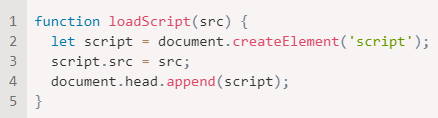
Многие действия в JavaScript асинхронные.

Например:

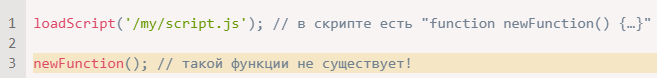


Эта функция загружает на страницу новый скрипт. Когда в тело документа добавится конструкция <script src="…">, браузер загрузит скрипт и выполнит его.



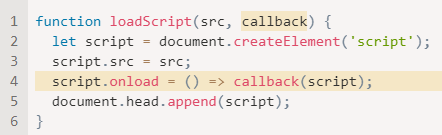
Если после вызова loadScript(…) есть какой-то код, то он не будет ждать, пока скрипт загрузится.

Допустим скрипт объявляет новую функцию, которую мы хотим выполнить, вызвать новую функцию сразу после строки вызова скрипта не сработает:



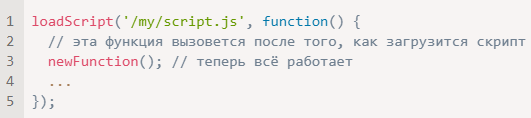
у браузера не было времени загрузить скрипт. Сейчас функция loadScript никак не позволяет отследить момент загрузки. Скрипт загружается, а потом выполняется. Но нам нужно точно знать, когда это произойдёт, чтобы использовать функции и переменные из этого скрипта.

Можно передать функцию callback вторым аргументом в loadScript, чтобы вызвать её, когда скрипт загрузится:



Событие onload выполняет функцию после загрузки и выполнения скрипта.

Теперь, нужна функция из скрипта, нужно вызвать ее в колбэке:



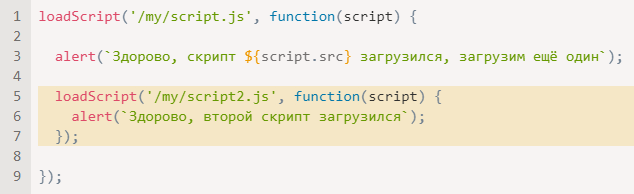
Смысл такой: вторым аргументом передаётся функция (обычно анонимная), которая выполняется по завершении действия. Например:



Такое написание называют **асинхронным программированием с использованием колбэков.** В функции, которые выполняют какие-либо асинхронные операции, передаётся аргумент callback — функция, которая будет вызвана по завершению асинхронного действия.

**[Колбэк в колбэке](https://learn.javascript.ru/callbacks" \l "kolbek-v-kolbeke)**

Чтобы загрузить два скрипта один за другим - сначала первый, а за ним второй, первое, что приходит в голову, вызвать loadScript ещё раз уже внутри колбэка:

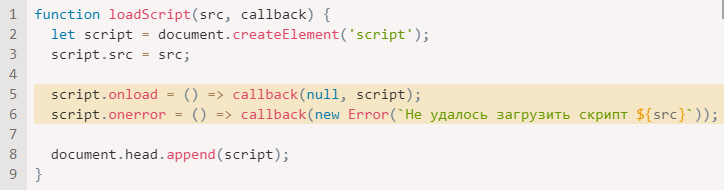


Когда внешняя функция loadScript выполнится, вызовется та, что внутри колбэка.

Этот вариант подойдёт, когда у нас одно-два действия, но для большего количества уже не удобно.

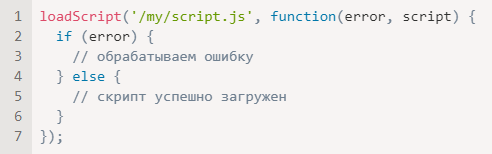
## [Перехват ошибок](https://learn.javascript.ru/callbacks" \l "perehvat-oshibok)

версия loadScript, которая умеет отслеживать ошибки загрузки:



Мы вызываем callback(null, script) в случае успешной загрузки и callback(error), если загрузить скрипт не удалось.

 Например:



Такой подход распространён и называется **«колбэк с первым аргументом-ошибкой» («error-first callback»)**.

Правила таковы:

1. Первый аргумент функции callback зарезервирован для ошибки. В этом случае вызов выглядит вот так: callback(err).
2. Второй и последующие аргументы — для результатов выполнения. В этом случае вызов выглядит вот так: callback(null, result1, result2…).

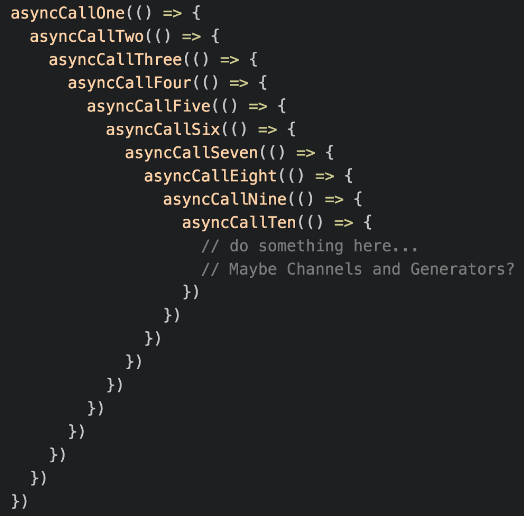
Одна и та же функция callback используется и для информирования об ошибке, и для передачи результатов.

## [Адская пирамида вызовов](https://learn.javascript.ru/callbacks" \l "adskaya-piramida-vyzovov)

Для одного или двух вложенных вызовов функции с асинхронными колбэками выглядят нормально.

Чем больше вложенных вызовов, тем наш код будет иметь всё большую вложенность, которую сложно поддерживать, особенно код, содержащий другие цепочки вызовов, условия и т.д.

Иногда это называют «адом колбэков» или «адской пирамидой колбэков».



Пирамида вложенных вызовов растёт вправо с каждым асинхронным действием.

Такой подход к написанию кода не приветствуется.

Можно попытаться решить эту проблему, изолируя каждое действие в отдельную функцию, например:



Этот код делает всё то же самое, но вложенность отсутствует, потому что все действия вынесены в отдельные функции.

Код абсолютно рабочий, но кажется разорванным на куски. Его трудно читать. Приходится прыгать глазами между кусками кода, когда пытаешься его прочесть. Это неудобно, особенно, если читатель не знаком с кодом и не знает, что за чем следует.

Кроме того, все функции step\* одноразовые, и созданы лишь только, чтобы избавиться от «адской пирамиды вызовов». Никто не будет их переиспользовать где-либо ещё. Таким образом, мы, кроме всего прочего, засоряем пространство имён.

Есть способ получше – промисы.