imaxe da dereita, para os datos proporcionados.

Para realizalo hai dúas posibilidades, aínda que sería bo que intentases as dúas opcións.

- Crear o código código HTML
- Crear os nodos da árbore.

7. Escribe unha función **crearCalendario(elemento, ano, mes)** que engada ao elemento pasado como parámetro un calendario do ano e mes indicados.

O calendario debe ser unha táboa, onde cada semana é un <tr> e cada día un <td>. A cabeceira da táboa está creada con <th>.

Por exemplo, o calendario resultado de chamar á función cos seguintes parámetros vese na imaxe seguinte. Observar que se aplicaron estilos CSS para mellorar o aspecto.

crearCalendario(calendario, 2022, 11);

L	M	Me	X	V	S	D
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

8. Ordena a seguinte táboa pola columna “Nome”. Ten en conta que a táboa pode ter máis filas.

```
<table id="taboaOrdenar">
   <thead>
    <tr>
      <th>Nome</th>
      <th>Apelido</th>
      <th>Idade</th>
    </tr>
   </thead>
   <tbody>
    <tr>
      <td>John</td>
      <td>Smith</td>
      <td>10</td>
    </tr>
    <tr>
      <td>Pete</td>
      <td>Brown</td>
      <td>15</td>
    </tr>
    <tr>
      <td>Ann</td>
      <td>Lee</td>
      <td>5</td>
    </tr>
   </tbody>
 </table>
```

9. Dada unha lista como a seguinte, escribe o código que engada o número de descendentes.

```
<ul id="listaAnimais">
  <li>
    Animals
    <ul>
      <li>
        Mammals
        <ul>
          <li>Cows</li>
          <li>Donkeys</li>
          <li>Dogs</li>
          <li>Tigers</li>
        </ul>
      </li>
      <li>
        Other
        <ul>
          <li>Snakes</li>
          <li>Birds</li>
          <li>Lizards</li>
        </ul>
      </li>
    </ul>
  </li>
  <li>
    Fishes
    <ul>
      <li>
        Aquarium
        <ul>
          <li>Guppy</li>
          <li>Angelfish</li>
        </ul>
      </li>
      <li>
        Sea
        <ul>
          <li>Sea trout</li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

- Animals [2]
  - Mammals [4]
    - Cows
    - Donkeys
    - Dogs
    - Tigers
  - Other [3]
    - Snakes
    - Birds
    - Lizards
- Fishes [2]
  - Aquarium [2]
    - Guppy
    - Angelfish
  - Sea [1]
    - Sea trout

Estilos e clases

Antes de aprender como se manexan en JavaScript as clases e estilos, convén recordar que hai dúas formas de aplicar estilos a un elemento:

- Crear unha **clase** CSS e engadila ao elemento: ** .** Versión recomendada.
- Escribir as propiedades directamente en style: ** **

Dende JavaScript pódense modificar os estilos CSS independentemente da forma usada para aplicalos.

Element.className é unha propiedade que permite ler/establecer o valor do atributo "class" do elemento.

NOTA: utilízase **className** como nome da propiedade en lugar de **class** por conflitos coa palabra reservada "class" en linguaxes que se utilizan para manipular o DOM.

```
<body class="main page">
  <script>
    console.log(document.body.className); // main page
  </script>
</body>
```

Observar que **className** obtén a lista de clases. Se se modifica o **className** modifícase toda a lista de clases.

Ás veces interesa agregar ou eliminar só unha clase, polo que é mellor utilizar a propiedade **Element.classList**.

Element.classList é uha propiedade de só lectura que devolve unha lista viva (DOMTokenList) cos atributos class do elemento. Aínda que a propiedade classList é de só lectura, é posible modificala usando os métodos add(), remove(), replace() e toggle().

```
<body class="main page">
  <script>
    document.body.classList.add('article');
    console.log(document.body.className); // main page article
  </script>
</body>
```

Métodos para modificar a DOMTokenList:

- DOMTokenList.add(tokens): engade os tokens á lista, omitindo os que están duplicados.
- DOMTokenList.remove(tokens): elimina os tokens especificados da lista.

- DOMTokenList.replace(oldToken, newToken): substitúe un token existente por outro novo.
- DOMTokenList.toggle(token): se o token está na lista elimínao e devolve **false**. Se o token non está na lista, engádeo e devolve **true**. O valor devolto indica se o token está na lista despois de executar o método.
- DOMTokenList.contains(token): devolve true se a lista contén o token e false en caso contrario.

A propiedade HTMLElement.style é un obxecto de tipo CSSStyleDeclaration, que corresponde ao texto escrito no atributo "style" do elemento. É posible ler/modificar as propiedades CSS accedendo directamente ás propiedades coa notación camelCase:

```
element.style.backgroundColor = "red"
```

A propiedade style só permite acceder aos estilos en liña asignados directamente a un elemento. Sen embargo, os estilos efectivos aplicados a un elemento son a consecuencia da aplicación de todas as regras CSS e herdanza. Para acceder aos estilos aplicados debe usarse Window.getComputedStyle()

```
<html lang="en">
  <head>
    <style> body { color: red; margin: 5px; font: 2rem/2 sans-serif;} </style>
  </head>
  <body>
    <script>
      console.log(document.body.style.color);
      console.log(document.body.style.marginTop);

      let computedStyle = getComputedStyle(document.body);
      console.log(computedStyle.marginTop);
      console.log(computedStyle.color);
      console.log(computedStyle.fontSize);
    </script>
  </body>
</html>
```

NOTAS:

- observar o valor devolto pola propiedade **computedStyle.fontSize** e comparalo co estilo CSS asignado. Fixarse que se fai o cálculo e asígnaselle o tamaño en px.
- getComputedStyle ten os valores de propiedades CSS con nomes longos (non usar os nomes abreviados). Por exemplo, a propiedade CSS **font** é o nome abreviado para propiedades como font-style, font-weight, font-size/line-height, font-family, … Sempre se debe utilizar a propiedade exacta. Usar os nomes abreviados non está estandarizado, polo que poden obterse resultados diferentes en distintos navegadores.

Referencias

Para a elaboración deste material utilizáronse, entre outros, os recursos que se enumeran a continuación:

- JavaScript | MDN
- Introduction to the DOM - Web APIs | MDN
- Client-side web APIs - Learn web development | MDN
- JavaScript Tutorial - w3schools
- Desarrollo Web en Entorno Cliente | materials
- Javascript en español - Lenguaje JS
- The Modern JavaScript Tutorial
- Eloquent JavaScript