

Cómo incluir Javascript en HTML5

25 respuestas



Qué es el Javascript y por qué me interesa en HTML5

Si estás leyendo este artículo, ya sabrás lo que es el **JavaScript**... si no es tu caso en la wikipedia lo definen como:

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos,³ basado en prototipos, imperativo, débilmente tipado y dinámico.

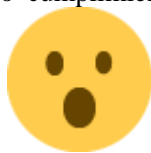
No creo necesario entrar en detalles, siempre se puede ir a la entrada de [wikipedia](#) para más información y obtener interesantes enlaces sobre el lenguaje.

De manera informal, el **JavaScript** es lo que se utiliza en los sitios web para hacer esos efectos molones de sliders, mover bloques de contenido (drag&drop), etc etc... básicamente cualquier tipo de interactividad entre el usuario y la página web utiliza Javascript, por ejemplo:

- Seguramente en más de una ocasión has oído hablar de [AJAX](#). Bien, AJAX significa “AsynchronousJavaScript And XML”... sí, **JavaScript**.
- [jQuery](#) también es una de esas palabrejas que está en boca de todos y no es ni más, ni menos, que un framework de **JavaScript**... eso sí, es tan maravilloso que muchos frontends utilizan jQuery por todos los beneficios que aporta (yo soy fan de jQuery), pero desconocen cómo se programa en Javascript, vamos que les quitas el jQuery y ni un `document.getElementById` saben hacer... ese es el gran peligro de los frameworks, son maravillosos y todos debemos utilizarlos sí o sí, pero antes hay que conocer el lenguaje, es **vital!!**

Norma para incluir nuestro Javascript

Realmente no existe una norma de obligado cumplimiento a la hora de colocar nuestro código Javascript dentro del



HTML... un momento... ¿dentro del HTML?

Sí, el código **JavaScript** lo interpreta nuestro navegador web, por lo tanto necesitamos incluirlo en el HTML. Cuando en un navegador se escribe la URL de un sitio web, se hace una petición al servidor que aloja el sitio, para que devuelva la información que debe pintar el ordenador en el navegador web.

Esta información se devuelve, normalmente, en forma de un código HTML que el navegador interpreta y pinta. Esto significa que si yo quiero que el navegador interprete un determinado código **Javascript** debo indicarle al navegador de alguna forma que existe un código Javascript que quiero que se ejecute y como el navegador lo que recibe es un código HTML... esta indicación tendrá que ir en el HTML.

Para esto, entre otras cosas, HTML tiene el tag script... lamentablemente ahí se acaba “la norma”.

Diferentes formas de incluir nuestro Javascript

Para explicar las diferentes posibilidades a la hora de incluir nuestro código **Javascript** vamos a utilizar como ejemplo una pantalla que muestre el texto “Clic aquí” y que, una vez se haga clic, muestre una alerta diciendo “Hola mundo!”.

Old-old-old-old-style todo en el HTML

Al principio de los tiempos era muy habitual encontrarnos con código javascript dentro de las propias etiquetas HTML, formando parte de un atributo de evento, como el evento *onclick*.

[View Raw Code?](#)

```
1. <!DOCTYPE html>
2. <html lang="es">
3. <head>
4. <meta charset="utf-8" />
5. <title>Incluyendo JS en HTML5</title>
6. </head>
7. <body>
8. <section>
9. <span onclick="alert('hola mundo!');">Clic aquí</span>
10. </section>
11. </body>
12. </html>
```

En este caso, tenemos una etiqueta HTML *span*, con un atributo *onclick*. Esto significa que cuando abras la web en el navegador verás lo siguiente:

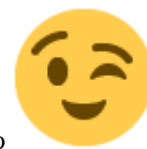


El avanzado clic aquí

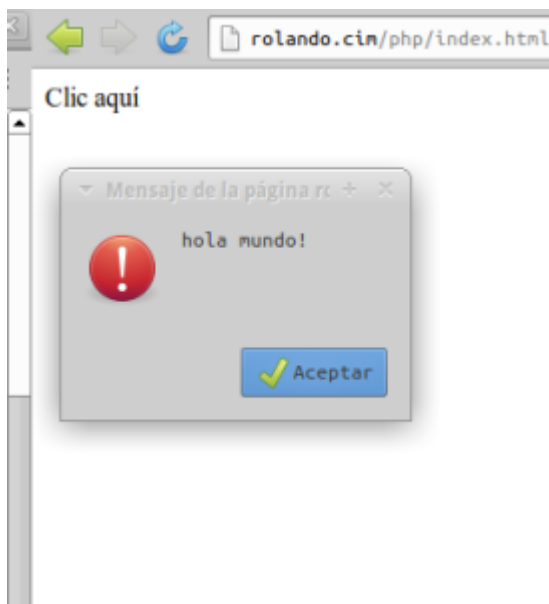
Buah, super-chula la pantalla



Lo importante no es que sea bonito, sino claro



Si vamos a esa pantalla y hacemos clic en “Clic aquí” veríamos lo siguiente:



Impresionante efecto!

Lo que ha ocurrido es que cuando se hizo clic en la frase, se activó el evento *onclick* por lo que el navegador ejecutó el código incluido en el atributo, o sea:

[View Raw Code?](#)

```
1. alert('hola mundo!');
```

Y ese código, es Javascript.

Old-old-old-style todo en el HTML

Aunque el código de ejemplo es muy sencillo, es bastante fácil ver el problema de incluir el código javascript dentro de las etiquetas HTML...

1. Es imposible reutilizar el código javascript. Si quieres tener varios eventos onclick que abran una alerta diciendo “hola mundo!” tendrás que escribir el código JavaScript en cada uno de los eventos... una lata!
2. La claridad del código brilla por su ausencia. En el ejemplo sólo tenemos una etiqueta HTML, pero es fácil imaginarse una pantalla con cientos de etiquetas HTML o un proyecto web de múltiples pantallas HTML... mantener el código de esta forma es una locura... **muerte segura.**

Así que, apareció otra forma de incluir el Javascript... y aquí entra, por primera vez, la etiqueta *script*:

[View Raw Code?](#)

```
1. <!DOCTYPE html>
2. <html lang="es">
3. <head>
4. <meta charset="utf-8" />
5. <title>Incluyendo JS en HTML5</title>
6. <script type="text/javascript">
7.   function alerta() {
8.     alert('hola mundo!');
9.   }
10. </script>
11. </head>
12. <body>
13. <section>
14. <span onclick="alerta();">Clic aquí</span>
15. </section>
16. </body>
17. </html>
```

Bien, el resultado será el mismo que en el primer ejemplo, ya que hace exactamente lo mismo, pero tenemos alguna pequeña diferencia:

- Dentro de la etiqueta *head* tenemos la etiqueta *script* con su apertura y su cierre... con el atributo “type” especificando al navegador que el script incluido dentro de la etiqueta es de tipo **text/javascript** o lo que es lo mismo, javascript.
- Dentro del *script* tenemos nuestro código javascript. En este caso se trata de una función que, una vez llamada, lanzará nuestra alerta.
- Nuestra etiqueta *span* mantiene el atributo *onclick*, pero en vez de tender entro el código javascript para lanzar la alerta, lo que tiene es una llamada a la función que creamos en la cabecera de nuestro documento.

De esta forma se conseguía tener todo el código **javascript** en la cabecera, de forma que fuese más sencillo modificarlo y, sobre todo, permitiendo reutilizar el código ya que podemos tener varias etiquetas HTML con un evento *onclick* que llame a la misma funcion de javascript que hemos declarado.

Old-old-style todo en el HTML

Aunque la última opción indicada de poner el Javascript es una clara mejora sobre la primera, seguimos teniendo el problema de la gestión del código (el problema 2). Para solucionar este problema de claridad y gestión, damos un pasito más en cómo implementar el Javascript en nuestro HTML:

[View Raw Code?](#)

```
1. <!DOCTYPE html>
2. <html lang="es">
3. <head>
4. <meta charset="utf-8" />
5. <title>Incluyendo JS en HTML5</title>
6. <script type="text/javascript">
7. document.getElementById('lanzar_alerta').onclick = function () {
8. alert('¡Hola mundo!');
9. }
10. </script>
11. </head>
12. <body>
13. <section>
14. <span id="lanzar_alerta">Clic aquí</span>
15. </section>
16. </body>
17. </html>
```

Bien, bien... esto va cogiendo forma... nuestra etiqueta *span* ya no tiene el atributo *onclick* y en su lugar le hemos agregado un identificador (ID) y en nuestro código **javascript** hemos creado el código para que el navegador sepa que tiene que lanzar la alerta cuando el usuario haga clic en “Clic aquí”

IMPORTANTE: Este ejemplo es la primera implementación pura del javascript ya que en las dos versiones anteriores estábamos utilizando un atributo de una etiqueta html para indicar que existía código javascript que ejecutarse en el evento onclick. Esta nueva opción sólo utiliza javascript para indicar este hecho.

Probamos nuestro código y comprobamos que el resultado es el de siempre!

mmmm mmmm o no???

Pues no! Tal y como tenemos nuestro código, cuando se pincha en “Click aquí” nuestro navegador nos devuelve un error (no visible por pantalla) como este:

Uncaught TypeError: Cannot set property 'onclick' of null

pffff y aquí empiezan los problemas! Este problema fue insalvable para muchos programadores que no entendían el problema... porque no entendían cómo funcionaba Javascript. Por eso es **VITAL** que uno conozca y entienda el funcionamiento de aquello que programa y las peculiaridades del lenguaje que utiliza. Vale, aquí nadie nace con la lección aprendida y cuando se empieza con un lenguaje nuevo es normal desconocer como funciona... y no vamos a tirarnos meses aprendiendo el funcionamiento del lenguaje antes de usarlo... se usa porque se tiene que usar y se hace lo que se puede. La diferencia está en quienes se molestan en entender lo que programan mientras lo programan y en los que no...

Es **inaceptable** que a un auto-etiquetado como “programador senior” o “experto” en javascript le hables de DOM y que diga ¿lo cual? o que simplemente sepa que exista, pero nada más...

Cuando un navegador carga la web, va descargando y ejecutando el contenido a medida que lo descarga. Sin entrar en detalles, esto significa que nuestro código HTML es leído e interpretado **ANTES** de que el navegador lea nuestra etiqueta *span*, por lo que en ese momento no existe, por lo que se genera el error de **javascript** indicado y, lo más importante, el navegador dejará de interpretar el código javascript, o sea, **no funcionará ningún código js posterior al error**.

En otras palabras: **FUEGO FUEGO FUEEEEEEEEGOOOOO!!!!**

Y qué viene con el fuego? Pues sí, **los bomberos**. Uno de esos grandiosos bomberos dijo algo como...

bueno, si el problema es que el javascript se interpreta antes de que el navegador lea la etiqueta HTML, pongamos el javascript dentro del body, antes de su cierre.

Tal que así:

[View Raw Code?](#)

```
1. <!DOCTYPE html>
2. <html lang="es">
3. <head>
4. <meta charset="utf-8" />
5. <title>Incluyendo JS en HTML5</title>
6. </head>
7. <body>
8. <section>
9. <span id="lanzar_alerta">Clic aquí</span>
10. </section>
11. <script type="text/javascript">
12. document.getElementById('Lanzar_alerta').onClick = function () {
13. alert('hoLa mundo!');
14. }
15. </script>
16. </body>
17. </html>
```

Y como con eso se solucionaba el problema, empezamos a ver el javascript antes del cierre del body... y tristemente se sigue usando mucho, pero mucho esta “solución”.

Rápido y efectivo... pero **NO!!!** Dentro del body sólo deberíamos tener HTML, tenemos el HEAD para los metas, las declaraciones de CSS, los scripts etc. Que el navegador lo lea igual no significa que sea el sitio correcto. Como en cualquier problema de nuestra vida, no debemos buscar la solución rápida, **sino que necesitamos la mejor solución, la más óptima, la ideal**.

Y la ideal para nosotros es que el Javascript esté donde tiene que estar: en el HEAD. Y para eso existe tenemos el maravilloso evento “onload” que podemos ejecutar en el objeto “window”, o sea:

[View Raw Code?](#)

```
1. <!DOCTYPE html>
2. <html lang="es">
3. <head>
4. <meta charset="utf-8" />
5. <title>Incluyendo JS en HTML5</title>
6. <script type="text/javascript">
7. window.onload = function() {
8. document.getElementById('Lanzar_alerta').onClick = function () {
9. alert('hoLa mundo!');
10. }
11. }
12. </script>
13. </head>
14. <body>
15. <section>
16. <span id="lanzar_alerta">Clic aquí</span>
17. </section>
```

```
18. </body>
19. </html>
```

Con esto estamos indicando, en **javascript**, que cuando el navegador termine la carga de la web (window.onload) ejecute una function de javascript que incluye nuestro código, o sea, nuestra llamada al evento onclick de nuestro span. Esto funciona y es correcto!

Bueno, los dos problemas que teníamos con nuestro primer ejemplo están solucionados! Pero, como siempre hay que buscar lo mejor de lo mejor, se fue más allá.

Old-style: El Javascript fuera del HTML

El desarrollo de la programación web y la profesionalización del sector hizo, entre otras cosas, que se quisiese reciclar el código javascript de forma sencilla, que el código javascript utilizado por una web empezó a ser realmente enorme... además los programadores crean “librerías” para poder compartirlas con los compañeros, amigo o con el mundo en general.

Y ante esta situación es **IMPOSIBLE** mantener el código **Javascript** dentro del HTML... la migración del mismo código de una web a otra sería complejo... los documentos a editar serían auténticas bestias en lo que a líneas de código se refiere... y nuestros entornos de desarrollo sufrirían mucho mucho... pupa de la grande.

Además, se supone que un documento HTML debería tener sólo código HTML y no mezclar Javascript, CSS, etc...

Así que, la forma de incluir el javascript en el HTML evolucionó un poquito más:

[View Raw Code?](#)

```
1. <!DOCTYPE html>
2. <html lang="es">
3. <head>
4. <meta charset="utf-8" />
5. <title>Incluyendo JS en HTML5</title>
6. <script type="text/javascript" src="./javascript.js"></script>
7. </head>
8. <body>
9. <section>
10. <span id="lanzar_alerta">Clic aquí</span>
11. </section>
12. </body>
13. </html>
```

Oh, belleza absoluta! En vez de poner el código **Javascript** dentro de la etiqueta *script*, lo que hemos hecho es incluirle a esta etiqueta el atributo src (source) y dándole como valor una URL relativa al documento, para que cargue su contenido como si fuese **javascript**.

Y, por supuesto, hemos creado un fichero llamado *javascript.js* al mismo nivel de nuestro HTML con el código javascript que antes teníamos dentro del *index.html*

[View Raw Code?](#)

```
1. window.onload = function() {
2. document.getElementById('lanzar_alerta').onclick = function () {
3. alert('hola mundo!');
4. }
5. }
```

Bien, objetivo conseguido:

- Nuestro javascript funciona.
- Nuestro javascript está cargado en la cabecera del documento html.
- Nuestro javascript está incluido en un fichero externo.
- Al tener el javascript en un fichero externo, podemos llevarlo a otro proyecto de forma extremadamente sencilla!!

Gracias a esta forma de incluir los ficheros JS podemos incorporar a nuestro sitio web efectos asombrosos en un tiempo récord... usar frameworks como jQuery que nos hacen la vida mucho más sencilla (pero mucho eh!)

Y así estuvimos felices durante toda la vida de nuestros documentos HTML4.01 y XHTML! Pero, amigos, llegó HTML5 y, como debe ser, dimos otra vuelta de tuerca más!

Incluir Javascript en HTML5

Con **HTML5** han venido muchos cambios, muchos orientados a que dediquemos menos tiempo a las “tonterías”.

Aunque la etiqueta script se puede usar para muchas cosas, principalmente se utiliza para incluir código javascript, por lo tanto HTML5 entiende que, si no se dice nada, el script va a cargar código javascript. Así pues, no necesitamos usar el atributo type si vamos a cargar javascript!

[View Raw Code?](#)

```
1. <!DOCTYPE html>
2. <html lang="es">
3. <head>
4. <meta charset="utf-8" />
5. <title>Incluyendo JS en HTML5</title>
6. <script src="./javascript.js"></script>
7. </head>
8. <body>
9. <section>
10. <span id="lanzar_alerta">Clic aquí</span>
11. </section>
12. </body>
13. </html>
```



Es un pequeño cambio, cierto es, pero eso que nos ahorramos

Optimizando la carga de Javascript en HTML5

El “gran problema” cuando implementamos JS en nuestra web es, como ya comentamos, que el código se descarga y se ejecuta según se llega a su declaración.

Y lo más importante, mientras un fichero JS está siendo descargado y ejecutado el navegador web no puede seguir leyendo el documento. O sea: la carga de la web se hace más lenta.

Aunque pueda parecer una tontería... los proyectos web modernos llevan [jQuery](#) como framework... es probable que todo el [jQueryUI](#) para dar una interfaz visual amigable... seguramente una extensión de jQuery para un slider... como tengamos posibilidad de escoger franjas de fechas tendremos un date picker.... en fin... un montón de librerías **Javascript** además del código que nosotros mismos escribamos para interconectar todas esas librerías con nuestro HTML para conseguir la experiencia deseada. Así que... importa, sí que importa este pequeño detalle del **Javascript**.

Para que nos entendamos, pasando nuestro ejemplo a javascript utilizando jquery como framework tendríamos:

[View Raw Code?](#)

```
1. <!DOCTYPE html>
2. <html lang="es">
3. <head>
4. <meta charset="utf-8" />
5. <title>Incluyendo JS en HTML5</title>
6. <script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
7. <script type="text/javascript" src="./javascript.js"></script>
8. </head>
9. <body>
10. <section>
11. <span id="lanzar_alerta">Clic aquí</span>
12. </section>
13. </body>
14. </html>
```

y nuestro javascript.js sería

[View Raw Code?](#)

```
1. $(document).ready(function() {  
2. $('#lanzar_alerta').click(function() {  
3. alert('hola mundo!');  
4. });  
5. });
```

Bien, la realidad es que tenemos dos ficheros javascript, uno es el jquery y otro es nuestro código javascript. Realmente, jquery es una librería que está ahí y necesitamos para que nuestro código funcione, pero es nuestro código javascript quien interactúa con el DOM (vamos, nuestro código HTML, así hablando mal y rápido)... por lo que lo ideal es que nuestro jquery se cargase como si de una imagen se tratase y nuestro javascript lo hiciese de la forma tradicional... o al final del documento, que a fin de cuentas es lo que “forzábamos” con el window.onload y ahora con el \$(document).ready... bueno,



esto es HTML5 así que tenemos carga asincrónica

HTML5 async y defer para JAVASCRIPT

- **defer:** Realmente ya existía en HTML4 y su uso significa que el script se ejecutará una vez se haya cargado la página.
- **async:** Se ejecutará de forma asíncrona, lo que significa que su ejecución no bloqueará el resto de descargas que existan en paralelo.

Gracias a async, nuestros scripts se podrán ejecutar sin bloquear la descarga de la web, peeeeeeeeero tiene un problemilla:

Los scripts ejecutados con async no siguen el orden natural de ejecución, lo que puede generar errores inesperados si no se usa con sentido.

En nuestro ejemplo vamos a utilizar sólo defer:

[View Raw Code?](#)

```
1. <!DOCTYPE html>  
2. <html lang="es">  
3. <head>  
4. <meta charset="utf-8" />  
5. <title>Incluyendo JS en HTML5</title>  
6. <script defer src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>  
7. <script defer type="text/javascript" src="./javascript.js"></script>  
8. </head>  
9. <body>  
10. <section>  
11. <span id="lanzar_alerta">Clic aquí</span>  
12. </section>  
13. </body>  
14. </html>
```

Y así estaría perfecto!!

El uso de async y defer nos puede traer grandes alegrías a la hora de la carga de nuestra web y si esto lo unimos a una maquetación y una declaración de estilos pensando en cómo se ve la web antes y después de que se ejecute nuestro javascript.... podemos obtener un resultado exquisito donde todo vaya cargando perfectamente cuadrado hasta que llegue el momento de ejecutar el JS... y todos los efectos surjan de golpe... bello, muy bello.

Igual en el futuro me animo y explico más en profundidad el async y cómo no se puede utilizar para cargar la librería



jQuery etc etc... si interesa claro-

En el artículo [carga asíncrona de javascript](#) profundizo en las bondades de la carga asíncrona y cómo solucionar su principal problema.

Verdades a medias de este artículo

Vale... alguno pensará que este artículo no es 100% verdad... bueno, cierto es... pero el objetivo no es ser estricto al 100% sino que se pueda entender de forma relativamente sencilla lo explicado.

Para muestra un botón: Vale, vale... Javascript no sólo es un lenguaje que se ejecuta del lado del cliente (navegador web), sino que existen implementaciones para tenerlo del lado del servidor como [node-js](#)



¿Alguien encuentra alguna otra verdad a medias?

Related Posts:

1. [Cómo incluir CSS3 en HTML5](#)
2. [Carga asíncrona de Javascript](#)
3. [localStorage en HTML5. El fin de las cookies](#)
4. [IndexedDB: Tu base de datos local en HTML5](#)

• [6 septiembre, 2013](#)

• [HTML5](#)

• [html5](#)

• [javascript](#)

[Share on facebook](#) [Share on twitter](#) [Share on linkedin](#) [Share on delicious](#) [Share on google+](#) [Share on pinterest](#) [Share on evernote](#) [Share on tumblr](#) [Share on blogger](#) [Share on wordpress](#) [Share on email](#) [Share on favorites](#) [More Sharing Services](#) [2](#)

Navegación de entradas

← [Pay with Tweet disponible como módulo para Drupal!](#) [localStorage en HTML5. El fin de las cookies](#) →

25 pensamientos en “Cómo incluir Javascript en HTML5”



1. [JoS11](#) 11 noviembre, 2013 en 22:07

[Responder](#) ↓

Gracias por la info. Me sirvió




2. [ISerra](#) 30 noviembre, 2013 en 20:18

[Responder](#) ↓

Muy buen post!

No es lo que buscaba pero ha sido entretenido. Muy bien explicado

Felicidades!!

3.  **lorens** 16 diciembre, 2013 en 19:03
[Responder](#) ↓


muy bueno pero me dio bastante sueño

4.  **Asier** 20 diciembre, 2013 en 16:37
[Responder](#) ↓

Muy bien explicado.
¿Podrías seguir más adelante?




Como introducción... mi cuerpo me pide más

5.  **lweb** 27 abril, 2014 en 18:15
[Responder](#) ↓


muy bueno me gustaría que explicaras sobre async. pdta: explicas muy bien

1.  **rolando.caldas** 28 abril, 2014 en 15:40
[Responder](#) ↓

Hola! Me apunto el tema y lo pongo en la lista de artículos a escribir en el blog, a ver si puedo llegar pronto a él!

6.  **maria** 5 mayo, 2014 en 23:10
[Responder](#) ↓

no logro entender...ayudaaaaa!!!

1.  **rolando.caldas** 5 mayo, 2014 en 23:55
[Responder](#) ↓

algo un poquito más concreto?

7.  **carlos caldeorn** 21 mayo, 2014 en 0:15
[Responder](#) ↓

muy bueno

8.  **Diego Rodríguez** 2 junio, 2014 en 2:55
[Responder](#) ↓

Gracias amigo, muy clara la información



9. **Marco Gallardo Casu** 12 junio, 2014 en 13:45

[Responder](#) ↓

Muy bueno el artículo, tengo que empezar a hacer una aplicación web con HTML5 + CSS + JavaScript con conexiones también a una base de datos y me quedé en HTML4 y CSS, y no había visto JavaScript hasta ahora así que me sirve mucho para empezar bien desde el principio.



Gracias por la información, muy bien explicada además



10. **Quique** 10 julio, 2014 en 10:22

[Responder](#) ↓

Muy buen artículo. Enhorabuena, y muchas gracias.



11. **Ramón** 20 septiembre, 2014 en 13:32

[Responder](#) ↓

Muy bien explicado. Enhorabuena.



12. **Edwin Raymundo** 18 octubre, 2014 en 21:23

[Responder](#) ↓

Mi mas sincera felicitación. Por lo que pude leer de tu post, tienes mucha práctica y experiencia en cuanto a Javascript. Estoy en un proceso de aprendizaje de toda esta temática, te agradezco por la forma en que explicas, y espero continuar en comunicación. Saludos desde Guatemala.



13. **Franco Valdes** 3 noviembre, 2014 en 17:20

[Responder](#) ↓

Algo que no entiendo es que cuando imprimes código html con jquery, en realidad no existe, hay alguna forma de volverlo mmm mas estable ? porque en realidad se ve pero al trabajarlo en tiempo real no se puede utilizar tan bien



1. **rolando.caldas** 3 noviembre, 2014 en 19:21

[Responder](#) ↓

Hola Franco!

En realidad, el código que cargas vía jQuery sí que existe. Lo que ocurre es que cuando imprimes HTML desde jQuery, éste evento se lleva a cabo una vez la página ha sido descargada y ejecutada por el navegador. El código fuente que recibe el navegador no incluye ese HTML, lo cual es normal, puesto que ese HTML se imprime del lado del cliente (navegador) y no del servidor (código fuente). Lo que ocurre es que, una vez se carga la página, el Javascript (en tu caso vía jQuery) altera el DOM de la página (a grosso modo el árbol HTML), incorporando nuevos contenidos. Estos contenidos no se pueden ver desde el código fuente, pero sí desde el código interpretado por el navegador. Para acceder a este último, sólo debes ejecutar las herramientas de desarrollador que tiene tu navegador (tanto Firefox, como Chrome, como IE).

saludos!



14. **Guillermo Suarez** 18 enero, 2015 en 16:37

[Responder](#) ↓

Muy buen explicación !!!!!

Tengo solo una duda, he utilizado tu información para cargar Datalist en HTML5 y me ha cuncionado perfecto.

Pero solamente he podido cargar un datalist de esta manera, cuando quiero hacer otro script e importarlo para que me



cargue 2 datalist no me funciona

Estos son los q importo del HEAD

Y estos son los scripts que estan en la carpeta JavaScripts

carcarListaSlots.js

```
window.onload = function() {
```

```
var options1 = ""  
var slot = new Array(16)
```

```
for(var i = 0; i < slot.length; i++)  
slot[i]= i;
```

```
for(var i = 0; i < slot.length; i++)  
options1 += ";
```

```
document.getElementById("listaSlots").innerHTML = options1;  
}
```

cargarListaPuertos

```
window.onload = function() {
```

```
var options1 = ""  
var puerto = new Array(64)
```

```
for(var i = 0; i < puerto.length; i++)  
puerto[i]= i;
```

```
for(var i = 0; i < puerto.length; i++)  
options1 += ";
```

```
document.getElementById("listaPuertos").innerHTML = options1;  
}
```

Es como si no se pudiera instancias mas la funcion que le paso a window.onload.

Probe intercambiar las funciones y me carga la lista de Puertos en lugar de la de Slots.

Es decir que los scrips funcionan y los datalist que los llaman también.

Luego supuse que solamente se podria llamar a una sola window.onload = function()

e hice los 2 script en uno asi

```
window.onload = function() {  
  
var options1 = ""  
var slot = new Array(16)  
var options2 = ""  
var puerto = new Array(64)  
  
for(var i = 0; i < slot.length; i++)  
slot[i]= i;  
  
for(var i = 0; i < slot.length; i++)  
options1 += "  
  
for(var i = 0; i < puerto.length; i++)  
puerto[i]= i;  
  
for(var i = 0; i < puerto.length; i++)  
options2 += "  
  
document.getElementById("listaPuertos").innerHTML = options2;  
document.getElementById("listaSlots").innerHTML = options1;  
  
}
```

Pero sigue cargandome solamente el datalist de Slots

Alguna sugerencia ???

Muchas gracias por el Post fue muy educativo.

Slds.



1. **rolando.caldas** 20 enero, 2015 en 1:06
[Responder](#) ↓

Buenas,

Lo que ocurre es que estás sobrescribiendo window.onload. Si quieres que se ejecuten varias funciones cuando se dispare el evento onload, lo que necesitas es agregar un listener al evento.

El HTML sería:

[View Raw Code?](#)

```
1. <!DOCTYPE html>  
2. <html>  
3. <head>  
4. <title>DataList</title>  
5. <script src="cargaListaPuertos.js"></script>  
6. <script src="cargaListaSlots.js"></script>  
7. </head>  
8. <body>  
9. <input list="listaPuertos" placeholder="Lista Puertos">  
10. <datalist id="listaPuertos">  
11.  
12. </datalist>  
13. <input list="listaSlots" placeholder="Lista Slots">  
14. <datalist id="listaSlots">  
15.
```

```
16. </datalist>
17. </body>
18. </html>
```

El fichero cargaListaPuertos.js

[View Raw Code?](#)

```
19. function loadPuertos() {
20.   var options1 = "";
21.   var puerto = new Array(64);
22.
23.   for(var i = 0; i < puerto.length; i++) {
24.     puerto[i] = i;
25.     options1 += '&#039;';
26.   }
27.
28.   document.getElementById("listaPuertos").innerHTML = options1;
29. }
30.
31. if (window.addEventListener) {
32.   window.addEventListener('load', loadPuertos, false);
33. } else if (window.attachEvent) {
34.   window.attachEvent('onload', loadPuertos );
35. }
```

y el cargaListaSlots.js

[View Raw Code?](#)

```
36. function loadSlots() {
37.   var options1 = "";
38.   var slot = new Array(16);
39.
40.   for(var i = 0; i < slot.length; i++) {
41.     slot[i] = i;
42.     options1 += '&#039;';
43.   }
44.
45.   document.getElementById("listaSlots").innerHTML = options1;
46. }
47.
48. if (window.addEventListener) {
49.   window.addEventListener('load', loadSlots, false);
50. } else if (window.attachEvent) {
51.   window.attachEvent('onload', loadSlots );
52. }
```

15.  **javier** [22 enero, 2015 en 21:13](#)

[Responder](#) ↓

Tu manera de explicar es muy buena, me gustaria que profundizaras mas sobre async y defer gracias.

1.  **rolando.caldas** [22 enero, 2015 en 23:12](#)

[Responder](#) ↓

Muchas gracias, creo que te resultará interesante: <https://rolandocaldas.com/html5/carga-asincrona-de-javascript>

16.  **Jorge** [7 febrero, 2015 en 4:52](#)

[Responder](#) ↓

Muy buena tu explicación, justo buscaba como integrar un js externo en html5. La duda de novato, si tengo varias rutinas js, las tengo que hacer en programas separados o hay una manera de incluirlas en un solo archivo, y si se puede como sería el

llamado a cada evento.
Muchas gracias.



1. **rolando.caldas** 11 febrero, 2015 en 0:31
[Responder](#) ↓

Buenas!

No tienes por qué ponerlas separadas, pueden ir todas juntas en un fichero, ahí depende de cómo quieras organizarlo, supongo que por lo que comentas sería más útil tenerlos separados. Luego sólo sería llamar a la función directamente desde el propio fichero o hacerlo a través de un eventListener al load del documento, si esto



último te suena a chino, por ahora lo llamas directamente



17. **RoccoDylan** 10 marzo, 2015 en 1:24
[Responder](#) ↓

Me gusto mucho el post, estaria bién más entradas de ejemplos interesantes con javascript.
Un saludo!



18. **jair** 9 mayo, 2015 en 15:49
[Responder](#) ↓

HOLA QUE TAL, TENGO UNA DUDA, TENGO QUE HACER UNA APLICACION PARA iOS PERO NO CUENTO CON UNA MACBOOK O iMAC Y ANDE BUSCANDO COMO PROGRAMAR EN WINDOWS, HASTA QUE UN PROFESOR ME COMENTO QUE PODIA HACER LA PROGRAMACION EN HTML5 Y JAVASCRIPT Y DESPUES PASAR EL CODIGO EN UNA MAC PARA SUBIR LA APLICACION A APP STORE, MI PREGUNTA ES: ES POSIBLE HACER ESO?



19. **Angelica** 28 mayo, 2015 en 0:59
[Responder](#) ↓

Una muy buena forma de explicar, para las personas que hasta ahora empezamos en el tema. Gracias