Clase 16

JavaScript Parte 5

JavaScript

Temas: Objetos - Iterar con for in -for of

Manipulación del DOM - Eventos

Objetos

¿Qué son los objetos?

En Javascript, existe un tipo de dato llamado objeto. No es más que una variable especial que puede contener más variables en su interior. De esta forma, tenemos la posibilidad de organizar múltiples variables de la misma temática dentro de un objeto. Veamos algunos ejemplos:
En muchos lenguajes de programación, para crear un objeto se utiliza la palabra clave new. En Javascript también se puede hacer:

```
const objeto = new Object(); // Esto es un objeto «genérico» vacío
```

Sin embargo, siempre que podamos, en Javascript se prefiere utilizar lo que se llaman los literales, un método abreviado para crear objetos directamente, sin necesidad de utilizar la palabra new.

```
const objeto = {}; // Esto es un objeto vacío
```

Pero hasta ahora, solo hemos creado un objeto vacío. Vamos a crear un nuevo objeto, que contenga variables con información en su interior:

```
// Declaración del objeto
const player = {
  name: "Manz",
  life: 99,
  strength: 10,
};
```

Estas variables dentro de los objetos se suelen denominar propiedades. Como se puede ver, un objeto en Javascript nos permite encapsular en su interior información relacionada, para posteriormente poder acceder a ella de forma más sencilla e intuitiva.

Objetos

Acceso a sus propiedades

Una vez tengamos un objeto, podemos acceder a sus propiedades de dos formas diferentes: a través de la notación con puntos o a través de la notación con corchetes.

```js
// Notación con puntos
console.log(player.name); // Muestra "Manz"
console.log(player.life); // Muestra 99

// Notación con corchetes
console.log(player["name"]); // Muestra "Manz"
console.log(player["life"]); // Muestra 99
```

El programador puede utilizar la notación que más le guste. La más utilizada en Javascript suele ser la notación con puntos, mientras que la notación con corchetes se suele conocer en otros lenguajes como «arrays asociativos».

Objetos

Vamos Buenos Aires

Añadir propiedades

También podemos añadir **propiedades** al **objeto** después de haberlo creado, aunque la sintaxis cambia ligeramente. Veamos un ejemplo equivalente al anterior:

JS

```
// Declaración del objeto
const player = {};

// Añadimos mediante notación con puntos
player.name = "Manz";
player.life = 99;
player.strength = 10;

// Añadimos mediante notación con corchetes
player["name"] = "Manz";
player["life"] = 99;
player["strength"] = 10;
```

Las **propiedades** del objeto pueden ser utilizadas como variables. De hecho, utilizar los objetos como elementos para organizar múltiples variables suele ser una buena práctica en Javascript.

Objetos

Tipos de Objetos

Hasta ahora, solo hemos visto los objetos «genéricos», en Javascript conocidos como tipo `OBJECT`, declarándolos con un **`new Object()`** o con un literal **`{}`**, dos formas equivalentes de hacer lo mismo. Al generar una variable de tipo `OBJECT`, esa variable «hereda» una serie de métodos (*del objeto* ***`Object`*** *en este caso*).

```js
const o = {};
o.toString(); // Devuelve '[object Object]' (Un objeto de tipo Object)
```

En este ejemplo, **`toString()`** es uno de esos métodos que tienen todas las variables de tipo `OBJECT`. Sin embargo, hasta ahora y sin saberlo, cuando creamos una variable de un determinado tipo de dato (*sea primitivo o no*), es también de tipo `OBJECT`, ya que todas las variables heredan de este tipo de dato. Por lo tanto, nuestra variable tendrá no sólo los métodos de su tipo de dato, sino también los métodos heredados de `OBJECT`:

```js
const s = "hola";
s.toString(); // Devuelve 'hola'
```

https://www.w3schools.com/js/js_objects.asp

DOM

¿Qué es el DOM?

Las siglas **DOM** significan **Document Object Model**, o lo que es lo mismo, la estructura del documento HTML. Una página HTML está formada por múltiples etiquetas HTML, anidadas una dentro de otra, formando un árbol de etiquetas relacionadas entre sí, que se denomina **árbol DOM** (*o simplemente DOM*).

En Javascript, cuando nos referimos al **DOM** nos referimos a esta estructura, que podemos modificar de forma dinámica desde Javascript, añadiendo nuevas etiquetas, modificando o eliminando otras, cambiando sus atributos HTML, añadiendo clases, cambiando el contenido de texto, etc...

DOM

El objeto document

En Javascript, la forma de acceder al DOM es a través de un objeto llamado **document**, que representa el árbol DOM de la página o pestaña del navegador donde nos encontramos. En su interior pueden existir varios tipos de elementos, pero principalmente serán ELEMENT o NODE:

- ELEMENT no es más que la representación genérica de una etiqueta: **HTMLElement**.
- NODE es una unidad más básica, la cuál puede ser ELEMENT o un **nodo de texto**.

DOM

Seleccionar elementos del DOM

Existen varios métodos, los más clásicos y tradicionales para realizar búsquedas de elementos en el documento. Observa que si lo que buscas es un elemento específico, lo mejor sería utilizar **`getElementById()`**, en caso contrario, si utilizamos uno de los 3 siguientes métodos, nos devolverá un ARRAY donde tendremos que elegir el elemento en cuestión posteriormente:

Métodos de búsqueda	Descripción
ELEMENT **`.getElementById(id)`**	Busca el elemento HTML con el id **`id`**. Si no, devuelve NULL.
ARRAY **`.getElementsByClassName(class)`**	Busca elementos con la clase **`class`**. Si no, devuelve **`[]`**.
ARRAY **`.getElementsByName(name)`**	Busca elementos con atributo name **`name`**. Si no, devuelve **`[]`**.
ARRAY **`.getElementsByTagName(tag)`**	Busca elementos **`tag`**. Si no encuentra ninguno, devuelve **`[]`**.

DOM

Seleccionar elementos del DOM

getElementById()

El primer método, **`.getElementById(id)`** busca un elemento HTML con el **id** especificado en **`id`** por parámetro. En principio, un documento HTML bien construído **no debería** tener más de un elemento con el mismo **`id`**, por lo tanto, este método devolverá siempre un solo elemento:

```js
const page = document.getElementById("page");   // <div id="page"></div>
```

DOM

Seleccionar elementos del DOM

getElementsByClassName()

Por otro lado, el método **`.getElementsByClassName(class)`** permite buscar los elementos con la **clase** especificada en **`class`**. Es importante darse cuenta del matiz de que el metodo tiene **`getElements`** en plural, y esto es porque al devolver **clases** (*al contrario que los id*) se pueden repetir, y por lo tanto, devolvernos varios elementos, no sólo uno.

JS

```
const items = document.getElementsByClassName("item");  // [div, div, div]

console.log(items[0]);      // Primer item encontrado: <div class="item"></div>
console.log(items.length);  // 3
```

DOM

Seleccionar elementos del DOM

> Estos métodos devuelven siempre un ARRAY con todos los elementos encontrados que encajen con el criterio. En el caso de no encontrar ninguno, devolverán un ARRAY vacío: `[]`.

Exactamente igual funcionan los métodos **`getElementsByName(name)`** y **`getElementsByTagName(tag)`**, salvo que se encargan de buscar elementos HTML por su atributo **name** o por su propia **etiqueta** de elemento HTML, respectivamente:

JS

```
// Obtiene todos los elementos con atributo name="nickname"
const nicknames = document.getElementsByName("nickname");

// Obtiene todos los elementos <div> de la página
const divs = document.getElementsByTagName("div");
```

DOM

Vamos Buenos Aires

Métodos modernos

Aunque podemos utilizar los métodos tradicionales que acabamos de ver, actualmente tenemos a nuestra disposición dos nuevos métodos de búsqueda de elementos que son mucho más cómodos y prácticos si conocemos y dominamos los selectores CSS. Es el caso de los métodos `.querySelector()` y `.querySelectorAll()`:

Método de búsqueda	Descripción
ELEMENT `.querySelector(sel)`	Busca el primer elemento que coincide con el selector CSS **`sel`**. Si no, NULL.
ARRAY `.querySelectorAll(sel)`	Busca todos los elementos que coinciden con el selector CSS **`sel`**. Si no, **`[]`**.

Con estos dos métodos podemos realizar todo lo que hacíamos con los **métodos tradicionales** mencionados anteriormente e incluso muchas más cosas (*en menos código*), ya que son muy flexibles y potentes gracias a los **selectores CSS**.

DOM

Métodos modernos

querySelector()

El primero, **.querySelector(selector)** devuelve el primer elemento que encuentra que encaja con el selector CSS suministrado en **selector**. Al igual que su «equivalente» **.getElementById()**, devuelve un solo elemento y en caso de no coincidir con ninguno, devuelve NULL :

```js
const page = document.querySelector("#page");          // <div id="page"></div>
const info = document.querySelector(".main .info");    // <div class="info"></div>
```

Lo interesante de este método, es que al permitir suministrarle un selector CSS básico ▸ o incluso un selector CSS avanzado ▸ , se vuelve un sistema mucho más potente.

El primer ejemplo es equivalente a utilizar un **.getElementById()**, sólo que en la versión de **.querySelector()** indicamos por parámetro un SELECTOR , y en el primero le pasamos un simple STRING . Observa que estamos indicando un **#** porque se trata de un **id**.

En el segundo ejemplo, estamos recuperando el primer elemento con clase **info** que se encuentre dentro de otro elemento con clase **main**. Eso podría realizarse con los métodos tradicionales, pero sería menos directo ya que tendríamos que realizar varias llamadas, con **.querySelector()** se hace directamente con sólo una.

DOM

Métodos modernos

querySelectorAll()

Por otro lado, el método **.querySelectorAll()** realiza una búsqueda de elementos como lo hace el anterior, sólo que como podremos intuir por ese **All()**, devuelve un ARRAY con todos los elementos que coinciden con el SELECTOR CSS:

```js
// Obtiene todos los elementos con clase "info"
const infos = document.querySelectorAll(".info");

// Obtiene todos los elementos con atributo name="nickname"
const nicknames = document.querySelectorAll('[name="nickname"]');

// Obtiene todos los elementos <div> de la página HTML
const divs = document.querySelectorAll("div");
```

En este caso, recuerda que **.querySelectorAll()** siempre nos devolverá un ARRAY de elementos. Depende de los elementos que encuentre mediante el SELECTOR, nos devolverá un ARRAY de **0** elementos o de **1**, **2** o más elementos.

> Al realizar una búsqueda de elementos y guardarlos en una variable, podemos realizar la búsqueda posteriormente sobre esa variable en lugar de hacerla sobre **document**. Esto permite realizar búsquedas acotadas por zonas, en lugar de realizarlo siempre sobre **document**, que buscará en todo el documento HTML.

DOM

Eventos del DOM

https://www.w3schools.com/js/js_htmldom_events.asp

Objetos

Bucles

`for..in`	Bucles sobre posiciones de un array.
`for..of`	Bucles sobre elementos de un array.

https://www.w3schools.com/jsref/jsref_forin.asp

https://www.w3schools.com/jsref/jsref_forof.asp