

**Часть 1**

# **Язык JavaScript**

# **JS**

**Илья Кантор**

**Сборка от 24 апреля 2024 г.**

Последняя версия учебника находится на сайте <https://learn.javascript.ru>.

Мы постоянно работаем над улучшением учебника. При обнаружении ошибок пишите о них на [нашем баг-трекере](#).

- [Введение](#)
  - [Введение в JavaScript](#)
  - [Справочники и спецификации](#)
  - [Редакторы кода](#)
  - [Консоль разработчика](#)

Здесь вы можете изучить JavaScript, начиная с нуля и заканчивая продвинутыми концепциями вроде ООП.

Мы сосредоточимся на самом языке, изредка добавляя заметки о средах его исполнения.

## Введение

Про язык JavaScript и окружение для разработки на нём.

## Введение в JavaScript

Давайте посмотрим, что такого особенного в JavaScript, чего можно достичь с его помощью и какие другие технологии хорошо с ним работают.

## Что такое JavaScript?

Изначально *JavaScript* был создан, чтобы «сделать веб-страницы живыми».

Программы на этом языке называются *скриптами*. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы.

Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска.

Это отличает JavaScript от другого языка – [Java](#) .

### Почему JavaScript?

Когда JavaScript создавался, у него было другое имя – «LiveScript». Однако, язык Java был очень популярен в то время, и было решено, что позиционирование JavaScript как «младшего брата» Java будет полезно.

Со временем JavaScript стал полностью независимым языком со своей собственной спецификацией, называющейся [ECMAScript](#) , и сейчас не имеет никакого отношения к Java.

Сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» [JavaScript](#) .

У браузера есть собственный движок, который иногда называют «виртуальная машина JavaScript».

Разные движки имеют разные «кодовые имена». Например:

- [V8](#) – в Chrome, Opera и Edge.
- [SpiderMonkey](#) – в Firefox.
- ...Ещё есть «Chakra» для IE, «JavaScriptCore», «Nitro» и «SquirrelFish» для Safari и т.д.

Эти названия полезно знать, так как они часто используются в статьях для разработчиков. Мы тоже будем их использовать. Например, если «функциональность X поддерживается V8», тогда «X», скорее всего, работает в Chrome, Opera и Edge.

### Как работают движки?

Движки сложны. Но основы понять легко.

1. Движок (встроенный, если это браузер) читает («парсит») текст скрипта.
2. Затем он преобразует («компилирует») скрипт в машинный язык.
3. После этого машинный код запускается и работает достаточно быстро.

Движок применяет оптимизации на каждом этапе. Он даже просматривает скомпилированный скрипт во время его работы, анализируя проходящие через него данные, и применяет оптимизации к машинному коду, полагаясь на полученные знания. В результате скрипты работают очень быстро.

## Что может JavaScript в браузере?

Современный JavaScript – это «безопасный» язык программирования. Он не предоставляет низкоуровневый доступ к памяти или процессору, потому что изначально был создан для браузеров, не требующих этого.

Возможности JavaScript сильно зависят от окружения, в котором он работает. Например, [Node.JS](#) поддерживает функции чтения/записи произвольных файлов, выполнения сетевых запросов и т.д.

В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером.

Например, в браузере JavaScript может:

- Добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили.
- Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.
- Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы (технологии [AJAX](#) и [COMET](#)).
- Получать и устанавливать куки, задавать вопросы посетителю, показывать сообщения.
- Запоминать данные на стороне клиента («local storage»).

## Чего НЕ может JavaScript в браузере?

Возможности JavaScript в браузере ограничены ради безопасности пользователя. Цель заключается в предотвращении доступа недобросовестной веб-страницы к личной информации или нанесению ущерба данным пользователя.

Примеры таких ограничений включают в себя:

- JavaScript на веб-странице не может читать/записывать произвольные файлы на жёстком диске, копировать их или запускать программы. Он не имеет прямого доступа к системным функциям ОС.

Современные браузеры позволяют ему работать с файлами, но с ограниченным доступом, и предоставляют его, только если пользователь выполняет определённые действия, такие как «перетаскивание» файла в окно браузера или его выбор с помощью тега `<input>`.

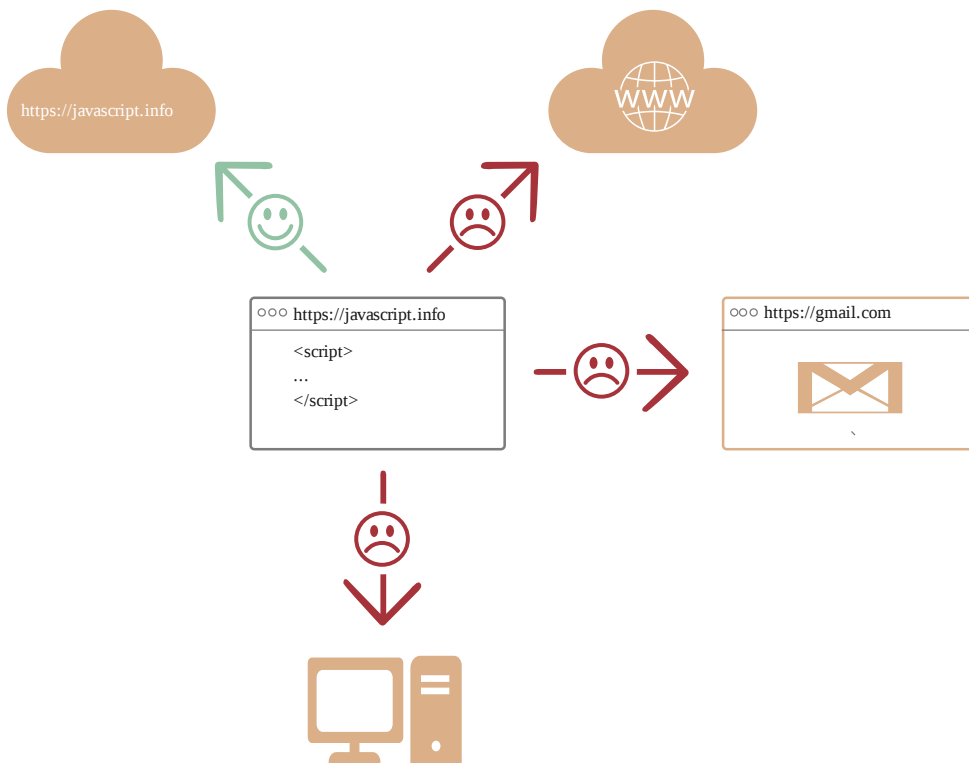
Существуют способы взаимодействия с камерой/микрофоном и другими устройствами, но они требуют явного разрешения пользователя. Таким образом, страница с поддержкой JavaScript не может незаметно включить веб-камеру, наблюдать за происходящим и отправлять информацию в [ФСБ](#) [↗](#).

- Различные окна/вкладки не знают друг о друге. Иногда одно окно, используя JavaScript, открывает другое окно. Но даже в этом случае JavaScript с одной страницы не имеет доступа к другой, если они пришли с разных сайтов (с другого домена, протокола или порта).

Это называется «Политика одинакового источника» (Same Origin Policy). Чтобы обойти это ограничение, обе страницы должны согласиться с этим и содержать JavaScript-код, который специальным образом обменивается данными.

Это ограничение необходимо, опять же, для безопасности пользователя. Страница `https://anysite.com`, которую открыл пользователь, не должна иметь доступ к другой вкладке браузера с URL `https://gmail.com` и воровать информацию оттуда.

- JavaScript может легко взаимодействовать с сервером, с которого пришла текущая страница. Но его способность получать данные с других сайтов/доменов ограничена. Хотя это возможно в принципе, для чего требуется явное согласие (выраженное в заголовках HTTP) с удалённой стороной. Опять же, это ограничение безопасности.



Подобные ограничения не действуют, если JavaScript используется вне браузера, например — на сервере. Современные браузеры предоставляют плагины/расширения, с помощью которых можно запрашивать дополнительные разрешения.

## Что делает JavaScript особенным?

Как минимум, *три* сильные стороны JavaScript:

- Полная интеграция с HTML/CSS.
- Простые вещи делаются просто.
- Поддерживается всеми основными браузерами и включён по умолчанию.

JavaScript — это единственная браузерная технология, сочетающая в себе все эти три вещи.

Вот что делает JavaScript особенным. Вот почему это самый распространённый инструмент для создания интерфейсов в браузере.

Хотя, конечно, JavaScript позволяет делать приложения не только в браузерах, но и на сервере, на мобильных устройствах и т.п.

## Языки «над» JavaScript

Синтаксис JavaScript подходит не под все нужды. Разные люди хотят иметь разные возможности.

Это естественно, потому что проекты разные и требования к ним тоже разные.

Так, в последнее время появилось много новых языков, которые *транспируются* (конвертируются) в JavaScript, прежде чем запустятся в браузере.

Современные инструменты делают транспилиацию очень быстрой и прозрачной, фактически позволяя разработчикам писать код на другом языке, автоматически преобразуя его в JavaScript «под капотом».

Примеры таких языков:

- [CoffeeScript](#) ➞ добавляет «синтаксический сахар» для JavaScript. Он вводит более короткий синтаксис, который позволяет писать чистый и лаконичный код. Обычно такое нравится Ruby-программистам.
- [TypeScript](#) ➞ концентрируется на добавлении «строгой типизации» для упрощения разработки и поддержки больших и сложных систем. Разработан Microsoft.
- [Flow](#) ➞ тоже добавляет типизацию, но иначе. Разработан Facebook.
- [Dart](#) ➞ стоит особняком, потому что имеет собственный движок, работающий вне браузера (например, в мобильных приложениях). Первоначально был предложен Google, как замена JavaScript, но на данный момент необходима его транспилиация для запуска так же, как для вышеперечисленных языков.
- [Brython](#) ➞ транспирует Python в JavaScript, что позволяет писать приложения на чистом Python без JavaScript.

Есть и другие. Но даже если мы используем один из этих языков, мы должны знать JavaScript, чтобы действительно понимать, что мы делаем.

## Итого

- JavaScript изначально создавался только для браузера, но сейчас используется на многих других платформах.
- Сегодня JavaScript занимает уникальную позицию в качестве самого распространённого языка для браузера, обладающего полной интеграцией с HTML/CSS.
- Многие языки могут быть «транспирированы» в JavaScript для предоставления дополнительных функций. Рекомендуется хотя бы кратко рассмотреть их после освоения JavaScript.

## Справочники и спецификации

Эта книга является *учебником* и нацелена на то, чтобы помочь вам постепенно освоить язык. Но когда вы хорошо изучите основы, вам понадобятся дополнительные источники информации.

### Спецификация

[Спецификация ECMA-262](#) ➞ содержит самую глубокую, детальную и формализованную информацию о JavaScript. Она определяет сам язык.

Вначале спецификация может показаться тяжеловатой для понимания из-за слишком формального стиля изложения. Если вы ищете источник самой достоверной информации, то это правильное место, но она не для ежедневного использования.

Новая версия спецификации появляется каждый год. А пока она не вышла официально, все желающие могут ознакомиться с текущим черновиком на <https://tc39.es/ecma262/> [↗](#).

Чтобы почитать о самых последних возможностях, включая те, которые «почти в стандарте» (так называемые «stage 3 proposals»), посетите <https://github.com/tc39/proposals> [↗](#).

Если вы разрабатываете под браузеры, то существуют и другие спецификации, о которых рассказывается во [второй части](#) этого учебника.

## Справочники

- **MDN (Mozilla) JavaScript Reference** – это справочник с примерами и другой информацией. Хороший источник для получения подробных сведений о функциях языка, методах встроенных объектов и так далее.

Располагается по адресу

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference> [↗](#).

Хотя зачастую вместо их сайта удобнее использовать какой-нибудь интернет-поисковик, вводя там запрос «MDN [что вы хотите найти]», например

<https://google.com/search?q=MDN+parseInt> [↗](#) для поиска информации о функции `parseInt`.

## Таблицы совместимости

JavaScript – это развивающийся язык, в который постоянно добавляется что-то новое.

Посмотреть, какие возможности поддерживаются в разных браузерах и других движках, можно в следующих источниках:

- <http://caniuse.com> [↗](#) – таблицы с информацией о поддержке по каждой возможности языка. Например, чтобы узнать, какие движки поддерживают современные криптографические функции, посетите: <http://caniuse.com/#feat=cryptography> [↗](#).
- <https://kangax.github.io/compat-table> [↗](#) – таблица с возможностями языка и движками, которые их поддерживают и не поддерживают.

Все эти ресурсы полезны в ежедневной работе программиста, так как они содержат ценную информацию о возможностях использования языка, их поддержке и так далее.

Пожалуйста, запомните эти ссылки (или ссылку на эту страницу) на случай, когда вам понадобится подробная информация о какой-нибудь конкретной возможности JavaScript.



## Редакторы кода

Большую часть своего рабочего времени программисты проводят в редакторах кода.

Есть два основных типа редакторов: IDE и «лёгкие» редакторы. Многие используют по одному инструменту каждого типа.

### IDE

Термином [IDE](#) (Integrated Development Environment, «интегрированная среда разработки») называют мощные редакторы с множеством функций, которые работают в рамках целого проекта. Как видно из названия, это не просто редактор, а нечто большее.

IDE загружает проект (который может состоять из множества файлов), позволяет переключаться между файлами, предлагает автодополнение по коду всего проекта (а не только открытого файла), также она интегрирована с системой контроля версий (например, такой как [git](#)), средой для тестирования и другими инструментами на уровне всего проекта.

Если вы ещё не выбрали себе IDE, присмотритесь к этим:

- [Visual Studio Code](#) (кросс-платформенная, бесплатная).
- [WebStorm](#) (кросс-платформенная, платная).

Для Windows есть ещё Visual Studio (не путать с Visual Studio Code). Visual Studio – это платная мощная среда разработки, которая работает только на Windows. Она хорошо подходит для .NET платформы. У неё есть бесплатная версия, которая называется [Visual Studio Community](#).

Многие IDE платные, но у них есть пробный период. Их цена обычно незначительна по сравнению с зарплатой квалифицированного разработчика, так что попробуйте и выбирайте ту, что вам подходит лучше других.

### «Лёгкие» редакторы

«Лёгкие» редакторы менее мощные, чем IDE, но они отличаются скоростью, удобным интерфейсом и простотой.

В основном их используют для того, чтобы быстро открыть и отредактировать нужный файл.

Главное отличие между «лёгким» редактором и IDE состоит в том, что IDE работает на уровне целого проекта, поэтому она загружает больше данных при запуске, анализирует структуру проекта, если это необходимо, и так далее. Если вы работаете только с одним файлом, то гораздо быстрее открыть его в «лёгком» редакторе.

На практике «лёгкие» редакторы могут иметь множество плагинов, включая автодополнение и анализаторы синтаксиса на уровне директории, поэтому границы между IDE и «лёгкими» редакторами размыты.

Следующие варианты заслуживают вашего внимания:

- [Sublime Text](#) (кроссплатформенный, условно-бесплатный).
- [Notepad++](#) (Windows, бесплатный).
- [Vim](#) и [Emacs](#) тоже хороши, если знать, как ими пользоваться.

## Не будем ссориться

Редакторы, перечисленные выше, известны автору давно и заслужили много хороших отзывов от коллег.

Конечно же, есть много других отличных редакторов. Выбирайте тот, который вам больше нравится.

Выбор редактора, как и любого другого инструмента, индивидуален и зависит от ваших проектов, привычек и личных предпочтений.

## Консоль разработчика

Код уязвим для ошибок. И вы, скорее всего, будете делать ошибки в коде... Впрочем, давайте будем откровенны: вы *точно* будете совершать ошибки в коде. В конце концов, вы человек, а не [робот](#).

Но по умолчанию в браузере ошибки не видны. То есть, если что-то пойдёт не так, мы не увидим, что именно сломалось, и не сможем это починить.

Для решения задач такого рода в браузер встроены так называемые «Инструменты разработки» (Developer tools или сокращённо — devtools).

Chrome и Firefox снискали любовь подавляющего большинства программистов во многом благодаря своим отменным инструментам разработчика. Остальные браузеры, хотя и оснащены подобными инструментами, но всё же зачастую находятся в роли догоняющих и по качеству, и по количеству свойств и особенностей. В общем, почти у всех программистов есть свой «любимый» браузер. Другие используются только для отлова и исправления специфичных «браузерозависимых» ошибок.

Для начала знакомства с этими мощными инструментами давайте выясним, как их открывать, смотреть ошибки и запускать команды JavaScript.

## Google Chrome

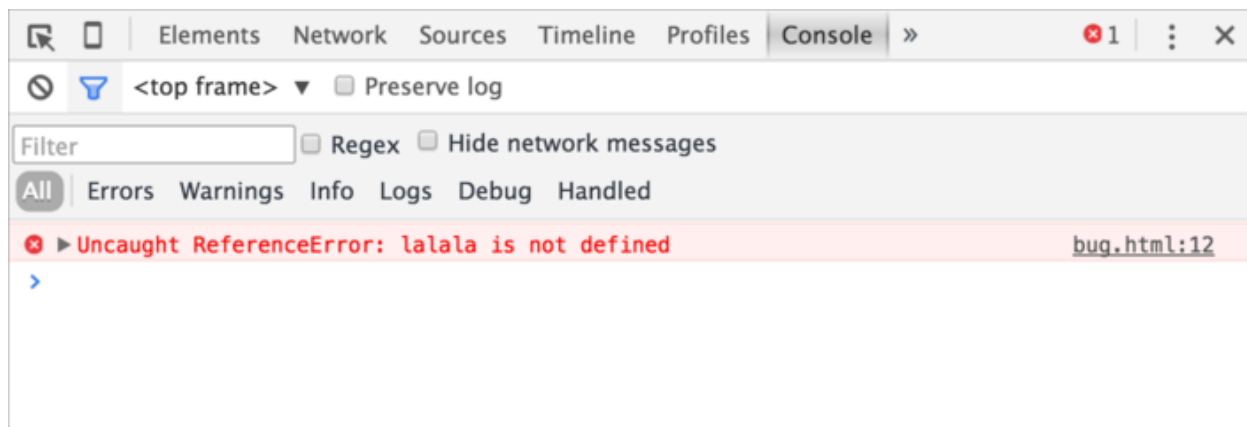
Откройте страницу [bug.html](#).

В её JavaScript-коде закралась ошибка. Она не видна обычному посетителю, поэтому давайте найдём её при помощи инструментов разработки.

Нажмите **F12** или, если вы используете Mac, **Cmd+Opt+J**.

По умолчанию в инструментах разработчика откроется вкладка Console (консоль).

Она выглядит приблизительно следующим образом:



Точный внешний вид инструментов разработки зависит от используемой версии Chrome. Время от времени некоторые детали изменяются, но в целом внешний вид остаётся примерно похожим на предыдущие версии.

- В консоли мы можем увидеть сообщение об ошибке, отрисованное красным цветом. В нашем случае скрипт содержит неизвестную команду «lalala».
- Справа присутствует ссылка на исходный код `bug.html:12` с номером строки кода, в которой эта ошибка и произошла.

Под сообщением об ошибке находится синий символ `>`. Он обозначает командную строку, в ней мы можем редактировать и запускать JavaScript-команды. Для их запуска нажмите `Enter`.

#### **i** Многострочный ввод

Обычно при нажатии `Enter` введённая строка кода сразу выполняется.

Чтобы перенести строку, нажмите `Shift+Enter`. Так можно вводить более длинный JS-код.

Теперь мы явно видим ошибки, для начала этого вполне достаточно. Мы ещё вернёмся к инструментам разработчика позже и более подробно рассмотрим отладку кода в главе [Отладка в браузере](#).

## Firefox, Edge и другие

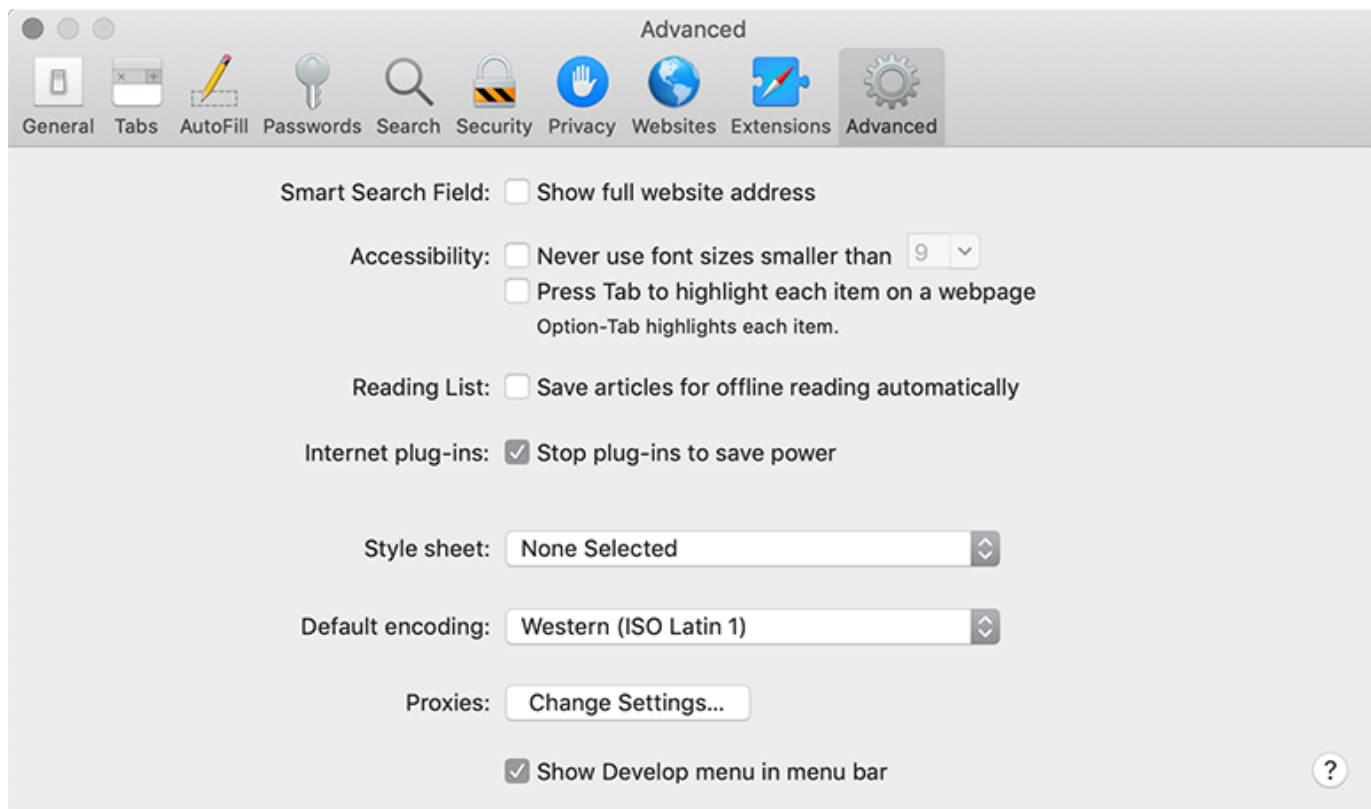
Инструменты разработчика в большинстве браузеров открываются при нажатии на `F12`.

Их внешний вид и принципы работы мало чем отличаются. Разобравшись с инструментами в одном браузере, вы без труда сможете работать с ними и в другом.

## Safari

Safari (браузер для Mac, не поддерживается в системах Windows/Linux) всё же имеет небольшое отличие. Для начала работы нам нужно включить «Меню разработки» («Developer menu»).

Откройте Настройки (Preferences) и перейдите к панели «Продвинутые» (Advanced). В самом низу вы найдёте чекбокс:



Теперь консоль можно активировать нажатием клавиш `Cmd+Opt+C`. Также обратите внимание на новый элемент меню «Разработка» («Develop»). В нем содержится большое количество команд и настроек.

## Итого

- Инструменты разработчика позволяют нам смотреть ошибки, выполнять команды, проверять значение переменных и ещё много всего полезного.
- В большинстве браузеров, работающих под Windows, инструменты разработчика можно открыть, нажав `F12`. В Chrome для Mac используйте комбинацию `Cmd+Opt+J`, Safari: `Cmd+Opt+C` (необходимо предварительное включение «Меню разработчика»).

Теперь наше окружение полностью настроено. В следующем разделе мы перейдём непосредственно к JavaScript.