

Curso de JavaScript

- JavaScript Essential Training
-

Contenido

Glosario	2
Links útiles.....	3
Capítulo 1. Introducción	4
Capítulo 2. Lo básico.....	4
Capítulo 3. Trabajando con datos.	5
Capítulo 4. Funciones y objetos	7
Capítulo 5. JavaScript and DOM	9
Capítulo 7. JavaScript DOM Eventos	13
Capítulo 8. Proyecto Typing Speed Tester.....	14
Capítulo 9. Loop	14
Capítulo 10. Troubleshooting	14

Glosario

- NaN: Not a Number
-

Links útiles

- <https://developer.mozilla.org/es/docs/Web/JavaScript/Closures>
- <https://developer.mozilla.org/en-US/docs/Web/API/Element>
- <https://developer.mozilla.org/en-US/docs/Web/Events>
-

Capítulo 1. Introducción

JavaScript

Es la capa de interacción. Es un lenguaje de *scripting* que se ejecuta en el navegador y se encarga de interactuar con el HTML y CSS para cambiar lo que se ve y lo que se puede hacer.

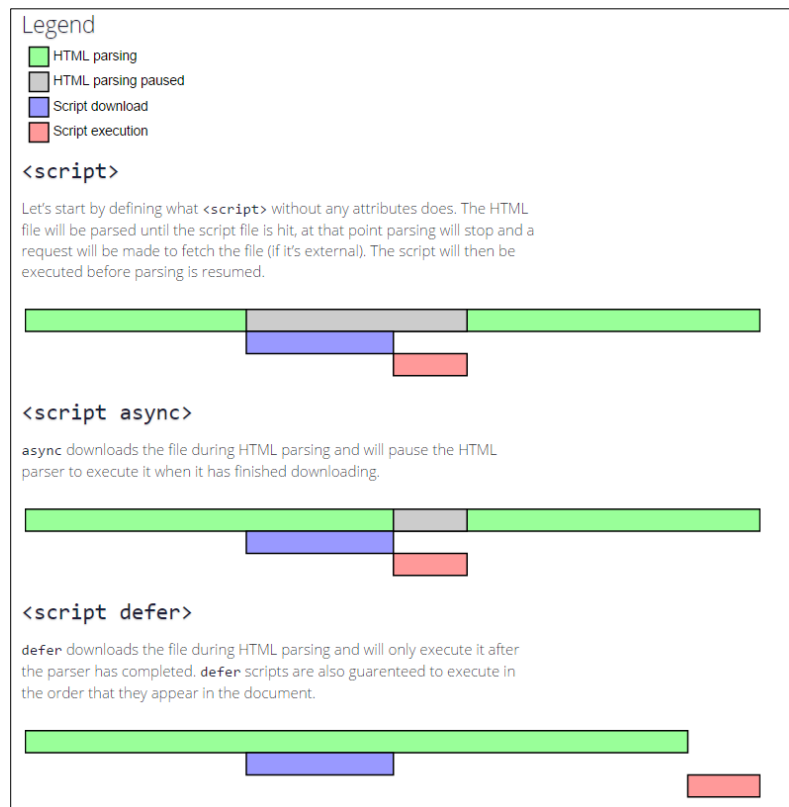
Capítulo 2. Lo básico.

Document.body

`document.body` es el elemento que contiene el contenido para el documento. En documentos con contenidos `<body>`, devuelven el elemento `<body>`, y en documentos de marco de sistema, esto devuelve el elemento extremo `<frameset>`.

Aunque `body` es programable, colocando un nuevo cuerpo en un documento efectivamente quitará a todos los hijos actuales del elemento existente `<body>`.

Cargar Script JS



A tener en cuenta JS

- JavaScript es *case sensitive*.
- Los nombres para definir métodos, funciones y atributos es *camelCase*
- Objetos y Clases con la primera letra en mayúscula
- Constantes todo en mayúsculas.
- Siempre añadir *var* cuando declarando una variable.
- ¡Añadir espacio para los humanos!

```
// With whitespace (human readable):
var date = new Date();
document.body.innerHTML = "<h1>" + date + "</h1>";

// Without whitespace (machine readable):
var date=new
Date();document.body.innerHTML="<h1>
"+date+"</h1>";
```

Capítulo 3. Trabajando con datos.

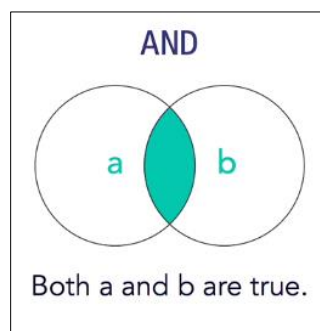
Typeof

Sirve para saber qué tipo de variable es

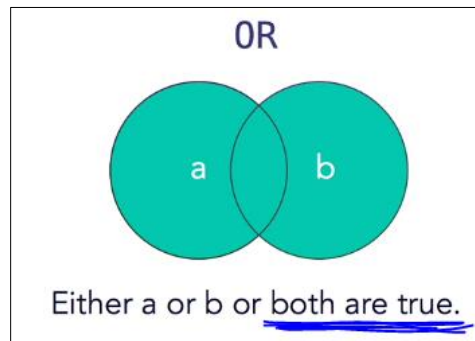
```
>> var ejemploString = "Mi nombre";
< undefined
>> console.log(typeof ejemploString);
string
< undefined
>> |
```

Condiciones y lógica

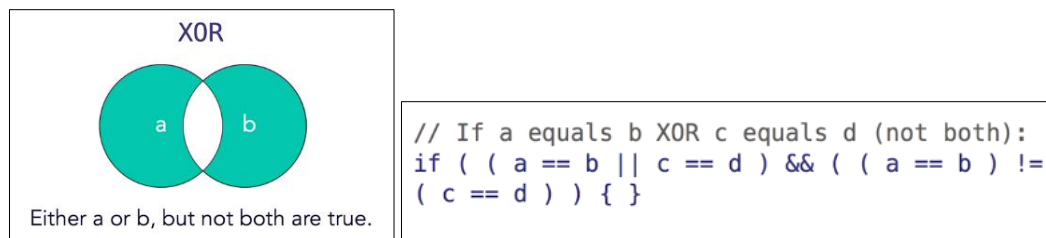
- **AND = &&**



- OR = ||



- XOR: No existe en JavaScript así que hay que crearlo.



- Operador condicional (ternario)

```
var elvisLives = Math.PI > 4 ? "Sip" : "Nop";
```

En este caso el (?) indica lo que se hará si la condición es verdadera y los (:) si es falsa

Propiedades y métodos.

- **Métodos:** Funciones que pertenecen al objeto. Pueden requerir argumentos que se añaden en los paréntesis. Una cosa es declarar una función y otra es llamarla.
- **Property:** Información en general sobre el objeto

Array.join

```
// Return the items in an array as a comma separated string.
var arrayString = pens.join(" ");
console.log("String from array: ", arrayString);
```

```
Before: ► Array(4) [ "red", "blue", "green", "orange" ]
String from array: red blue green orange
```

Capítulo 4. Funciones y objetos

Funciones

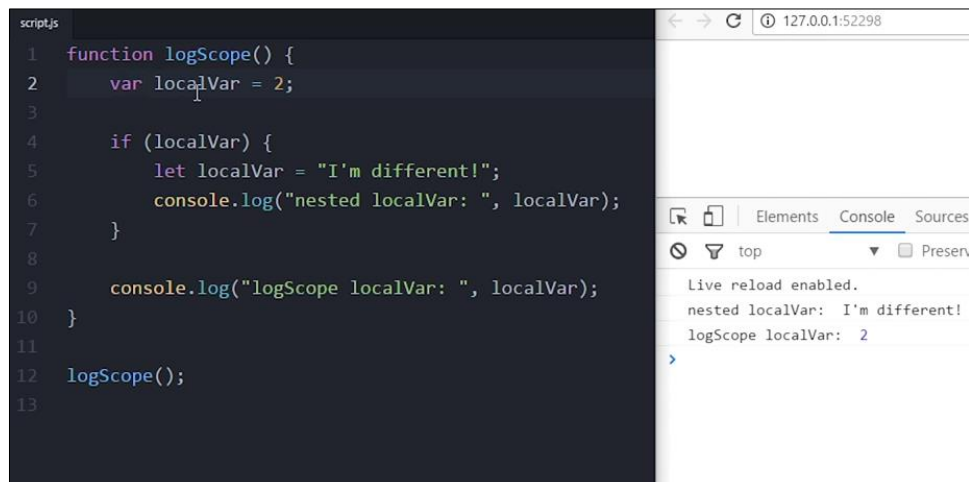
Son pequeños “programas” dentro del Script. Sirven para sementar el código y hacer más fácil su gestión, o para repetir operaciones. Hay tres tipos

- **Named Functions**
- **Anonymous**
- **Immediately invoked.**

Se crean primero las funciones y después se llaman.

Const y Let

- **Const:** Constante, no se puede cambiar una vez definida (debe escribirse todo en mayúscula como buena práctica)
- **Let:** Una variable con menor alcance que *var*. Se le interpreta como “lo que queda entre dos corchetes”. En la imagen se observa como dentro del *if* a pesar de asignarle un nuevo valor, una vez sale del *if* retoma su valor anterior.



The screenshot shows a web browser with a JavaScript file named `script.js` loaded. The code defines a function `logScope()` that uses `var` and `let` to demonstrate variable scope. The console output shows that `logScope` logs the value of `localVar` as 2, while the nested `if` block logs it as "I'm different!".

```
1 function logScope() {
2   var localVar = 2;
3
4   if (localVar) {
5     let localVar = "I'm different!";
6     console.log("nested localVar: ", localVar);
7   }
8
9   console.log("logScope localVar: ", localVar);
10 }
11
12 logScope();
13
```

Console output:

```
Live reload enabled.
nested localVar: I'm different!
logScope localVar: 2
```

Objetos



The screenshot shows a code editor with two files: `script.js` and `index.html`. The `script.js` file contains a JavaScript object `myObject` with properties `title`, `level`, and `views`, and a nested function `updateviews` that increments the `views` property.

```
1 var myObject = new Object({
2   title: "Hola",
3   level: 1,
4   views: 0,
5   updateviews: function(){
6     return ++myObject.views;
7   }
8 });
```

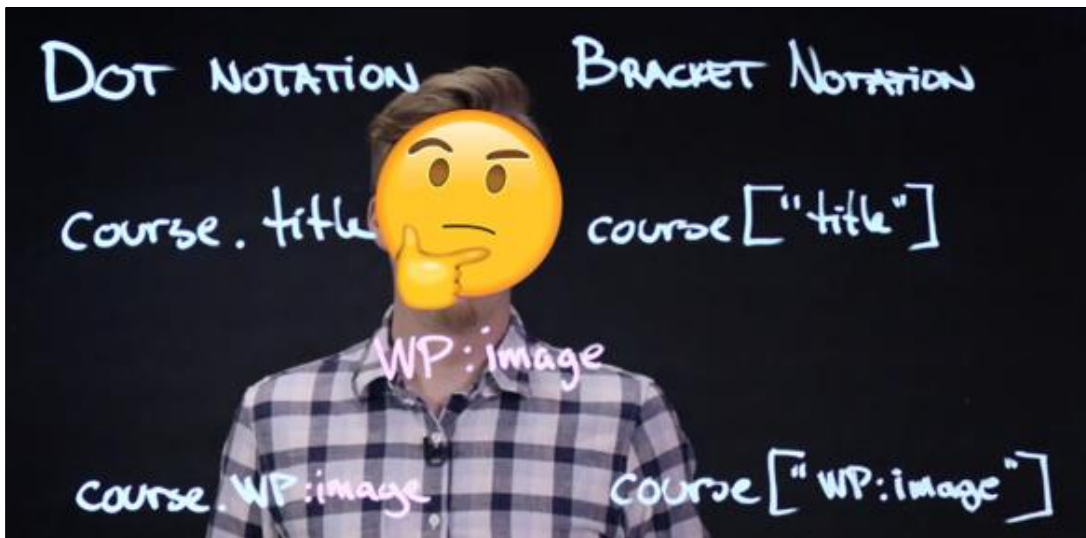
Constructor Objetos

Así se pueden crear objetos constructores, para no tener que hacer el ejemplo anterior. Como regla, cuando se define un constructor de objeto se empieza con Mayúscula.

```
script
1  function Course(title,instructor,level,published,view
2      this.title = title;
3      this.instructor = instructor;
4      this.level = level;
5      this.published = published;
6      this.views = views;
7      this.updateViews = function() {
8          return ++this.views;
9      };
10 }
11
12 |
```

Notación con Punto y con Brackets

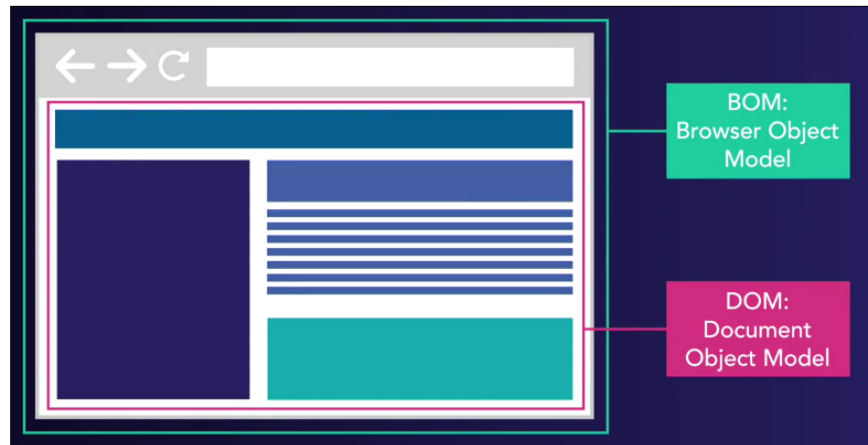
Dos tipos de notaciones para obtener lo mismo. Como se explica, los corchetes son usados en caso de que la notación de la propiedad tenga conflictos con JavaScript



Closures

Conocido también como clausura es la combinación de una función y el ámbito léxico en el que se declaró dicha función

Capítulo 5. JavaScript and DOM



Referenciar elementos en el DOM

```
// Get the element with a specified ID:
document.getElementById("some-ID");

/* Get all elements with a specified class name
as array: */
document.getElementsByClassName("classname");

/* Get all elements with a specified HTML tag as
array: */
document.getElementsByTagName("HTML tag");
```

Existen dos referencias más nuevas que logran referenciar a los elementos usando sintaxis de CSS.

```
/* Get the first element matching specified
selector(s): */
document.querySelector(".main-nav a");

/* Get all elements matching specified
selector(s): */
document.querySelectorAll(".post-content p");
```

Acceder y cambiar elementos/clases

<https://developer.mozilla.org/en-US/docs/Web/API/Element>

```

>> document.querySelector(".masthead").classList
< ▶ DOMTokenList [ "masthead", "clear" ]
>> document.querySelector(".masthead").classList.add("new-class");
< undefined
>> document.querySelector(".masthead").classList
< ▶ DOMTokenList(3) [ "masthead", "clear", "new-class" ]
>> document.querySelector(".masthead").classList.remove("clear");
< undefined
>> document.querySelector(".masthead").classList
< ▶ DOMTokenList [ "masthead", "new-class" ]

```

Acceder y cambiar atributos

- Element.hasAttribute()
- Element.getAttribute()
- Element.setAttribute()
- Element.removeAttribute()

```

const CTAELEMENT = document.querySelector(".cta a");

if(CTAELEMENT.hasAttribute("target")){
  console.log(CTAELEMENT.getAttribute("target"));
}
else{
  /*Attribute, value */
  CTAELEMENT.setAttribute("target", "_blank");
}

console.log(CTAELEMENT.attributes);

```

Añadir elementos DOM

1. Crear el elemento
2. Crear el nodo de texto que va dentro del elemento
3. Añadir el nodo de texto al elemento
4. Añadir el elemento al árbol DOM.

Métodos necesarios:

```

.createElement(); // Create an element.

.createTextNode(); // Create text node.

.appendChild(); /* Place one child node
inside another. */

```

```
const FEATURED = document.querySelector(".featured-image");
const THEIMAGE = FEATURED.querySelector("img");

var altText = THEIMAGE.getAttribute("alt");

var captionElement = document.createElement("figcaption");

var captionText = document.createTextNode(altText);

captionElement.appendChild(captionText);

console.log(captionElement);
FEATURED.appendChild(captionElement);
THEIMAGE.setAttribute("alt", "");
```

También se puede hacer de una manera más sencilla:

```
const FEATURED = document.querySelector(".featured-image");
const THEIMAGE = FEATURED.querySelector("img");

var altText = THEIMAGE.getAttribute("alt");

var captionElement = document.createElement("figcaption");

captionElement.append(altText);

//var captionText = document.createTextNode(altText);
//captionElement.appendChild(captionText);

console.log(captionElement);
FEATURED.append(captionElement);
THEIMAGE.setAttribute("alt", "");
```

El método *append* reemplaza a *appendChild* y *createTextNode*. Sin embargo, puede no ser soportado en navegadores antiguos.

Aplicar estilos “inline” CSS a un elemento.

Como regla se sabe que un “*inline*” tiene mayor prioridad que una hoja de estilos CSS. Es por eso, que en la mayoría de los casos la mejor practica es crear reglas de CSS y usar JavaScript para administrar esas clases para aplicar las reglas al elemento.

Para aplicar estilos directos en “inline” se hace de la siguiente forma. Como se ve *background-color* propiedad en CSS se usa con camelCase en JS.

```
> document.querySelector(".cta a").style.color = "green"
< "green"
> document.querySelector(".cta a").style.backgroundColor = "blue"
< "blue"
```

También para añadir varios comandos de CSS se usa con la propiedad `cssText`.

```
> document.querySelector(".cta a").style.cssText = "padding: 1em; color: white; background-color: red;"
```

Recordar que estos estilos siempre serán *inline* y jamás modificarán la hoja de estilos

```
▼ <div class="cta">
  <a href="#" style="padding: 1em; color: white; background-color: red;">Book Now!</a> == $0
</div>
```

Capítulo 7. JavaScript DOM Eventos

<https://developer.mozilla.org/en-US/docs/Web/Events>

Eventos típicos

Eventos en el Navegador:

- Load: Cuando se termina de cargar los recursos
- Error: Cuando un recurso no cargo
- Online/offline: El navegador tiene (o pierde) acceso a internet
- Resize: cuando el *viewport* cambia de tamaño
- Scroll: cuando el *viewport* ha sido *scrolled* arriba/abajo/derecha/izquierda.

Eventos en el DOM:

- Focus: Cuando el elemento está en enfoque. Ha sido clickeado, se le hizo tab, etc.
- Blur: Cuando el elemento pierde el enfoque. Como por ejemplo salir de un formulario
- Resets/submit: Eventos de formularios. Se ha presionado el botón de resetear o de enviar.
- Mouse Events: click, mouseover, drag, drop, etc.

Capítulo 8. Proyecto Typing Speed Tester.

AddEventListener()

Este método adjunta un *evento handler* a un elemento específico. Tiene tres parámetros:

```
element.addEventListener(event, function, useCapture)
```

- Event: Especifica el nombre del evento. NO SE USA EL prefijo “on”. Es decir, en vez de decir: *onclick* se usa *click*
- Function: Especifica la función que se ejecutara cuando el evento ocurra
- useCapture: Es opcional, especifica si el evento debería ser ejecutado en la fase de captura o de burbujeo (*bubbling phase*)

Capítulo 9. Loop

Keywords Break y Continue

- Break: Termina el loop
- Continue: Termina la iteración y sigue con la siguiente.

Capítulo 10. Troubleshooting

- *Console.info*
- *Console.error*

Dos tipos de mostrar información en consola muy útiles.

JSLint

Linting (v) se refieren al sitio web JS�int o **JSHint** para verificar errores del script.

Minificar código

Eliminar espacios en blanco.