**Сайты:**

<https://learn.javascript.ru>

# Основы JavaScript

Программы на языке JavaScript можно вставить в любое место HTML при помощи тега SCRIPT

<script>

alert( 'Привет, Мир!' );

</script>

Чтобы отобразить русский шрифт в HEAD  <meta charset="utf-8">. Если вы будете открывать файл с диска, то именно он укажет браузеру кодировку.

# Внешние скрипты, порядок исполнения

Если JavaScript-кода много – его выносят в отдельный файл, который подключается в HTML:

<script src="/path/to/script.js"></script>

Чтобы подключить несколько скриптов, используйте несколько тегов:

<script src="/js/script1.js"></script>

<script src="/js/script2.js"></script>

...

## [Асинхронные скрипты: defer/async](https://learn.javascript.ru/external-script" \l "asinhronnye-skripty-defer-async)

Браузер загружает и отображает HTML постепенно. Особенно это заметно при медленном интернет-соединении: браузер не ждёт, пока страница загрузится целиком, а показывает ту часть, которую успел загрузить.

Если браузер видит тег <script>, то он по стандарту обязан сначала выполнить его, а потом показать оставшуюся часть страницы.

Например, в примере ниже – пока все кролики не будут посчитаны – нижний <p> не будет показан:

<!DOCTYPE HTML>

<html>

<head>

<meta charset="utf-8">

</head>

<body>

<p>Начинаем считать:</p>

<script>

alert( 'Первый кролик!' );

alert( 'Второй кролик!' );

alert( 'Третий кролик!' );

</script>

<p>Кролики посчитаны!</p>

</body>

</html>

Такое поведение называют «синхронным». Как правило, оно вполне нормально, но есть важное следствие.

**Если скрипт – внешний, то пока браузер не выполнит его, он не покажет часть страницы под ним.**

То есть, в таком документе, пока не загрузится и не выполнится big.js, содержимое <body> будет скрыто:

<html>

<head>

<script src="big.js"></script>

</head>

<body>

Этот текст не будет показан, пока браузер не выполнит big.js.

</body>

</html>

И здесь вопрос – действительно ли мы этого хотим? То есть, действительно ли оставшуюся часть страницы нельзя показывать до загрузки скрипта?

Есть ситуации, когда мы не только НЕ хотим такой задержки, но она даже опасна.

Например, если мы подключаем внешний скрипт, который показывает рекламу или вставляет счётчик посещений, а затем идёт наша страница. Конечно, неправильно, что пока счётчик или реклама не подгрузятся – оставшаяся часть страницы не показывается. Счётчик посещений не должен никак задерживать отображение страницы сайта. Реклама тоже не должна тормозить сайт и нарушать его функциональность.

А что, если сервер, с которого загружается внешний скрипт, перегружен? Посетитель в этом случае может ждать очень долго!

Вот пример, с подобным скриптом (стоит искусственная задержка загрузки):

<p>Важная информация не покажется, пока не загрузится скрипт.</p>

<script src="https://js.cx/hello/ads.js?speed=0"></script>

<p>...Важная информация!</p>

Что делать?

Можно поставить все подобные скрипты в конец страницы – это уменьшит проблему, но не избавит от неё полностью, если скриптов несколько. Допустим, в конце страницы 3 скрипта, и первый из них тормозит – получается, другие два его будут ждать – тоже нехорошо.

Кроме того, браузер дойдёт до скриптов, расположенных в конце страницы, они начнут грузиться только тогда, когда вся страница загрузится. А это не всегда правильно. Например, счётчик посещений наиболее точно сработает, если загрузить его пораньше.

Поэтому «расположить скрипты внизу» – не лучший выход.

Кардинально решить эту проблему помогут атрибуты async или defer:

**Атрибут async**

Поддерживается всеми браузерами, кроме IE9-. Скрипт выполняется полностью асинхронно. То есть, при обнаружении <script async src="..."> браузер не останавливает обработку страницы, а спокойно работает дальше. Когда скрипт будет загружен – он выполнится.

**Атрибут defer**

Поддерживается всеми браузерами, включая самые старые IE. Скрипт также выполняется асинхронно, не заставляет ждать страницу, но есть два отличия от async.

Первое – браузер гарантирует, что относительный порядок скриптов с defer будет сохранён.

То есть, в таком коде (с async) первым сработает тот скрипт, который раньше загрузится:

<script src="1.js" async></script>

<script src="2.js" async></script>

А в таком коде (с defer) первым сработает всегда 1.js, а скрипт 2.js, даже если загрузился раньше, будет его ждать.

<script src="1.js" defer></script>

<script src="2.js" defer></script>

Поэтому атрибут defer используют в тех случаях, когда второй скрипт 2.js зависит от первого 1.js, к примеру – использует что-то, описанное первым скриптом.

Второе отличие – скрипт с defer сработает, когда весь HTML-документ будет обработан браузером.

Например, если документ достаточно большой…

<script src="async.js" async></script>

<script src="defer.js" defer></script>

Много много много букв

…То скрипт async.js выполнится, как только загрузится – возможно, до того, как весь документ готов. А defer.jsподождёт готовности всего документа.

Это бывает удобно, когда мы в скрипте хотим работать с документом, и должны быть уверены, что он полностью получен.

**async вместе с defer**

При одновременном указании async и defer в современных браузерах будет использован только async, в IE9- – только defer (не понимает async).

**Атрибуты async/defer – только для внешних скриптов**

Атрибуты async/defer работают только в том случае, если назначены на внешние скрипты, т.е. имеющие src.

При попытке назначить их на обычные скрипты <script>…</script>, они будут проигнорированы.

Тот же пример с async:

<p>Важная информация теперь не ждёт, пока загрузится скрипт...</p>

<script async src="https://js.cx/hello/ads.js?speed=0"></script>

<p>...Важная информация!</p>

При запуске вы увидите, что вся страница отобразилась тут же, а alert из внешнего скрипта появится позже, когда загрузится скрипт.

**Эти атрибуты давно «в ходу»**

Большинство современных систем рекламы и счётчиков знают про эти атрибуты и используют их.

Перед вставкой внешнего тега <script> понимающий программист всегда проверит, есть ли у него подобный атрибут. Иначе медленный скрипт может задержать загрузку страницы.

**Забегая вперёд**

Для продвинутого читателя, который знает, что теги <script> можно добавлять на страницу в любой момент при помощи самого javascript, заметим, что скрипты, добавленные таким образом, ведут себя так же, как async. То есть, выполняются как только загрузятся, без сохранения относительного порядка.

Если же нужно сохранить порядок выполнения, то есть добавить несколько скриптов, которые выполнятся строго один за другим, то используется свойство script.async = false.

Выглядит это примерно так:

function addScript(src){

var script = document.createElement('script');

script.src = src;

script.async = false; // чтобы гарантировать порядок

document.head.appendChild(script);

}

addScript('1.js'); // загружаться эти скрипты начнут сразу

addScript('2.js'); // выполнятся, как только загрузятся

addScript('3.js'); // но, гарантированно, в порядке 1 -> 2 -> 3

Более подробно работу со страницей мы разберём во второй части учебника.

**[Итого](https://learn.javascript.ru/external-script" \l "itogo)**

* Скрипты вставляются на страницу как текст в теге <script>, либо как внешний файл через <script src="путь"></script>
* Специальные атрибуты async и defer используются для того, чтобы пока грузится внешний скрипт – браузер показал остальную (следующую за ним) часть страницы. Без них этого не происходит.
* Разница между async и defer: атрибут defer сохраняет относительную последовательность скриптов, а async – нет. Кроме того, defer всегда ждёт, пока весь HTML-документ будет готов, а async – нет.
* В примере ниже подключены два скрипта small.js и big.js.
* Если предположить, что small.js загружается гораздо быстрее, чем big.js – какой выполнится первым?
* <script src="big.js"></script>
* <script src="small.js"></script>
* А вот так?
* <script async src="big.js"></script>
* <script async src="small.js"></script>
* А так?
* <script defer src="big.js"></script>
* <script defer src="small.js"></script>