



2.1. Conceptos básicos

Vamos a ver cómo incluir JavaScript en una página web:

“código” entre etiquetas `<script></script>`



```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<title></title>
</head>
<body>
  <script>
    document.write("Código JS dentro de HTML");
  </script>
</body>
</html>
```



```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <p onclick="alert('Un mensaje lanzado por JavaScript')">
    Esto es un párrafo con un evento asociado.
  </p>
</body>
</html>
```

“src” en el head o en el body

Se trata de la **inclusión de ficheros .js**, la forma más común y que proporciona más claridad, modularidad y facilidad de mantenimiento del código.


Fichero micodigo.js


```
document.write("Código JS en un fichero externo");
```

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <script src="micodigo.js"></script>
  <title>Incluyendo JS externo</title>
</head>
<body>
  <p>
    Un párrafo cualquiera.
  </p>
</body>
</html>
```

En cuanto a rendimiento, lo mejor es ponerla **al final del <body>** para que no se detenga el renderizado de la página mientras se descarga y se ejecuta el código js.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <script src="micodigo.js"></script>
  <title>Incluyendo JS externo</title>
</head>
<body>
  <p>
    Un párrafo cualquiera.
  </p>
</body>
</html>
```

A green arrow originates from the right side of the slide, pointing towards the code block. It then curves downwards and to the right, pointing specifically at the closing tag of the script element, </script>, which is highlighted in green in the code block. This visualizes the recommendation to place the script at the end of the body for better performance.




También podemos ponerlo en el <head> pero usando los atributos **async** y/o **defer**.

Cuando el **navegador carga** el HTML y se encuentra con una etiqueta **<script>...</script>**, no puede continuar construyendo el DOM. Debe ejecutar el script en el momento.

Lo mismo sucede con los scripts externos **<script src="..."></script>**, el navegador tiene que esperar hasta que el script sea descargado, ejecutarlo y solo después procesa el resto de la página.



Esto nos lleva a dos importantes problemas:

- Los scripts no pueden ver los elementos del DOM que se encuentran debajo de él por lo que no pueden agregar controladores de eventos, etc.
 - Si hay un script muy pesado en la parte superior de la página, este “bloquea la página”. Los usuarios no pueden ver el contenido de la página hasta que sea descargado y ejecutado.
- 

```
<p>...contenido previo al script...</p>
```

```
<script src="https://javascript.info/article/script-async-defer/long.js?speed=1"></script>
```

```
<!-- Esto no es visible hasta que el script sea cargado -->
```


```
<p>...contenido posterior al script...</p>
```

Posible solución → podemos poner el script en la parte inferior de la página por lo que podrá ver los elementos sobre él y no bloqueará la visualización del contenido de la página.

```
<body>
```

```
...todo el contenido está arriba del script...
```

```
  <script src="https://javascript.info/article/script-async-defer/long.js?speed=1"></script>  
</body>
```

Problema → el navegador solo se dará cuenta del script (y podrá empezar a descargarlo) después de descargar todo el documento HTML. Para documentos HTML extensos eso puede ser un retraso notable.

Este tipo de cosas son imperceptibles para las personas que usan conexiones muy rápidas, pero muchas personas en el mundo todavía tienen velocidades de internet lentas y utilizan una conexión de internet móvil que está lejos de ser perfecta.

Solución definitiva → uso de los atributos **async** y/o **defer**

Atributo defer

Indica al navegador que no espere por el script. En lugar de ello, debe seguir procesando el HTML, construir el DOM. El script carga “en segundo plano” y se ejecuta cuando el DOM esta completo.

```
<p>...contenido previo script...</p>  
  
<script defer src="https://javascript.info/article/script-async-defer/long.js?speed=1">  
  // ...  
</script>  
  
<!-- Inmediatamente visible -->  
<p>...contenido posterior al script...</p>
```

Ventajas:

- Los scripts con **defer** nunca bloquean la página.
- Los scripts con **defer** siempre se ejecutan cuando el DOM está listo.
- Los scripts con **defer** mantienen su orden relativo.

Si tenemos dos scripts diferidos, **long.js** (muchas líneas de código) y luego **small.js** (más corto):

```
<script defer src="https://javascript.info/article/script-async-defer/long.js"></scri  
<script defer src="https://javascript.info/article/script-async-defer/small.js"></scr
```

El navegador analiza la página en busca de scripts y los descarga en paralelo para mejorar el rendimiento.

El atributo **defer** asegura que el orden relativo se mantenga. Entonces incluso si **small.js** se carga primero, aún espera y se ejecuta después de **long.js**.

→ Es importante para casos donde necesitamos cargar una librería JavaScript y entonces un script que depende de ella.

Atributo **async**

El atributo **async** es de alguna manera como **defer**. También hace el script no bloqueante. Pero tiene importantes diferencias de comportamiento.

El atributo **async** significa que el **script** es **completamente independiente**:

- El navegador no se bloquea con scripts **async** (como **defer**).
- Otros scripts no esperan por scripts **async**, y scripts **async** no espera por ellos.

En otras palabras, los scripts **async** cargan en segundo plano y se ejecutan cuando están listos. El DOM y otros scripts no esperan por ellos, y ellos no esperan por nada. Un script totalmente independiente que se ejecuta en cuanto se ha cargado.

```
<p>...contenido previo a los scripts...</p>
```

```
<script>
```

```
  document.addEventListener('DOMContentLoaded', () => alert("¡DOM listo!"));
```

```
</script>
```

```
<script async src="https://javascript.info/article/script-async-defer/long.js"></script>
```

```
<script async src="https://javascript.info/article/script-async-defer/small.js"></script>
```

```
<p>...contenido posterior a los scripts...</p>
```


Ejemplo similar al que vimos con **defer**: Dos scripts **long.js** y **small.js**, pero ahora con **async** en lugar de defer.

Los unos no esperan por los otros. El que cargue primero (probablemente small.js), se ejecuta primero.

```
<p>...contenido previo a los scripts...</p>
```

```
<script>
```

```
  document.addEventListener('DOMContentLoaded', () => alert("¡DOM listo!"));
```

```
</script>
```

```
<script async src="https://javascript.info/article/script-async-defer/long.js"></script>
```

```
<script async src="https://javascript.info/article/script-async-defer/small.js"></script>
```

```
<p>...contenido posterior a los scripts...</p>
```

Scripts dinámicos

Hay otra manera importante de agregar un script a la página.

Podemos **crear un script y agregarlo dinámicamente al documento** usando JavaScript:

```
let script = document.createElement('script');  
script.src = "/article/script-async-defer/long.js";  
document.body.append(script); // (*)
```

El script comienza a cargar tan pronto como es agregado al documento (*).



Los scripts dinámicos se comportan como async por defecto:

- Ellos no esperan a nadie y nadie espera por ellos.
- El script que carga primero se ejecuta primero (load-first order).

Esto puede ser cambiado si explícitamente establecemos **script.async=false**. Así los scripts serán ejecutados en el orden del documento, tal como en **defer**.

```
function loadScript(src) {  
    let script = document.createElement('script');  
    script.src = src;  
    script.async = false;  
    document.body.append(script);  
}
```

```
// long.js se ejecuta primero a causa del async=false  
loadScript("/article/script-async-defer/long.js");  
loadScript("/article/script-async-defer/small.js");
```

En la práctica,

defer es usado para scripts que necesitan todo el DOM y/o si su orden de ejecución relativa es importante.

async es usado para scripts independientes, como contadores y anuncios donde el orden de ejecución no importa.

La página sin scripts debe ser utilizable

Si usas **defer** o **async**, el usuario verá la página *antes* de que el script sea cargado.

En tal caso algunos componentes gráficos probablemente no estén listos.

No olvides poner alguna señal de “cargando” y deshabilitar los botones que aún no estén funcionando.

Esto permite al usuario ver claramente qué puede hacer en la página y qué está listo y qué no.

Estructura de carpetas de un proyecto web

