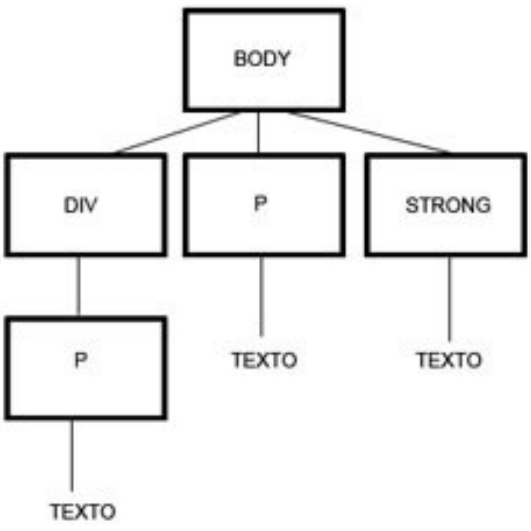
Introducción al DOM 11

Cualquier página web que se precie esta llena de diferentes elementos: párrafos, enlaces, contenedores... Todos estos elementos presentes en una web van formar una estructura informática denominada 'árbol' llamada DOM: Document Object Model.

Un árbol es una estructura informática usada para representar datos que está compuesta por nodos (los diferentes elementos web) y que se interconectan entre sí imitando la forma de un árbol o matorral invertido (arriba la raíz y el árbol se expande hacia abajo).

Con Javascript vamos a poder acceder de diferentes formas a nodos de ese árbol (etiquetas de mi página) con objeto de alterar sus atributos HTML o sus propiedades CSS. Haciendo esto vamos a poder crear páginas con elementos interactivos y dinámicos que van a cambiar o alterar su comportamiento cuando el visitante lo decida.




```
<body>
  <div>
    <p>
      texto
    </p>
  </div>
  <p>
    texto
  </p>
  <strong>
     texto
  </strong>
</div
```

11.1 Accediendo correctamente al DOM

Antes de aprender cómo modificar los diferentes elementos del árbol DOM de una página es fundamental saber dónde colocar nuestro código, dado que de no hacerlo correctamente, lo más normal es que no se obtengan los resultados esperados.

Lo primero es saber que el árbol DOM se crea una vez que se han terminado de cargar todos los elementos de la página.
Sin embargo, lo normal es que el Javascript se cargue antes de que lo haga toda la página (hay elementos que tardan más en cargarse como fotos, archivos multimedia...). ¿Qué ocurre si sucede esto? Pues que el Javascript va a intentar acceder a un árbol DOM que aún no se ha creado y, por tanto, no podrá hacer su trabajo correctamente y dará un error.

Para solucionar esto debemos forzar al Javascript a que espere a que el árbol DOM esté creado. Para ello tenemos varias formas:

1. Colocar el código Javascript al final de la página.
2. Haciendo uso de los eventos `load` o `DOMContentLoaded`.
3. Haciendo uso de herramientas de librerías externas (pej, JQuery)

Quitando la última forma (que no se va a explicar en este manual), las dos primeras formas son sencillas de aplicar:

La primera forma es simplemente colocar todo el código usado para alterar el árbol DOM al final de la página:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Mi pagina</title>
    <script>
        //codigo Javascript que no toca el DOM
    </script>
```

```
</head>
<body>
   elementos HTML de mi pagina
</body>
</html>
<script>
        //código que accede al arbol DOM
</script>
```

Sin embargo, en proyectos de gran complejidad, la mayoría de las veces esto no es posible y, aparte, nos hace colocar código Javascript en otra zona que no es la cabecera y esto, hoy día, no es elegante.

La **segunda forma** es mucho más recomendable: es usar algún evento que nos indique cuando se ha cargado el árbol DOM. En el siguiente capítulo veremos en profundidad los eventos. No obstante, podemos adelantar un evento que nos será muy útil en este punto: El evento `load`

Este evento espera a que todos los elementos de la página se carguen y posteriormente lanza el Javascript que tiene asociado y lo ejecuta.

La forma de actuar es muy sencilla. Basta con colocar el manejador de dicho evento (en el próximo capitulo veremos que es eso) y dentro el código que se encargará de tocar los nodos del DOM

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Mi pagina</title>

    <script>
        //codigo Javascript que no toca el DOM
```

```
        window.onload = function(){
            //codigo que accede al DOM
        }
    </script>
</head>
<body>
   elementos HTML de la pagina
</body>
</html>
```

Nota: Aunque todavía se suele usar este evento para estos casos, es mucho mejor usar el evento `DOMContentLoaded` que, como su nombre indica, espera a que se cargue el árbol DOM para lanzar su código Javascript asociado. Esto es mucho más eficiente a esperar que se cargue toda la pagina (imágenes y contenido multimedia incluidos) como hace el evento `load`.

Sin embargo, `DOMContentLoaded` solo se puede usar con funciones asociadoras de eventos y eso es algo que se explica en la parte avanzada de este manual, por lo que, de momento, debemos usar el evento `load`.

11.2 Acceso a nodos

Cuando creamos una página o aplicación web usando HTML, automáticamente se va a crear el árbol DOM asociado. Como se ha indicado antes, un nodo del árbol DOM no es más que un elemento HTML de la estructura de la página/aplicación.

Con Javascript vamos a poder acceder directamente a cualquier elemento o un grupo de elementos del árbol DOM de nuestra pagina/aplicación y guardar esa referencia en una variable. De esta forma vamos a poder editar/añadir/quitar propiedades HTML y CSS a nuestro gusto e incluso vamos a poder crear y/o eliminar nodos del árbol a nuestro gusto.

Lo primero que debemos hacer cuando queremos "tocar" nodos del árbol DOM es conseguir una referencia al nodo o grupo de nodos que queremos. Para esto, Javascript nos proporciona las llamadas **funciones especializadas**:

11.2.1 getElementsByTagName

La función `document.getElementsByTagName(nombre_de_una_etiqueta)` obtiene todos los elementos de nuestra página que cuya etiqueta sea la misma que la que indicamos como parámetro. La función devuelve un array[8] con los elementos encontrados. En cada celda de ese array tenemos una referencia al elemento correspondiente (el orden será el orden de aparición en el código).

```
var parrafos = document.getElementsByTagName("p")
```

Con la línea de código anterior vamos a obtener un array con todos los párrafos de la página y dicho array lo almacenamos en la variable `parrafos`. En la posición 0 del array tendremos una referencia al primer párrafo que aparece en el código HTML de la página, en la posición 1, al siguiente párrafo...

¿Y que es eso de "una referencia"? En realidad, una referencia no es más que un objeto (no un número, ni una cadena, ni un booleano). Como nosotros aún no dominamos ese concepto de programación, vamos a suponer que es 'indicador' (un 'puntero') que apunta al elemento correspondiente de nuestra página.

Así pues, el array almacenado en `parrafos` va a contener punteros que apuntan a los párrafos de la página.

11.2.2 getElementsByName

La sintaxis es: `document.getElementsByName(nombre)`.

Esta función nos selecciona todos aquellos elementos cuyo atributo `name` sea el mismo que el indicado como parámetro.

El lector debería saber que el valor del atributo `name` normalmente es único para cada elemento de nuestra página (salvo para los botones radio del mismo conjunto). Por tanto

8 En realidad, estas funciones devuelven una estructura llama `HTMLCollection`, pero como son muy similares a los array, nosotros vamos a suponer que son arrays (aunque no es correcto).

la mayoría de las veces vamos a obtener un 'puntero' al elemento con el nombre indicado.

```
var elemento = document.getElementsByName("especial");


. . .
<input type='text' name='especial' … >
```

Con el código anterior vamos a tener en `elemento` una referencia a la caja de texto con nombre `especial`.

11.2.3 getElementsByClassName

La sintaxis es: `document.getElementsByClassName(nombre_de_clase).`

Esta función nos selecciona todos aquellos elementos que pertenezcan a la clase CSS especificada entre paréntesis. Es decir, cuyo atributo `class` sea el mismo que el indicado como parámetro.

Como las clases se usan para crear agrupaciones, esta función siempre va a devolver una array (incluso cuando la clase solo tenga un elemento).

```
var elementos = document.getElementsByClassName("Emenu");
. . .
<nav>
  <a href='#' class='Emenu'> Inicio </a>
  <a href='#' class='Emenu'> Nosotros </a>
  <a href='#' class='Emenu'> Proyectos </a>
  <a href='#' class='Emenu'> Productos </a>
  <a href='#' class='Emenu'> Contacto </a>
</nav>
```

Con el código anterior vamos a tener en `elementos` un array de referencias a todos los elementos de la clase `Emenu`.

11.2.4 getElementById

Su sintaxis es: `document.getElementByName(id).`

Posiblemente, esta es la forma más común de acceder a un nodo del árbol DOM. Esta función nos selecciona aquel elemento cuyo atributo `id` coincide con el indicado como parámetro.

```
var elemento = document.getElementById("yo");
. . .
<div id='pepe'>
  <p name='trozo'> ... </p>
  <a href='#' id='yo'> Pulsa aquí </a>
</div>
```

Con el código anterior vamos a obtener una referencia al enlace dado que es el elemento con `id='yo'`

El lector debería saber que el valor del atributo `id` debe ser único para cada elemento de mi página, por tanto esta función siempre devolverá una referencia a un elemento.

Aparte de las funciones especializadas vistas en los apartados anteriores, a partir de la versión 6, Javascript proporciona otras dos herramientas para acceder a nodos del árbol DOM. A estas herramientas se las conoce como **funciones base.** Estas funciones nos permiten seleccionar nodos del árbol usando cualquiera de los selectores de CSS actuales, siendo, por tanto, mucho más versátiles que las funciones especializadas.

11.2.5 querySelector

Su sintaxis es: `querySelector(selector_css).`

Devuelve una referencia al primer elemento que cumpla con el criterio dado por el selector que va entre paréntesis.

```
var elemento = querySelector("div p");
```

```
. . .
<div id='pepe'>
  <p> primero </p>
  <p> segundo </p>
  <p> tercero </p>
</div>
```

Con el código anterior, en la variable `elemento` obtenemos una referencia al primer párrafo dentro de un div (en el código será `primero`).

Recuerda: `querySelector` siempre devuelve una referencia. Incluso cuando lo usamos con selectores que afectan a un grupo de elementos. Ej: `elemento = querySelector(".Emenu");`

11.2.6 querySelectorAll

Su sintaxis es: `querySelectorAll(selector_css).`

Devuelve un array[9] con referencias a todos los elementos que cumplas con el criterio dado por el selector que va entre paréntesis.

```
var elementos = querySelector("div p");
. . .
<div id='pepe'>
  <p> primero </p>
  <p> segundo </p>
  <p> tercero </p>
</div>
```

Con el código anterior, en la variable `elementos` obtenemos un array con referencias a todos los párrafos párrafo dentro de un div.

9 En realidad, estas funciones devuelven una estructura llama `NodeList`, pero como son muy similares a los array, nosotros vamos a suponer que son arrays (aunque no es correcto).

Recuerda: `querySelector` siempre devuelve un array de referencias. Incluso cuando lo usamos con selectores que afectan a un solo elemento. Ej: `elementos = querySelectorAll("#pepe");`

Ya hemos aprendido varias formas de acceder a distintos elementos de nuestra página. Como se ha explicado, con estas funciones obtenemos referencias que apuntan a los elementos deseados. La pregunta lógica que se puede hacer todo lector es ¿y ahora qué? ¿qué hago con esas referencias?

Obtener esas referencias es el primer paso para poder alterar los elementos de mi página. A continuación vamos a ver como cambiar/crear/eliminar propiedades de los elementos seleccionados.

Importante: Aunque las estructuras `HTMLCollection` y `NodeList` se comportan de forma "similar" a un array, no son estructuras idénticas y las funciones para array que se muestran mas adelante, no funcionarán en ellas.
Si alguna vez necesitamos convertir alguna de esas dos estructuras en array forzosamente, podemos hacerlo así:

```
//obtenemos una HTMLCollection
var lista = document.getElementsByTagName(…)
//Se transforma en array
var nuevoArray = Array.from(lista);
```

11.3 Acceso a atributos HTML

Una vez nos hemos posicionado sobre un elemento, podemos consultar o cambiar los valores de todos los atributos HTML de ese elemento de la siguiente forma:

```
//Obtengo la referencia al elemento
var elemento = document.getElementById("tabla");


//Muestro el valor de su atributo altura
alert(elemento.height);


//cambio valor del atributo border
elemento.border = '5px';


. . .


<table id="tabla" width='230px' height='300px'
border='1px'>
     . . .
</table>
```

Como se ve en el ejemplo, el acceso a atributos HTML de un elemento es tan sencillo como colocar la variable con la referencia a ese elemento seguido de punto y el nombre del atributo a consultar:

```
var elemento = document.getElementById("tabla");

alert(elemento.height);
```

Si lo que queremos es cambiar el valor de un atributo, tan sencillo como indicar ese atributo (usando el punto) y asignarle un nuevo valor correcto:

```
elemento.border = '5px';
```

Destacar que si a la hora de asignar un nuevo valor al atributo colocamos un valor incorrecto, ese atributo quedará sin valor (y por tanto ele elemento quedará afectado de alguna manera) hasta que se vuelva a cargar la página.

Si asignamos valor aun atributo que no tiene el elemento, a partir de esa asignación, al elemento se le añadirá el atributo asignado:

```
var elemento = document.getElementById("parra");


elemento.align= "center"
//A partir de este punto, el párrafo quedará alineado al
centro


. . .


<p id= "parra">
  Soy un párrafo con alineación a la izquierda por
defecto.
</p>
```

Como es natural, los atributos `id` y `name` también se pueden consultar y alterar aunque esto último no es recomendable hacerlo porque podríamos perder la referencia de ese objeto en el árbol DOM.

11.3.1 Atributo innerHTML

Todos los elementos de nuestra página van a tener un atributo llamado `innerHTML` el cual guarda el contenido HTML de ese elemento, es decir, todo lo que hay entre las etiquetas de apertura y cierre de ese elemento (ya sea texto u otras etiquetas).

Ejemplo: Para el siguiente código…

```
<div id='padre'>
  <p id='contenido'>
     En un lugar de la Málaga profunda.
  </p>
</div>
```

… si hacemos:

```
var elemento = document.getElementById('padre');

//cambia el contenido del párrafo
console.log(elemento.innerHTML)
```

Vamos a obtener:

```
<p id='contenido'>
     En un lugar de la Málaga profunda.
</p>
```

Es decir, todo el código HTML contenido en ese nodo.

El contenido de este atributo se puede cambiar. Así pues, aunque no sea la forma más elegante, nosotros podemos alterar el contenido de un elemento a través del innerHTML:

```
var elemento = document.getElementById('contenido');

//cambia el contenido del párrafo
elemento.innerHTML = "Cambio el <b>texto</b>";
. . .
<p id='contenido'>
   En un lugar de la Málaga profunda.
</p>
```

Al ejecutarse el código anterior el contenido del párrafo del ejemplo cambiará por el indicado en el atributo innerHTML (hasta que la página se vuelva a cargar).

Obviamente, el contenido de innerHTML puede consultarse, mostrarse por pantalla, copiarse a otra variable...

```
var p1 = document.getElementById('contenido');
var p2 = document.getElementById('salida');

//copio el contenido de p1 a p2
p2.innerHTML = p1.innerHTML;
. . .
<p id='contenido'>
   En un lugar de la Málaga profunda.
</p>

<p id='salida'>
   No voy a durar mucho así escrito.
</p>
```

11.3.2 Atributo textContent

Al igual que `innerHTML`, todos los elementos de nuestra página van a tener el atributo llamado `textContent`. Este atributo se usa para consultar y modificar solo el texto de ese elemento y el de todos sus hijos e ignora las etiquetas HTML que encuentre. Sin embargo, respeta los saltos de lineas y los espacios entre palabras.

Ejemplo: Para el mismo código del apartado anterior…

```
<div id='padre'>
   Soy texto fuera del parrafo.
  <p id='contenido'>
     En un lugar de la Málaga profunda.
  </p>
</div>
```

… si hacemos:

```
var elemento = document.getElementById('padre');

//muestra el contenido del párrafo
console.log(elemento.textContent)
```

Vamos a obtener:

```
Soy texto fuera del parrafo.

   En un lugar de la Málaga profunda
```

Es decir, todo el texto que posee ese elemento y sus hijos sin etiquetas HTML.

Como es lógico, el contenido de este atributo se puede cambiar. De esta forma, nosotros podemos alterar el texto de un elemento:

```
var elemento = document.getElementById('padre');
//cambia el texto del DIV
elemento.textContent= "Cambio solo el texto";
. . .
<div id='padre'>
   Soy texto fuera del parrafo.
```

```
  <p id='contenido'>
     En un lugar de la Málaga profunda.
  </p>
</div>
```

Es importante destacar que si cambio el texto de un elemento de esta forma, se eliminan todos los hijos que tiene ese elemento. Por lo que el DIV del ejemplo anterior quedaría:

```
<div id='padre'>
   Cambio solo el texto
</div>
```

Destacar que también existe el atributo **`innerText`**, el cual es similar a `textContent` (devuelve el texto del elemento y de sus hijos) pero no respeta los saltos de lineas ni los espacios.

11.3.3 Atributo *class*

Como el lector ya debe saber, el atributo class sirve para indicar a que clase CSS pertenece ese elemento. Al ser un atributo HTML podemos mostrarlo o alterarlo con Javascript sin problemas.

Tan sólo hay que tener en cuenta que Javascript conoce a ese atributo por `className` (dado que la palabra `class` se usa para definir clases en Javascript)

Sabiendo esto podemos cambiar fácilmente la clase a la que pertenece un elemento usando Javascript:

```
var elemento = document.getElementById("tabla");

//mostramos la clase a la que pertenece
alert(elemento.className);

//cambiamos la clase
elemento.className = 'listaUsuarios';
. . .

<table id="tabla" class='usuarios'>
 . . .
</table>
```

Obviamente, si cambiamos de clase a un elemento, este perderá todas las propiedades de la antigua clase y obtendrá las propiedades de la clase nueva.

11.4 Acceso a propiedades CSS

Al igual que hemos podido acceder a atributos, con Javascript vamos a poder acceder a las propiedades CSS que tenga un objeto seleccionado.

El acceso es casi tan sencillo como en el caso anterior:

1. Accedo al elemento deseado.
2. Cambio el valor de una propiedad colocando: la variable donde tengo la referencia seguida de punto, la palabra `style`, otro punto y el nombre de la propiedad CSS a la que queremos alterar su valor:

```
#tabla{
   border: 1px solid black;
   width: 300px;
   height: 300px;
   padding: 10px;
```

```
   margin-left: 20px;
   background-color: aqua;
}
. . .
var elemento = document.getElementById("tabla");
elemento.style.border = '2px solid red';
elemento.style.padding = '0.75em';
elemento.style.width = '500px'
```

En el código anterior estamos cambiando los valores de las propiedades `border`, `padding` y `width`, Es importante colocar valores entre comillas (son cadenas) y además deben ser correctos para cada propiedad. Si no hacemos eso, estaremos asignando un valor incorrecto a la propiedad y el navegador no la tendrá en cuenta.

Si lo que queremos es alterar el valor de una propiedad con nombre compuesto (pej: `margin-left`) debemos actuar de la siguiente forma:

- Se elimina el guión (-) del nombre y se juntan las palabras.
- La primera letra de la segunda palabra irá en mayúsculas.

```
#tabla{
   border: 1px solid black;
   width: 300px;
   height: 300px;
   padding: 10px;
   margin-left: 20px;
   background-color: aqua;
}
. . .
var elemento = document.getElementById("tabla");
elemento.style.marginLeft = 0;
elemento.style.backgroundColor = 'blue';
```

Hay una forma alternativa (y más elegante) de hacer lo que acabamos de aprender: usando la función `setProperty`:

```
referencia.style.setProperty(propiedad css, valor)
```

Para el ejemplo anterior, podemos hacer:

```
var elemento = document.getElementById("tabla");


elemento.style.setProperty('margin-left','2px');
elemento.style.setProperty('background-color','blue');
elemento.style.setProperty('width','500px');
```

Destacar que, usando esta forma, las propiedades compuestas se escriben igual que en código CSS.

Tanto usando la forma `.style.propiedad` como `.style.setProperty`, lo que estoy haciendo en realidad es añadir valores al atributo `style` de esa etiqueta (y si no lo tiene, lo añade). No estoy tocando realmente la hoja de estilos.

11.4.1 Consultando valores CSS

En el apartado anterior hemos visto como cambiar los valores de propiedades CSS del atributo `style` del elemento referenciado. Con el nivel que ya tenemos, lo normal es pensar que podríamos consultar/mostrar/copiar valores de propiedades CSS de la misma forma, pej:

```
console.log(elemento.padding);
```

Sin embargo, **esto no es correcto** si lo que queremos es mostrar valores de propiedades de la hoja de estilos. Si usamos esta forma, toda aquella propiedad que no esté en el atributo `style`, va a devolver una cadena vacía, aunque la tengamos declarada en la hoja de estilos.

Supongamos el siguiente escenario:

```
p{
   border:  2px solid black;
   width:  500px;
   padding: 10px;
}
```

```
 <p id="parra">
     Texto muy largo para que se vea bien la anchura y el
paddind del elemento.
</p>
```

```
function consulta(){
    var elemento = document.getElementById('parra');
    console.log(elemento.style.padding);
}
```

Al ejecutar la función, vamos a obtener por consola: `<empty string>`

Esto es debido a que esa propiedad no está definida en el atributo `style` (que en este caso, ni siquiera está definido).

Además, aunque estuviera esa propiedad definida en el atributo `style`, seguimos teniendo otro problema: podemos tener que un navegador nos muestre `10px,` otro nos muestre `10px 10px 10px 10px` u otro muestre `10px 10px`

Es por eso que, para consultar valores de propiedades CSS, debemos hacer uso de un par de herramientas de Javascript que, por desgracia, complican algo que se podría hacer de forma sencilla tal y como hemos mostrado.

Para consultar valores de propiedades CSS debemos hacer los siguiente:

1. Obtener una referencia al elemento.
2. Obtener los estilos del elemento con la función `window.getComputedStyle(referencia)`

3. usar la función `getPropertyValue` para obtener el valor de la propiedad deseada.

Para los códigos del ejemplo anterior:

```
p{
   border:  2px solid black;
   width:  500px;
   padding: 10px;
}
```

```
 <p id="parra">
     Texto muy largo para que se vea bien la anchura y el
paddind del elemento.
</p>
```

Obtendríamos el valor de `padding`:

```
function consulta(){

    //paso 1
    var elemento = document.getElementById('parra');

    //paso 2
    var estilos = window.getComputedStyle(elemento);
    //paso 3
    var valor = estilo.getPropertyValue('padding')

    console.log(valor);
}
```

Si deseamos consultar el valor de una propiedad compuesta, la escribiríamos tal y como se escribe en código CSS:

```
var valor = estilo.getPropertyValue('font-size')
```

Aún usando esta forma para obtener valores CSS, hay que tener presente que en la mayoría de los casos, no se nos va mostrar correctamente las unidades. Pej: los valores de colores los va a devolver siempre con notación RGB aunque usemos nombres y siempre va a devolvernos valores en pixeles independientemente de que usemos puntos, porcentajes, em, auto … (obviamente, nos hace el cálculo correcto para transformar un valor de una unidad a pixeles).

También es importante destacar que los valores devueltos siempre serán del tipo cadena, ya que también devuelve las unidades. Nunca va a devolver números (aunque el valor no tenga unidad definida en la hoja de estilos).

11.4.2 Obteniendo el tamaño de un elemento

Como el lector debería saber ya, el tamaño que ocupa un elemento en la página no es el que indica la propiedad `width/height` de su CSS. A ese valor hay que sumarle los valores (si los hubiera) del padding, del borde y, en algunos casos, del margin.

Cuando se alteran propiedades de nodos del árbol DOM, el programador no tiene por qué saber los valores de las propiedades CSS. Sin embargo, puede ser necesario conocer cuanto ocupa ese elemento dentro de la estructura de la página.

Para obtener ese valor podemos hacer uso de las siguientes funciones de JavaScript:

Función	Descripción
`clientWidth`	Obtiene la *anchura* del elemento incluyendo el padding.
`clientHeight`	Obtiene la *altura* del elemento incluyendo el padding.
`offsetWidth`	Obtiene la *anchura* del elemento incluyendo el padding y el borde.
`offsetHeight`	Obtiene la *altura* del elemento incluyendo el padding y el borde.

Para el siguiente código HTML:

```
<div id='padre'>
  Qué tamaño tengo...
</div>
```

Con las siguientes propiedades CSS:

```
#padre{
    width: 400px;
    padding: 10px;
    border: 5px solid black;
    margin: 20px;
}
```

Si ejecutamos el siguiente código:

```
var elemento = document.getElementById('padre');

console.log("Anchuras: ")
console.log(elemento.clientWidth)
console.log(elemento.offsetWidth)

console.log("Alturas: ")
console.log(elemento.clientHeight)
console.log(elemento.offsetHeight)
```

Obtenemos:

```
Anchuras:
420
430
Alturas:
38
48
```

Destacar que, si bien la anchura se puede calcular de forma análoga obteniendo los valores de las propiedades `width`, `padding` y `border` como se vio en el apartado anterior, el cálculo de la altura no es tan sencillo debido a que, en las propiedades CSS no aparece la propiedad `height` por ningún lado.

Si usamos las funciones explicadas hacemos que sea el propio Javascript el que calcule todos los valores de forma correcta independientemente de si aparecen las propiedades CSS correspondientes o no.