Учимся кодить на JavaScript

Джереми Мориц

Иллюстрации
Кристин Мориц

JavaScript для подростков

Jeremy
Moritz

Illustrated by
Christine Moritz

Code for Teens

The Awesome Beginner's Guide to Programming

Volume 1: JavaScript

www.mascotbooks.com

Джереми
Мориц

Иллюстрации
Кристин Мориц

Учимся кодить на JavaScript

JavaScript для подростков

ПИТЕР®

Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону
Самара • Минск

2019

ББК 32.973.2-018.1
УДК 004.43
М79

Мориц Джереми

М79 Учимся кодить на JavaScript. — СПб.: Питер, 2019. — 256 с.: ил. — (Серия «Вы и ваш ребенок»).

ISBN 978-5-4461-0959-3

Ты любишь играть онлайн? Общаться с друзьями через ВКонтакте, Фейсбук и Инстаграм? Смотреть видеоролики на смартфоне? Все, чем ты пользуешься, было придумано обычными людьми, которые когда-то решили, что хотят заняться программированием. Умение писать код — это современная суперспособность, отличающая магов от маглов. И логичнее всего начать с изучения языка JavaScript, на котором написано более 90 % всех веб-сайтов.

«Учимся кодить на JavaScript» поможет тебе самостоятельно, без помощи родителей и учителей, написать программный код; ведь если говорить начистоту, большинство взрослых слабо представляют себе, как это делается. В книге много задачек и упражнений (с ответами, только чур сразу не подглядывать!). Ответов может быть несколько, главное, чтобы у тебя получился рабочий вариант. Стань круче Илона Маска! Пришло время закатать рукава и приступить к прокачке новой суперспособности!

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018.1
УДК 004.43

ISBN 978-1684019601 англ.
ISBN 978-5-4461-0959-3

Посвящается моим старшим детям — **Анджеле** и **Тони**. Спасибо вам за вдохновение, благодаря которому и была написана эта книга; за то, что помогли мне взглянуть на программирование ясным детским взором.

А также остальным четверым моим замечательным детям.
Благодарю вас за бесконечные часы радостного отдохновения.

Предисловие (для родителей)

Сара Фелпс, преподаватель информатики

Когда Джереми попросил меня написать предисловие к этой книге, я была страшно обрадована и польщена. Должна сразу оговориться: я не профессиональный программист. Но я прекрасно знаю, как найти общий язык с ребёнком, поскольку вот уже десять лет преподаю компьютерные науки в начальных классах. Я также разработала учебную программу по информатике, призванную взрастить и поддерживать интерес к программированию у детей и подростков на протяжении всего периода обучения в школе. Во время работы над учебной программой я перелопатила целую тонну специальной литературы, посещала конференции по IT и то и дело обращалась за консультацией к своему личному специалисту по разработке программного обеспечения — мужу. Накопленный опыт и понимание специфики работы с детьми помогли мне разработать эффективный курс обучения основам программирования, который уже зарекомендовал себя. Так что я решила поделиться успешным опытом с коллегами и стала выступать на технологических конференциях, а также в детских научных кружках и мастерских.

В одной из подобных мастерских мы и познакомились с Джереми, Кристин и их детьми. От старших до самых младших — все Морицы горят программированием и в целом полны жаждой познания; Джереми и Кристин невероятно сильно вдохновили своих детей познавать весь окружающий мир. Я могу лишь надеяться, что дети, которых я учу, получают от меня хотя бы толику подобного вдохновения.

Мы с Джереми полностью сошлись во взглядах на обучение детей программированию. Как он говорит в этой книге, «программировать — это как иметь суперспособность», ведь применение навыков, которые появляются и развиваются по мере обучения программированию, выходят далеко за границы компьютерного мира. Компьютерное, то есть «вычислительное», мышление помогает становлению и развитию мышления логического и в целом улучшает гибкость ума. Умение быстро и эффективно найти решение задачи применимо абсолютно в любой обла-

сти! Навыки программирования вкупе со всем вышеперечисленным существенно повысят шансы на нахождение отличной работы — такого кандидата с руками оторвёт любой работодатель.

«Учимся кодить на JavaScript» — великолепное пособие для обучения самому популярному в мире языку программирования. Книга написана просто, понятно и вместе с тем в яркой и увлекательной форме. Материал подаётся порциями и отлично сбалансирован, включая в себя практические и теоретические задания. И это не говоря об остроумных рисунках Кристин Мориц!

Но самое главное — эта книга помогает детям обучаться самостоятельно! А ведь именно этого зачастую недостаёт школьному образованию. Подавая материал, учителя предлагают детям подождать, посмотреть, чтобы затем понять, и так далее; хотя на самом деле всё упирается в то, что вся эта важная информация останется непонятной ученикам без вмешательства мудрого взрослого. В этой книге вы найдёте противоположный подход: ждать ничего не нужно — а что нужно, так это двигаться как раз в точности со скоростью собственного понимания!

Для проверки пройденного материала в книге есть увлекательные упражнения. Без паники! В конце книги вы найдёте все ответы (если вы, как я, никогда не знаете, когда начать помогать ребёнку). Словом, книга устроена таким образом, чтобы вы могли работать вместе с ребёнком. Почему нет? Просматривая текст книги, я не удержалась, тоже открыла новую вкладку и выполнила пару упражнений.

«Учимся кодить на JavaScript» станет отличным пособием как для классной, так и для домашней работы, а также крутым подарком для пытливого и любознательного подростка, готового к новым свершениям. Поразительное ощущение, когда учишь другого человека изменять окружающий мир! Не могу дождаться возможности познакомить с этой книгой своих учеников и ещё более того — своих детей, которые, я надеюсь, продолжат изучать программирование и будут усердно поддерживать свою «обучаемостность» (как говорит Джереми) на протяжении всей будущей жизни.

Сара Фелпс

Предисловие к русскому изданию

Сегодня JavaScript — один из самых популярных и востребованных языков программирования, который постоянно развивается, используется почти повсеместно. Это невероятно простой и в то же время достаточно сложный язык программирования, возможности и области применения которого с каждым днем становятся все шире. И книга «Учимся кодить на JavaScript», выходящая в издательстве «Питер», вполне способна стать для вашего ребенка окном и лестницей в огромный мир языка JavaScript, достаточно ее открыть и сделать первые шаги.

Разделенная на главы и охватывающая практически все основы языка, эта книга, как гид, проведет вашего ребенка по начальным ступеням в мире языка JavaScript, давая теоретические объяснения различным нюансам программирования на данном языке, и сразу обеспечивающие читателя возможностью полученные знания применить на интересных практических задачах. От первой команды вывода на экран фразы "Привет, Мир" (или же "Hello, World"), до написания функций, выполняющих поставленные автором различные задачи.

В процессе ее прочтения, как действующий разработчик (программист), я делал заметки практически на каждой странице в духе «здесь чего-то не хватает», «как-то слишком легко» или «маловато». Однако позже, когда уже почти дочитал книгу, пришел к выводу, что ее автор весьма обдуманно и тщательно подошел к содержанию, он умело рассортировал порядок подачи теоретического и практического материала, превращая процесс ее прочтения в увлекательное путешествие в программирование на языке JavaScript, где нет ничего сложного и запутанного. Ибо всего того, что сможет заставить вашего ребенка надолго задуматься в процессе прочтения, быть для него непонятным или показаться ему чересчур сложным, в этой книге нет. Да, непросто поверить в то, что процесс

изучения языка программирования может быть таким легким, интересным, веселым и увлекательным.

От себя отмечу и как преподаватель, и как разработчик, что эта книга достаточно легко читается и не дает вам заскучать среди строк. Во всяком случае, если ваш ребенок действительно хочет научиться программировать — уверен, что эта книга без особых сложностей поможет ему начать писать код на языке JavaScript. Книга подробно расскажет о том, как «легко» начать, даст подробные объяснения всех базовых основ и позволит применить их на практических примерах, а в конце подскажет направление дальнейшего пути в мире программирования.

Мне она понравилась, думаю, понравится и вам.

Темир Сангаджиев, преподаватель курсов по JavaScript
«Международная школа программирования для детей CODDY»,
Frontend-разработчик

Введение

Зачем учиться программировать?

С первой же главы этой книги вы начнёте обучаться читать и писать программный код. Сотни страниц разъяснений, упражнений и заданий будут направлены на достижение результата. И прежде чем двинуться далее, быть может, самое время задать себе этот немаловажный вопрос: а зачем мне вообще учиться программировать?

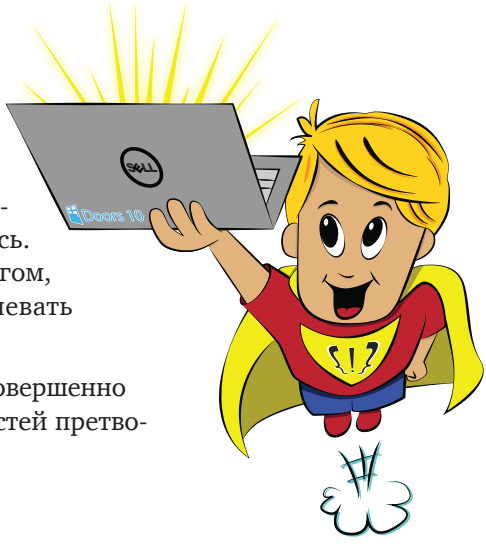
Если развёрнуто отвечать на этот вопрос, пожалуй, стоило бы упомянуть о чём-нибудь вроде «разработки программного обеспечения» — одной из наиболее быстрорастущих сфер на мировом рынке; неплохо было бы провести тонкое сравнение с прошлым, отметив, что программирование играет в современном мире схожую роль с каким-нибудь непростым ремеслом; и напоследок можно добавить, что спрос на рынке труда на хороших программистов только растёт, что гарантирует отличную зарплату, условия работы, гибкий график, удовлетворение от собственной дейстельности, многолетнюю успешную карьеру и бла-бла-бла, и всё в таком духе.

Но на самом деле ответ намного проще.

Суперспособности.

Гарри Поттера приняли в «Хогвартс», чтобы там он смог развить и отточить свои магические способности. Одарённые подростки собирались под крылом Профессора Икс. Так и ты, дорогой читатель, вскоре научишься управлять самой ценной суперспособностью в мире — писать программный код.

Вас когда-нибудь осеняла гениальная идея о том, что такой-то компании или такому-то человеку просто необходимо именно то, что вы придумали? Или, скажем, вас попросили выполнить какое-нибудь скучное, монотонное задание, на которое к тому же ещё уйдёт ооооочень много времени, а вы бы мечтали, чтобы была какая-нибудь машина, чтобы просто сделать всю эту работу за вас? Или вы играли в компьютерную игру и думали: «Ага! Это, конечно, неплохо, но было бы намного лучше, если бы...»?



Когда о чём-то подобном думают простые маглы, им не остаётся ничего, кроме как горестно вздохнуть и отмахнуться. Ведь подобные штуки не для простых смертных, верно? Что ж, дочитав эту книгу до конца, вы убедитесь, что обладаете намного большей силой, чем вам казалось. Вы убедитесь, что обладаете поразительным мозгом, способным общаться с машинами и даже повелевать ими, заставляя исполнять ваши приказания.

Учась программировать, вы открываете двери в совершенно новый мир, полный возможностей — возможностей претворять свои лучшие идеи в реальность.

С чего начать?

Едва вы решили научиться программировать, как хоп! — перед вами рассыпалась целая куча разнообразных способов это сделать.

Ведь научиться писать программный код это почти то же, что выучить иностранный язык. Скажем, вы хотите выучить какой-нибудь язык. Прежде чем начать, вы, вероятно, зададитесь вопросом: выучить какой язык было бы наиболее полезно? Быть может, ваша бабушка или дедушка говорят по-китайски? Или ваша тётя родом из Японии? Или, может, в семье вашего лучшего друга все свободно говорят по-французски? Любой подобный фактор может повлиять на выбор изучаемого языка. Впрочем, если вы живёте в США, то, наверное, самый очевидный выбор — это испанский язык, просто потому что в Штатах на нём говорят больше, чем на всех остальных языках вместе взятых (кроме, конечно, английского).

Схожим образом дело обстоит и с языками программирования; существует огромное множество отличных языков — Java, Python, C#, PHP, Go, C++ и так далее. Однако один из них всё же возглавляет этот список, и это —

JavaScript!

JavaScript является наиболее популярным языком программирования, на нём написано более 90 % всех веб-сайтов. Именно JavaScript управляет подвижными элементами на странице, прописывает, что и когда должно произойти в ответ на то или иное действие пользователя. Также JavaScript широко применяется в сфере разработки компьютерных игр и мобильных приложений. Всё больше людей пользуется мобильным интернетом, всё больше компаний переносит свой бизнес в онлайн, так что с каждым годом ценность языка JavaScript растёт.

В общем, угадайте: какой язык программирования мы будем с вами изучать в этой книге?!

Ух ты! Неужели угадали?! Ну конечно JavaScript!! (Ох, ну вы даёте! Видимо, придётся в следующий раз придумать вопрос посложнее.)

Повышаем обучаемостность и весельеполучение!

Когда я писал этот заголовок, я был искренне уверен, что выдумал слово «обучаемостность»… оказалось — нет! Оно уже было кем-то выдумано из «обучения» и «способности» до меня. Неплохо, правда? Никогда не угадаешь, где придётся узнать что-то новенькое. Поразительно!

Ну да ладно… о чём мы там говорили? Ах, да!

Книга, которую вы держите в руках, поможет вам самостоятельно, не прибегая к помощи родителей или учителей, написать программный код. Серьёзно, давайте наконец взглянем правде в глаза: большинство взрослых в любом случае довольно слабо представляет себе, как именно это делается. А даже если кто-то из них и умеет это делать, то он, конечно же, невероятно занят спасением мира от нависших над ним опасностей, так что и времени объяснить вам, что к чему, у него нет.

Но не бойтесь, мои дорогие читатели! Эта книга ясно и понятно расскажет вам, «что» и «к чему». В конце каждой главы вы найдёте краткий обзор и набор упражнений по пройденной теме. Если по ходу главы вы упустили что-то важное, то сможете наверстать этот момент, просто заглянув в обзор темы, а занимательные упражнения помогут вам надёжно закрепить материал. Некоторым даже покажется, что подобных обзоров и упражнений в книге чересчур много. Если вы из их числа, я бы всё же рекомендовал вам мельком пробегать их, чтобы понимать, знаете ли вы, как решить ту или иную задачку, или нет. Будьте честны с самими собой, не упускайте возможности помочь себе улучшить собственные навыки!

Попробуйте, например, при вводе кода, который указан в книге, что-нибудь в нём менять и смотреть, что после этого будет происходить.

Все темы, представленные в книге, связаны между собой по порядку, так что, пожалуйста, прежде чем двигаться вперёд, убедитесь, что вы точно уяснили только что пройденный материал. Также в конце книги вы найдёте глоссарий, который поможет вам выяснить значение слова, если вы вдруг запамятовали (такая «мозговая икота» порой нападает на каждого).

И ещё хорошая новость: если уж вы намертво застряли с какой-нибудь задачкой, то ВСЕ ответы вы найдёте в конце книги (будто бы у вас на руках реальная «книга для учителя»!). Впрочем, стоит отметить, что иногда вы встретите упражнение с более чем одним верным ответом. Постарайтесь вникать в логику каждого представленного здесь упражнения — оно того, несомненно, стоит; но всё же не нужно «умирать» над поиском точного ответа, как в конце книги, если ваш вариант слегка отличается, но тоже работает.

Ну и наконец, программировать — это весело! Быть может, не всегда, но большую часть времени процесс должен ощущаться сродни решению увлекательной головоломки или задачки. Ни в коем случае не расстраивайтесь, если не понимаете чего-то сразу же, пробуйте подойти к проблеме с разных сторон, но если видите, что прогресса всё нет, не стоит буксовать часами — просто идите дальше! Загляните в ответы и — вперёд, а к этому трудному месту вернитесь чуть позже. Всё это — части большого путешествия в мир программирования!

Что ж, закатаем рукава и приступим к прокачке ваших новых суперспособностей!

Слово к родителям

Совершенно секретно: детям вход воспрещён!

Следующая пара страниц является посланием к родителям! Дети, если вы читаете это конфиденциальное, совершенно секретное и так далее вступление, то сейчас — самое время выполнить свою часть сделки и передать эту книгу старшим… и получить её обратно две страницы спустя.

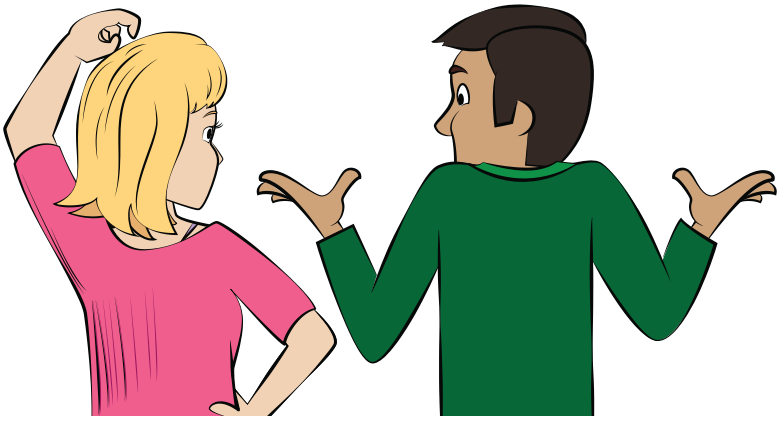
Итак, родители, если вам посчастливилось принять замечательное решение приобрести эту книгу своему маленькому или уже не очень чаду, то вы, вероятно, понимаете, насколько полезным является изучение основ программирования. Разработчик программного обеспечения (то есть программист) является одной из наиболее востребованных и высокооплачиваемых профессий в мире на сегодняшний день, и спрос на хороших программистов с каждым годом только растёт. Изучение программирования развивает логическое мышление, а также тренирует множество других весьма полезных навыков и способностей, которые прослужат подростку всю его дальнейшую жизнь.

Впрочем, иметь представление о том, что что-то имеет ценность, и понимать, каким именно образом можно применить это знание на практике, — это, что называется, две большие разницы! Может статься, что вы в жизни не написали и строчки программного кода! Может, вы даже могли бы спутать клавиатуру с ординатурой или браузер со шнауцером!

А может, и наоборот: вы профессиональный программист, и вам бы очень хотелось, чтобы и ваши дети пошли по вашим стопам, обладали бы востребованной профессией и гарантированным светлым будущим, в которое вы смотрите каждый день. Но откуда взять столько времени, терпения и, главное, педагогические навыки, чтобы научить ребёнка всему необходимому? Ведь для обучения чему-либо, очевидно, недостаточно просто знать, как это «что-либо» делается.

Так или иначе, эта книга для ВАС… чтобы вы вручили её ребёнку, а тот её прочёл.

Садясь за работу над этой книгой, мне хотелось написать её так, чтобы любой ребёнок или подросток (словом, всякий, кто уже сносно умеет читать) мог, изучив её, сразу же писать собственный программный код без помощи родителей или учителей.



Как отец шестерых детей, обучающий всех их на дому, я имею богатый практический опыт по части детского обучения. Так вот, по собственному опыту могу сказать, что найти комплексные учебные материалы по программированию, ориентированные на детей, невероятно сложно. Мои дети, бывало, показывали мне какой-нибудь сделанный проект, но объяснить свой собственный код были не в состоянии! Это происходило оттого, что большая часть кода была написана ими «под диктовку» того или иного пособия. Именно желание подарить моим детям более полезное пособие по программированию и вдохновило меня начать работу над этой книгой.

При обучении программированию детям зачастую нужно куда больше задачек и упражнений на повторение, чем взрослым. Им почти не нужны вводные объяснения и заумные формулы: лучше как можно скорее перейти к практическому заданию, к которому «примешаны» небольшие разъяснения. В противном случае знания и навыки весьма легко забываются, если не идут рука об руку с реальным их испытанием.

Вы убедитесь, что материалы, собранные в книге, поданы с прицелом на понимание и скорость работы ребёнка. На каждой странице вас встретят красочные и весёлые иллюстрации за авторством моей чудесной и талантливой жены Кристин, а также отменные… ну, словом, «папины шутки юмора», чтобы не дать вам заскучать и всё такое.

Несмотря на то что эта книга написана таким образом, чтобы ребёнок мог обучаться без посторонней помощи, будет ещё лучше, если у вас — как у любящего и заботливого родителя — получится принять участие в этом процессе. Сразу скажу: для этого вам не потребуется НИКАКИХ специальных знаний о программировании! На начальных этапах ваш ребёнок создаст и сохранит новый документ (который мы будем называть Рабочей тетрадью) и будет заносить в него все свои ответы. В конце книги есть раздел с правильными ответами, так что ребёнок сможет всё проверить самостоятельно.

Вместе с тем, когда ребёнок предоставлен самому себе и должен самостоятельно по ответам проверить свою работу, он порой так до выполнения этой работы и не доходит, удовлетворяясь знанием ответов. В общем, предупреждён — значит вооружён! Надеюсь, вы сидели, когда читали эти строки. ;-) Серьёзно: если бы вы сами, с ответами в руках, иногда проверяли работу вашего ребёнка, этот дополнительный момент контроля со стороны помог бы ему более серьёзно воспринимать сам процесс обучения и более усердно подходить к нему.

Что также может быть весьма полезным — время от времени просить ребёнка показать вам, что он уже сделал. Вам не нужно даже пытаться ничего понять — просто дайте ребёнку самому вам объяснить, что к чему. Не стесняйтесь проявить энтузиазм, когда ребёнок показывает вам свой проект. Вовсе не обязательно знать хоть что-то о программировании, чтобы порадоваться за своё чадо, ободрить и похвалить его.

И последнее: пожалуйста, прочтите выше коротенькое «Введение». Оно даёт ясное представление о том, каким образом устроена эта книга и как ваш ребёнок может извлечь из неё максимум пользы.

Я невероятно счастлив, что ваш ребёнок вот уже почти совершил первый шаг на пути обучения замечательному и полезному ремеслу программирования. Надеюсь, эта книга поможет вам превратить этот путь в большое и увлекательное путешествие!

1 Привет, Мир!

Вы уже готовы написать свои первые строчки **кода**?
Сейчас-сейчас, только соберём парочку вещей в дорогу. За дело!

Компьютер, браузер, консоль

Первым делом вам понадобится **компьютер**: стационарный, ноутбук или Chromebook (но не смартфон или планшет!). Если у вас нет собственного компьютера, можете воспользоваться компьютером в библиотеке. Или можете прийти домой к другу, у которого есть компьютер. Ну или можете попробовать собрать компьютер из проволоки, светодиодов и кастрюли с остывшим пюре (не рекомендуется).

Далее: на компьютере должна быть установлена **операционная система**: Windows (если у вас PC), MacOS (если у вас компьютер от Apple), ChromeOS (если у вас Chromebook) или Linux (если вы компьютерный гик). Вы должны знать, какая именно операционка установлена у вас на компьютере.

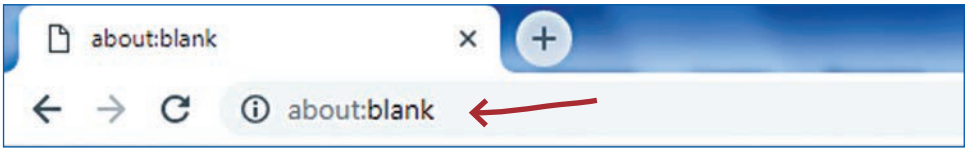
У вас должен быть установлен Google **Chrome**. Chrome — это **браузер**, то есть программа для просмотра веб-страниц. Вполне вероятно, что он уже установлен на вашем компьютере: поищите на рабочем столе иконку, похожую на разноцветный покебол[1].

Кстати, вы заметили, что некоторые слова выделены **жирным шрифтом**? Так я отмечаю важные для нашей работы слова, определение которых вы найдёте в глоссарии в конце книги. «Что же такое **глоссарий**?» — спросите вы. Глоссарий — это как словарь, только специальный, в котором собраны термины, использующиеся в этой книге. Короче, загляните в конец книги и посмотрите!

[1] Покебол (*англ.* — Pokéball) — мяч для ловли покемонов, монстров из одноимённой анимационной серии мультфильмов. — *Здесь и далее примеч. пер.*

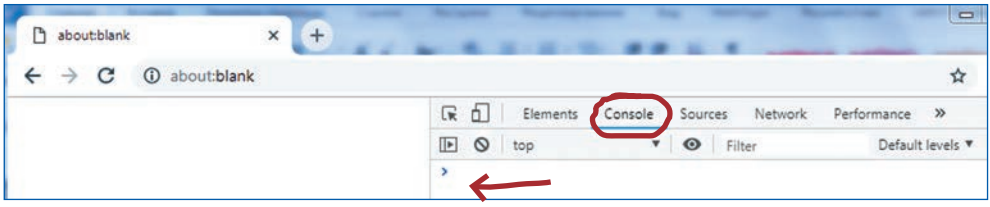
Если у вас всё же не установлен Chrome, то можете скачать его при помощи любого другого браузера (Firefox, Safari, Edge или IE), просто вбив в поисковой строке «скачать Chrome» и следуя дальнейшим инструкциям.

Отлично. После скачивания и установки Chrome откройте его и в адресной строке введите: `about:blank`. Нажмите ENTER (или RETURN, если у вас Mac). Браузер откроет новую, пустую страницу. Обратите внимание: клавиши, которые следует нажимать, также ВЫДЕЛЕНЫ в тексте, но их определений вы в глоссарии не найдёте, потому… ну, короче, потому что это просто кнопки на вашей клавиатуре.



Итак, пришло время познакомиться с консолью! **Консоль** является одним из мегасекретных инструментов Chrome, предназначенных специально для разработчиков. Раз вы теперь *начинающий программист-разработчик, ею можете пользоваться и вы!* Чтобы вызвать консоль, просто нажмите вместе клавиши CTRL, SHIFT (или же COMMAND и OPTION, если у вас Mac) и J. В будущем все подобные комбинации мы будем прописывать в виде CTRL+SHIFT+J (или COMMAND+OPTION+J для Mac соответственно).

После нажатия комбинации клавиш откроется новая строка меню, а в левом верхнем углу страницы вы увидите угловую скобку. Если же комбинация CTRL+SHIFT+J (или COMMAND+OPTION+J) по каким-то причинам *не работает*, вы можете войти в консоль, нажав на пустое место на странице правой кнопкой мыши, выбрать в контекстном меню пункт Inspect и затем нажать Console в строке меню (как показано на скриншоте ниже).



Кстати, если переместить курсор к краю светло-серой области, где он превратится в двойную стрелку, то, нажав и удерживая левую кнопку, можно растянуть окно консоли; обязательно сделайте это, чтобы у вас было больше места для введения кода.

Теперь, когда перед вами открыто окно консоли и вы растянули его удобным образом, нажмите мышкой рядом с угловой скобкой (>). В строке появится мигающий курсор, означающий, что можно начинать вводить текст. Сейчас вы наберёте свои первые строчки кода. Введите следующий текст (вместе с кавычками), не забыв точку с запятой «;» (находится чуть правее центра на клавиатуре) — важный символ, который будет постоянно заключать строку кода:

```
"Привет, Мир!";
```

Нажмите ENTER (или RETURN на Mac). Консоль вернёт вам в следующей строке ваш текст: `"Привет, Мир!"`. Обратите внимание: код, который вы должны будете ввести в консоль, `отображается синим цветом`, ответ же консоли `отображается красным`. Это сделано для того, чтобы вам легче было понять, что именно следует вводить в строку консоли. Позднее вы встретитесь также с примерами кода, `выделенного серым цветом`, — эти строки вам нужно будет лишь прочесть (а не вводить в строку).

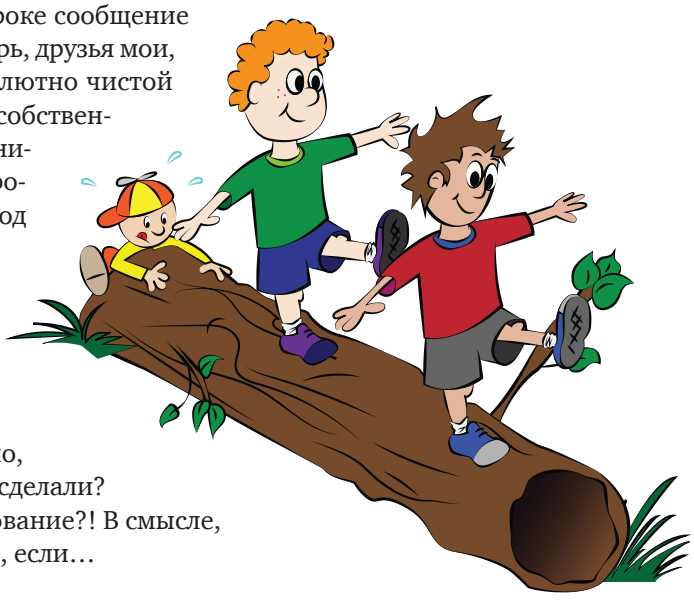
Теперь нажмите клавишу UP_ARROW (стрелка вверх); вы увидите свой текст `"Привет, Мир!";`. Переместите курсор в начало строки и наберите вместо этого текста следующий:

```
var приветствие = "Привет, Мир!";
```

и нажмите ENTER (помните, что ENTER всегда означает RETURN, если у вас Mac). В консоли вы увидите ответ: `undefined.` Всё идёт по плану! Теперь наберите:

```
приветствие;
```

и нажмите ENTER. Если вы всё сделали верно, то консоль вернёт в строке сообщение `"Привет, Мир!"`. Что ж, теперь, друзья мои, вы можете всегда и с абсолютно чистой совестью заявлять, что вы собственноручно написали и выполнили (хоть и совсем микрокрохотулечный, но всё же…) код на языке **JavaScript**!



Повторяй за мной!

Ну и… в чём же, собственно, смысл? Что мы только что сделали? И ЭТО и есть программирование?! В смысле, да кому вообще какое дело, если…

Таааак! Придержите коней! Очень рад, что спросили. Честное-пречестное! Я просто сгораю от нетерпения, чтобы всё это вам объяснить, но сперва давайте выполним маленькое упражнение на доверие. Мы сыграем в «повторяй за мной» — просто набирайте в консоли всё, что указано ниже (и `выделено синим цветом`!). Каждую строку завершайте нажатием клавиши ENTER и наслаждайтесь результатом:

```
5 + 8;
```

Консоль ответит вам: `13`

```
7 - 3 + 6;
```

И ответ: `10`

Далее наберите (обратите внимание на символ `*`, означающий *умножение*):

```
var x = 3; 4 * x;
```

И получите в ответе: `12`

Теперь:

```
x + x;
```

Ответ: `6`

Обратите внимание, что последнее вычисление было выполнено, используя полученные выше значения `x`. Едем дальше:

```
x = 200;
```

Мы присвоили `x` *новое* значение. Теперь дважды нажмём стрелку вверх, чтобы вернуться к:

```
x + x;
```

Нажав теперь ENTER, мы получим в ответ — `400`, поскольку значение `x` уже изменилось. Ладно, давайте ещё парочку, и я всё подробно объясню.

Наберите (символ `/` означает *деление*):

```
x / 100;
```

Ответ: `2`

Теперь:

```
(x - 50) / 5;
```

И получите ответ: `30`

```
(x - 50) / (5 - 1);
```

Ответ: `37.5`

```
"Хочу прожить до " + x + " лет! ";
```

И ответ: `"Хочу прожить до 200 лет!"`

```
"Хмм… хотя, пожалуй, и до " + (x / 2) + " было бы уже неплохо.";
```

Ответ: `"Хмм… хотя, пожалуй, и до 100 было бы уже неплохо."`

Ну как вам? Удивило ли вас что-нибудь? Я пока ещё ничего не объяснял, но думаю, что что-то вы уже начали понимать, верно?

Были ли у вас какие-нибудь ошибки? Надеюсь, что да. Ведь **разработчики**, пишущие на JavaScript, совершают ошибки в коде по десять раз на дню! Важнейшей чертой хорошего **программиста** является умение выявить собственные ошибки — их ещё называют **багами**[1] — и исправить (то есть раздавить) их. К этому мы ещё обязательно вернёмся; но сперва давайте выясним, из чего состоит ваш первый код на JavaScript — что означает и как интерпретируется (то есть понимается компьютером) каждая его часть.

Синтаксис

JavaScript, подобно множеству прочих **языков программирования**, обладает строгим **синтаксисом**. Синтаксис — это раздел грамматики, изучающий предложения и способы сочетания слов внутри предложения. Компьютеры не столь сообразительны, как люди: они не в состоянии «просто понять», что вы имели в виду. Машина сможет понять вас только в том случае, если вы будете общаться с ней строго в тех формах выражения, которые она от вас ожидает. Эта ожидаемая форма выражения и называется синтаксисом.

Мы сейчас ещё немного поработаем в консоли, но перед этим, пожалуйста, давайте убедимся, что вы уже точно умеете её вызывать. Итак, закроем окно Chrome (даже если вы специально оставили его открытым на будущее) и начнём сначала. Введём в адресной строке `about:blank`, затем при помощи комбинации CTRL+SHIFT+J (или, как вы помните, COMMAND+OPTION+J для Mac) вызовем консоль (обязательно запомните эти комбинации — вы будете ими пользоваться постоянно!). Должно открыться чистое, ожидающее ваших распоряжений окно консоли.

Рассмотрим пару примеров грамотного применения синтаксиса JavaScript. Перепечатывайте в консоль все строки кода, отмеченные синим (пожалуйста, набирайте код В ТОЧНОСТИ как указано далее) и в конце каждой строки нажимайте клавишу ENTER. Приступим:

```
var первая_часть_приветствия = "Привет,";
```

[1] От *англ.* bug — жук. По легенде, в 1945 году учёные Гарвардского университета, тестировавшие вычислительную машину Mark II Aiken Relay Calculator, нашли мотылька, застрявшего между контактами электромеханического реле, и вклеили насекомое в технический дневник с сопроводительной надписью: «First actual case of bug being found» (Первая фактическая ошибка (насекомое) была найдена). — *Примеч. ред.*

Объяснение: перед вами так называемые **предложения** JavaScript. Предложения зачастую заканчиваются точкой с запятой (`;`). Ключевое слово `var` в начале строки объявляет компьютеру (ну то есть технически, конечно, не напрямую компьютеру, а всего лишь встроенному в браузер **интерпретатору** JavaScript), что следующее за ним `первыйЧастьПриветствия` является **переменной**. Далее знак равенства (`=`) указывает компьютеру (тьфу… *интерпретатору*), что мы присваиваем значение `"Привет"` нашей переменной `первыйЧастьПриветствия`, чтобы в дальнейшем можно было бы производить с ней операции. Что-то многовато длинных-умных слов, да? Знаю-знаю, но, пожалуйста, постарайтесь их запомнить, потому что к ним мы будем обращаться практически постоянно. Если забудете, что какое-то там слово означает, помните, что всегда можно заглянуть в глоссарий в конце книжки!

```
let вторыйЧастьПриветствия = "Мир";
```

Консоль ответит нам: `undefined`

Объяснение: в этом предложении мы также произвели операцию **присваивания** значения. Ключевое слово `let` делает то же самое, что и `var`. Впрочем, некоторые различия между ними, конечно, есть, но их подробное описание сейчас было бы излишне, так что ладно. Просто имейте в виду, что несмотря на то, что по большей части мы здесь будем использовать `let`, не менее часто при дальнейшем изучении JavaScript вы встретитесь и с `var`. В нашем предложении мы присвоили новой переменной `вторыйЧастьПриветствия` **строку** (`"Мир"`). Слово `"Мир"` называется строкой, поскольку заключено в кавычки (об этом мы подробно поговорим в одной из следующих глав). Ещё вы, наверное, обратили внимание, что в ответ консоль выдала `undefined`. Не стоит волноваться из-за этого — подобный ответ всего лишь означает, что вы пока не определили, что именно должен вернуть (RETURN) вам в качестве ответа интерпретатор JavaScript. Вроде того, что вы бы, например, сказали интерпретатору: «Слушай, запиши-ка вот это, чтобы в будущем мы что-нибудь с этим могли сделать». Интерпретатор послушно исполнил ваше приказание и теперь терпеливо дожидается дальнейших инструкций.

Кстати, если во время перепечатывания кода отсюда в консоль вы опечатались где-то и произошли какие-нибудь странные ошибки и всё такое, то просто обновите страницу Chrome (значок обновления похож на кольцо со стрелочкой; находится в верхней части окна браузера) и попробуйте проделать всё заново. В будущем вы научитесь решать подобные проблемы и другими способами, но пока довольно и этого. Теперь наберём следующий код:

```
let одиночныйПробел = " ";
```

Ответ: `undefined`

Объяснение: наше предложение присвоило переменной `одиночныйПробел` значение строки (содержащей один пробел, заключённый в кавычки). Обратите внимание: *при операции присваивания слева от знака равенства всегда должна*

быть лишь одна переменная. Например, `let x = 2 + 2;` является для JavaScript грамотным предложением, в то время как `2 + 2 = let x;` или, скажем, `2 + 2 = x;` — *нет*.

```
let полноеПриветствие = перваяЧастьПриветствия + одиночныйПробел +
вораяЧастьПриветствия;
```

Ответ: `undefined`

Объяснение: это очередное присваивание. Кстати говоря, вы заметили, что внутри всех переменных — `перваяЧастьПриветствия`, `вораяЧастьПриветствия`, `одиночныйПробел` или `полноеПриветствие` — слова пишутся с прописной буквы? А что между словами нет пробелов? Это вовсе не случайно! В именах переменных не должно быть пробелов, а сами эти имена, по-хорошему, должны быть написаны «верблюжьимРегистром» (**camelCase**). Этот стиль подразумевает, что имя переменной начинается со строчной буквы, а каждое следующее слово (или аббревиатура), входящее в имя переменной, начинается с прописной. Своё название этот стиль получил оттого, что прописные буквы в середине выглядят будто верблюжьи горбы. Если вы пока не поняли — просто продолжайте читать: вы столько раз ещё увидите перед собой «верблюжийРегистр», что в один момент горбы просто появятся, будто всегда там и были.

```
полноеПриветствие = полноеПриветствие + "!!";
```

Ответ: `"Привет, Мир!!"`

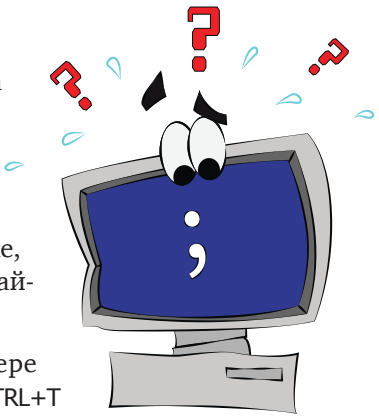
Объяснение: в данной операции мы добавляем новую строку (`"!!"`) в конец уже знакомой нам переменной `полноеПриветствие`, присваивая затем всю эту удлинённую строку нашей переменной. Вы заметили отсутствие `let`? Как так вышло? А так вышло оттого, что `let` используется, когда мы создаём (ещё говорят — *объявляем*) новую переменную. В нашем же случае мы не создали никакой новой переменной. Переменная `полноеПриветствие` уже существует, так что для работы с ней нет необходимости прибегать к `let`.

```
полноеПриветствие;
```

Ответ: `"Привет, Мир!!"`

Объяснение: раз уж мы не совершили нового присваивания или ещё чего, компьютеру (да что ж такое! — *интерпретатору*!) уже есть что нам вернуть (RETURN) в ответ! Как если бы вы обратились к интерпретатору со словами: «Слушай, а теперь выдай-ка (RETURN) мне значение той переменной, помнишь?» Соответственно, консоль и выдала вам в ответ нынешнее значение переменной `полноеПриветствие`.

Самые частые ошибки



Теперь, когда мы немного научились писать на языке JavaScript, давайте наделаем немного ошибок! Ответственно заявляю как программист: синтаксические ошибки при написании кода случаются практически каждый день. Штука в том, чтобы уметь прочесть сообщение об ошибке, которое выдаст вам машина, чтобы оперативно найти и исправить недочёт в написанном коде.

Итак, давайте откроем новую *вкладку* в браузере Chrome. Это можно сделать, нажав комбинацию CTRL+T (или соответственно COMMAND+T на Mac). Наберите в адресной строке `about:blank` и, используя известный вам сверхсекретный код, вызовите консоль (если забыли сочетание клавиш, вернитесь назад, найдите его и на этот раз постарайтесь запомнить!). Откроется чистое, готовое к вашим услугам окно консоли. Наберём в нём:

```
5 + ;
```

Ответ: `Uncaught SyntaxError: Unexpected token ;`

Подобный ответ консоли выглядит просто дико, а? Ненайденная синтаксическая ошибка? Неожиданный токен?! Штаа?!

Так, ладно… Теперь давайте внимательно вчитаемся. `Uncaught SyntaxError` — ненайденная синтаксическая ошибка, означает всего лишь, что в синтаксисе вашего предложения содержится ошибка. Другими словами, то, что вы написали, не является правильным с точки зрения языка JavaScript. Интерпретатор… тьфу, я имел в виду компьютер… Хотя стоп, ведь я имел в виду ИМЕННО *интерпретатор*! — он ведь интерпретирует предложения, написанные на языке *JavaScript*… Так вот, интерпретатор не может прочесть ваше предложение и не знает, что с ним нужно делать. `Unexpected token ;` означает, что интерпретатор прочёл что-то, чего прочесть вовсе не ожидал: в нашем случае — точку с запятой (`;`).

В общем, интерпретатор JavaScript как ни в чём не бывало читал ваше предложение и дошёл до точки с запятой, означающей для него, что предложение закончилось. И интерпретатор весь такой: «Чего?! `5 + ;`? Да как так-то? 5+ ЧТО?! Ерунда какая-то! Здесь точно какая-то ошибка! Я совсем не ожидал, что тут будет эта `;` и теперь не знаю, что мне делать!»

Чем больше вы изучаете JavaScript, тем больше начинаете ценить эти **сообщения об ошибках** — зачастую они очень верно показывают, где же вы допустили промах. Давайте совершим ещё парочку ошибок:

```
6+7) ;
```

Ответ: `Uncaught SyntaxError: Unexpected token )`

Объяснение: у вас есть закрывающая скобка «)», но нет скобки открывающей! А ведь скобки всегда ходят парами — не может быть, чтобы была закрывающая, но не нашлось открывающей скобки, и наоборот.

```
(1+2;)
```

Ответ: `Uncaught SyntaxError: Unexpected token ;`

Объяснение: точка с запятой `;` указывает интерпретатору JavaScript, что конец предложения достигнут ДО ТОГО, КАК закрылась скобка. Именно поэтому `;` и стала «неожиданным токеном». Понятно?

```
3 + новаяПеременная;
```

Ответ: `Uncaught ReferenceError: новаяПеременная is not defined`

Объяснение: теперь перед нами не синтаксическая ошибка (SyntaxError), а ошибка обращения (ReferenceError). Быть может, вы уже заметили, в чём здесь дело? Давайте внимательно прочтём сообщение об ошибке (ведь для этого, в конце концов, оно и выдаётся!). Там сказано: `новаяПеременная is not defined` — новая переменная не задана, вот в этом-то и кроется наша проблема! Сперва мы должны **объявить** переменную каким-нибудь, скажем, таким образом:

```
let новаяПеременная = 24;
```

Ответ: `undefined`

Пусть ответ консоли `undefined` вас не смущает — он лишь означает, что на данный момент ей больше нечего вам ответить. Давайте теперь дважды нажмём стрелку вверх (чтобы вернуться к нашему предыдущему предложению) и нажмём ENTER:

```
3 + новаяПеременная;
```

Ответ: `27`

Объяснение: теперь наша переменная и *объявлена*, и **определена** (то есть она теперь имеет конкретное значение), и наше предложение прекрасно работает! То, что вы только что проделали, является простейшим примером **дебаггинга**, или отладки. Вы обнаружили в коде ошибку, то есть «жука», и раздавили его, то есть исправили ошибку! Давайте раздавим ещё парочку, а?

```
let любимыйЦвет = "красный";
```

Ответ: `undefined // пока всё идёт по плану (undefined лишь означает «мне нечего вам сказать»)`

```
let любимыйЦвет = "синий";
```

Ответ: `Uncaught SyntaxError: Identifier 'любимыйЦвет' has already been declared`

Объяснение: сможете сами понять, в чём проблема, прочитав сообщение об ошибке? Всё просто: мы использовали ключевое слово `let` для объявления

переменной, хотя она уже была прежде объявлена. И тогда всё прошло хорошо, поскольку переменная `любимыйЦвет` объявлялась нами впервые. После этого мы уже не вправе снова использовать для неё `let`. Если его убрать, то и наше второе предложение прекрасно сработает:

```
любимыйЦвет = "синий";
любимыйЦвет;
```

Ответ: `"синий"`

Всё понятно? Мы прочли сообщение об ошибке, убрали из предложения `let` и тем самым изменили значение переменной (а может, вместе с ней и любимый цвет, кто знает?!) с «`красного`» на «`синий`».

Вы догадываетесь, что мы лишь едва прогулялись по бережку огромного океана всевозможных ошибок, которые могут нам повстречаться на пути, однако важно вынести отсюда вот что: не бойтесь ошибок! *Сообщение об ошибке — ваш друг!* Внимательно читайте их. Если после прочтения вы всё ещё не понимаете, в чём проблема, попробуйте скопировать текст сообщения об ошибке и вставить его в поисковую строку Google. В поисковой выдаче вы точно найдёте совет, как легко и быстро поправить дело.

Повторяй за мной: калькулятор хот-догов

Мы приступаем к первому настоящему проекту в этой книжке «Повторяй за мной». Мы начнём с очень простых заданий, а затем перейдём к выполнению более сложных.

Прежде чем мы продолжим, пожалуйста, либо закройте и заново откройте браузер Chrome, либо просто откройте новую вкладку и наберите `about:blank` (выглядит так, будто бы мы зря тратим на это время, но частое повторение подобных вещей поможет вам назавтра вспомнить, о чём тут вообще шла речь!). Теперь откроем чистое окно консоли.

Итак, всё готово — приступим!

Нам нужно вычислить среднюю стоимость комбо-набора в забегаловке «Сосиски и картошка у Веснушчатого Фреда» (почему нужно? Ну, потому что… ну нужно и нужно, короче). Вот что нам известно:

Комбо A: Альфа-пёс (хот-дог, средняя картошка и напиток на выбор) — $6.75.

Комбо B: Волкодав (хот-дог, большая картошка и два напитка на выбор) — $7.50.

Комбо C: Борзая (хот-дог, две маленькие картошки и соус на выбор) — $5.75.

Комбо D: Степной волк (хот-дог, большая картошка и маленький хот-дог) — $8.

Теперь давайте вместе, обсуждая каждый шаг, приступим к выполнению задания. Во-первых, определим, что нам нужно, а чего нам вовсе не нужно: нужны ли нам названия комбо? Не-а! Нужен ли их состав? Тоже нет (хотя, как по мне, в «Борзую» неплохо бы добавить ещё один соус, но я, впрочем, не настаиваю)! В общем, всё, что нам нужно — это цена каждого набора, ведь нам предстоит найти СРЕДНЮЮ стоимость комбо в этом заведении.

Чтобы получить среднее арифметическое из любого числового ряда, нам необходимо сложить вместе все члены этого ряда (найти их сумму), а затем поделить эту сумму на количество всех чисел в нашем ряду. Например, чтобы получить среднее от 3 и 5, мы сложим их, получим в сумме 8 и поделим её на два (потому что в нашем ряду всего два числа): в результате мы получим, что среднее арифметическое от 3 и 5 будет 4. В выражении школьной арифметики это выглядит так: (5+3) / 2 = 4.

Итак, начинаем писать наш код. Сперва давайте присвоим цены каждого комбо своим переменным — почти всегда это верный первый шаг в написании кода на JavaScript:

```
let комбоA = 6.75;
let комбоB = 7.5;
let комбоC = 5.75;
let комбоD = 8;
```

Каждому комбо теперь назначена своя переменная. Что же дальше? Нужно сложить их вместе!

```
let суммаВсехКомбо = комбоA + комбоB + комбоC + комбоD;
```

А теперь, чтобы получить среднюю цену на комбо, осталось просто поделить полученный результат на количество членов в ряду — то есть на общее количество комбо (помните, что символ `/` означает деление)!

```
let количествоКомбо = 4;
let средняяЦена = суммаВсехКомбо / количествоКомбо;
```

Ну и наконец, можно с чистой совестью попросить консоль вывести наш ответ:

```
средняяЦена;
```

Ответ: `7`

Итак, средняя стоимость комбо-набора у «Веснушчатого Фреда» составляет $7. Шикарно! Босс будет страшно рад, когда мы ему расскажем об этом…

Стоп! Срочная новость! Похоже, в меню у «Веснушчатого Фреда» появилось новое комбо!

Комбо E: Колли (два необжаренных хот-дога, маленькая картошка и загадочный приз) — $8.25

Придётся нам внести парочку мелких изменений, чтобы обновить среднее значение цены за комбо:

```
let комбоE = 8.25;
суммаВсехКомбо = суммаВсехКомбо + комбоE;
```

Обратите внимание, что мы НЕ использовали `let` во второй строке. Всё потому, что `суммаВсехКомбо` *уже* была объявлена раннее! Объявить её повторно мы не можем, так как получим в ответ ошибку. Вместо этого мы просто изменим нынешнее значение переменной на неё же ПЛЮС стоимость нового комбо.

Также давайте исправим и количество комбо в списке (заметьте, что мы НЕ пользуемся `let`, так как изменяем уже существующие переменные, а не объявляем новые):

```
количествоКомбо = 5;
```

Теперь, нажимая стрелку вверх, прокрутим список до тех пор, пока не доберёмся до следующей строки (ENTER пока не нажимайте!):

```
let средняяЦена = суммаВсехКомбо / количествоКомбо;
```

Поместите курсор в начало строки (самый левый край) и удалите `let` (поскольку переменная уже существует); у вас должна получиться такая строка (нажимаем ENTER!):

```
средняяЦена = суммаВсехКомбо / количествоКомбо;
```

Отлично! Теперь, после всех внесённых изменений, выведем новый результат:

```
средняяЦена;
```

Ответ: `7.25`

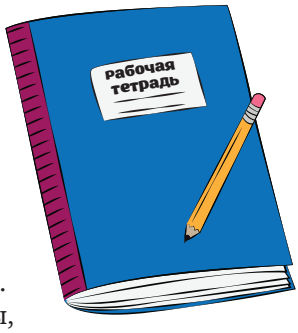
Ураа! Теперь мы выяснили также и НОВУЮ среднюю стоимость на комбо-набор в «Сосиски и картошка у Веснушчатого Фреда»!

Надеюсь, у вас не возникло проблем с этим небольшим проектом. Если же они всё-таки возникли, я советую вам открыть в Chrome новую вкладку и попробовать ещё раз с самого начала, прежде чем двигаться дальше.

ВИКТОРИНА

Все главы в этом разделе оканчиваются небольшими викторинами. Настоятельно рекомендую вам проходить их, по возможности не подглядывая в ответы. Цель этих мини-опросов — выявить, насколько хорошо вы усвоили основные моменты, которые мы обсуждали в прочитанной главе.

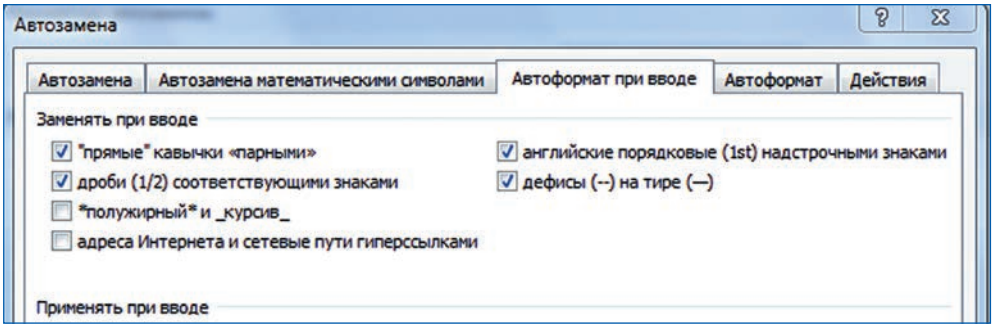
Очень важно: все свои ответы вы должны заносить в отдельный текстовый документ, чтобы в дальнейшем к ним можно было вернуться. Это можно проделать множеством различных способов. Например, если у вас или у родителей есть аккаунт Google (ну или если вы как раз собирались зарегистрироваться), то можно, открыв новую вкладку в браузере, перейти по адресу http://docs.google.com и создать новый документ, в котором будут храниться ваши ответы. Если вы не знаете, как это сделать, уверен — ваши родители с радостью вам помогут. Если же аккаунта на Google у вас нет, то можно воспользоваться любым доступным текстовым редактором вроде Microsoft Word, iWork Pages или OpenOffice. А можете и вовсе заносить свои ответы в Блокнот (Notepad) или TextEdit. Словом, вам решать, как и куда, но ответы на викторины, тесты, DIY-проекты и так далее должны у вас быть под рукой. Если они нам понадобятся, я буду называть документ с ними **Рабочей тетрадью**.



И ещё одно: обязательно отключите «умные кавычки»!

Если ваша Рабочая тетрадь будет в Google, то просто найдите в меню Инструменты в верхней части страницы, выберите Настройки и СНИМИТЕ галочки с пунктов «Писать слова с заглавной буквы» и «Заменять прямые кавычки парными».

Если же вы пользуетесь Microsoft Word, OpenOffice или Pages, то просто загуглите, как это делается: «Отключить автоматическую замену кавычек в <название_вашей_программы>».



Просто поверьте мне. Если что-то не выходит, попросите родителей помочь, но не оставляйте всё как есть; в противном случае ваш код может потом выдавать какие-нибудь неожиданные ошибки!

И ещё одно: помните, что все ответы вы найдёте в конце книги. Закончив отвечать на вопросы, вы должны сверить свои результаты с ответами, чтобы убедиться, что ничего не упустили. Даже если вы уверены, что всё правильно, — всё равно, пожалуйста, проверьте себя. Вам встретятся неочевидные вопросы и задачи с подвохом, ответы на которые необходимо будет проверить. Если папа или мама или ещё кто-то согласится вам помочь с ответами — будет ещё лучше!

Так, ладно, от слов к делу:

1. Какой URL (адрес) необходимо ввести в адресной строке Chrome, чтобы открыть в браузере совершенно пустую вкладку?
2. Какой комбинацией клавиш открывается консоль в Chrome?
3. Какие два ключевых слова используются для объявления новых переменных?
4. Какой символ зачастую ставится в конце предложения?
5. Каким символом обозначается операция *деления*?
6. Каким образом всегда ходят скобки?
7. Какую ошибку вы получите, запустив подобное предложение:

   ```
   let сумма = (9 + ; 3)
   ```

8. Какой способ использования прописных букв нужен при наименовании переменных в JavaScript?
9. Если у вас есть открывающая _____, то всегда должна быть и закрывающая _____.
10. Есть ли в следующем коде синтаксические ошибки (и если есть, то какие)?

    ```
    let моёНастроение = "Вот бы научиться писать код на JavaScript";
    let моёНастроение = "Не терпится использовать мои новые
                        суперспособности";
    моёНастроение;
    ```

11. Что означает одиночный знак равенства?

КЛЮЧЕВЫЕ ЭПИЗОДЫ

Все главы этого раздела также будут заканчиваться беглым обзором основных пройденных тем (плюс-минус в том же порядке, что и в главе). Пробегитесь по списку: если вы вдруг наткнётесь на что-то незнакомое или что-то, что вы не до конца поняли, — обязательно вернитесь к этой теме.

Итак, вот что мы делали в этой главе:

- Изучили необходимый минимум.
- Открыли пустую вкладку в браузере Chrome.
- Открыли консоль в Chrome.
- Проделали ряд операций с переменными.
- Проделали арифметические действия.
- Изучили синтаксис JavaScript.
- Изучили верблюжийРегистр.
- Рассмотрели сообщения об ошибках.

УПРАЖНЕНИЯ

Итак, мы дошли до секции упражнений; всё, что мы будем тут делать, необходимо прописывать в консоли. Если вы что-то не поняли и появились какие-то затруднения — без промедления заглядывайте в конец книжки! Даже если вы уже всё поняли и знаете, как и что тут нужно делать, эти упражнения помогут надёжно закрепить материал.

I. Введите следующие правильные фрагменты кода в консоль.

1.
```
let любимаяДобавка = "пепперони";
let втораяЛюбимаяДобавка = "сосиски";
let шикарнаяПицца = любимаяДобавка + " и " + втораяЛюбимаяДобавка;
шикарнаяПицца;
```

2.
```
var всего = 6 + 7 + 8;
всего;
```

3.
```
(8 * 3) / 6;
```

4.
```
var возрастБрата = 11;
var возрастСестры = 13;
var количествоБратьевИлиСестёр = 2;
var среднийВозраст = (возрастБрата + возрастСестры) /
                     количествоБратьевИлиСестёр;
среднийВозраст;
```

5.
```
let крутоеЧисло = 5;
let крутойОтвет = (20 / крутоеЧисло) + ((8 * крутоеЧисло) / (6 - 2));
крутойОтвет;
```

```
6. 6 + " футов " + 7 + " футов";

7. "Работаю с " + 9 + " до " + 5;

8. 7 + "3"; // здесь вы, быть может, удивитесь!

9. 12 / 0; // и здесь вы, быть может, тоже удивитесь!
```

II. Что не так с каждым из фрагментов?

Здесь вы должны будете определить, в чём проблема с каждой из указанных строк. Вам нужно будет открыть Рабочую тетрадь (тот документ, что мы создавали для ваших ответов), записать туда свои соображения, а затем (обязательно!) сверить их с ответами в конце книги. Подсказка: если на первый взгляд вам кажется, что всё нормально, то вбивайте каждую строку в консоль и внимательно следите: вдруг вам встретятся на пути наши старые добрые друзья-помощники — Сообщения об Ошибках?!

```
1. мояНеОбъявленнаяПеременная = 5;

2. let известнаяЦитата = "повар спрашивает повара;

3. 5 + 3 = x;

4. (4 * 7;)

5. var чтоТоНеТак = 2 + (9 / (1 + 2);

6. let чтоТоЕщёНеТак = 4 + (5 - 2));

7. 12 + x = 15;

8. let аРазвеНеНужнаТочкаСЗапятой = "Конечно нужна!"

9. let аэтоневерблюжийрегистр = "Поэтому довольно трудно прочесть";

10. var АЭТОЧТОЛУЧШЕ = "Не особо";

11. let а_может_быть_так = "Так, конечно, лучше, но удобнее и принято всё
                           же по-другому";

12. let ВотВродеТакПохоже = "Да, похоже! Попробуй ещё!";

13. let ну-вот-так-уж-наверное-точно = "Шутишь что ли? Там же вообще
                                       минусы между словами!";

14. let аЕслиТак= Вот так — хорошо!
```

Внимание: закончив упражнения, обязательно обратитесь к ответам в конце — помимо собственно ответов, там вас ожидает ещё парочка весёлых бонусов по поводу разных регистров и тому подобного.

Сделай сам: средний возраст моих родных

В конце каждой главы помещаются специальные проекты «Сделай сам»; в работе над ними вы будете самостоятельно применять навыки и информацию из пройденной главы (и глав, пройденных ранее). Никто вас не осудит, если для выполнения проекта вы даже откроете консоль и будете делать его там! Закончив писать свой код, выделите его и скопируйте (используя комбинацию клавиш CTRL+C или COMMAND+C для Mac) в свою Рабочую тетрадь (вставить скопированный фрагмент можно комбинацией CTRL+V или COMMAND+V для Mac). Затем откройте ответы в конце и сравните свой вариант с предложенным там. В идеале, конечно, сделать это после того, как вы закончили проект, но если вы застряли, то тоже ничего страшного.

Этот проект как две капли воды похож на тот, который мы уже делали ранее в разделе «Повторяй за мной». От вас требуется:

Определить средний возраст ваших ближайших родственников (мамы, папы, братьев, сестёр и, конечно, себя).

Закройте-откройте Chrome, откройте пустую вкладку, а затем консоль и пишите код. Обязательно пользуйтесь переменными вроде `возрастМладшегоБрата`!

Ещё раз: если вы не знаете, что дальше делать, обязательно обратитесь к «Рекомендациям для DIY-проекта» в конце книжки.

2 Время оперировать!

Вне зависимости от языка программирования, который вы изучаете, вы должны хоть чуть-чуть иметь представление о типах данных. Любая ваша переменная будет обладать определённым типом данных; умение определить, каким именно, является очень полезным навыком, который поможет вам избежать (или найти и исправить) множество ошибок в коде.

Числа

Итак, начнём с простого типа данных — с **чисел**!

Числа, как вы знаете, бывают **положительными** и **отрицательными** (а ещё есть 0 — нуль, который ни положительный, ни отрицательный). Любое число, пусть даже десятичная дробь с уймой знаков после запятой, никогда не берётся в кавычки. Вот несколько простых примеров чисел (не забывайте, что текст, выделенный синим, необходимо занос[illegible] в консоль; если забыли, как её открыть, верните[illegible] в начало первой главы):

```
4
282038273
-38
51.9
0
-0.000087
```

Мегапросто, да? Едем дальше!

Операции с числами

Арифметические действия

Врача! Врача! Скорее! Эти числа долго не протянут… Нужна ОПЕРАЦИЯ!

В предыдущей главе мы немного поиграли с такими крутыми штуками под названием **операторы** (сложением). Теперь мы подробнее рассмотрим их, а также выучим ещё парочку новых. Для начала быстренько пробежимся по уже известным нам:

Знак плюс (`+`) используется для выражения *сложения*: `4 + 4;`

Ответ: `8`

Минус (`-`) — для *вычитания*: `7 - 6;`

Ответ: `1`

Звёздочкой (`*`) изображается *умножение*: `3 * 4;`

Ответ: `12`

А прямым слэшем (`/`) — *деление*: `15 / 5;`

Ответ: `3`

Если в строке совершается более одного действия, то, чтобы отделить их друг от друга, а также сделать код более читабельным, мы пользуемся — `(`скобками`)`. Давайте наберём следующие предложения в консоли. Ответ по каждому из них должен состоять только из одной цифры:

```
3 * (2 + 1);
(3 + 9) / (10 - 6);
(2 + (3 * 4)) / (6 + 1);
(2 * (5 - (8 / 2))) * (3 + 1);
```

Отлично! На этом наш обзор выполнения простых арифметических действий при помощи консоли закончен. Надеюсь, это всё было вам уже известно, потому что мы переходим к чему-то *новому*…

Комбинированное присваивание (присваивание со сложением, присваивание с вычитанием и т. д.)

Вы уже не раз видели, как при помощи математических операторов происходит присваивание значений переменным. Например, так:

```
let сумма = 5 + 2; // значение суммы — 7
```

А ещё вы, наверное, не успели позабыть, что в любой момент можно изменить значение уже известной переменной (разве что вы решите снова прибегнуть к помощи `let`):

```
сумма = сумма + 3; // теперь значение суммы стало 10
```

С этим всё понятно; теперь проделаем другой занятный трюк. Можно пользоваться `+=` (читается как «присваивание со сложением») для того, чтобы быстро увеличить значение переменной! Это называется комбинированным присваиванием, поскольку сперва имеет место операция над переменной — собственно то, с чем комбинируется следующее затем присваивание переменной нового значения. Вот вам несколько примеров (обязательно пропишите их все в консоли):

```
let значение = 5;
значение += 2; // значение теперь 7 (то же самое, что значение =
               // значение + 2;)
значение += 3; // значение теперь 10 (то же самое, что значение =
               // значение + 3;)
значение = значение + значение; // 20 (а можно просто значение +=
                                // значение;)
значение += значение; // 40 (то же самое, что значение = значение +
                      // значение;)
```

Вы ведь уже догадались, что подобные штуки работают и с прочими математическими действиями, да?!

```
значение -= 25; // значение теперь 15 (то же, что и значение = значение -
                // 25;)
значение *= 2;  // значение теперь 30 (то же самое, что значение =
                // значение * 2;)
значение /= 3;  // значение теперь 10 (то же самое, что значение =
                // value / 3;)
значение; // Ответ: 10
```

Сейчас будет ещё несколько примеров; постарайтесь прежде чем прогнать их все через консоль, самостоятельно угадать ответ:

```
let ответ = 0;
ответ += 2;
ответ *= 30;
ответ -= 12;
ответ /= 6;
ответ *= 7 - 5;
ответ += ответ;
ответ;
ответ /= 4;
ответ -= ответ;
ответ;
```

Надеюсь, всё здесь было понятно практически интуитивно (в том смысле, что полученные результаты не шокировали вас своей неожиданностью). Если какие-либо моменты остались вам непонятны — просто вернитесь назад и пробегитесь по пройденному материалу с начала главы ещё раз. Если же и тогда что-то останется неясным — просто двигайтесь дальше (мы ещё не раз будем к этому возвращаться, так что рано или поздно оно усвоится).

Инкремент и декремент (плюс плюс и минус минус)

Хочу с вами разучить ещё парочку хитрых штуковин. Итак, инкремент — это такой оператор, который берёт наше число и прибавляет к нему единицу (иногда даже говорят — «инкрементирует»). Выглядит это так:

```
let счетчик = 0;
счетчик++;
счетчик++; // прибавить 1 (при помощи инкремента)
```

И, соответственно, декремент — противоположный оператор, эту единицу отнимающий (то есть «декрементирующий» число):

```
счетчик--; // вычесть 1 (при помощи декремента)
```

Для более чёткого понимания вот вам три способа, которыми можно выполнить операцию сложения (в подобных случаях зачастую лучше использовать инкремент):

```
счетчик = счетчик + 1;
счетчик += 1;
счетчик++; // инкремент (предпочтительный способ)
```

И, соответственно, три способа выполнить операцию вычитания (опять же — лучше использовать декремент):

```
счетчик = счетчик - 1;
счетчик -= 1;
счетчик--; // декремент (предпочтительный способ)
```

Оператор modulo (деление с остатком)

Ну как, готовы к чему-то посложнее? Отлично!

Итак, знаком процента (`%`) мы будем обозначать операцию взятия *остатка*: `9 % 2;`

Ответ: `1`

Видали когда-нибудь подобное? Сперва эта штука всем кажется странной, но после кажется очень дельной и нужной для грамотного написания кода! Знак процента (`%`) в программировании (выводится при помощи комбинации SHIFT+5)

обозначает использование оператора **modulo** (или просто *mod*), который осуществляет взятие **остатка** при **целочисленном делении**! Ясно?

В смысле «вы ещё сильнее не поняли, чем раньше?!» Хмм… ну… ладно, давайте попробуем по-другому.

Помните, когда вы в школе проходили деление, учитель предлагал вам примеры, которые преспокойно решались и так — безо всяких там остатков, десятичных и прочих дробей? Что-нибудь вроде:

9 / 3 = 3

10 / 2 = 5

8 / 4 = 2

Всё это делится весьма просто и гладко. Но что, если выбрать что-нибудь менее гладкое? Скажем:

5 / 2 = ?

Ну, вы ответите мне: «5 поделить на 2? Да легко! Это 2,5 (два с половиной)!» Что ж, конечно, вы будете правы. Но что, если нельзя прибегнуть к десятичным и любым другим дробям?

К примеру, перед вами 5 людей, которые пытаются поделить между собой 2 машины… По два человека уже разместились в каждой из машин, но ведь остался ещё один, который не сел ни в одну из них, — он и будет этим самым остатком. Не станете же, в конце концов, вы пилить его надвое (по крайней мере, я *надеюсь*)!

Мы рассмотрели стандартный пример «целочисленного деления» (или «**евклидового деления**», если вы уж прям такой педант). При таком делении числа никоим образом не дробятся — ведь они целые. Всё, что требуется — это найти, сколько раз одно число «помещается» (при делении) в другое; то, что останется после этого, и будет *остатком*! На уроках математики вам встретятся задачки подобного рода:

3 / 2 = 1 ост (1) // эта математическая запись читается так: 3 поделить на 2 равняется 1 и 1 в остатке

4 / 2 = 2 ост (0) // 4 поделить на 2 равняется 2 и 0 в остатке

5 / 2 = 2 ост (1) // 5 поделить на 2 равняется 2 и 1 в остатке

6 / 2 = 3 ост (0) // 6 поделить на 2 равняется 3 и 0 в остатке

7 / 2 = 3 ост (1) // 7 поделить на 2 равняется 3 и 1 в остатке

8 / 2 = 4 ост (0) // 8 поделить на 2 равняется 4 и 0 в остатке

Логика ясна? Вот так работает деление целых чисел с остатком (в отличие от дробей). Так вот, при работе с оператором modulo (`%`) нас будет интересовать *только остаток*. Оператор производит обычное деление, но *вместо* результата *сразу* выдаёт вам то, что осталось после него (то есть остаток).

Что ж, теперь давайте с помощью нашего нового оператора прорешаем те же примеры, что и выше. Набирайте в консоли (помните — только то, что выделено синим!):

```
3 % 2;
```

Ответ: `1` // 3 / 2 = 1 (результат деления, который оператор *игнорирует*) и 1 в остатке (то, что возвращается в качестве *ответа*)

```
4 % 2; // Ответ: 0 // 4 поделить на 2 будет 2 (результат) и 0 в остатке
5 % 2; // 1 // 5 поделить на 2 будет 2 и 1 в остатке
6 % 2; // 0 // 6 поделить на 2 будет 3 и 0 в остатке
7 % 2; // 1 // 7 поделить на 2 будет 3 и 1 в остатке
8 % 2; // 0 // 8 поделить на 2 будет 4 и 0 в остатке
9 % 2; // 1 // 9 поделить на 2 будет 4 и 1 в остатке
```

Всё понятно? Оператор возвращает только остаток и *больше ничего*. Ещё несколько примеров:

```
6 % 3;
```

Ответ: `0`

Ну как, ожидаемый ответ? `6` поделить на `3` будет `2`… верно? Верно. Ну а сколько же останется после такого деления? Нисколько! `6` делится на `3` нацело, то есть без *остатка*, так что и в ответ нам возвращается `0`.

```
7 % 4;
```

Ответ: `3`

Объяснение: `7` поделить на `4` (нацело) будет `1` и `3` в остатке.

```
4 % 5;
```

Ответ: `4`

Объяснение: `4` поделить на `5` (опять же — нацело) будет `0` (поскольку `5` «помещается» в `4` ноль раз). Таким образом, после неудачной попытки деления `4` на `5` в *остатке* у нас остаётся `4`.

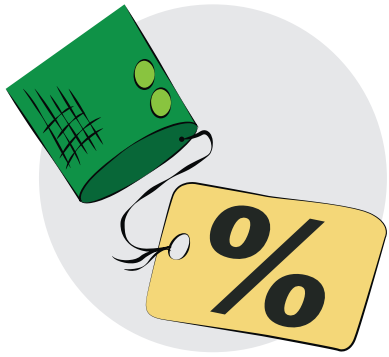
```
15 % 2;
```

Ответ: `1`

`15` поделить на `2` будет `7` (игнорируемый результат) и `1` в остатке.

Теперь я попрошу вас перенести все дальнейшие строки к себе в консоль и попытаться угадать ответ: *прежде* чем вы нажмёте клавишу ENTER, я хочу, чтобы у вас уже был готов свой ответ, который можно было бы сверить с ответом консоли.

```
3 % 3; // Ответ: 0
4 % 3; // Ответ: 1
5 % 3; // 2
6 % 3; // 0
7 % 3; // 1
8 % 3; // 2
9 % 3;
10 % 3;
11 % 3;
12 % 3;
13 % 3;
14 % 3;
15 % 3;
16 % 3;
17 % 3;
18 % 3;
```



Уловили модель? `0`, `1`, `2`, `0`, `1`, `2`, `0`, `1`, `2`, `0`, `1`, `2`… Совпадение? Не думаю! Эта модель повторится, даже если вы возьмёте тысячи (и проделаете с ними `x % 3;`). Подобным образом будут вести себя все примеры на взятие остатка.

Например, если у последовательности чисел брать остаток при делении на 2 (`x % 2;` — помните? мы это уже делали), то вам вернутся следующие ответы:

`0`, `1`, `0`, `1`, `0`, `1`, `0`, `1`, `0`, `1`, `0`, `1`, `0`, `1`, `0`, …

Если мы делили на 4 (`x % 4;`), то:

`0`, `1`, `2`, `3`, `0`, `1`, `2`, `3`, `0`, `1`, `2`, `3`, `0`, …

На 5 (`x % 5;`):

`0`, `1`, `2`, `3`, `4`, `0`, `1`, `2`, `3`, `4`, `0`, `1`, `2`, `3`, `4`, `0`, …

Словом, наибольшее число в ряду всегда будет меньше делителя (второго числа в уравнении, того, на которое мы делили).

И напоследок: помните, при делении на 2 у нас выходило `0`, `1`, `0`, `1`, `0`, `1`…? Так вот, благодаря очевидности этого ряда становится очень легко узнать, чётное число или нечётное! Любое число, возвращающее после деления `0`, является чётным, а любое, оставляющее `1`, нечётное! Убедитесь сами:

```
8 % 2;       // 0: значит 8 — чётное число!
17 % 2;      // 1: 17 — нечётное!
101 % 2;     // 1: 101 — нечётное!
```

```
5590 % 2;    // 0: 5590 — чётное число!
41703 % 2;   // 1: нечётное!
826834 % 2;  // 0: чётное!
```

Спасибо, что дослушали моё объяснение до конца. Тема довольно трудная (мои дети даже считают, что она самая-самая трудная во всей книге), но надеюсь, вы всё поняли. Если нет — попробуйте ещё раз пройтись по примерам в этой главе. Если всё равно не понимаете — просто двигайтесь дальше. Может, когда будете делать проект с этим оператором, всё встанет на свои места!

Повторяй за мной: сыграем в вышибалы

Наверное, самая трудная задача, когда-либо вставшая перед любым программистом, это найти способ избежать унижения быть вечно последним, когда набирают команды в вышибалы. Именно этим займёмся и мы. Мы разработаем простой и честный метод, пользуясь которым мы разобьём группу учеников на равные команды.

Но сперва, как обычно, откроем в Chrome новую вкладку, `about:blank`, и консоль — всё это вы уже должны уметь проделать, хоть посреди ночи вас растолкай!

Итак, вот что нам известно:

- на физкультуру ходят 12 учеников;
- у каждого из них есть ученический билет (личный ID учащегося) с номером от 1 до 12;
- для игры в вышибалы нужны ТРИ (да-да, я знаю! просто «нужно» и всё) равные команды;
- разбивка учеников по командам происходит ТОЛЬКО по номерам их ученических билетов (то есть ID), а НЕ по уровню их игры (или же отсутствию его);
- каждое предложение должно выбирать команду для одного из учеников.

С чего бы начать? Что ж, давайте сперва дадим название каждой из команд! Весь класс проголосовал, и выяснилось, что следующие названия оказались наиболее популярными: «Команда 1», «Команда 2» и «Команда 0» (хм, а я разве не упомянул, что у ребят в этом классе с фантазией не очень?).

Теперь давайте присвоим одной переменной значение количества команд:

```
let количествоКоманд = 3;
```

а следующей — номер билета первого ученика Эндрю:

```
let номерId = 1;
```

БТВ, а вы заметили, что у переменной `номерId` буква `D` — строчная? Почему же d, а не D — не прописная, чтобы было `номерID`? ID — **аббревиатура**, сокращение, при котором берутся лишь первые буквы составляющих её слов… так же как ЛОЛ, или БТВ, или вот моя любимая — ТЛ; ДР[1] (к этой книжке не имеющая, конечно, никакого отношения!). Так вот, «I» в ID — это сокращение от английского слова *identifier*… Короче, ID — это сокращение-аббревиатура, а они легче читаются, когда написаны верблюжьимРегистром вместе с прочими нормальными словами. В дальнейшем мы ещё вернёмся к этому.

А теперь будет круто! Помните оператор modulo, который брал остаток от деления? Вы, наверное, решили, что он довольно бесполезный и работы у него вообще никогда не будет, так? А вот нет! Глава ещё даже не закончилась, а он нам уже понадобился. Вот дела!

Вот та самая простенькая строчка кода, которая точно укажет, какой ученик попадёт в какую команду:

```
номерId % количествоКоманд;
```

Ответ: `1`

Команда 1! Пока всё работает отлично! Переходим к следующему ученику; Террелл говорит, что номер его ученического — `2`. Чтобы код, указанный выше, работал, нам необходимо *обновить* значение переменной `номерId` и вновь запустить наш код. Помните, что раз мы всего лишь *меняем* значение переменной (а не создаём новую), то использовать `let` нам не нужно.

```
номерId = 2;
```

Теперь, изменив значение нашей переменной, мы дважды нажимаем стрелку вверх, чтобы вернуться к нашему рабочему коду, и вновь запускаем его:

```
номерId % количествоКоманд;
```

Ответ: `2`

Команда 2! Настал черёд лучшего друга Террелла — Суреша (ученический билет номер `12`); ему бы очень хотелось попасть с Терреллом в одну команду. Но мы отвечаем ему, что любимчиков у нас нет и взяток мы не берём (а он так и так ничего вкусненького нам не предложил). Мы просто вновь запускаем наш алгоритм

---

[1] Прочно вошедшие в молодёжный интернет-сленг (в том числе и в русский) сокращения. ЛОЛ (LOL) — *англ.* laugh out loud/lots of laugh — громко хохотать/много смеяться; БТВ (BTW) — *англ.* by the way — между прочим, кстати; ТЛ; ДР (TL; DR) — *англ.* too long; didn't read — слишком длинно; не прочёл (часто: много букв; не осилил).

выбора. Давайте узнаем, сможем ли мы в одной и той же строке изменить переменную `номерId` и запустить наш код с оператором modulo! Сможем! — благодаря тому, что между ними стоит точка с запятой (`;`):

```
номерId = 12; номерId % количествоКоманд;
```

Ответ: `0`

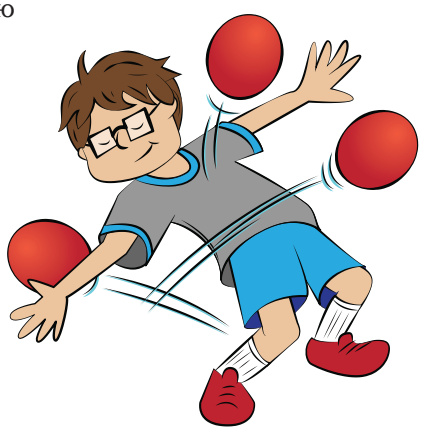
Команда 0! К сожалению, Терреллу и Сурешу не суждено сыграть в одной команде, но составы наших команды как минимум равные. Давайте распределим по командам оставшихся в классе учеников. Для каждого предложения вам нужно будет нажать стрелку вверх, изменить соответствующим образом значение переменной `номерId` и, нажав ENTER, выполнить код:

```
номерId = 3; номерId % количествоКоманд;  // 0
номерId = 4; номерId % количествоКоманд;  // 1
номерId = 5; номерId % количествоКоманд;  // 2
номерId = 6; номерId % количествоКоманд;  // 0
номерId = 7; номерId % количествоКоманд;  // 1
номерId = 8; номерId % количествоКоманд;  // 2
номерId = 9; номерId % количествоКоманд;  // 0
номерId = 10; номерId % количествоКоманд; // 1
номерId = 11; номерId % количествоКоманд; // 2
```

Шикарно! Наш код грамотно распределил всех учеников по командам, и никто не остался грустить на лавке (по крайней мере, пока кому-нибудь не расквасили мячом нос…)!

ВИКТОРИНА

Как и всякий раз, я настоятельно рекомендую отвечать на вопросы, не подглядывая в ответы. Записывайте ответы в Рабочую тетрадь (см. выше, в главе 1, по поводу тетради). Ответы для проверки вы, как обычно, найдёте в конце книжки — обязательно воспользуйтесь ими, после того как самостоятельно ответите на все вопросы!



1. Какое значение имеет символ `+` в математических операциях на JavaScript?
2. Каким символом обозначается умножение в JavaScript?

3. Какой это тип данных?

   `3456`

4. Какое значение имеет символ `/` в математических операциях на JavaScript?

5. К какому типу данных будет принадлежать переменная `чтоЗаТипДанных` после запуска следующей строки:

   ```
   let чтоЗаТипДанных = 5;
   ```

6. Что используется для разделения, группировки и определения порядка выполнения математических операций, а также для того, чтобы написанное легче читалось?

7. С помощью какого символа можно было бы упростить следующее предложение:

   ```
   мояПеременная = мояПеременная * 2;
   ```

8. Какой символ используется в JavaScript для оператора modulo?

9. Найдите простейший (то есть кратчайший) способ записи; как называется использованный символ?

   ```
   мояПеременная = мояПеременная + 1;
   ```

10. Найдите простейший (то есть кратчайший) способ записи; как называется использованный символ?

    ```
    мояПеременная = мояПеременная - 1;
    ```

11. Каким образом надо *изменить* следующее предложение, чтобы оно означало «три умножить на сумму четырёх и единицы» (подсказка: ответ консоли должен быть `15`):

    ```
    3 * 4 + 1;
    ```

12. Если бы вы писали код, который должен был бы определять чётные и нечётные числа, какую цифру вы бы использовали с оператором modulo?

13. Используя какой символ можно было бы упростить следующее предложение:

    ```
    моёЗначение = моёЗначение - 8;
    ```

14. Сколько уникальных (то есть *различных*) чисел в ряду ответов оператора modulo вы бы получили, если бы делали серию делений целых чисел на 5 (скажем, `11 % 5;` `12 % 5;` `13 % 5;` и так далее)?

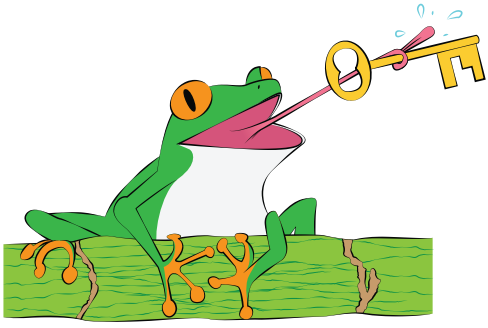
15. Как называют группу символов, которые сперва производят действие с переменной, а затем присваивают этой же переменной новое значение (**например**, `+=`, `-=`, `*=`, `/=`)?

16. Вы решили написать код на JavaScript, который бы задавал следующую последовательность мигания светодиодов: красный, синий, зелёный, красный, синий, зелёный, красный, синий, зелёный и так далее. Какой из известных нам математических операторов был бы наиболее полезен при написании подобного кода?

КЛЮЧЕВЫЕ ЭПИЗОДЫ

Как и во всех прочих «Ключевых эпизодах» в этой книге, пробегитесь по списку и, если вдруг найдётся что-то, что вы не до конца поняли, вернитесь к этой теме и попробуйте проработать задания по ней ещё раз. Итак, в этой главе мы обсуждали:

- Типы данных.
- Числа.
- Математические операции.
- Роль скобок в математических операциях.
- Комбинированное присваивание.
- Присваивание со сложением.
- Присваивание с вычитанием.
- Присваивание с умножением.
- Присваивание с делением.
- Инкремент (`++`).
- Декремент (`--`).
- Оператор modulo (взятие остатка).
- Знаки математических операций.
- Закономерности при работе с оператором modulo.
- Нахождение чётных и нечётных чисел.



УПРАЖНЕНИЯ

Вы можете решать следующие упражнения так, как вам больше нравится (я бы опять же посоветовал заносить все ответы в Рабочую тетрадь). Обязательно пробуйте решать их, заносить в консоль и в случае чего, не медля ни секунды, спешите заглянуть в ответы в конце! Даже если всё это вы и так уже знаете, эти упражнения помогут вам лучше запомнить изученный материал.

I. Введите следующие правильные фрагменты кода в консоль.

1. `8 * (7 - 5);`
2. `15 / (6 - 4 + 1);`
3. `(7 + (3 * 1) - 2) / (3 - 1);`
4. `((5 + 3) / (10 - 6)) * 3;`
5.
```
let сумма = 6;
сумма *= 5;
сумма -= 10;
сумма /= 4;
сумма += сумма * 3;
сумма;
```
6.
```
let сколькоМонетВКармане = 5;
сколькоМонетВКармане++;
сколькоМонетВКармане++;
сколькоМонетВКармане--;
сколькоМонетВКармане++;
сколькоМонетВКармане;
```
7. `14 % 3;`
8. `(20 - 10) % (8 - 4);`
9. `4 * (500 % 5);`
10. `2018 % 100;`
11. `15 % (3 + 1);`
12.
```
let количествоКоманд = 4;
let всегоУчеников = 17;
let номерКоманды = всегоУчеников % количествоКоманд;
номерКоманды;
```

II. Что не так с каждым из фрагментов?

Подсказка: вбейте строку в консоль, чтобы узнать, не выдаст ли она какое-нибудь полезное сообщение об ошибке?

1. `let этоЧисло % 2 = 0;`
2. `var 1 = 4 % 3;`
3.
```
let моёЛюбимоеЧисло = 5;
моёЛюбимоеЧисло = моёЛюбимоеЧисло += 6;
```
4. `(4 + (3 * 1) % 2;`
5. `let числоДляВзятияОстатка == 5 % 5;`
6. `let этоОстаток = делимое % делитель;`

```
7. let другойОстаток = 5 + 2) % 25;
8. 6 +* 3;
9. 5 % 25 = конечныйРезультат;
10. let большоеЧисло += 4;
11. let новоеЧисло = 5;
    новоеЧисло * 3 += 6;
12. let техническиНеОшибкаНоВсёЖеНеСамыйЛучшийСпособ = 5.25 % 4;
13. var хммммКакаяСтраннаяОшибка = 20 +/ 5;
```

КОМПЛЕКСНЫЙ ОБЗОР

В конце этой главы нас ждёт новая рубрика, которая станет в дальнейшем постоянной, — «Комплексный обзор». «Комплексный» означает «собранный воедино» (из разрозненных фрагментов). Задачки в этой рубрике похожи на известные нам «Викторины» и «Упражнения» в конце каждой главы, однако же они нацелены не на повторение и закрепление пройденного в главе материала, но на понимание и применение всего изученного до сих пор.

В дальнейшем вы убедитесь, что эта секция — одна из наиболее полезных и ценных, поскольку не позволит вам упустить из виду и забыть ничего из пройденного. *Все* ваши ответы следует заносить в Рабочую тетрадь. И не забудьте обязательно проверить себя по ответам в конце!

1. Как называется этот стиль: `вЭтойПеременнойЕстьПрописныеБуквы`?
2. Какой комбинацией клавиш открывается консоль в Chrome?
3. Является ли следующее предложение правильным (и если нет — почему)?
   ```
   var 1ыйНомерId = 1;
   ```
4. Является ли следующее предложение правильным (и если нет — почему)?
   ```
   let номерIdУченика12 = 12;
   ```
5. Так или нет: сообщения об ошибках — это помощь в написании кода.
6. Какую ошибку выдаст консоль, если в строке есть открывающая скобка, но нет закрывающей?
7. Каким оператором стоит воспользоваться, чтобы узнать, делится ли некое число нацело на 6?
8. Что необходимо ввести в адресной строке браузера, чтобы открыть пустую вкладку?

9. Каким символом можно воспользоваться, чтобы следующее предложение не изменило смысл, но было записано более коротко?

```
матемЗначение = 7 + матемЗначение;
```

10. Является ли следующий код правильным (и если нет — почему)?

```
var любимыеХлопья = 'Kix'; любимыеХлопья = 'Froot Loops';
любимыеХлопья;
```

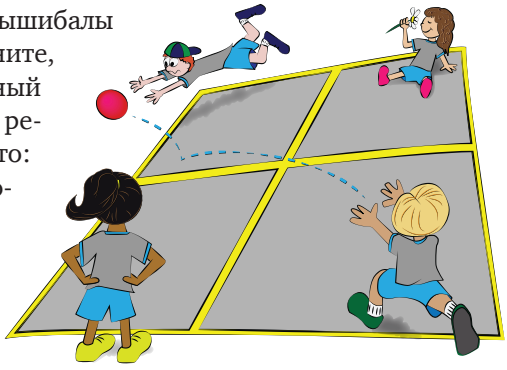
11. Является ли следующий код правильным (и если нет — почему)?

```
let любимыеХлопья = "Cap'n Crunch";
let любимыеХлопья = "Lucky Charms";
любимыеХлопья;
```

Сделай сам: сыграем в «квадрат»!

Ну и ну! Очередной проект «Сделай сам»! Ну вы помните, да? В этих проектах мы на практике и самостоятельно пытаемся применить идеи, которые обсуждали на протяжении главы. А ещё такие проекты часто очень похожи на задания из «Повторяй за мной»; так что если вы застряли — смело подглядывайте туда!

Итак, вот ваша задача.



Ребята из того же класса, что играли в вышибалы во время «Повторяй за мной» (вы помните, у каждого в этом классе есть уникальный номер ученического билета?), теперь решили сыграть в «квадрат». А ещё вот что: в классе появилось четыре новичка (номера ID — 13, 14, 15 и 16)!

Но вот незадача: они никак не могут поделить между собой «квадрат»! Большой квадрат разделён на четыре равных квадрата поменьше (`0`, `1`, `2` и `4`), каждый из которых вмещает ровно 4 учеников. Поскольку ваши блестящие способности помогли ребятам без труда разбиться по командам для игры в вышибалы, то они вновь обратились к вам с просьбой возглавить организационный процесс; то есть определить, на каком квадрате какая «четвёрка» будет играть (основываясь, конечно же, на `номераxId`)! Словом, чтобы определить порядок игры для всех 16 учеников, вам нужно будет написать формулу, схожую с той, что мы использовали для их игры в вышибалы.

На какую информацию вы будете опираться при написании кода:

- в классе 16 учеников;
- у каждого из них есть ученический билет с уникальным номером от 1 до 16;

- по правилам игры необходимо задействовать все четыре «малых» квадрата;
- распределение учеников по квадратам происходит исходя из номеров их ученических билетов;
- для распределения каждого ученика необходимо использовать отдельное предложение.

Закройте и заново откройте Chrome, откройте пустую вкладку, а затем консоль и запишите в неё свой код. Обязательно пользуйтесь переменными с понятными названиями вроде `номерКвадрата`. И не забывайте, что к уже написанному выше коду можно вернуться нажатием стрелки вверх. Всё это упростит и ускорит работу над проектом.

Если у вас не получается, не расстраивайтесь и не сидите часами над решением. Сперва попробуйте вернуться к игре в вышибалы (в «Повторяй за мной»), чтобы освежить в памяти порядок действий, но если и это не помогает, то *смело* листайте в конец книги — там вы найдёте советы и рекомендации по выполнению самостоятельных проектов. Но после того — не ленитесь и, посмотрев решение, закройте его и постарайтесь всё же выполнить проект от начала и до конца. Делайте так сколько угодно раз, пока наконец не убедитесь, что вы вполне можете делать подобные штуки!

3 Комментирование строк

В предыдущей главе мы встретились с понятием типов данных в JavaScript и сразу же рассмотрели числа в качестве таковых. Быть может, вы решили, что всё, на что годится JavaScript, так это на то, чтобы сделать из компьютера большой калькулятор?

Что ж, спешу вас успокоить: существуют и другие типы данных. В этой главе мы рассмотрим, пожалуй, наиболее важный и полезный из них, а именно — строки!

Попутно мы также познакомимся с ещё одной крайне важной языковой конструкцией на JavaScript…

Комментарии

Порой при написании кода вам необходимо будет снабдить его пояснениями, предназначенными только для людей, а НЕ для компьютерных машин. Возможно, это будет некое послание к себе из будущего (к Будущему Мне!) по поводу той или иной вещи в программном коде. А может, это будет просто указание, что запустить нужно лишь такую-то часть кода, а прочее надлежит опустить. Словом, в любом подобном случае вам необходимо будет прибегнуть к помощи **комментариев** (напоминаю, что определения встречающихся впервые и выделенных жирным шрифтом слов вы всегда найдёте в глоссарии в конце книги).

Приготовьтесь к работе в консоли. Закройте-откройте браузер, откройте пустую вкладку, запустите консоль (если не помните, как это делается, вернитесь к главе 1).

Однострочные комментарии

В общем-то, я уже несколько раз по ходу дела использовал однострочные комментарии, но как-то всё отмалчивался по этому поводу. Выглядит однострочный комментарий следующим образом (набирайте у себя в консоли):

```
// Привет! А я — комментарий!
```

Ответ: `undefined`

Когда интерпретатор видит два слэша подряд (`//`), он сразу понимает, что всё, что идёт далее до конца строки, можно смело проигнорировать. Это работает, если в строке есть и написанный код:

```
5 + 1; // это просто... я бы мог и в уме посчитать...
3 * 4; // а, теперь понятно...
8 + 2; // интерпретатор запускает код, но игнорирует комментарий
       // после него
```

Всё понятно?

Перенос

Мы обязательно продолжим разбираться с комментариями, но для того чтобы вы грамотно смогли перепечатать к себе в консоль дальнейший материал, сперва нужно разобраться, как же выполнить в консоли **перенос** на новую строку. В обычном текстовом редакторе вы просто жмёте ENTER (которая на Mac, конечно же, является клавишей RETURN), и курсор послушно перепрыгивает на новую строку.

Но вы ведь уже заметили, что нажатие ENTER в консоли производит несколько другой эффект, верно? Нажатием ENTER вы даёте машине понять, что вы закончили предложение (или предложения) и теперь желали бы ознакомиться с ответом интерпретатора. Именно таким образом мы пользовались клавишей ENTER на протяжении всех предыдущих глав. Но что, если вы всего-то лишь и хотели перейти строкой ниже? Для этого вам следует воспользоваться комбинацией SHIFT+ENTER (зажмите клавишу SHIFT, а затем нажмите ENTER). Кстати говоря, на вашей клавиатуре есть две клавиши SHIFT (слева и справа), вы вольны выбрать любую, которая вам больше по нраву.

Давайте наберём следующий код (вместе с комментариями) в консоль, используя перенос строки:

```
// Нажмите SHIFT+ENTER после этого комментария
var простоеСложение = 2 + 2; // и ещё раз
простоеСложение; // теперь просто ENTER. Консоль должна выдать
                 // в ответ "4"!
```

Если всё сделано верно, то в ответ вы получите `4` и не получите `undefined` где-то в процессе. Всё получилось? Если нет, и вы получили в каком-то месте в ответ

`undefined`, это означает, что вы таки не выполнили перенос строки комбинацией SHIFT+ENTER. В общем, попробуйте заново и убедитесь, что на этот раз перенос выполнен правильно.

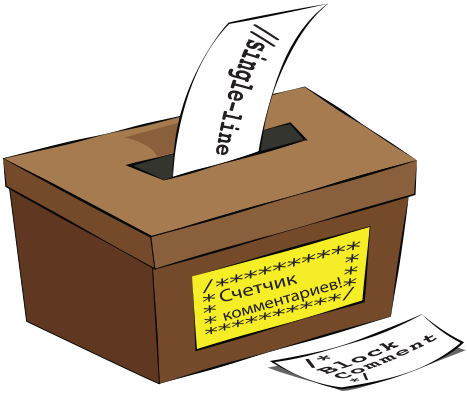
Кстати: как вы понимаете, никаких синтаксических ограничений и правил для комментариев не существует. Словом, в комментариях невозможно получить синтаксическую ошибку, поскольку они пишутся для того, чтобы их прочёл человек (который, конечно, намного умнее любой машины, так что вам не нужно следить за каждой мельчайшей закорючкой, коль скоро понятен основной смысл).

Давайте наберём следующий код в консоли (не забывайте про переносы):

```
// моя нынешняя зарплата (если округлять до ближайшего миллиона)
let моягодоваяЗарплата = 1000000;
// 365 — количество дней в году
let заработокЗаДень = моягодоваяЗарплата / 365;
заработокЗаДень; // вот столько я получаю в день (ну, может быть,
                 // чуть-чуть меньше)
```

Всё так? Едем дальше? Отлично! Тогда перейдём к главному блюду этой главы — к строкам!

Блок комментариев



Чего? Блок комментариев? А строки-то где? Опять «потом»?! Да сколько можно уже!

Ну да, ну да, строки будут чуть ниже. Я постараюсь разобраться с этим как можно скорее, чтобы уж точно перейти к самым что ни на есть строкам.

Итак, **блоки комментариев** похожи на однострочные комментарии и ведут себя схожим образом: только вместо того, чтобы игнорировать всего *одну* строчку, интерпретатор будет игнорировать *все*, пока не обнаружит окончание блока комментариев. Объясняю: блок комментариев открывается таким образом — `/*` и закрывается таким — `*/`. Как только интерпретатор видит `/*`, он думает: «Ага, значит, пока я не увижу `*/`, то могу ни на что здесь не обращать никакого внимания!» Вот пример (не забывайте использовать SHIFT+ENTER для переноса!):

```
/* вот вам комментарий, в котором нужно
использовать перенос строки. Интерпретатор
будет всё игнорировать, покуда он не увидит...
*/
4 + 9; /* а вот это уже сработает! */
```

Сработало? Тогда ещё один:

```
let любимыйСуперГерой = "Зелёный Фонарь";
/* Шучу-шуткую!
Никто в здравом уме не считает Фонаря своим любимым супергероем!
любимыйСуперГерой = "Бэтмен"; // и что, интерпретатор прочтёт это?
                              // НЕТ, КОНЕЧНО!
*/
любимыйСуперГерой; // Ахаха!
/* Мы ведь всё ещё дурим консоль: она же не может читать то, что указано
   в блоке комментариев! Говорил же, что люди куда умнее машин! */
```

Ну как, всё ясно? По поводу блоков комментариев можно было бы ещё долго разглагольствовать, но мне, как и вам, уже не терпится перейти к чему-нибудь новенькому, так что — едем далее!

Строки

Когда-то в первой главе мы уже касались темы строк, но тогда я довольно бегло объяснил этот момент, потому что информации и так было крайне много, и я просто ~~поленился~~ не хотел излишне перегрузить вас.

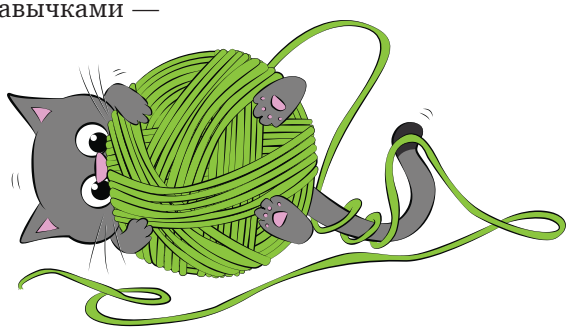
Пожалуй, из всех типов данных строками вы будете пользоваться наиболее часто. Чтобы создать новую строку, необходимо использовать либо `'одинарные'`, либо `"двойные кавычки"`. Рассмотрим парочку примеров:

```
"Привет, Мир!"
'Привет'
"В строке может быть слово, предложение, хоть целый параграф или даже
несколько страниц текста"
'12345'
```

У-у-пс! А последнее-то как сюда затесалось? Тот самый неловкий момент!.. Это ж числа, разве нет?

На самом деле — нет! Это — *строка*, поскольку заключена в одинарные кавычки! Ну что, как? Надурил я вас (ну хоть на секундочку)?

В общем, внимательно следите за кавычками — одинарные они или двойные: если вы начинаете строку одинарной кавычкой (`'`), то, хотя внутри и могут быть двойные, завершить её надлежит также одинарной (`'`). И, соответственно, если строка начинается двойной кавычкой (`"`), то и завершиться она должна также двойной.



А теперь, пока мы ещё не возомнили себя мегакрутыми хакерами, давайте-ка наделаем пару-тройку ошибок. Но сначала закройте все открытые программы (чтобы перед вами был лишь рабочий стол) и по памяти сделайте всё необходимое, чтобы открыть рабочее окно консоли (если что — см. главу 1).

Всё готово? Давайте рассмотрим наши ошибки и выясним, как же их исправить; набирайте в консоли:

```
let чтотоНеТак = 'Скажите, д'Артаньян, разве я не прав? ';
```

И консоль отвечает: `Uncaught SyntaxError: Unexpected identifier`

```
" "Разумеется, правы, чёрт побери!" — вскричал он.";
```

И в ответ: `Uncaught SyntaxError: Unexpected identifier`

Уловили, что там было не так? Если нет, приглядитесь повнимательнее, присмотритесь к выделению цветом… Ну как, увидели? Цветом выделена часть, которую интерпретатор и счёл нашей строкой — то есть, по его мнению, апостроф в `д'Артаньяне` был ЗАКРЫВАЮЩЕЙ строку одинарной кавычкой; а ведь строка-то ещё и не думала кончаться! Точно так же двойные кавычки перед словом `Разумеется` интерпретатор расценил как закрытие всей строки.

К счастью, всё это несложно поправить: вам нужно просто *исправить* пару кавычек, которые должны открыть и закрыть строку, с одинарных на двойные или наоборот (в зависимости от того, в чём была ошибка). Выглядит это так:

```
let вотТеперьПорядок = "Скажите, д'Артаньян, разве я не прав?" ; // как
                    // видно, эта строка теперь заключена в двойные кавычки
' "Разумеется, правы, чёрт побери!" — вскричал он.'; // а эта —
                    // в одинарные
```

Всё понятно? Вот ещё пара примеров на использование кавычек:

```
let строкаВодинарныхКавычках = 'Красный цвет — "дороги нет", а зелёный
                                свет — "кати!" ';
let строкаВдвойныхКавычках = "Ну вот, теперь я — мегапрограммист
                              на JavaScript'е.";
```

Знаки

Строки состоят из элементов, которые принято называть **знаками**. Пожалуй, вы бы сказали «буквами», но ведь в строках частенько присутствуют числа или символы. Вот, например:

```
"а"; // строка из одного знака
"абв 123"; // строка из семи знаков (пробел тоже считается за символ)
"секретный_П@рОль!!1"; // строка из восемнадцати знаков
```

Кстати, *знаком* мы также называем и любую букву, цифру и символ, которые находятся в *строке* программного кода (даже когда они *не* являются частью строки). Скажем:

```
"где же точка с запятой?"; // точка с запятой находится (за пределами
          // строки, ) в самом конце и, соответственно, является 26 знаком
"а где знак вопроса?"; // вопросительный знак (находится внутри строки
          // и -) является здесь 20 по счёту знаком
```

Обратный слэш

Теперь немного усложним дело:

```
let аВотЕщёОднаОшибка = ' "Лицемер!" — воскликнул д'Артаньян';
```

Ответ: `Uncaught SyntaxError: Unexpected identifier`

Ёлки зелёные! Да тут же в строке есть *и* одинарные, *и* двойные кавычки! И что нам теперь делать? К счастью, если уж нас совсем припёрли к стенке, то на выручку спешит наш верный друг и товарищ — \\\\\\\\\\\ **ОБРАТНЫЙ СЛЭШ**! \\\\\\\\\\\\\\

Поищите обратный слэш (\) на своей клавиатуре; часто он расположен над или сбоку от ENTER. Некоторые, впрочем, почему-то не доверяют нашему новому другу и товарищу — обратному слэшу… Быть может, они боятся, что если они слишком сблизятся с ним, то он всё равно обернётся к ним спиной, и они будут грустить и скучать? Но не бывать тому: не таков наш верный друг и товарищ обратный слэш! Он всегда готов прийти на помощь в трудную минуту. И вот как он это делает:

```
let строкаСобратнымСлэшем = ' "Лицемер!" — воскликнул д\'Артаньян';
let другойВариантСобратнымСлэшем = "Лицемер!\" — воскликнул д'Артаньян";
```

Такие штуки мы, программисты, называем «экранированием». В общем, всё сводится к простому: строку можно заключить либо в одинарные, либо в двойные кавычки; если же вам хочется использовать *внутри* строки *те же* кавычки, что стоят и снаружи, то их нужно будет **экранировать** при помощи нашего друга — обратного слэша. Всё ясно? Вот вам ещё несколько примеров на закрепление:

```
"Мой доктор настоятельно советует принимать \"Миланту!\" ";
'Я не буду использовать в английском "ain\'t", потому что "ain\'t" —
неверная форма глагола.';
```

```
'А ведь когда-то "подъезд" и "объявление" писали как "под\'езд"
и "об\'явление".';
```

Если всё ещё не совсем поняли, попробуйте поиграть с кавычками и слэшем в консоли — уверен, всё станет на свои места! Едем дальше!

Объединение строк

Сейчас я поясню ещё один момент из первой главы. Во время работы со строками (ну вы помните, такими штуками с одинарными и двойными кавычками вокруг, да?) особое значение приобретает знак плюс (`+`); он означает здесь не *сложение*, а *объединение*! Набирайте в консоли:

```
"Саймон"+"считает"+"что"+"можно"+"и"+"без"+"пробелов"+"прекрасно"+
"печатать.";
'Но '+'ведь '+'с '+'пробелами '+'гораздо '+'удобнее '+'читать '
+'напечатанное!';
"В "+"этом "+"предложении "+ 4 +" пробела.";
'Ух ты, глядите! А ведь цифру "' + 7 + '" тоже можно объединить!';
```

В общем, всё это в ответе консоли вернётся цельной строкой. Так работают типы данных. Если к цифре (*числу*) добавить другую, то всё равно получится *число*. Соответственно, если к строке добавить (**объединить** с ней) другую (третью, четвёртую…), то и получится *строка*. Вопрос: а какой тип данных у вас получится, если объединить строку с числом (как мы проделали только что)? Ответ прост: ещё одна строка.

Именно поэтому в возвращаемом консолью значении этих предложений все они являются строками (как видите, все ответы также заключены в кавычки). В консоли:

```
"1"; // и это строка, а не цифра
'1' + '2';
"12" + 3;
'1' + ("2" + "3") + 4;
1 + ("234" + "5");
12 + '345' + 6;
var последниеДвеЦифры = 67; "12345" + последниеДвеЦифры;
'1234567' + (4 * 2);
let началоСтроки = '1234'; началоСтроки += 56 + ("78" + (3 * 3));
началоСтроки;
```

Заметили присваивание со сложением (`+=`) в последнем предложении? В предыдущей главе мы при помощи этой штуки прибавляли к нашей переменной число, а затем присваивали ей получившееся новое значение. Так вот, тут почти то же самое, за исключением того, что вместо «сложения» мы «объединяем» строки между собой, а затем присваиваем новое значение.

За все годы моей компьютерной жизни я, пожалуй, ни разу не встретил человека, который бы заявил, что объединение строк является его любимым занятием во время написания кода. Это скорее такой базовый навык, который нужно просто

освоить, чтобы потом понимать более крутые и сложные вещи. Надеюсь, эта глава вам в этом поможет.

Повторяй за мной: пишем биографию знаменитого писателя

А теперь смастерим очередной забавный и интересный проект. Но! Сперва знаете что? Всё закройте и снова откройте, чтобы иметь перед собой чистое, готовое к новой работе окно консоли. Можете ворчать и сетовать на меня, зануду, но в будущем (вот увидите!) вы не раз помянете эти надоедающие повторения добрым словом!

Итак, вот наша задача.

После оглушительного успеха романа «Буду вторым» знаменитый писатель Гюго Первый захотел на своём веб-сайте вывесить биографию. В общем, он обратился к нам с этой просьбой. Там должны быть указаны его личные данные вроде имени, возраста, последней изданной книги, хобби и любимой цитаты из собственной книги. Также он пожелал, чтобы мы снабдили наш код однострочными и блочными комментариями (уж не знаю, зачем ему это понадобилось, ну да ладно). И наконец, он бы хотел, чтобы наш код был снабжён универсальными переменными, чтобы его потом можно было использовать как макет для таких же страниц других писателей, сотрудничающих с его издательством!

Как и всегда с нашими проектами, я бы хотел начать с того, что мы запишем то, что нам известно (не забывайте использовать SHIFT+ENTER для написания блока комментариев):

```
/*
Переменные, которые мы будем использовать
*/
let имяПисателя = 'Гюго';
let фамилияПисателя = 'Первый'; // — нужно только для того, чтобы потом
                                // создать "полноеИмяПисателя"
// let отчествоПисателя = 'В.'  // просто комментим для количества
                        // комментариев (его отчество нам не понадобится).
let полноеИмяПисателя = имяПисателя + ' ' + фамилияПисателя; // понимаете?
let названиеКниги = "Буду вторым";
let возраст = 25; // помните? когда число присоединяется к строке,
                  // оно также становится строкой
let хобби = 'нырять со скал';
let цитата = 'If at first you don\'t succeed, maybe cliff diving isn\'t
for you.';[1]
```

А теперь давайте всё объединим в цельную биографию нашего молодого классика:

[1] Перевод с английского: если у вас не вышло с первого раза, то, возможно, скалоныряние — не ваше.

```
let bio = полноеИмяПисателя + ' — '
+ возраст + '-летний писатель, автор нашумевшего романа "'
+ названиеКниги + '". В свободное время ' + имяПисателя
+ ' любит ' + хобби + ' и проводить время в кругу родных и близких. '
+ "Вот любимая цитата " + имяПисателя + ' из "'
+ названиеКниги +'": "'+ цитата+'"';
```

Вы обратили внимание на широкое применение одинарных и двойных кавычек, а также обратного слэша? Очень надеюсь, что вас это нисколько не смутило. В общем-то, я хотел лишь показать, что вам есть из чего выбирать; лично я предпочитаю одинарные кавычки и, когда это уместно, использую их для строк в JavaScript (скажем, когда в строке присутствует апостроф, я пользуюсь двойными кавычками). Кто-нибудь, конечно, наоборот — предпочитает по умолчанию пользоваться двойными кавычками. Словом, можно и так, и так.

Что ж, теперь давайте посмотрим на конечный результат:

```
// показать полностью bio
bio;
```

И ответ будет (если всё верно): `"Гюго Первый — 25-летний писатель, автор нашумевшего романа "Буду вторым". В свободное время Гюго любит нырять со скал и проводить время в кругу родных и близких. Вот любимая цитата Гюго из "Буду вторым": "If at first you don't succeed, maybe cliff diving isn't for you.""`

Получили вы верный ответ консоли? Если нет, то сперва проверьте, не было ли каких сообщений об ошибке — они могут весьма ускорить процесс «ловли жуков» (то есть исправления багов). Далее убедитесь, что в точности перепечатали текст кода из книги в консоль. Обязательно отдельно проверьте все одинарные и двойные кавычки, апострофы, знаки плюс и обратные слэши.

Если думаете, что вам удалось найти и исправить неполадку, попробуйте поменять писателя и книгу (скажем, на приключенческий роман Говарда Йю «Я в порядке») и посмотреть, что из этого выйдет. Не забывайте, что если вы меняете уже объявленные переменные, то следует убирать `let` (иначе вы получите синтаксическую ошибку!).

ВИКТОРИНА

Заносите свои ответы в Рабочую тетрадь. Не подглядывайте в ответы! Проверьте себя по ответам в конце, только когда закончите (даже если вы суперуверены, что ответили на всё правильно!).

1. Комментарии к коду будет читать _____:
    а) интерпретатор JavaScript;
    б) человек;
    в) консоль.

2. Какой тип данных является наиболее часто применяемым в работе с JavaScript?
3. Интерпретатор JavaScript _____ все встреченные им комментарии:
   а) исполнит (запустит);
   б) скомпилирует;
   в) проигнорирует.
4. Каким символом обозначается однострочный комментарий?
5. Какой комбинацией клавиш делается перенос строки в консоли?
6. Строка в JavaScript заключается в _____.
7. Каким символом открывается блок комментариев?
8. Каким символом закрывается блок комментариев?
9. Истина/Ложь: при работе со строками намного лучше применять двойные кавычки, нежели одинарные.
10. Строка состоит из одного или более _____.
11. Что означает знак `+` при работе со строками в JavaScript?
12. Какой символ нужно *добавить* в следующее предложение, чтобы избежать синтаксической ошибки?

```
let цитатаИзФильма = 'The name's Bond. James Bond.';
```
[1]

13. Как бы можно было *изменить* (ничего не добавляя) 2 знака в следующем предложении, чтобы избежать синтаксической ошибки?

```
let ещёОднаЦитатаИзФильма = 'Here's looking at you, kid.';
```
[2]

КЛЮЧЕВЫЕ ЭПИЗОДЫ

Как и всегда в «Ключевых эпизодах», здесь я советую вам пробежаться по нижепредставленному списку и, если в нём обнаружится что-то, вам непонятное, вернуться на пару страниц назад и повторить материал. Итак:

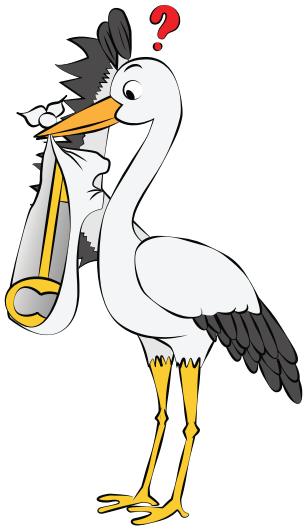
- Комментарии.
- Однострочные комментарии.

---

[1] С *англ.* — «Меня зовут Бонд. Джеймс Бонд».

[2] Цитата из знаменитого фильма 1942 года «Касабланка», переведённая (не совсем точно) в дублированной версии фильма как «Удачи тебе, малышка». — *Примеч. пер.*

- Перенос строки.
- Блоки комментариев.
- Строка как тип данных.
- Одинарные и двойные кавычки при работе со строками.
- Символы.
- Экранирование обратным слэшем.
- Объединение строк.
- Присваивание со сложением при работе со строками.
- Объединение чисел со строками.



УПРАЖНЕНИЯ

I. Введите следующие правильные фрагменты кода в консоль.

1. `// Однострочный комментарий`

2.
```
/* Блок комментариев. Не забывайте пользоваться сочетанием SHIFT+ENTER
для переноса строки в консоли. */
```

3.
```
/*******************************
* Ну, раз уж блок комментариев  *
* может вмещать больше одной    *
* строчки кода, то можно делать *
* такие вот премилые рамочки!   *
*******************************/
```

4.
```
var любимаяШоколадка = "Алёнка"; // Хмм… ну, нет, пожалуй
любимаяШоколадка = 'Snickers'; // Да, вот так-то лучше
любимаяШоколадка += ' двойной, с орехами'; // не жирно ли?
// любимаяШоколадка = KitKat; // сначала так написал, потом
// в комментарии отправил
любимаяШоколадка; // "Snickers двойной, с орехами" (потому что "KitKat"
                  // был сослан в комментарии).
```

5.
```
let строкаВОдинарныхКавычках = 'Мы уже приехали?';
строкаВОдинарныхКавычках;
```

6. `let строкаИзЧетырёхСимволов = 'Нет.';  строкаИзЧетырёхСимволов;`

7.
```
let объединённаяСтрока = 'Мы уже приехали? '  + 'Мы уже приехали? '  +
'Мы ' + 'уже ' + 'приехали?';
объединённаяСтрока;
```

```
8. let строкаВДвойныхКавычках = "Мы приедем тогда, когда приедем.";
   строкаВДвойныхКавычках;

9. let строкаВОдинарныхКавычкахСкавычкамиВнутри = 'Но мама сказала:
   "Ты и опомниться не успеешь, '+ 'как мы приедем"!'
   строкаВОдинарныхКавычкахСкавычкамиВнутри;

10. let строкаВДвойныхКавычкахСобратнымиСлэшамиВнутри = "А ещё я тебе
    говорила: "+ "\"Хватит дёргать папу, когда он за рулём\"!";
    строкаВДвойныхКавычкахСобратнымиСлэшамиВнутри;

11. let строкаВОдинарныхКавычкахСАпострофамиИОбратнымиСлэшамиВнутри =
    'Но яйаа таааак ' + 'устааала, а мы всё под\'езжаем да под\'езжаем,
    а никак не приедем, а уснуть, прислонившись к окну, я не могу.';
    строкаВОдинарныхКавычкахСАпострофамиИОбратнымиСлэшамиВнутри;

12. let объединённыеСтрокиСАпострофом = "Если не прекратишь канючить
    и коверкать 'подъезжаем' в 'под'езжаем', " + "я сейчас же остановлю
    машину, и мы пойдём пешком!";
    объединённыеСтрокиСАпострофом;
```

II. Что не так с каждым из фрагментов?

Подсказка: набирайте фрагменты в консоли и ловите полезные сообщения об ошибках!

```
1. // let чтоМоглоПойтиНеТак = 'Ах-ох';
   чтоМоглоПойтиНеТак;

2. let любимыйСериал = 'Леди Баг и Супер-Кот'; /* Просто чудовищно!
   любимыйСериал;

3. let ледиБагГероиня = 'Прямо как Жанна Д'Арк!';

4. //* Просто-напросто лучший
       блок комментариев
   **/

5. /** ну ещё один // блок комментариев // и чего ? **//

6. var кТестуГотов = 'но ' + 'что-то ' * "пошло " + "не так!";

7. var песняИзФильмаПроЛедиБаг = 'the ' + 'luck' + i + 'est';

8. let кто = "выпустил " + \""собак?\"";[1]
```

КОМПЛЕКСНЫЙ ОБЗОР

Упражнения и вопросы в этом разделе могут касаться любой пройденной нами темы. Если будет необходимость, обязательно проверяйте код в консоли; не ле-

[1] Известная некогда песня *Who let the dogs out* группы *Baha Men*.

нитесь заносить свои ответы в Рабочую тетрадь. И не забудьте потом проверить себя по ответам в конце!

1. Какой комбинацией клавиш открывается в Chrome консоль?
2. Правильно ли написано следующее предложение (и если нет — почему?):
   ```
   (10 + (4 * 2) - 5) / (6 - 7);
   ```
3. Правильно ли написано следующее предложение (и если нет — почему?):
   ```
   let гонщикНомер8 = 8;
   ```
4. Что нужно ввести в адресной строке браузера, чтобы открыть пустую вкладку?
5. Правильно ли написано следующее предложение (и если нет — почему?):
   ```
   let допустимоеЧисло % 4 = 2;
   ```
6. Истина/Ложь: сообщения об ошибке являются показателем того, что что-то не так.
7. Как можно было бы короче записать вторую строчку в этом коде, получив при этом тот же результат:
   ```
   let возраст = 12;
   возраст = 1 + возраст;
   ```
8. Где разработчик может легко и просто проверить работу написанного кода и получить немедленный ответ интерпретатора?
9. Какой ошибкой на следующий код отреагирует консоль?
   ```
   let letterBeforeM = 'N'; let letterBeforeM = 'L'; letterBeforeM;
   ```
10. Правильно ли написано следующее предложение (и если нет — почему?):
    ```
    let делимое = 10;
    let делитель = 3;
    let сообщениеОбОстатке = 'При евклидовом делении ' + делимое
        + ' на ' + делитель
        + ' мы имеем в остатке ' + (делимое % делитель);
    сообщениеОбОстатке;
    ```
11. Chrome, Firefox, Internet Explorer, Edge и Safari, все они являются _____.
12. Каким образом следует писать имена переменных на JavaScript?
13. Что используется для разделения, группировки и определения порядка выполнения математических операций, а также для того, чтобы написанное легче читалось?
14. Что это за тип данных?
    ```
    '2000'
    ```
15. Что это за тип данных?
    ```
    "Чашечка отменного кофе из медного кофейника"
    ```

16. Что означает одиночный знак равенства в предложении?

17. Как можно было бы короче записать следующий код, получив при этом тот же результат:

```
let любимоеБлюдо = "Курочка";
любимоеБлюдо = любимоеБлюдо + " Буррито";
```

18. Как называется оператор JavaScript, при котором используется знак процента?

19. Правильно ли написан следующий код (и если нет — почему?):

```
let любимыеЧипсы = "Fritos";
// let любимыеЧипсы = "Doritos";
// любимыеЧипсы;
```

20. Правильно ли написан следующий код (и если нет — почему?):

```
/* let любимыйПопкорн = "С маслом";
let любимыйПопкорн = "С карамелью"; */
let любимыйПопкорн;
```

Сделай сам: ваша биография

Все уже в курсе, какой замечательный код вы смастерили для знаменитых литераторов, так что вы и сами стали настоящей знаменитостью! Ну и теперь все хотят знать биографические подробности о супер-мега-знаменитом JavaScript-прогере; они хотят узнать о ваших хобби, домашних животных, навыках в программировании, школьных оценках и любимых мультиках!

Обязательно используйте переменные, однострочные комментарии, блоки комментариев и объединение строк (как в «Повторяй за мной») и создайте самую лучшую биографию для своей собственной нежданно-негаданно прославившейся персоны!

Не забывайте про Рабочую тетрадь! Если возникнут затруднения, загляните в ответы в конце, но не перепечатывайте ответ! Переверните страницу и попробуйте вновь самостоятельно. Помните: повторение — мать учения!

4 Вы хотите функций? Их есть у нас!

Если вы никогда не изучали языки программирования, то, пожалуй, слово «функция» не произведёт на вас никакого эффекта. Но как человеку, который сталкивается с этими самыми функциями каждый день, мне просто не терпится начать рассказывать вам о них! Все те проекты в «Сделай сам» и «Повторяй за мной», которые мы до сих пор выполняли… я каждый раз ловил себя на мысли: «Вот на самом деле я бы это делал через функцию, но ведь мы их ещё не обсуждали… Терпение… Ещё немного: вот-вот уже подойдёт глава 4!». Что ж, вот наконец и она, глава 4! Скорее за дело!

Функции: ключевые понятия

«Но что же такое эти функции?» — спросите вы. Рад, что спросили! В программировании **функция** — это отдельный (самостоятельный) блок программного кода, который вызывается специально для выполнения конкретного действия. Прежде чем мы двинемся дальше, пожалуйста, закройте-откройте всё, что нужно, чтобы иметь перед собой чистое рабочее окно консоли.

Объявление функции

Ниже представлен пример простейшей функции. Обратите внимание на фигурные скобки (зачастую они расположены недалеко от клавиши ENTER). И не забывайте пользоваться SHIFT+ENTER для переноса:

```
function вывестиЦифруПять() {
    return 5;
}
```

То, что мы только что проделали, называется *объявлением* функции. После того как функция была **объявлена**, то есть когда мы «познакомили» наш код с ней, её уже можно будет использовать в дальнейшей работе. Давайте рассмотрим всё это шаг за шагом:

- Ключевое слово `function` указывает интерпретатору на то, что следующий далее код является **пользовательской**, то есть созданной вами, а не встроенной, функцией.
- Написанное верблюжьимРегистром `вывестиЦифруПять` является пользовательским *названием* этой функции. Для интерпретатора в общем-то нет разницы, как именно называется эта функция, но лучше давать функциям названия, из которых чётко следует, что именно они делают.
- `(`Скобки`)` — обязательный элемент любой функции. Порой в скобки заключается одна, две и так далее переменных; в нашем случае в скобках нет ничего.
- В `{`фигурные скобки`}`, зачастую расположенные над ENTER, на клавише с `[`квадратными скобками`]`, должно быть заключено само тело функции (то есть всё, что в неё непосредственно входит.
- Само тело функции принято выделять *отступами* справа (зачастую при помощи клавиши TAB); это *не обязательно*, но *приветствуется*, поскольку значительно облегчает работу.
- Ключевое слово `return` означает, что всякий раз, когда мы вызываем эту функцию, значение, которое сейчас же будет выведено (оно называется «возвращаемое значение функции»), будет передано (то есть «возвращено») интерпретатору.

Вы, вероятно, обратили внимание на ответ консоли — `undefined`. Это, как вы помните, просто означает, что кроме этого консоли пока нечего больше вам ответить. Ведь всё, что вы сделали, — это *объявили* функцию, так что и вернуть вам пока нечего (точно так же, как с объявлением переменных).

Вызов функции

Теперь нам нужно будет **вызвать** нашу функцию. *Вызов* функции — это такой красивый термин для её *запуска*. Вызов функции выглядит следующим образом:

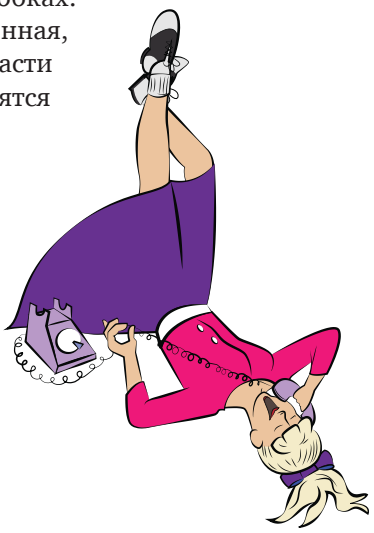
```
вывестиЦифруПять ();
```

Ответ: `5`

То есть мы, *объявив* функцию, а затем *вызвав* её, получаем в результате *возвращаемое значение* этой функции.

Параметры и аргументы функций

Объявим ещё одну функцию с переменной в скобках. Когда при функции в скобках есть ещё и переменная, мы будем называть её **параметром**. У большей части функций будут параметры, но иногда нам встретятся функции и без них.



```
function прибавитьТри(начальноеЧисло) {
    return начальноеЧисло + 3;
}
```

После объявления функции и её параметров мы можем вызвать (запустить) её точно так же, как и предыдущую, с той лишь разницей, что теперь полученное в результате *значение* мы ещё и присваиваем параметру нашей функции; такое присваивание значения мы будем называть **аргументом**:

```
прибавитьТри(4);
```

Ответ: `7`

В нашем примере `4` является *аргументом* функции; как только наша функция получила свой аргумент, она тотчас же присвоила его значение переменной `начальноеЧисло` (которое является *параметром* нашей функции). Далее в теле функции над параметром была произведена нехитрая математическая операция, и нам было *возвращено* в ответе новое значение.

Попробуйте шутки ради дать в качестве аргумента *строку*. Как думаете, что произойдёт?

```
прибавитьТри('Моё любимое число это ');
прибавитьТри('4');
```

Сможете объяснить разницу между параметрами и аргументами? Если ещё нет, то вкратце она состоит в следующем: *аргумент* — это значение, которое мы «выдали» функции, когда её вызвали, и которое присвоили *параметру* этой функции. Или так: *параметр* — это название переменной, указанное в объявлении функции и ожидающее присваивания значения в момент вызова функции; присваиваемое параметру значение и есть *аргумент*. Всё понятно? Отлично. А ещё вот что: множество программистов используют эти термины (*аргумент* и *параметр*) взаимозаменяемо. Вы не поверите, но мне даже… пришлось сперва справиться об определениях этих терминов, когда я это писал, поскольку я и сам запамятовал, что из них что.

Давайте попробуем объявить функцию, параметры которой будут ожидать двух строк в качестве своего значения (обратите внимание, что если у вас более одного параметра, то их необходимо отделять друг от друга запятой).

```
function искреннийКомплимент(имя, черта) {
    let комплимент = 'Ух ты, ' + имя + '!  Твоя '
        + черта + ' выглядит просто восхитительно!';
    return комплимент;
}
```

Когда мы вызовем эту функцию, мы должны будем задать ей в качестве аргументов две строки (не забывайте пользоваться одинарными или двойными кавычками!), отделённых друг от друга запятой:

```
искреннийКомплимент('Джереми', 'причёска');
```

Ответ: `"Ух ты, Джереми!  Твоя причёска выглядит просто восхитительно!"`

О, большое спасибо, компьютер! Пожалуй, стоит почаще спать на левом боку!

DRY (Don't Repeat Yourself/ Принцип сухости)

А вы знали, что крутые разработчики любят, когда всё излагается сухо и по делу? В смысле, когда ничего по древу не растекается, всё происходит без лишних повторов, лаконично и так далее. В общем, все хорошие программисты придерживаются принципа сухости, или **DRY**[1] (Don't Repeat Yourself), — то есть стараются избегать ненужных повторений.

Очень часто нам попадаются стандартные модели, которым нужно просто следовать, не повторяясь по многу раз. Ведь если каждый раз придётся переписывать уже известные данные, то самое простое задание тут же превратится в монструозный и нечитабельный массив кода.

Вот вам дурацкий пример:

Ваша младшая сестрёнка: А как узнать, что машина — красная?

[1] «Dry» в переводе с английского — сухой.

Вы: Ну, если её двери — красные, капот — красный, багажник — красный и весь кузов тоже красный, то, пожалуй, можно с уверенностью заключить, что эта машина — красная.

Сестрёнка: Понятно! А как узнать, что машина — синяя?

Вы: Ну, если её двери — синие, капот — синий, багажник — синий и весь кузов тоже синий, то, пожалуй, можно с уверенностью заключить, что эта машина — синяя.

Сестрёнка: Понятно! А как узнать, что машина — оранжевая?

Вы: Ну, если её двери — оранжевые, капот — оранжевый, багажник — оранжевый и весь кузов тоже оранжевый, то, пожалуй, можно с уверенностью заключить, что эта машина — оранжевая.

Сестрёнка: Понятно! А как узнать, что машина — лиловая?

Вы: Ну, если её двери — лиловые, капот — лиловый, багажник — лило… хмм… Кажется, я начинаю повторяться… О скольких цветах машин ты ещё собиралась спросить?

Сестрёнка: Пока не знаю. А сколько цветов в коробке с мелками?

Какой многословный и повторяющийся вышел диалог, не находите? Тут бы вам с вашей очаровательной сестрёнкой очень пригодилась небольшая пользовательская функция (а ведь всем известно, что младшие сестрёнки просто ОБОЖАЮТ кстати написанные пользовательские функции!). И не забывайте о SHIFT+ENTER для переноса!

```
function аКакУзнатьЧтоМашина(цвет) {
    return "Ну, если её двери —  " + цвет
        + ", капот — " + цвет
        + ", багажник — " + цвет
        + ", и весь кузов тоже " + цвет
        + ", то, пожалуй, можно с уверенностью заключить, что эта машина —
          " + цвет + ".";
}
```

Теперь всякий раз, как ваша сестрёнка обнаружит новый цвет в коробке с пастелью и ей захочется узнать о цвете той или иной машины, ей всего лишь нужно будет запустить функцию и задать ей цвет! Давайте посмотрим, как это работает. Чтобы использовать одну функцию несколько раз, просто возвращайтесь к ней нажатием стрелки вверх (и не забывайте менять аргумент, то есть цвет!):

```
аКакУзнатьЧтоМашина('серого цвета');
```

Ответ: `Ну, если её двери — серого цвета, капот — серого цвета, багажник — серого цвета и весь кузов тоже серого цвета, то, пожалуй, можно с уверенностью заключить, что эта машина — серого цвета.`

```
аКакУзнатьЧтоМашина('лазоревого цвета');
аКакУзнатьЧтоМашина('цвета аквамарин');
аКакУзнатьЧтоМашина('цвета жжёной сиены'[1]);
аКакУзнатьЧтоМашина('зеленовато-голубого цвета');
аКакУзнатьЧтоМашина('макаронно-сырного цвета');
аКакУзнатьЧтоМашина('цвета толчёной киновари'[2]);
```

Как вам? Стало значительно короче и проще, правда? Эта функция позволит вашей сестрёнке всегда найти ответы на свои вопросы, займёт её на много часов вперёд, а вас освободит от необходимости повторяться. Ах, какой же вы чудесный и замечательный старший брат, не правда ли?

Но вы, вероятно, подметили, что и с этой функцией все повторения не ушли: приходится как минимум писать то же самое и менять цвет. Почему бы нам сразу не «запихнуть» в нашу функцию целую коробку пастельных мелков, вместо того чтобы всякий раз вводить по-одному? Если эта мысль вас посетила, то вы, без сомнения, рассуждаете как первоклассный программист-разработчик! Мы без труда с этим справимся… в главе 9, в которой мы будем обсуждать *массивы* и *циклы*. Вот тогда-то мы по-настоящему развернёмся, а наши функции будут прям суше некуда (помните? — «DRY»)!

Встроенные функции

alert()

Мы рассмотрели примеры простых пользовательских функций; быть может, вы уже догадались, что в JavaScript существует также и целый ряд встроенных? Под **встроенной** понимается такая функция, которая уже существует на уровне самого языка программирования, так что её можно использовать в любой момент.

Итак, откроем пустую вкладку, запустим чистую консоль и попробуем немного поиграть со встроенными функциями!

```
alert('Бу!');
```

[1] Коричневато-красная краска (также называется «итальянская земля»).
[2] Ртутный минерал, с древности использовавшийся для получения алой краски.

Испугались? Обратите внимание, что встроенные функции действуют по тем же законам, что и те, которые мы с вами мастерили собственноручно (названия в верблюжьемРегистре, а за ними скобки с аргументом). Главное отличие в том, что эти функции вам не нужно объявлять! Они как бы уже знакомы JavaScript «с рождения». Давайте ещё парочку для закрепления:

```
alert('А это необходимо?');
alert("Если б это был настоящий сайт, такая штука, наверное, очень бы
доставала?");
alert('Ну ладно, повеселились и будет.');
alert('Эй! Ну хватит уже!!');
```

Скажу честно: подавляющее большинство пользователей (и разработчиков) терпеть не могут функцию `alert()`. Это выплывающее сообщение их донельзя раздражает. Короче, можно очень позабавиться с этой нашей новообретённой суперспособностью; но помните, что она — весьма коварная и легко может перетянуть вас на тёмную сторону!

console.log()

Так что давайте лучше употребим свои силы во имя добра! Например, вот секретный способ вывести (**логгировать**) в консоль отладочную информацию только для разработчиков (пользователи её увидеть не смогут; как вы знаете, большинство людей не подозревает даже о существовании самой консоли, а не то что о секретных «логах»!):

```
console.log('Совершенно секретно! Только для разработчиков!');
```

Как явствует из названия функции, мы выводим в консоль «лог» (то есть информацию о работе системы); этой доброй суперспособностью разработчики пользуются постоянно. Скажем, когда у вас были сообщения об ошибках, вы видели в консоли именно это — интерпретатор выдавал («логгировал») в консоль информацию о работе системы, чтобы вы могли прочесть и исправить нужные параметры. Словом, очень полезная штука. Вам не раз и не два придётся прибегнуть к помощи `console.log`, так что запомните эту функцию!

```
console.log('Выведите любое сообщение, какое пожелаете');
console.log('просто введите сюда какую-нибудь ' + "строку");
let сообщение = 'А ещё в качестве аргумента можно использовать
                 переменные!';
console.log(сообщение);
let чтоНужноСпасти = 'положение'; console.log('Я спасу ' +
                     чтоНужноСпасти);
```

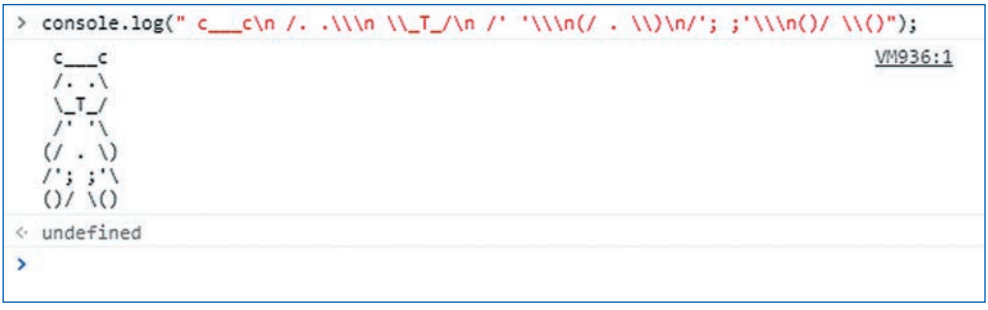
Пока вы играете с логами в консоли, я покажу вам ещё один трюк. Его мне поможет выполнить наш старый добрый друг — обратный слэш! При его содействии (`\n`) можно разбивать свой код по строчкам. Попробуйте сами:

```
console.log('Можно разбить любой текст\n на многие\n строчки.');
console.log("Вот первая.\nА вот 2-я.\n3-я\n4-я\n5-я!");
```

Эта штука с `\n` вообще-то имеет множество полезных практических применений; например, можно нарисовать в кодировке ASCII мишку в консоли. Набирайте вместе со мной в консоли (всё в одну строку и внимательно проверяйте каждый `/`, `\`, пробел и всё такое):

```
console.log(" c___c\n /. .\\\n \\_T_/\n /' '\\\n(/ . \\)\n/';-;'\\\n()/ \\
()");
```

Понятно, почему так выходит? Каждый `\n` задаёт переход на новую строчку, так что последующие символы как бы «спускаются» ниже, и *вуаля*! Мы с вами ~~внесли серьёзный вклад в развитие общества~~ нарисовали премилого мишку!



Math.random()

А вот ещё одна очень полезная встроенная функция: `Math.random()`. Обратите внимание, что, в отличие от предыдущих, название этой функции начинается с прописной буквы `'M'`. Тому есть причины, но о них мы пока не будем. В каждом языке, будь то русский, английский или JavaScript, всегда есть какие-то исключения из правил. Вот это — одно из них. Все прочие переменные и функции, которые вы встретите в этой книге, следует писать со строчной буквы; помните, что регистр — это важная штука и, если название Math.random() начать с math, то функция просто не запустится.

```
Math.random();
Math.random();
Math.random();
Math.random();
```

Вот так штука: каждый раз новое число! Не просто новое: функция выдаёт случайное число между `0` и `1`. Если вам вдруг понадобится случайное число или ряд в этом диапазоне, то эта функция вам очень пригодится.

Math.floor()

Эта функция требует число или цифру в качестве аргумента и, получив, округляет их в *меньшую* сторону до ближайшего *целого* (то есть без каких-либо десятых долей после запятой).

```
Math.floor(10.4);
Math.floor(5);
Math.floor(214.19723);
let почти33 = 32.99; Math.floor(почти33);
```

А теперь всё вместе!

Попробуем замешать небольшой коктейль из только что выученных функций. Если хотите, можете вполне обойтись и без комментариев (как обычно, начинаются с `//`), но не забывайте о SHIFT+ENTER для переноса! Для следующего этапа очень важно, чтобы всё, что мы пока написали, оставалось одним цельным блоком кода:

```
let случайноеЧислоМежду0и1 = Math.random();
// Умножим на 100 случайное число от 0 до 1, чтобы получить число
// между 0 и 100
let большоеСлучайноеЧисло = случайноеЧислоМежду0и1 * 100;
// Применим Math.floor() для округления В МЕНЬШУЮ СТОРОНУ
// до ближайшего целого числа ("Цел")
let большоеСлучайноеЦел = Math.floor(большоеСлучайноеЧисло);
let выбериЧисло = 'Выбери число от 0 до 100: ';
console.log(выбериЧисло + большоеСлучайноеЦел);
```

Надеюсь, ничего из сделанного не оказалось для вас трудной задачкой, и вы всё поняли. Если нет, то не торопясь просмотрите каждую строчку и комментарии к ней. Если же и после этого у вас остались вопросы, то советую вам вернуться к началу разговора о встроенных функциях и внимательно всё перечитать. Если и после этого какие-то моменты остались неясны, тогда просто продолжайте с нами до конца главы и, быть может, всё само прояснится по ходу дела.

А теперь я попрошу вас сделать кое-что, что спровоцирует интерпретатор выдать нам ошибку. Нажмите стрелку вверх, чтобы в строке вернулся наш код, а затем нажмите ENTER. Если всё верно, то вы увидите следующее сообщение:

```
Uncaught SyntaxError: Identifier 'случайноеЧислоМежду0и1' has already been
declared at <anonymous>:1:1
```

Понимаете, откуда взялась эта ошибка? Если собираетесь ответить: «Нет», то позвольте спросить: «А вы вообще *читали* само сообщение?» Помните: *сообщения об ошибках — ваши друзья*! Они призваны помочь вам. Так что не ленитесь читать их!

Ну а если вы всё же удосужились прочесть сообщение об ошибке, то, вероятно, обнаружили, что загвоздка была в самом *первом* члене самой *первой* строчки кода (именно к этому относится `:1:1` в сообщении). Как вы помните, ключевое слово `let` должно использоваться лишь для объявления *новых* переменных, а мы же, напротив, пытались применить его для уже объявленной.

Один способ решения данной проблемы мы уже знаем — просто убрать `let`; теперь я покажу вам ещё один. Пожалуйста, отнеситесь к нему со всем вниманием. Вновь нажмите стрелку вверх, чтобы вернуть наш код, и припишите в самом конце, после завершающей `;` закрывающую фигурную скобку `}`. Затем стрелками переместите курсор в самое начало, комбинацией SHIFT+ENTER сделайте перенос и наберите следующий код:

```
function logСлучайноеЧисло() {
```

В общем, у вас должен получиться такой код:

```
function logСлучайноеЧисло() {
let случайноеЧислоМежду0и1 = Math.random();
let большоеСлучайноеЧисло = случайноеЧислоМежду0и1 * 100;
let большоеСлучайноеЦел = Math.floor(большоеСлучайноеЧисло);
let выбериЧисло = 'Выбери число от 0 до 100: ';
console.log(выбериЧисло + большоеСлучайноеЦел);
}
```

Теперь можете победоносно нажать ENTER и получить в ответ `undefined`. Мы только что проделали вот что: взяли наш код и «завернули» его в упаковку нашей пользовательской *функции*, которую мы *объявили* и назвали. Теперь можно преспокойно вызвать её, просто набрав:

```
logСлучайноеЧисло();
```

Попробуйте сами пару-тройку раз и убедитесь (если всё было сделано верно), что каждый раз результат будет иным:

```
logСлучайноеЧисло();
logСлучайноеЧисло();
logСлучайноеЧисло();
```

Получилось? Очень надеюсь, что да! Информации уйма, так что не расстраивайтесь, если у вас не вышло с первого раза. Внимательно изучите весь код, убеди-

тесь, что в нём нет никаких ошибок и опечаток. Внимательность — невероятно важный навык для программиста. Помните: «практически уверен» — это не уверен!

Повторяй за мной: инструкция по применению машины времени

И снова «Повторяй за мной»! Не забывайте перед началом каждого нового задания закрывать и открывать всё, что нужно, чтобы перед вами было лишь чистое, готовое к работе окно консоли (кстати, если вы ещё не в курсе — окно консоли можно растягивать, чтобы было удобнее с ней работать).

Итак, знаменитый профессор Уммнек фон Генийберг почти закончил работу над своим главным изобретением — машиной времени для своих кота, пса, хорька и волнистого попугайчика! Он уже откалибровал хронометрические тумблеры, установил и настроил несущую панель, модифицировал турбофазотронную систему и сумел преодолеть фундаментальные изъяны *спотолкавзятого амулита* с помощью пластично-логарифмичной обшивки. Словом, большая часть работы уже сделана!

Но профессору всё ещё необходим алгоритм выбора года для работы машины (какой-нибудь год между 0 и 2500-м нашей эры), чтобы отправить в путешествие во времени всех своих питомцев. Доверить столь ответственный выбор самим питомцам профессор не может, поскольку до сих пор ещё не закончил работу над переводчиком с животного. В то же время он не хочет, чтобы потом придумали, что у него были любимчики и он кого-то отправил в более удачную эпоху. Словом, профессор хочет, чтобы процесс выбора происходил случайным образом. Он не удосужился прочесть мою книжку, и поэтому как пользоваться функциями для выбора случайных чисел, он не знает; в общем — ему нужна ваша помощь! Ах да, и наш код должен быть повторяемым, чтобы все звери могли путешествовать из эпохи в эпоху, когда только захотят.

Итак, нам предстоит:

- Случайным образом определить год между 0 и 2500-м нашей эры.
- Операция выбора должна быть повторяемой.

Перво-наперво давайте создадим функцию и наполним её жёстко закодированными значениями (то есть такими, которые не изменяются сами по себе):

```
function путешествиеВоВремени() {
    let питомец = 'кот';
    let год = 1950;
    return 'Ваш ' + питомец + ' отправляется во времени в год ' +
    год + '!';
}
```

Наша функция объявлена, теперь мы можем вызывать её столько раз, сколько нам заблагорассудится… но проблема в том, что она всегда возвращает одно и то же! Смотрите сами:

```
путешествиеВоВремени();
путешествиеВоВремени();
путешествиеВоВремени();
путешествиеВоВремени();
```

Нам нужно сделать функцию динамически изменяющейся; для этого, во-первых, переменную `питомец` мы сделаем параметром нашей функции (то есть поместим её в скобки):

```
function путешествиеВоВремени(питомец) {
    let год = 1950;
    return 'Ваш ' + питомец + ' отправляется во времени в год ' +
    год + '!';
}
```

И теперь, вызывая нашу функцию, мы можем предлагать питомцев в качестве её аргументов!

```
путешествиеВоВремени('кот');
путешествиеВоВремени('пёс');
путешествиеВоВремени('волнистый попугайчик');
путешествиеВоВремени('хорёк');
```

Итак, мы научились посылать разных питомцев во времени, однако всех их мы пока что шлём лишь в `1950` год.

Конечно, «стабильность — признак мастерства», скажете вы, и я с вами соглашусь, но всё же толика разнообразия не помешает. Мы поправим это дело точно так же, как и в случае с питомцами профессора, — мы сделаем переменную `год` параметром! Но ведь это означает, что всякий раз придётся вручную вводить год отправления того или иного питомца (например: `путешествиеВоВремени('кот', 2250);`), а это подаст повод недругам профессора обвинить его в пристрастности, так что этот вариант нам не подойдёт.

Нам необходимо добавить в нашу функцию механизм определения случайных чисел; на помощь нам придёт встроенная функция `Math.random()`, которая, как вы, несомненно, помните, выдаёт нам случайные числа в диапазоне от `0` до `1`.

Полученное случайное число мы затем умножим на наш максимальный год планирующихся временных путешествий — то есть на `2500` (год нашей эры):

```
function путешествиеВоВремени(питомец) {
    let максГод = 2500; // дальше этого года мы посылать наших зверей
                        // не будем
    let год = Math.random() * максГод;
    return 'Ваш ' + питомец + ' отправляется во времени в год ' +
    год + '!';
}
```

Попробуем:

```
путешествиеВоВремени('пёс');
путешествиеВоВремени('хорёк');
```

О, мы так близко! Я уже чувствую запах успеха! Но что же делать с этими знаками после запятой, что же это за `год` такой десятичный получается?! А ведь мы уже округляли десятичные дроби, помните? Мы это делали при помощи `Math.floor()`! Эту функцию можно даже объединить с уже существующей строкой!

```
function путешествиеВоВремени(питомец) {
    let максГод = 2500; // дальше этого года мы посылать наших зверей
                        // не будем
    let год = Math.random() * максГод;

    return 'Ваш ' + питомец + ' отправляется во времени в год '
        + Math.floor(год) + '!';
}
```

А теперь поотправляйте разных зверей:

```
путешествиеВоВремени('кот');
путешествиеВоВремени('волнистый попугайчик');
console.log(путешествиеВоВремени('пёс'));
console.log(путешествиеВоВремени('хорёк'));
alert(путешествиеВоВремени("старый футбольный мяч"));
alert(путешествиеВоВремени('злой соседский кот'));
```

Поздравляю! Вы в прямом смысле слова «попали» в историю!

Кстати говоря, вы заметили, что в четырёх последних примерах мы вызывали нашу функцию внутри другой функции?! Это называется **вложенной** функцией, когда одна функция находится (буквально — вкладывается) и действует внутри другой; этот приём очень популярен среди программистов. При работе с вложенными функциями интерпретатор всегда будет **вычислять значение**, начиная с внутренней. Вычислением значения функции называют весь процесс её обработки: запуск, вызов и возврат значения. Таким образом, вычислить значение, начиная с *внутренней* функции, означает, что интерпретатор (компьютер) сперва «увидит» и запустит именно вложенную внутрь функцию.

Например, в `alert(путешествиеВоВремени('злой соседский кот'));` интерпретатор сперва запускает (то есть начинает вычислять значение) внутреннюю функцию `путешествиеВоВремени`, затем берёт значение, которое возвращает эта функция (в нашем случае возвращаемым значением была целая строка, повествующая о грядущем путешествии наших зверей) и передаёт его (значение, то есть строку) в качестве аргумента к внешней функции `alert()`.

Надеюсь, вы всё поняли. Если же нет, пожалуйста, вернитесь на пару абзацев выше и попробуйте медленно, с чувством, с толком и всем прочим перечитать их. Это очень важная штука, так что постарайтесь. А теперь — викторина!

ВИКТОРИНА

Не забывайте заносить свои ответы в Рабочую тетрадь. Не подглядывайте в ответы раньше времени! Сверьтесь с ними после того, как всё закончите.

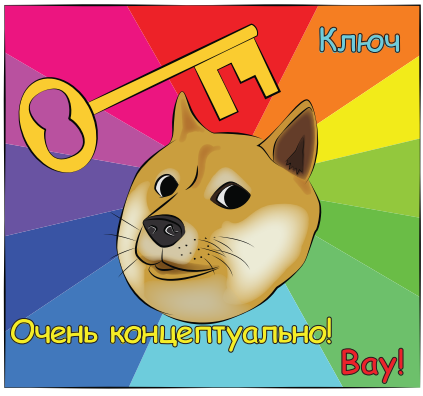
1. Как называется отдельный блок кода, который можно вызвать для выполнения какого-либо действия?
2. В какие символы заключается функция?
3. Создав функцию, я могу использовать её в будущем. Как называется запуск функции по команде?
4. В каком стиле должно быть прописано название функции?
5. Как называется часть функции, которая *что-то выдаёт нам назад*, после вызова функции?
6. Когда мы объявляем функцию, в её скобках может быть заключена переменная (или переменные), которые называются _____. Когда мы вызываем функцию, то тому, что заключено в скобки, мы можем присвоить значения, называемые _____.
7. Назовите английскую аббревиатуру, обозначающую принцип, согласно которому разработчик не должен повторять один и тот же блок кода по многу раз.
8. Какая из четырёх рассмотренных в главе встроенных функций наиболее раздражающе действует на пользователей (особенно если встречается им не один раз)?
9. Когда предполагается задать аргументам функции более одного значения, каким символом они должны отделяться друг от друга?
10. Какая встроенная функция позволяет очень легко и просто оставлять суперсекретные сообщения для разработчиков (или для себя самого, если вы работаете над улучшением собственного кода)?

11. Даже если к функции ничего нельзя прибавить или внести в неё изменения, какие символы необходимы для её запуска?
12. Встроенная функция нахождения случайных чисел возвращает значение больше _____ и меньше _____.
13. Какая встроенная функция округляет число в меньшую сторону до ближайшего целого?
14. Истина/Ложь: функция может быть «упакована» в другую функцию.
15. Истина/Ложь: когда происходит вычисление значения вложенной функции, интерпретатор сперва вычисляет значение внутренней, а затем внешней функции.
16. Истина/Ложь: во время работы с вложенной функцией интерпретатор будет использовать параметр внутренней функции и передаст его в качестве аргумента внешней.
17. При помощи какой клавиши можно выделить код отступом, чтобы он легче читался (например, внутри функции)?
18. Как задать переход на новую строчку в консоли?

КЛЮЧЕВЫЕ ЭПИЗОДЫ

Как обычно, пробегитесь по списку и обязательно вернитесь и повторите те темы, в понимании которых вы не совсем уверены:

- Что такое функция?
- Объявление функции.
- Синтаксис и наименования всех составляющих функции.
- Возвращаемое значение функции.
- Вызов функции.
- Параметры функции.
- Аргументы функции.
- D.R.Y. (Принцип сухости).
- Встроенные функции.
- `alert()`.
- `console.log()`.
- `Math.random()`.
- `Math.floor()`.
- Вложенные функции.



УПРАЖНЕНИЯ

I. Введите следующие правильные фрагменты кода в консоль.

Обратите внимание: можете перефразировать или вовсе проигнорировать всё, что находится в `//комментариях`; нет никакой надобности в точном перепечатывании комментариев, если вы понимаете, о чём в них сказано.

1.
```
function получитьПрогнозПогоды() {
 return 'Дикий зной со снегопадом';
 }
```

2.
```
// Прогноз погоды на сегодня
получитьПрогнозПогоды();
```

3.
```
/*
Теперь выведем лог прогноза
в консоль
*/
function логПрогноза() {
    let прогноз = 'облачно, возможны осадки в виде фрикаделек';
    let сообщениеПрогноза = "Погода на сегодня: " + прогноз;
    console.log(сообщениеПрогноза);
    return прогноз;
}
```

4.
```
alert('Прогноз погоды: \n' + логПрогноза());
```

5.
```
function какТебяЗовут(имя) {
   let сообщение = 'Тебя зовут \n' + имя;
   console.log(сообщение);
   alert(сообщение);
   return сообщение;
}
```

6.
```
какТебяЗовут('<напиши_своё_имя>');
```

7.
```
function твояЛюбимаяИгрушка(любимаяИгрушка, возраст) {
    return 'Тебе уже ' + возраст + ', а тебе всё ещё нравится ' +
            любимаяИгрушка + '?';
}
```

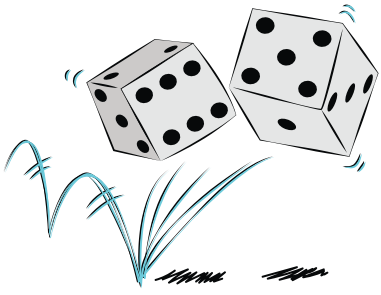
8.
```
твояЛюбимаяИгрушка('<назови_любимую_игрушку>', '<сколько_тебе_лет>');
```

9.
```
function бросаемИгральныйКубик() { // обычный игральный кубик с шестью
                                   // сторонами
    /* нам придётся прибавить 1, поскольку в противном случае
       Math.floor() будет выдавать результат  в диапазоне от 0 до 5 */
    return Math.floor(Math.random() * 6) + 1;
}
```

```
10. бросаемИгральныйКубик(); // попробуйте несколько раз, чтобы убедиться,
                             // что всё работает

11. function параИгральныхКубиков() { // теперь бросим два игральных
                                      кубика
        let кубик1 = бросаемИгральныйКубик(); // используем функцию,
                                              // объявленную выше
        let кубик2 = бросаемИгральныйКубик();
        let сумма = кубик1 + кубик2;
        console.log('У вас выпало ' + сумма + '! (' + кубик1 + ' и ' +
                    кубик2 + ')');
        return сумма;
    }

12. alert(параИгральныхКубиков()); // попробуйте несколько раз (обратите
                                   // внимание на сообщения в консоли)
```

II. Что не так с каждым из фрагментов?

```
1. function неБылоОбъявления();

2. function даКомуОниВообщеНужны()
       return "И так сойдёт.";

3. function вКабинетеСтоматолога() [
       console.log('Быть может, вам понадобятся брекеты...');
   ]

4. function аПончикиЭтоВкусно() {
       return 'Ага!';
   }
   аПончикиЭтоВкусно(console.log());

5. console.log('Целое число от 0 до 1: ' + Math.random(Math.floor()));

6. alert('Все любят всплывающие сообщения!');

7. console.log('Случайное число: ' + Math.random();)

8. function какойтоЦвет {
       return 'фиолетовый';
   }
   console.log(какойтоЦвет());

9. function любимыйРесторанчик() {
       return 'Мой любимый ресторан ' + ресторан;
   }
   любимыйРесторанчик('McDonald\'s');
```



КОМПЛЕКСНЫЙ ОБЗОР

1. Какой тип данных заключается в одинарные или же двойные кавычки?
2. Какие два способа объявления переменных мы рассматривали?
3. Истина/Ложь: интерпретатор JavaScript игнорирует комментарии.
4. Правильно ли написано следующее предложение (и если нет — почему?):

```
var почемуБыИспугалась7 = 'потому что 7 съела 9';
```

5. Какой знак нужно добавить в следующее предложение, чтобы избежать синтаксической ошибки?

```
let славнаяПесенка = 'They Can't Take That Away From Me.';
```

6. Истина/Ложь: благодаря использованию функций можно эффективно следовать принципу W.E.T.[1] (знаменитому принципу мокрости!).
7. Выполняет ли следующий код то, что хочет от него разработчик?

```
let днейВгоду = 365; /* вернёмся к этому позже
```

8. Какой это тип данных?

```
0
```

9. Какой символ используется для однострочных комментариев?
10. Строка в JavaScript состоит из отдельных _____.
11. Какой символ соответствует оператору modulo?
12. Истина/Ложь: строка не должна содержать двойных кавычек, кроме тех случаев, когда строка заключена в одинарные кавычки.
13. Какой результат рассчитывает получить разработчик, используя оператор modulo?
14. Истина/Ложь: сообщения об ошибках предназначены только для людей.
15. Какого цвета трава?
16. Где разработчик может легко и просто проверить работу написанного кода и получить немедленный ответ интерпретатора?
17. Правильно ли написано следующее предложение (можете попробовать набрать его в консоли), и если нет — почему?

---

[1] Игра слов: *dry* — сухой, *wet* — мокрый. Шуточный принцип W.E.T. расшифровывается как «Write Everything Twice» (Пиши всё по два раза) или «We Enjoy Ryping» (Нам нравится печатать).

```
function найтиОстаток(делимое, делитель) {
    let сообщениеОбОстатке = 'При евклидовом делении ' + делимое
        + ' на ' + делитель
        + ' мы имеем в остатке ' + (делимое % делитель);
    return сообщениеОбОстатке;
}
найтиОстаток(10, 3);
```

18. Каким оператором стоит воспользоваться, чтобы узнать, делится ли некое число нацело на 6?

19. При помощи какого символа можно было бы упростить следующее присвоение?

```
огромноеЧисло = огромноеЧисло + 19;
```

20. Какой это тип данных?

```
'75'
```

21. Правильно ли написано следующее предложение (и если нет — почему?):

```
function необходимыйВозраст(5) {
    // дети младше 5 на карусель не допускаются;
    return 5;
}
необходимыйВозраст(5);
```

22. Для чего в консоли применяется комбинация SHIFT+ENTER?

23. При помощи какой клавиши можно выделить код отступом, чтобы он легче читался (например, внутри функции)?

24. Верен ли следующий код (и если нет — почему?):

```
function найтиСлучайноеЧисло(мин, максЧисло) {
    return Math.floor(Math.random() * максЧисло);
}
найтиСлучайноеЧисло(20); // нужно число, которое было бы меньше 20
```

25. Если вы вызываете функцию, которая находится внутри другой функции, то такую функцию называют _____.

Сделай сам: городская лотерея

Город избрал вас в качестве ответственного лица за проведение лотереи в этом году! Все участники ооочень заинтересованы в том, чтобы выбираемые числа были полностью случайные. Участники лотереи выбирают 3 числа, каждое из которых может быть одно- или двузначным в диапазоне от 0 до 99 (то есть любое целое число меньше 100). Все билеты выглядят следующим образом: `"25-4-92"`, или `"46-81-7"`, или `"18-60-98"` и так далее.

Город возлагает на вас большие надежды! Вы должны не подвести жителей и создать функцию, которая бы возвращала три случайным образом отобранных числа (отделённых друг от друга дефисами и `log`-сообщениями с результатом в консоли). Удачи! Хотя к чему она вам? Она скорее пригодится участникам городской лотереи!

И помните: если вы застряли, загляните в рекомендации к «Сделай сам» в этой главе! Но ознакомившись с ними, сразу же закрывайте ответы и продолжайте работать самостоятельно.

5 Сравню ли с...[1]

В этой главе будет много нового, но уж очень сложно быть не должно: ведь, в общем-то, всё вертится вокруг простой идеи — истина или ложь?

До сих пор мы всегда имели дело лишь с примитивными типами данных — с *числами* и *строками*. Быть может, вам уже встречался термин «**примитивный**» применительно к программированию? Если нет, я поясню: «примитивный» (ещё говорят «простой») означает, что этот тип данных не является объектом (к этому моменту мы ещё вернёмся) и не имеет встроенных методов работы (то есть функций).

Всего же существует пять примитивных (или простых) типов, и в этой книжке мы обсудим их все (ну да, технически типов всего шесть, но шестой несколько трудноват и вообще в обозримом будущем вам не понадобится точно. Так что представим, что всего типов — пять). Хорошие новости: последние три типа — самые простые!

Булевый (логический) тип/boolean

Итак, третий из типов данных, которые вам обязательно понадобятся, называется **булевым** (boolean), или логическим. Булевый (рифмуется с «всё разруливай») тип всегда имеет значение либо `true` (истина), либо `false` (ложь). И только так, и никак иначе! Он либо врёт, либо говорит правду — пан или пропал, свет включен или выключен, либо есть, либо нет. Вы либо сделали домашнее задание, либо нет.

[1] Начало 18 сонета Шекспира (*англ.*: Shall I compare…).

Сейчас мы подробно разберёмся со *всеми* булевыми значениями и поработаем с ними *всеми* в консоли. Как обычно, закройте и откройте браузер и запустите консоль. Наберём (без кавычек):

```
true;
false;
```

Ну как, ясно? Это и есть все возможные булевы значения! Если нужно время — пожалуйста, пробегитесь по списку ещё раз, чтобы убедиться, что вы всё правильно поняли и запомнили.

Операторы сравнения

Булевы значения приходятся как нельзя кстати, когда нам необходимо что-то сравнить в JavaScript. Когда такая необходимость появляется, мы сразу же вызываем команду наших высококвалифицированных **операторов сравнения**.

Сейчас мы последовательно изучим все восемь операторов сравнения, но штука в том, что в результате работы каждого из них мы всё равно всегда останемся с булевым значением — либо `true`, либо `false`. Наберите парочку из приведённых ниже примеров в консоль. Ну, не ленитесь! Так, тогда наберём в консоли их *все* (как, в общем-то, и должно происходить с *любым* текстом в этой книге, выделенным синим). Внимательно следите за реакцией консоли на вводимый код.

«Тройное равно», или «равноравноравно» (===)

```
'мой дом' === "мой дом"; // true
1 === true; // false
false === false; // true
false === 0; // false
"Я запутался" === 'Я запутался'; // true
3 === 3; // true
3 === '3'; // false
```

Обращаю ваше внимание особенно на последний пример; `===` означает «строго равно». Оператор чётко следит за тем, чтобы *и* значение, *и* тип были строго идентичными. В случае с `3 === '3'` значение, конечно, идентично, однако тип — нет: ведь первое — это число, а второе — строка.

«Двойное равно», или «равноравно» (==)

```
3 == 3; // true
3 == '3'; // true
0 == 'ноль'; // false
1 == true; // true
5 == true; // false
false == 0; // true
```

Удивило ли вас что-нибудь здесь? Штука в том, что `==` означает «в достаточной степени равно». Так что выходит, что `false` и правда *в достаточной степени равно* `0`, а `3`, опять же, *в достаточной степени равно* `"3"`, несмотря на разность типов данных.

«Неравноравно», или «БАЦравноравно» (!==)

```
5 !== 8; // true
true !== 1; // true
0 !== false; // true
"Я запутался" !== 'Я запутался'; // false
3 !== 3; // false
3 !== '3'; // true
```

Восклицательный знак (`!` — программисты часто говорят вместо длинного «восклицательный знак» короткое и смешное звукоподражание «бац!») в подобных случаях всегда означает «не» и является одним из способов добавить к чему-то его отрицание. Таким образом, `!==` означает обратное `===`; как `===` следит за чёткой идентичностью значения и типа, так и `!==` следит за их различием. Это можно читать как «*строго* НЕ равно ...».

«Неравно», или «БАЦравно» (!=)

```
5 != 8; // true
true != 1; // false
0 != false; // false
3 != 3; // false
3 != '3'; // false
```

Надеюсь, тут уже всё достаточно интуитивно? `!=` всегда будет противоположен `==`. Можно читать это как «*в достаточной степени* НЕ равно».

Чтобы лучше понять разницу между `!==` и `!=`, представьте, что учитель на доске пишет следующее:

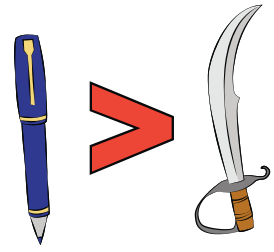
- Истина или ложь: число 1 *строго* НЕ равно true (то есть «истине») — `1 !== true`.

- Если вы внимательно следили за ходом его объяснений и всё поняли, то, конечно, скажете, что это истина (`true`). С другой стороны, сразу после этого учитель чуть меняет написанное:
- Истина или ложь: число 1 *в достаточной степени* НЕ равно true (то есть «истине») — `1 != true`.
- Теперь вы будете вынуждены указать, что это ложь (то есть `false`), так как `1` и `true` в достаточной степени равны между собой (как минимум в программировании это так), несмотря на разность типов данных.

И ещё одно: несмотря на то, что я только что объяснил (и, надеюсь, понятно) вам, что такое `==` и `!=`, я бы не хотел, чтобы вы вообще когда-либо их использовали. Почему так? А потому, что такой необходимости, в общем-то, *никогда* и нет. В любом возможном случае, когда вы можете их использовать, всегда можно использовать и строгие `===` и `!==`. Если же вы желаете большей гибкости в ответе (скажем, чтобы с одинаковым успехом принималось как `1`, так и `'1'` или `true`), то можно просто включить нужные варианты ответа в сам код (не меняя при этом `===`). Это значительно облегчит работу с кодом как вам, так и другим разработчикам, которым сразу будут ясны ваши намерения, вместо того чтобы ломать голову по поводу того, что «тут имелось в виду что-то такое, а может, и в достаточной степени ещё и такое» и так далее. Не волнуйтесь, если пока не совсем понимаете, зачем я вообще об этом заговорил (не сомневайтесь — в будущем обязательно поймёте); просто уясните правило: *вообще никогда* не используйте `==` или `!=`.

«Больше чем» (>)

```
8 > 5; // true
3 > 5; // false
'8' > 5; // true
2 > 2; // false
```



Ну, тут, я надеюсь, всё понятно интуитивно. Оператор «больше чем» (`>`) проверяет, больше ли значение, указанное слева, значения, указанного справа. Если да, то в ответе будет `true`, если же нет — `false`.

«Меньше чем» (<)

```
8 < 5; // false
3 < 5; // true
2 < 2; // false
```

То же самое, только наоборот: оператор проверяет, *меньше ли* значение слева значения справа; да — `true`, нет — `false`.

«Больше или равно» (>=)

```
8 >= 5; // true
3 >= 5; // false
2 >= 2; // true
```

Тут, опять же, всё просто: этот оператор делает то же самое, что и «больше чем», но он *также* выдаст логическое `true`, если два значения окажутся равными.

«Меньше или равно» (<=)

```
8 <= 5; // false
3 <= 5; // true
2 <= 2; // true
```

И последний, но не менее важный оператор — «меньше или равно»; он делает то же, что и «меньше чем», плюс, если сравниваемые величины будут равны, — выдаст булево `true`.

Вкратце подытожим: мы познакомились со всеми 8 операторами сравнения, проверяющими наши величины на равенство, неравенство и сравнивающими их. Шесть из них вы будете частенько использовать в самых разных ситуациях (`===`, `!==`, `<`, `<=`, `>`, и `>=`). Два других же (`==` и `!=`) вы, вполне вероятно, увидите в чьём-нибудь коде, но никогда не будете использовать в собственном.

Запомнили, что делает каждый оператор? Если нет, тогда обязательно вернитесь на пару страниц назад, повторите и попробуйте в консоли приведённые примеры (а ещё лучше — придумайте свои). Помните, что в том, чтобы «пролистать» эту книжку, большого толку нет. Толк будет только в том случае, если вы будете всё делать внимательно и неспешно. Работайте с книгой с той скоростью, с которой вам комфортно. Скорость — вовсе не главное: главное, чтобы вы понимали и запоминали пройденный материал.

Условные конструкции

Вы, должно быть, думаете: «Ну, все эти булевы-логические штуки — это было очень просто… Наверное, они довольно бесполезны, и используют их нечасто». Как бы не так! Логические значения используются в программировании более чем постоянно и всего чаще — в форме условных конструкций (или выражений).

А что такое «условная конструкция»?

Хороший вопрос! **Условная конструкция** — это такое предложение, которое используется для запуска определённых блоков кода согласно заданному **условию**. Условие (например, при сравнении `x === y`) всегда возвращает логическое значение — либо `true`, либо `false`. Соответственно, если значение `true`, то код следует

запустить, в противном же случае блок кода следует пропустить. Рассмотрим несколько примеров.

Условные выражения с if

Большинство условий в JavaScript выполнены при помощи оператора `if`. Наберём в консоли:

```
if (true) {
    console.log('булево значение в скобках — true! Код запустится!');
}

if (false) {
    console.log('булево значение — false; не пойдёт.');
}
```

В зависимости от булева значения в скобках оператор `if` либо запускает, либо не запускает код. Вроде пока всё просто и понятно, правда? То же самое можно проделать и с помощью самодельной переменной:

```
let булевоЗначение = true;
if (булевоЗначение) {
    console.log("Надеюсь, вы не удивитесь, если я скажу, что этот код —
                запустится.");
}
```

«Правка» и «ложка»[1]

Условные выражения могут работать *не только* с булевыми значениями. В следующих примерах мы будем иметь дело с небулевыми переменными, то есть с такими, которые *не точно* `true` или `false` (то есть не точно «истинны» или «ложны»). Они, знаете, какие-то такие — «**правка**» и «**ложка**», вроде «не совсем», но что-то близкое. Так что мы, в общем-то, спокойно можем использовать их в скобках, как и булевы значения. Давайте попробуем:

```
let значениеПравки = 1; // всё равно "правка", хоть это и число,
                        // (а не булево значение)
if (значениеПравки) {
    console.log('Это "Правка"!');
}
if (0) { // ноль — число (а не булево значение), но всё ж значение
         // его можно считать "ложкой"
    console.log('Не запустится!');
}
```

[1] Жаргонные слова *truthy* и *falsy* для обозначения ситуации, когда значение переменной заранее преобразуется в логическое «истинно» или «ложно» соответственно. — *Примеч. ред.*

```
if ('строка с хотя бы одним символом — уже "правка"') {
    console.log('Ура, работает!');
}
if ('') { // а эта строка — пустая, так что "ложка"
    console.log('неа.'); // код не будет запущен
}
if (-3) { // все целые числа (кроме нуля) — "правки"
    console.log('Ага!');
}
```

Операторы сравнения в выражениях с if

До сих пор мы с вами имели дело со сравнениями или с условными выражениями с `if`. Но пока мы не использовали их вместе; а ведь они просто созданы друг для друга, как машина с колёсами или принтер с чернилами!

Давайте откроем новую консоль в `about:blank` и попробуем следующее:

```
// сравнения выдают в результате логические значение. итак, true
// или false?
let сравнение = 3 < 9;
if (сравнение) {
    console.log('запустится ли этот код?');
}
if ('строка' === "строка") { // поместим оператор сравнения прямо в скобки
    console.log('они что — СТРОГО равны??');
}
if (4 !== '4') { // если они не строго идентичны, то будет true
    console.log('Что правда "true"?');
}
// выражение с if может быть и в одну строчку, если вам так больше
// нравится
if (8 <= 7) {console.log('8 меньше или равно 7?');}
if ('abc' > '') {console.log('а короткая строка это больше, чем пустая
                              строка? Ну да!');}
let количествоДетейВкоманде1 = 5;
let количествоДетейВкоманде2 = 6;
if (количествоДетейВкоманде1 < количествоДетейВкоманде2) {
    console.log('В Команде 2 больше игроков!');
}
```

if...else

Выражения с `if` — отличная и полезная штука, когда вы хотите, чтобы какой-нибудь код запустился, если (`if`) логическое значение — `true`. Хорошие программисты, однако, знают, что всегда неплохо бы иметь и запасной план. Именно тут и вступает в игру `if...else`.

```
let иванТопорышкинПошёлНаОхоту = false;
if (иванТопорышкинПошёлНаОхоту) {
    console.log("С ним пудель вприпрыжку пошёл, как топор. Иван повалился
бревном на болото"); // это мы сейчас пропустим
} else {
    console.log("А пудель в реке перепрыгнул забор."); // а вот это — мы
                                                        // запустим
}
```

Если логическое значение — `true` («истина» или хотя бы «правка»), то интерпретатор запустит соответствующий код, указанный в блоке `if`. Если же `false` («ложь», ну или «ложка»), тогда будет запущен код из блока `else`.

Теперь поменяем булевы значения и попробуем снова. Нажмём стрелку вверх и внесём поправки:

```
иванТопорышкинПошёлНаОхоту = true; // не используйте let!
if (иванТопорышкинПошёлНаОхоту) {
    console.log("С ним пудель вприпрыжку пошёл, как топор. Иван повалился
бревном на болото"); // теперь это мы запустим
} else {
    console.log("А пудель в реке перепрыгнул забор."); // а вот это — мы
                                                        // пропустим
}
```

А теперь упакуем всё это в функцию!

Давайте-ка теперь завернём парочку наших `if`...`else` выражений в красивую функцию, а?! Вы ведь ещё не забыли, как это делается и что вообще такое функция? Что ж, приступим (в комментарии `//план:` указан наш дальнейший план, так что этот комментарий нам также нужен):

```
function можноМнеДетскийНаборПожалуйста() {
    //ПЛАН: наполнить "внутренности" функции
}
```

Вот мы и задали функцию! Проблема в том, что в ней пока ничего нет. Нажмите стрелку вверх и вместо комментария `//план:` вставьте тело функции:

```
function можноМнеДетскийНаборПожалуйста() {
    let детскиеНаборыТолькоДляДетейНеСтарше = 10;
    let возрастРебёнка = 11; // можете внести сюда свой настоящий возраст,
                             // если хотите
    if (возрастРебёнка <= детскиеНаборыТолькоДляДетейНеСтарше) {
        console.log("Пожалуйста, вот ваш детский набор!");
    } else {
        console.log('Прошу прощения. Вам нужно сделать заказ из меню для
                взрослых.');
    }
}
```

А теперь просто *вызовем* функцию, написав её имя со скобками:

```
можноМнеДетскийНаборПожалуйста();
```

Ответ: `"Прошу прощения. Вам нужно сделать заказ из меню для взрослых."`

Вот же ж досада! Всего лишь год! Ну как так-то?!

Так, стоп! А что, если к окошку подойдёт кто-нибудь помладше? Каким *тогда* будет ответ?

```
let маленькийРебёнок = 5;
можноМнеДетскийНаборПожалуйста();
```

Ответ: `"Прошу прощения. Вам нужно сделать заказ из меню для взрослых."`

Вы что, шутите?! Как можно отказывать пятилетнему мальчику, который всего-то и хотел, что детский набор заказать?! Просто абсурд какой-то!

Штука в том, что в нашей функции жёстко закодирована (то есть «встроена», задана с самого начала) переменная `возрастРебёнка` со значением `11`. Чтобы это исправить, надо несколько раз нажать стрелку вверх, чтобы вернуть нашу функцию и переменить в ней *пару моментов* (а именно — добавить в скобках параметр функции и убрать строчку, жёстко задающую `возрастРебёнка`):

```
function можноМнеДетскийНаборПожалуйста(возрастРебёнка) {
    let детскиеНаборыТолькоДляДетейНеСтарше = 10;
    if (возрастРебёнка <= детскиеНаборыТолькоДляДетейНеСтарше) {
        console.log("Пожалуйста, вот ваш детский набор!");
    } else {
        console.log('Прошу прощения. Вам нужно сделать заказ из меню
                    для взрослых.');
    }
}
```

Как видно, вместо того чтобы жёстко «встраивать» `возрастРебёнка` в саму функцию, мы сделали эту переменную параметром, который потом, при вызове функции, можно будет изменить, задав значение аргументу! Попробуем:

```
// маленькийРебёнок теперь будет нашим аргументом, так его мы будем
// предлагать нашей функции
можноМнеДетскийНаборПожалуйста(маленькийРебёнок);
```

Ответ: `"Пожалуйста, вот ваш детский набор!"` // Ура-а-а-а!

```
можноМнеДетскийНаборПожалуйста(48); // Но надо и меру знать!
```

Ответ: `"Прошу прощения. Вам нужно сделать заказ из меню для взрослых."`

```
можноМнеДетскийНаборПожалуйста(10); // А для 10-летнего сработает?
можноМнеДетскийНаборПожалуйста(11); // Да ну ладно вам! У неё день
                                     // рожденья-то всего на прошлой
                                     // неделе был!
```

В общем, я надеюсь, что из всего этого вы вынесли достаточно полное понимание того, как работает условная конструкция `if`...`else`. В дальнейшем мы ещё не раз будем работать с разными условными конструкциями, так что я надеюсь, вы были внимательны. Ну а теперь можно из всего этого смастерить что-нибудь весёленькое! Закроем-откроем браузер и всё-всё остальное, откроем `about:blank` и запустим чистое окно консоли.

Повторяй за мной: рассчитываем минимальный рост для прохода на аттракционы

В город приехала ярмарка! Ураaa!

О нет… стойте… Да ведь туда пускают только жирафов… Ну как так-то! :-(

Вот невезуха! Впрочем, дело поправимо. На жирафье колесо обозрения и электромобили пускают всех без разбору, но чтобы попасть на жирафью карусель, нужно быть хотя бы 259 сантиметров ростом! Словом, вам нужно написать программку, которая бы оценивала рост жирафов (в метрах и сантиметрах, вроде 2,8 или 1,78) и отвечала бы соответствующим сообщением — достаточно ли высок жираф, чтобы прокатиться на карусели, или нет.

Звучит недурно! За дело! Сперва давайте объявим функцию и дадим ей известные нам параметры (и не забудем про `//план:`):

```
function ростЖирафа(ростВМетрах, сантиметры) {
    //план: здесь скоро будет код!
}
```

Теперь наша задача перевести рост нашего жирафа в сантиметры. Нам известно, что в каждом метре всегда 100 сантиметров, так что давайте сразу с этим разберёмся (всё это мы, конечно, набираем *там*, где был наш `//план:`):

```
    let сантиметровВМетре = 100;
    let ростЖирафа = ростВМетрах * сантиметровВМетре;
```

Сюда же нужно добавить оставшиеся сантиметры:

```
    ростЖирафа += сантиметры; // вы ведь помните что означает +=?
```

А затем нужно написать условную конструкцию со сравнением, чтобы выяснить, достаточно ли высок жираф:

```
    let минимальныйРостЖирафа = 259;
    if (ростЖирафа >= минимальныйРостЖирафа) {
        console.log("Ты уже дорос для карусели!");
    }
```

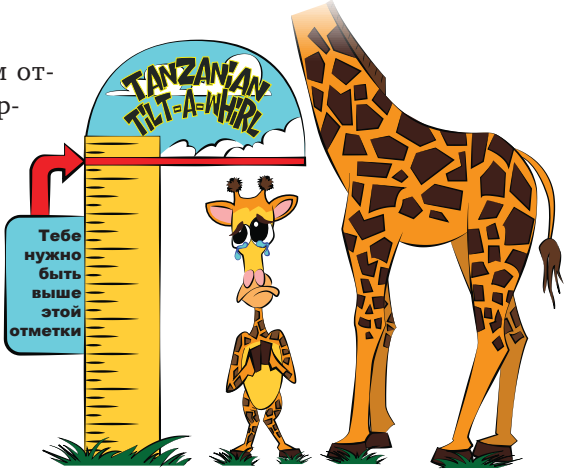
Ещё нам нужен блок с `else` на случай, если оператор сравнения выявит `false` и понадобится запустить другой блок кода:

```
    } else {
        console.log("Ты ещё маловат. Прокатись лучше на колесе
                    обозрения.");
    }
```

Соберём нашу функцию воедино:

```
function ростЖирафа(ростВМетрах, сантиметры) {
    let сантиметровВМетре = 100;
    let ростЖирафа = ростВМетрах * сантиметровВМетре;
    ростЖирафа += сантиметры;
    let минимальныйРостЖирафа = 259;
    if (ростЖирафа >= минимальныйРостЖирафа) {
        console.log("Ты достаточно высок для карусели!");
    } else {
        console.log("Ты ещё маловат. Прокатись лучше на колесе
                    обозрения.");
    }
}
```

А теперь можно одного за другим отправлять наших жирафов на проверку, вызывая для каждого нашу отличную функцию!



Итак, вот наша очередь: рост первого жирафа — 2,23 (то есть 2 метра 23 сантиметра), второго — 4,11 (4 м 11 см), третьего — 3,32, четвёртого — 5,54, пятого — 1,86, шестого — 2,62 и, наконец, седьмого — 5,79. И их всех мы можем легко, просто и быстро взять да проверить!

```
ростЖирафа(2,23); //  достаточно ли будет 2 метров и 23 сантиметров?
ростЖирафа(4,11); //  а как насчёт 4 м и 11 см?
ростЖирафа(3,32); // 3,32?
ростЖирафа(5,54);
```

```
ростЖирафа(1,86);
ростЖирафа(2,62);
ростЖирафа(5,79);
```

Ну как? Если вы всё сделали верно, то вы теперь в курсе, что лишь двое из наших жирафов оказались слишком малорослыми для карусели (хотя один из счастливчиков прошёл благодаря всего лишь трём сантиметрам!). Всё так?

Разве не здорово, что мы вот так вот запросто написали такую шикарную функцию, которую сколько угодно раз можно вызывать, и она каждый раз будет разрешать нужную проблему? Конечно — здорово! Вообще, это один из ключевых моментов в программировании: написать код один раз, а затем использовать его по мере необходимости хоть тысячу, хоть миллион, хоть тысячу миллионов раз.

ВИКТОРИНА

Как обычно, заносите все решения в Рабочую тетрадь, стараясь не подглядывать в ответы. Когда закончите, обязательно проверьте себя по ответам в конце.

1. Как называется примитивный тип данных, имеющий строго два значения — либо `true`, либо `false`?
2. Каким символом обозначается оператор сравнения, проверяющий, меньше ли значение слева значения справа?
3. Истина/Ложь: `"Тут есть какой-то подвох." !== 'Тут есть какой-то подвох.';`
4. Какой оператор сравнения наиболее предпочтителен, если необходимо проверить, что два значения не равны?
5. Какие два оператора сравнения я вам рекомендовал никогда не использовать?
6. Какой оператор сравнения проверяет, что значение справа меньше или равно значению слева?
7. Какой тип предложений используется для запуска блоков кода на основании данных условий?
8. Есть некоторое значение, которое не строго булево (то есть оно не строго `true` или `false`); однако если использовать его в качестве условия и заключить его в скобки (в качестве условия) при операторе `if`, то он всё же будет запускать блок кода. Таким образом, можно сказать, что это значение является _____. Если же на основании такого условия блок кода запускаться не будет, то можно сказать, что это значение является _____.
9. Какая часть условного выражения запускает код, если значение при блоке оператора `if` — `false`?

10. Истина/Ложь: операторы сравнения всегда выдают результаты в виде `true` или же `false`.
11. Истина/Ложь: операторы сравнения часто используются внутри скобок при блоке оператора `if`.
12. Истина/Ложь: операторы сравнения порой могут быть использованы и в скобках в блоке `else`.
13. Истина/Ложь: конструкция `if`…`else` может быть применена лишь за пределами функции.

КЛЮЧЕВЫЕ ЭПИЗОДЫ

Пробегитесь по списку и, если вдруг обнаружите что-то неизвестное или что-то, что не совсем понятно, обязательно вернитесь и повторите этот момент.

- Булевы (логические) значения.
- 8 операторов сравнения.
- «Равноравноравно» и «НЕравноравно».
- «Больше чем» и «меньше чем».
- «Больше или равно» и «меньше или равно».
- Никогда не пользуйтесь «двойным равно» и «НЕравно».
- Условные выражения (конструкции).
- Выражения с `if`.
- «Правка» и «ложка».
- Конструкция `if`…`else`.

УПРАЖНЕНИЯ

I. Введите следующие правильные фрагменты кода в консоль.

Набрав пример в консоли, прежде чем нажать ENTER, спросите себя: `true` или `false` сейчас будет в ответе? Затем сверьтесь с мнением интерпретатора. Если ваши мнения разойдутся, попробуйте определить, почему так вышло.

```
1. 'Эта строка' === "Эта Строка";
2. if (5 !== '5') {
      alert("хмм, кто бы мог подумать.");
}
3. 7 >= 7;
4. 9 <= 12;
5. if ('abc' > 'ab') {
       console.log('правка');
   } else {
       console.log('ложка');
   }
6. 0 !== false;
7. if (false != 0) {
      console.log('ха, плёвое дело!');
   } else {
      console.log('хм, чот странное');
   }
8. true == 5;
9. if (1 == true) {
      alert('Теперь понимаете, почему не стоит эти операторы использовать?
             Странные штуки!');
   } else {
      alert('А я думал, будет true!');
   }
10. -7 > 0;
11. 4 < -20;
12. true === 'true';
13. if (1 === '1') {
       console.log('трушно!');
   }
14. // пожалуйста, будьте внимательны с этим упражнением!важно, чтобы
    // вы всё поняли
   function какоеЧислоБольше(первоеЧисло, второеЧисло) {
       if (первоеЧисло > второеЧисло) {
           console.log(первоеЧисло + ' больше ' + второеЧисло);
       } else {
           if (первоеЧисло < второеЧисло) {
               console.log(второеЧисло + ' больше ' + первоеЧисло);
           } else {
               console.log('Да они же равны!');
           }
       }
   }
15. какоеЧислоБольше(212, 301);
16. какоеЧислоБольше(155, -800);
```

```
17. какоеЧислоБольше(12, 12);
18. какоеЧислоБольше('абвг', 'деёж');
19. какоеЧислоБольше('24', 24);
```

II. Что не так с каждым из фрагментов?

```
1. function чтоМеньше(первоеЧисло, второеЧисло) {
      if (первоеЧисло < второеЧисло) {
          console.log(первоеЧисло + ' меньше!');
      } else {
          console.log(второеЧисло + ' меньше!');
      }
  }
2. if {
      true;
  } else {
      false;
  }
3. if (true) {
      console.log('правка');
  } else (false) {
      console.log('ложка');
  }
4. 3 === 3 < 4;
5. 5 = 5;
6. let числовоеЗначение === 5;
7. let ещёОдноЧисло != 3;
```

КОМПЛЕКСНЫЙ ОБЗОР

1. Истина/Ложь: блоки комментариев заставляют интерпретатор JavaScript игнорировать всё, что указано с начала блока и до конца строчки.
2. Правильно ли написано следующее предложение (и если нет — почему?):

   ```
   let аПиццаВкусная = 'Шеф' !== 'Оскар Ворчун с Улицы Сезам';
   ```

3. Какие символы необходимо добавить в следующее предложение, чтобы избежать синтаксической ошибки?

   ```
   let грустнаяПесня = "It's So Hard to Say "Goodbye" to Yesterday";
   ```

4. Что означает D.R.Y. (принцип сухости) применительно к программированию?

5. Какой тип выражений используется в программировании для запуска блока кода, согласно известным условиям?
6. Выполняет ли следующий код замысел разработчика (и если нет — почему?):

   ```
   let днейВНеделе = 7; // для расчётов
   ```

7. Какой это тип данных?

   ```
   'true'
   ```

8. Каким символом (или символами) отмечается блок комментариев?
9. Если для работы условного выражения нам необходимы небулевы данные, которые мы всё же считаем `true` или `false`, то такие данные мы называем соответственно _____ и _____.
10. Что в JavaScript означает оператор `%`?
11. В каком стиле следует писать названия функций в JavaScript?
12. Истина/Ложь: строку, в которой есть апострофы, следует заключить в двойные кавычки.
13. Какую ошибку вы получите, запустив следующий код?

    ```
    let firstInitial = 'J';
    let firstInitial = "M";
    ```

14. Какого результата ожидает разработчик, использовавший оператор modulo?
15. Правильно ли написано следующее предложение (и если нет — почему?):

    ```
    (15 + (7 * 9) - 4) / 8 - 57);
    ```

16. Истина/Ложь: сообщения об ошибках направлены на сокрытие причины ошибки, дабы предотвратить хакерские атаки.
17. Что означает одиночный знак равенства в предложении?
18. Каким сочетанием клавиш открывается консоль в браузере Chrome?
19. Правильно ли написан следующий код (если хотите — попробуйте в консоли)? Если нет — почему?

    ```
    function годенЛиДляАрмии(возраст) {
        let призывнойВозраст = 18;
        if (возраст > призывнойВозраст) {
            console.log('Годен!');
        } else {
            console.log('Молодой ещё!');
        }
    }
    let мойВозраст = 18;
    годенЛиДляАрмии(мойВозраст);
    ```

20. При помощи какого символа можно было бы упростить следующее присваивание?

```
приветствие = приветствие + ', ' + имя;
```

21. Какой это тип данных?

```
false
```

22. Правильно ли написано следующее предложение (и если нет — почему?):

```
function вернутьНовоеЧисло(начальноеЧисло) {
    return начальноеЧисло + 4;
    начальноеЧисло += 3;
    console.log('Новое число: ' + начальноеЧисло);
}
вернутьНовоеЧисло(15);
```

23. Какой адрес нужно ввести в строке Chrome, чтобы открыть пустую вкладку?
24. Chrome, Internet Explorer, Edge, Firefox и Safari — это всё _____.
25. Какие два оператора сравнения не стоит использовать?
26. Какой оператор сравнения проверяет, что значение справа больше или равно значению слева?
27. При помощи какой клавиши можно выделить код отступом, чтобы он легче читался (например, внутри функции)?
28. Правильно ли написан следующий код (и если нет — почему?):

```
let булевоУсловие = true !== false;
if (булевоУсловие) {
    console.log('Это правка!');
} else {
    console.log("Это ложка!");
}
```

29. Если вы вызываете функцию, которая находится внутри другой функции, то такую функцию называют _____.
30. Истина/Ложь: операторы сравнения всегда выдают булевы значения.
31. При помощи какой встроенной функции можно сгенерировать случайное число в диапазоне от `0` до `1`?

Сделай сам: детская церковь

Пастор Проповедий объявил, что готовится открытие новой детской церкви. Похоже, все прихожане заинтересованы и хотят туда наведаться. Представляете? И взрослые, и дети — все! Ясное дело, что места на всех не хватит. Словом, пастор

Проповедий очень хочет, чтобы туда могли попасть лишь ребята в возрасте от 6 до 13 лет.

Напишите функцию, которая бы выручила пастора Проповедия из столь затруднительного положения. Функция должна принимать в качестве аргумента возраст желающего войти внутрь, а затем выводить лог-сообщение о том, может ли он это сделать (то есть подходит ли его возраст под нижний и верхний лимит).

Попробуйте свою функцию с разными возрастами, чтобы убедиться, что всё сделано верно. Как закончите (или до того, если где-то застряли), сверьтесь с рекомендациями в конце книжки.

6 Логические операции

Если вы успешно справились с предыдущей главой, то эта вам покажется сущей легкотнёй! В той главе мы разобрали третий из пяти типов данных; в этой мы познакомимся с оставшимися двумя. И знаете чего? Эти два — самые лёгкие из всех!

Null

Четвёртым примитивным типом данных является **null**. Null — это просто-напросто *ничто*. «Ничто» в смысле `0`? Нет, нуль — число. Может, в смысле `""`? Тоже нет, это просто пустая строка. Может, `false`? Опять нет, `false` — это булево (логическое) значение. Так что же это? Это — `null`. Ничто. И это — наш четвёртый тип данных. Приведём пример его использования:

```
let мояПеременная = null;
мояПеременная;
```

Ответ: `null`

Это очень полезная штука в тех случаях, когда нужно присвоить некоторое значение чему-то, что получит значение только в дальнейшем. Присваивая переменной значение null, вы, таким образом, заявляете интерпретатору, что эта переменная нам ещё понадобится и в будущем будет изменена.

Скажем, вы желаете создать некую форму, которую пользователь должен будет заполнить. В вашей форме есть три графы: `имя`, `возраст` и `цветМашины`. Во время создания формы вам нужно будет дать некоторые начальные значения трем переменным; это можно сделать, например, так:

```
let имя = null;
let возраст = null;
let цветМашины = null;
```

После того как пользователь заполнит форму, у вас появятся реальные значения для этих переменных:

```
имя = Каори;
возраст = 13;
```

Стойте-ка! Каори ведь ещё слишком юна, чтобы водить, какой `цветМашины`?! Никаких проблем! Значение переменной останется `null`, так как Каори оставила графу незаполненной. Когда же мы приступим к обработке нашей формы (об этом мы поговорим позже), мы просто проигнорируем поле с переменной `цветМашины`, поскольку она имеет всё то же значение — `null`.

Ещё обратите внимание, что `null` — *«ложка»*! Вы ещё помните, что это такое? Если нет, не ленитесь и обязательно загляните в глоссарий! А я пока поясню (набирайте в консоли):

```
let тестовоеУсловие = null;
if (тестовоеУсловие) {
    console.log('Похоже, что ' + тестовоеУсловие + ' всё же "правка"! '
       + 'Джереми просто не в курсе, о чём болтает!');
} else {
    console.log(тестовоеУсловие + ' — несомненная "ложка". '
       + 'И как вообще я мог сомневаться в таком великом таланте,
          как Джереми?!');
}
```

Короче, вы видите, насколько всё просто: у типа данных `null` есть всего одно возможное значение — `null`. Когда мы будем говорить об объектах, мы вернёмся к `null`, и он нам ещё не раз пригодится; в некотором смысле `null` даже считается простейшим из возможных объектов.

А знаете? Выбросьте это из головы, только путаница лишняя выйдет. Не будем бежать впереди паровоза!

Итак, мы знакомы с четырьмя из пяти простых типов данных; что ж, пора нам, пожалуй, закончить этот список.

Undefined

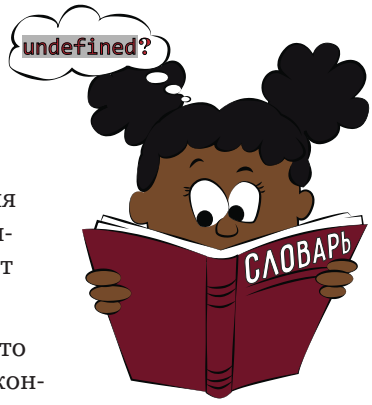
Пятый (и последний) примитивный тип данных — это… `undefined`!

Ух ты! Вот это поворот! Оказывается, последний тип данных — это та штука, которую мы знали чуть ли не с начала книжки!

В качестве типа данных `undefined` означает, что пока никакого значения, даже `null`, переменным присвоено не было[1]. `undefined` является значением переменной по умолчанию. Наберите в консоли и убедитесь сами:

[1] Undefined в переводе с английского означает «неопределённый».

```
let мояНоваяКлёваяПеременная;
мояНоваяКлёваяПеременная;
```



Вы, наверное, впервые *не* поставили `=` в строке, где был использован `let`, да? Всё потому, что если мы хотим значением нашей переменной получить неопределённость этого самого значения (то есть `undefined`), то мы не должны ей присваивать (`=`) ничего. В результате переменная получит значение по умолчанию — `undefined`.

И наконец, вот ещё что: попробуйте угадать, а что будет, если использовать `undefined` в условной конструкции? Как думаете, оно будет «правкой» или «ложкой»? Давайте проверим:

```
let новаяНеопределённаяПеременная;
if (новаяНеопределённаяПеременная) {
    console.log(новаяНеопределённаяПеременная + ' — это "правка"! Так
я и думал!');
} else {
    console.log('Так и знал! ' + новаяНеопределённаяПеременная + ' —
                это "ложка"!');
}
```

Ну что, вы оказались правы? Надеюсь, что да!

Итак, теперь мы знаем все пять примитивных типов данных в JavaScript! И шансов использовать все пять у нас ещё будет предостаточно! А пока давайте подберём ещё парочку полезных инструментов для вашей JavaScript-мастерской.

Логические операторы

В предыдущей главе мы обсуждали операторы сравнения вроде `<`, `>=`, `===` или `!==`. В этой же мы познакомимся с их близкими родственниками — **логическими операторами**.

Всего существует три логических оператора: `&&`, `||` и `!`. Как и операторы сравнения, логические операторы часто используются с булевыми значениями. Впрочем, можно их применять и с другими типами данных (вроде строк или чисел). В то же время, в отличие от операторов сравнения, возвращаемый результат не всегда будет строго логическим (то есть `true` или `false`), но может обладать значением **любого** типа, эти значения будут или «правкой», или «ложкой», которые, в свою очередь, будут приниматься как `true` и `false` соответственно. Благодаря такой «всеядности», логические операторы прекрасно сотрудничают с условными выражениями вроде `if`…`else`.

М-да… я несколько раз переписывал предыдущий абзац, а понятнее мои объяснения не стали ни на йоту! Пожалуй, лучше будет просто разобрать несколько примеров, чтобы всё расставить по местам.

Логическое И (&&)

Первый логический оператор, который мы будем разбирать, это логическое И; он обозначается двойным амперсандом (получается при нажатии SHIFT и 7) — `&&`. Этот оператор применяется, когда требуется выяснить, *оба* ли значения являются правдивыми (или хотя бы «правками»).

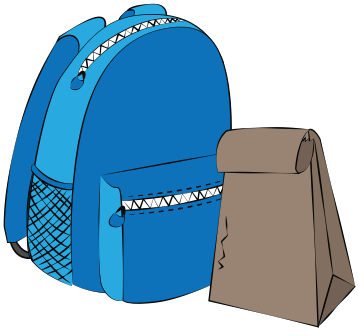
Прежде чем разобрать пример использования `&&` (логического И, хотя многие разработчики говорят просто «и-и») в коде, давайте рассмотрим ситуацию из реальной жизни. Например, ваша мама очень бы желала узнать, готовы ли вы наконец ехать в школу; она говорит: «Чтобы идти в школу, нужно собрать рюкзак и не забыть контейнер с обедом. Всё готово?»

«Ну, ма-ам… А что, если у меня ни рюкзака, ни обеда нет?»

«Ясно… значит, ничего ещё не готово (очевидно!)».

«Ну ладно-ладно! Я собрал рюкзак. Но обед мне искать лень. Может, я так пойду?»

«Ни в коем случае!»



«Вот! Я нашёл в холодильнике контейнер с обедом… правда, забыл рюкзак на столе…»

(Фэйспалм)

«Всё-всё! Я готов: вот рюкзак. И в нём — обед! Едем?»

«Да! Теперь ты [наконец-то] готов идти в школу!»

Как видите, успешно завершился лишь последний сценарий (то есть лишь он был «правкой»); все остальные были «ложками».

Закройте все открытые программы, затем откройте Chrome, пустую вкладку и запустите консоль. Набирайте в консоли примеры и пробуйте угадать ответы интерпретатора:

```
false && false;
true && true;
true && false;
false && true;
```

Вторая строка оказалась `true`, потому что лишь в ней с *обеих* сторон от оператора `&&` находятся «правки»; во всех остальных случаях присутствует хотя бы одна «ложка». Понимаете?

Или можно сказать наоборот, что `&&` вернёт *«ложку»*, если хотя бы *с одной стороны* найдётся «ложка». Звучит, пожалуй, довольно странно, но если вы внимательно посмотрите на примеры выше, то увидите, что это так.

Теперь попробуем использовать логическое И в функции. Я как раз, по случаю, имел разговор с вашим классным руководителем, и тот ясно дал мне понять, что вашу школу *необходимо* посещать лишь обутым ученикам! (Эй-эй, не смотрите на меня! Это же не мои правила!). Короче, в нашей функции должен быть ещё один параметр (в добавление к «собранному рюкзаку» и «обеду»):

```
function готовЛиЯКШколе (собралРюкзак, взялОбед, наделБотинки) {
    if (собралРюкзак && взялОбед && наделБотинки) {
        console.log('Теперь можно идти в школу!');
    } else {
        console.log("Ты ещё не готов! Поторопись или опоздаешь!");
    }
}
/* порядок значений (то есть аргументов) соответствует
* порядку параметров функции, заявленных строчкой выше.
* Таким образом, первое булево значение
* прикрепляется к рюкзаку, второе к обеду и так далее.
*
* Кстати говоря, когда пишете комментарии, их вовсе
* не обязательно перепечатывать отсюда слово в слово!
* Можете просто выписать основную идею, чтобы не забыть.
*/
готовЛиЯКШколе (false, false, true);
готовЛиЯКШколе (true, false, true);
// если аргумент не упомянут, то это означает то же, что и значение
// undefined
готовЛиЯКШколе (true, true);
готовЛиЯКШколе (true, true, true);
```

Логическое ИЛИ (||)

Второй наш логический оператор — это логическое ИЛИ (*англ.* — or; многие разработчики просто говорят «или-или», обозначающееся двойной вертикальной чертой (`||`). Этот оператор проверяет, является ли хотя бы *одно* из значений «правкой». Простой пример:

```
false || false;
true || true;
true || false;
false || true;
```

Как видно, лишь первая строчка вернула `false`, поскольку *ни одно* из значений не является «правкой». Остальные же три имеют в своём составе *как минимум одну* «правку», что и выражается в возвращаемом ответе. Надеюсь, тут всё понятно?

Ну или, опять же, можно сказать и наоборот: оператор `||` вернёт *«ложку»* лишь в том случае, когда *обе* стороны являются «ложками». И вновь это может показаться странным, но, присмотревшись к нашему примеру, вы увидите, что всё так.

Ещё один случай из реальной жизни, когда логическое ИЛИ (`||`) может иметь место: перед тем, как отправиться в школу, мама говорит, что нужно позавтракать. Вы можете съесть на завтрак яичницу, а можете позавтракать хлопьями с молоком (или всем вместе). Короче, вам придётся съесть хотя бы что-нибудь из этого списка (ведь первый приём пищи — самый важный!). Так вот: вы уже позавтракали? Попробуем разобраться:

```
function ужеПозавтракал (съелЯичницу, съелХлопья) {
    if (съелЯичницу || съелХлопья) {
        console.log("Ты уже позавтракал!");
    } else {
        console.log("Ты ещё не завтракал! Давай быстрее!");
    }
}
```

Не забывайте, что можно «вернуть» уже использованный в консоли код нажатием стрелки вверх. Сможете угадать ответы на каждый из вариантов?

```
ужеПозавтракал(false,true);
ужеПозавтракал(true,true);
ужеПозавтракал(true,false);
ужеПозавтракал(false,false);
```

Надеюсь, теперь разница между `&&` и `||` вполне очевидна?

Что ж, раз так, давайте попробуем использовать их вместе. Вы помните всё, что нужно сделать, если уж вы собираетесь покинуть отчий дом сегодня поутру, верно? Нужно позавтракать (яичницей или хлопьями), собрать рюкзак `И` обед `И` не забыть надеть ботинки.

Сможете ли вы сами написать единую функцию, включающую все эти пункты? Если да, то обязательно приступайте и не читайте дальше! Возвращайтесь, когда закончите! Если же нет, тогда оставайтесь с нами, и будем писать её вместе (но попутно следите за тем, какие части вы бы смогли написать самостоятельно, а какие — нет). Приступим:

```
function можноУжеВыходить (съелЯичницу, съелХлопья, собралРюкзак,
                            взялОбед, наделБотинки) {
    if ((съелЯичницу || съелХлопья) && (собралРюкзак && взялОбед &&
         наделБотинки)) {
        console.log("Отлично! Можно выходить!");
    } else {
        console.log("Пока нет! Ты ещё не готов!");
    }
}
```

А теперь попробуйте угадать ответы:

```
можноУжеВыходить(false,true,true,true,false);
можноУжеВыходить(true,true,false,true,true);
можноУжеВыходить(true,false,true,true,true);
// как вы помните, если аргумент опущен, то это то же, что и undifened
// (то есть "ложка")
можноУжеВыходить(true,true,true,true);
можноУжеВыходить(false,true,true,true,true)
```

Всё ясно? Круто!

Всегда «правка»/«ложка»… не всегда истина/ложь (true/false)

Глядя на всё это, вы, быть может, задавались вопросом: «Чего это он вечно со своими *«правками»* (truthy) и *«ложками»* (falsy)? Почему бы просто и сразу не писать *истина* (true) и *ложь* (false)?!» Впрочем, может, вы вовсе и не задавались подобным вопросом, но я всё же на него отвечу.

В случае с нашими `&&` и `||` возвращаемое значение не всегда будет строго `true` или `false`. Ответы часто будут располагаться где-то недалеко от той или иной стороны строго булева значения! Попробуйте в консоли следующее (если не хотите, можете комментарии не перепечатывать):

```
true && 5;
0 && 5; // 0 — "ложка", так что 5 — игнорируется
'привет' && undefined && true; // возвращается первая же "ложка"
                               // (остальное — опускается)
true && 1 && null && 'добрый';
true && 'добрый' && 'честный' && 'благородный'; // "ложек" нет, так что
                                    // возвращается последнее значение

true || 5; // возвращается первая же "правка" (остальное — опускается);
0 || 5;
'верный' || 'ласковый' && 'честный' || false; // возвращается первая же
                                              // "правка"
false || 0 || undefined || "" || null; // "правок" нет, так что возвраща-
                                       // ется последнее значение
```

Вы видите, что в ответе вовсе не всегда бывают логические значения. Но, поскольку все значения так или иначе являются либо «правками», либо «ложками», всё работает гладко, так как «правки» и «ложки» заменяют настоящие и строгие булевы `true` и `false`.

Надеюсь, я смог ответить на ваш незаданный вопрос. Едем дальше — к последнему логическому оператору!

Логическое НЕ (!)

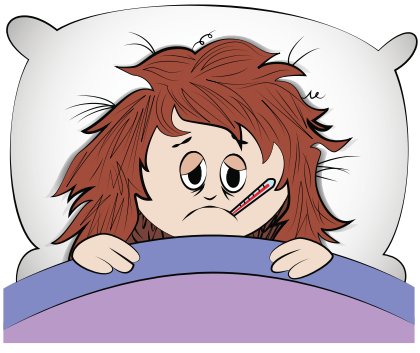
Третий и последний логический оператор — это обозначаемое восклицательным знаком (`!` — SHIFT+1) «**логическое НЕ**» (хотя многие разработчики называют его «нет» или просто, как мы уже говорили, «бац»). Посмотрите на следующие примеры:

```
!true; // false
!false; // true
```

Логическое НЕ просто отъявленный *бунтарь*! Что бы вы ему ни предложили, он всё наоборот вывернет! Вы, положим, говорите: «Какой чудесный погожий денёк, не правда ли?», а логическое НЕ такое: «Ага, конечно! Мерзопакостная слякоть и дождь, куда уж лучше! Ещё и тьма такая, хоть глаз выколи», — ну или просто булево `false`.

Стоит заметить, что, в отличие от логического И и ИЛИ, логическое НЕ *всегда* возвращает булево значение (либо `true`, либо `false`). Наберите в консоли и внимательно следите за ответами интерпретатора:

```
!0; // true (поскольку 0 — "ложка", ! делает его булевым true)
!5; // false (потому что 5 — "правка")
!(100 > 1);
!(5 !== 5);
!'строка любой длины';
!""; // пустая строка (хоть с двойными, хоть с одинарными кавычками)
!(true && true);
!(false || false);
!!true; // два ! (произносится "не не")
!!0; // то же, что и !(!0);
```



Теперь испробуем логическое НЕ в функции. Перво-наперво нам нужно будет… о, кажется, мне звонят… секундочку… похоже, мне придётся ответить. Я скоро вернусь.

Так, я здесь. Звонил ваш классный руководитель. Похоже, у вас в школе всерьёз обеспокоены угрозой эпидемии (у мно-

гих уже температура и тошнота). Так вот, согласно новым правилам, ученик НЕ должен приходить на уроки, если у него температура выше 37,7 ИЛИ его тошнит без перерыва. Как-то неловко про это тут писать, но что поделаешь? Такое вот распоряжение. Как думаете, сможете самостоятельно написать функцию (с использованием логического НЕ), которая бы определяла, не болеете ли вы? Если да, тогда начинайте! Если же нет, тогда попробуйте вот что:

```
function неБоленЛиЯ(мояТемпература, тошнитБезПерерыва) {
    let максЗдороваяТемпература = 37.7;
    if (!(мояТемпература > максЗдороваяТемпература || тошнитБезПерерыва)) {
        console.log("Ты здоров! Можешь идти в школу");
    } else {
        console.log("Лучше укутайся потеплее! Ты, похоже, заболел!");
    }
}
```

Обратите внимание, что первый параметр ожидает аргументом число, а второй — булево значение. Давайте вызовем нашу функцию с разными аргументами и посмотрим, как она справляется:

```
неБоленЛиЯ (37.7, false);
неБоленЛиЯ(37.6, true); // фууу… ну и кто это всё убирать будет?!
неБоленЛиЯ(65.5, false); // э-эмм… дай-ка я встряхну градусник, и ещё раз
                         // попробуем…
неБоленЛиЯ(37, false); // А, ну вот! Я ж говорю, что всё в порядке! Сего-
                       // дня что, контрольная по математике должна быть?
```

Отлично! А теперь соберём все кусочки воедино. Вы, наверное, догадываетесь, что наш пример излишне запутан и намеренно усложнён, в сравнении с тем, что встретилось бы вам в настоящем программном коде. Не сомневайтесь — мы ещё научимся всё делать красиво и аккуратно, избегая подобных неповоротливых конструкций, когда в последующих главах будем говорить об *отладке*. А пока давайте просто прицепим к нашей старой доброй функции ещё одну штуковину. Можете «достать» функцию стрелкой вверх из истории консоли. Приступим:

```
function нуТакМыИдёмИлиНет (мояТемпература, тошнитБезПерерыва,
съелЯичницу, съелХлопья, собралРюкзак, взялОбед, наделБотинки) {
    let = максЗдороваяТемпература = 37.7;
    if  (!(мояТемпература > максЗдороваяТемпература || тошнитБезПерерыва)
        && (съелЯичницу || съелХлопья)
        && (собралРюкзак && взялОбед && наделБотинки)) {
        console.log ("Ура-а-а! Наконец-то можно идти в школу! Отлично!");
    } else {
        console.log("Вот уж дудки! Ты или болен, или не готов!");
    }
}
/**
* Обратите особое внимание на порядок предлагаемых значений (то есть
"аргументов").
* Каждый аргумент соответствует своему параметру.
```

```
*/
нуТакМыИдёмИлиНет(36.6, false, false, true, true, true, true);
нуТакМыИдёмИлиНет(38.3, false, true, false, true, true, true);
нуТакМыИдёмИлиНет(36.7, true, true, true, true, true, true);
нуТакМыИдёмИлиНет(36.7, false, true, true, false, true, true);
нуТакМыИдёмИлиНет(37.7, false, true, false, true, true, true);
```

Ух! Какую штуку мы провернули, а? Едем дальше! Если вы до сих пор всё понимали, то дальнейшее покажется вам вообще легкотнёй!

if...else if...else

Вы, должно быть, уже считаете себя экспертом по части условных конструкций вроде `if`…`else`, да? Боюсь вас расстроить, но это так пока лишь отчасти. Если уж кто-то считается экспертом по части `if`...`else`, то он должен знать и… `else if`, а то какой же он эксперт, если он этого не знает!

Ну, например, вам подарили на день рождения новый будильник. Вы хотите написать программку, которая бы будила вас вовремя. С понедельника по пятницу вы встаёте в 6:30 и идёте в школу. Но в субботу ведь можно и поспать вволю! Итак, вы пишете функцию:

```
function времяПодъёма(деньНедели) {
    if (деньНедели === "суббота") {
        return null; // сегодня — поспим без будильника!
    } else {
        return "6:30";
    }
}
времяПодъёма("вторник");
времяПодъёма("суббота");
```

Шикарно работает! Хотя, стоп… погодите! Ведь в воскресенье же вы встаёте в 7:30, чтобы успеть на службу в церковь. Так значит, у нас теперь целых *три* варианта. Выглядит как работёнка для `else if`! Нажмём несколько раз стрелку вверх, чтобы вернуть функцию, и изменим её так:

```
function времяПодъёма(деньНедели) {
    if (деньНедели === "суббота")  {
        return null; //  сегодня — поспим без будильника!
    } else if ((деньНедели === "воскресенье") ) {
        return '7:30';
    } else {
        return '6:30';
    }
}
времяПодъёма("воскресенье");
времяПодъёма("пятница");
времяПодъёма("суббота");
```

Словом, вы понимаете, что если мы хотим, чтобы `else if` работал правильно, то нам нужно ещё одно условие (в скобках, конечно), после этого второго `if` (который следует сразу за первым `else`). И если значение будет «правкой», то запустится код из блока `if`, а если нет, то из `else` (который второй и ниже).

Слушайте, а я тут подумал: а почему бы нам вот этот только что выученный `else if` не применить в нашей `нуТакМыИдёмИлиНет` функции, чтобы она ещё более полезными сообщениями об ошибках нас завалила? Сейчас быстренько всё переделаем и внесём изменения! Вы помните, что SHIFT+ENTER позволит вам просто и комфортно скакать со строчки на строчку? Поехали:

```
function нуТакМыИдёмИлиНет (мояТемпература, тошнитБезПерерыва,
съелЯичницу, съелХлопья, собралРюкзак, взялОбед, наделБотинки) {
    let = максЗдороваяТемпература = 37.7;
    let сообщение; // undefined (пока);

    if  (мояТемпература > максЗдороваяТемпература) {
        сообщение = "Ты весь горишь! А ну быстрее в постель.";
    } else if (тошнитБезПерерыва) {
        сообщение = "И ты правда в таком виде хочешь в школу? "
            + "Иди-ка лучше в душ, а футболку кинь в корзину с грязным
бельём!";
    } else {
        if (съелЯичницу || съелХлопья) {
            if (!собралРюкзак) {
                сообщение = "Собери наконец рюкзак!";
            } else if (!взялОбед) {
                сообщение = "Ты опять забыл свой обед!";
            } else if (!наделБотинки) {
                сообщение = "А ботинки кто надевать будет?";
            } else {
                сообщение =  "Наконец-то можно идти в школу! Отлично!";
            }
        } else {
            сообщение = "Завтрак — это самый важный приём пищи!";
        }
    }
    console.log(сообщение);
}
```

Да уж! Вот это махина! Таких громадных функций мы ещё не писали! Но вместе с тем она весьма логична и по-своему даже удобочитаема (пожалуй, даже больше, чем ранние короткие версии)! Теперь давайте вызывать новую функцию с разными аргументами. Каждый раз до того, как нажмёте ENTER, попробуйте спрогнозировать ответ интерпретатора:

```
нуТакМыИдёмИлиНет(37.2, false, true, true, false, true, true);
нуТакМыИдёмИлиНет(38.8, false, true, false, true, false, true);
нуТакМыИдёмИлиНет(36.6, false, true, true, true, true);
нуТакМыИдёмИлиНет(36.7, true, true, true, true, true, true);
```

```
нуТакМыИдёмИлиНет(37.3, false, true, false, true, false, true);
нуТакМыИдёмИлиНет(36.7, false, false, false, true, true, true);
нуТакМыИдёмИлиНет(37.7, false, false, true, true, true, true);
```

Ну как вам? Всё ли было понятно? Если нет, пожалуйста, вернитесь и повторите ещё раз то, что не поняли! Поверьте, много времени это не займёт, зато потом — очень много сэкономит! А ещё, если вам попалось слово, значение которого вы не знаете или позабыли, обязательно найдите его в глоссарии. Не старайтесь читать «на скорость», будто у вас завтра экзамен по этой книжке! Главное, чтобы вы всё понимали и запоминали. Не сомневаюсь, вам это по силам!

Повторяй за мной: билеты в кино

За помощью к вам обратился местный кинотеатр. Они просят написать для них программу, которая бы сама определяла, достаточно ли ребёнку лет, чтобы пройти на сеанс того или иного фильма. Итак, на данный момент в кинотеатре показывают фильмы с рейтингом G, PG-13 и R. Рейтинг G (то есть general — «общий») означает, что на сеанс допускаются все; PG-13 — что ребёнок должен быть старше 13 лет; а вот фильмы с рейтингом R доступны к просмотру лишь тем, кто старше 17. Что ж, попробуем написать функцию, чтобы она определяла, продавать ли кассиру билет или нет.

Если вы знаете, как и что нужно делать (или хотя бы с чего начать), — обязательно приступайте, не читая далее! Если же нет, то, в общем-то, просто повторяйте за мной! Но сперва — вы знаете — закройте-откройте всё, что нужно, чтобы перед вами осталось лишь чистое окно консоли. Готово? Отлично! За дело:

```
function можноМнеБилетНаФильм(рейтингФильма, возраст) {
    let минВозрастДляR = 17;
    let минВозрастДляPg13 = 13;
    let сообщение = null;

    if (рейтингФильма === 'R' && возраст >= минВозрастДляR) {
        сообщение = 'Пожалуйста, ваш билет на фильм с рейтингом R.';
    } else if (рейтингФильма === 'PG-13' && !
              (возраст < минВозрастДляPg13)) {
```

```
        сообщение = 'Пожалуйста, ваш билет на фильм с рейтингом PG-13.';
    } else if (рейтингФильма === 'G') {
        сообщение = 'На фильмы с рейтингом G нет ограничений по возрасту;
                     пожалуйста, вот ваш билет.';
    } else {
        сообщение = 'К сожалению, вы не сможете посетить данный сеанс.';
    }

    console.log(сообщение);
}
```

Убедимся, что всё работает:

```
можноМнеБилетНаФильм('PG-13', 13);
можноМнеБилетНаФильм('R', 15);
можноМнеБилетНаФильм('G', 7);
можноМнеБилетНаФильм('R', 17);
```

Работает! Превосходно! И снова с помощью наших суперспособностей мы выручили… что, простите? А… Ну-у, ладно… Звонили из кинотеатра, сказали, что к ним стали поступать жалобы на работу нашего алгоритма: дескать, ограничения по возрасту не распространяются на отпрысков, пришедших в кино вместе с родителями. Словом, ребёнок любого возраста имеет право посещения любого сеанса, если вместе с ним идёт взрослый. Это значит, что нам нужно немного скорректировать работу нашей функции. Пожалуй, ничего сложного, как считаете? Смелее вперёд, на баррикады! Штурмуйте их своими догадками и идеями (помните, существует несколько способов выполнить одно и то же задание; ну и, конечно, если вы сможете справиться самостоятельно, то это будет много полезнее!).

Приступим:

```
function можноМнеБилетНаФильм(рейтингФильма, возраст, вместеСоВзрослым) {
    let минВозрастДляR = 17;
    let минВозрастДляPg13 = 13;
    let сообщение = null;

    if (рейтингФильма === 'R' && (возраст >= минВозрастДляR ||
        вместеСоВзрослым)) {
        сообщение = 'Пожалуйста, ваш билет на фильм с рейтингом R.';
    } else if (рейтингФильма === 'PG-13'
        && (!(возраст < минВозрастДляPg13) || вместеСоВзрослым)) {
        сообщение = 'Пожалуйста, ваш билет на фильм с рейтингом PG-13.';
    } else if (рейтингФильма === 'G') {
        сообщение = 'На фильмы с рейтингом G нет ограничений по возрасту;
                     пожалуйста, вот ваш билет.';
    } else {
        сообщение = 'К сожалению, вы не сможете посетить данный сеанс.';
    }

    console.log(сообщение);
```

```
}

можноМнеБилетНаФильм('PG-13', 9, false);
можноМнеБилетНаФильм('PG-13', 9, true);
можноМнеБилетНаФильм('G', 7); // помните? если аргумент не указан, то —
                              // false
можноМнеБилетНаФильм('R', 16);
можноМнеБилетНаФильм('R', 15, true);
```

И опять вы — герой всего города! Плёвое дело; рабочие будни простых супергероев! (Звук развевающегося по ветру плаща.)

ВИКТОРИНА

Выполняйте задания в Рабочей тетради. Когда закончите, попросите папу или маму проверить вас по ответам в конце (ну или сами себя проверьте, если родители заняты).

1. Как называется примитивный тип данных, означающий, что ещё никакого значения присвоено не было?
2. Как и операторы сравнения, _____ операторы (`&&`, `||` и `!`) отлично подходят для работы с условными конструкциями.
3. Как называется примитивный тип данных, который *ничего* не означает (в смысле означает ничего — не нуль, не пустую строку, не `undefined`, не `false` и так далее)?
4. Если в объявлении функции указано четыре параметра, но при вызове было предложено лишь два значения аргументов, какие значения получат оставшиеся два параметра?
5. Какой логический оператор вернёт в ответе «ложку», только если *оба* значения также являются «ложками»; как обозначается этот оператор?
6. Назовите все возможные значения для типа данных null.
7. Истина/Ложь: `53 >= 53 && !(51 <=52)`.
8. Если условие при блоке `if` вернуло ложь (или «ложку»), какую синтаксическую конструкцию можно применить, чтобы добавить ещё один вариант условия, до того как выполнить код блока `else`?
9. Назовите *единственный* логический оператор, который всегда возвращает булево значение; как обозначается этот оператор?
10. Истина/Ложь: так или иначе, любое значение в JavaScript можно считать либо «правкой», либо «ложкой»?

11. Какой логический оператор вернёт в ответе «правку», если хотя бы одно из значений будет «правкой»; как обозначается этот оператор?
12. Истина/Ложь: `true || false;`
13. Истина/Ложь: `false || (true && false);`
14. Истина/Ложь: значение и `null`, и `undefined` — «ложка».
15. Какой логический оператор вернёт в ответе «ложку», если *хотя бы одно* из значений будет «ложкой»; как обозначается этот оператор?
16. Истина/Ложь: `null || false !== undefined;`
17. Назовите значение переменной `моёСообщение`:

```
let моёСообщение;
```

18. Сколько существует возможных значений для типа данных `undefined`?
19. Истина/Ложь: `(null || false) || ((0 || true) || undefined);`
20. Назовите значение переменной `я`:

```
var я = !'уверен в себе! всё понимаю' || !!'растерян. ничего тут
не понимаю';
```

21. Какой логический оператор вернёт в ответе «правку», только если *оба* значения также являются «правками»; как обозначается этот оператор?
22. `null`, `false`, `' '`, `0` и `undefined` являются _____; а `true`, `'строка'` и `1` — _____.
23. Назовите значение переменной `большаяИзНих`:

```
var большаяИзНих= 'вера' && 'надежда' && 'любовь';[1]
```

КЛЮЧЕВЫЕ ЭПИЗОДЫ

Как обычно, пробегитесь по списку и обязательно вернитесь назад, чтобы повторить, если увидите что-то, что вам не знакомо или не совсем понятно:

- Тип данных null и его единственное возможное значение — `null`.
- Тип данных undefined и его единственное возможное значение — `undefined`.
- Три логических оператора.
- Логическое И (`&&`).

[1] Цитата из послания ап. Павла (1. Кор. 13:13).

- Логическое ИЛИ (`||`).
- Логическое НЕ (`!`).
- Возвращаемые значения операторов `&&`, `||` и `!`.
- `if...else` `if...else` .

УПРАЖНЕНИЯ

I. Введите следующие правильные фрагменты кода в консоль.

```
1. true && false && true;
```

```
2. true || false || true;
```

```
3. if (5 < 4) {
       console.log('A');
   } else if (5 > 4) {
       console.log('B');
   } else {
       console.log('C');
   }
```

```
4. 3 && (null || ((15 / 3) - 5));
```

```
5. if ('abc' > 'd') {
       console.log('A');
   } else if (0) {
       console.log('B');
   } else {
       console.log('C');
   }
```

```
6. if (5 && 'десятка' && null) {
       console.log('A');
   } else if (0 || undefined || '') {
       console.log('B');
   } else if (false || -1 || !0) {
       console.log('C');
   } else {
       console.log('D');
   }
```

```
7. // уделите этому примеру особое внимание! важно, чтобы вы его продумали
   // и поняли
   function ктоСтаршеТотИПервый(возраст1Человека, возраст2Человека) {
       let ктоПервый;
```

```
        if ((! возраст2Человека && возраст1Человека) || возраст1Человека >
            возраст2Человека) {
            ктоПервый = 1;
        } else if ((!возраст1Человека && возраст2Человека) ||
                    возраст2Человека > возраст1Человека) {
            ктоПервый = 2;
        } else {
            ктоПервый = null;
        }
        if (ктоПервый) {
            console.log( ктоПервый + '-й человек' + ' старше, а значит,
                         ходит первым.');
        } else if (!(возраст1Человека || возраст2Человека)) {
            console.log('Нужно предложить возрасты этих двоих в качестве
                         аргументов!');
        } else {
            console.log("Да они же ровесники! Давайте выберем случайным
                         образом? "
                + (Math.floor(Math.random() * 2) + 1) + ' ходит первым!');
        }
    }
8. ктоСтаршеТотИПервый(80, 10);
9. ктоСтаршеТотИПервый(15, 17);
10. ктоСтаршеТотИПервый(20);
11. ктоСтаршеТотИПервый(null, null);
12. ктоСтаршеТотИПервый(13, 13);
13. ктоСтаршеТотИПервый();
14. ктоСтаршеТотИПервый(null, 5);
15. ктоСтаршеТотИПервый(7, 7);
16. function неизвестныйЧеловек(имя) {
        console.log('Его зовут ' + (имя || 'Иван Иваныч') + '.');
    }
17. неизвестныйЧеловек('Толька С.-Корей');
18. неизвестныйЧеловек();
19. неизвестныйЧеловек('Большеголо, Вова');
20. неизвестныйЧеловек(null);
```

II. Что не так с каждым из фрагментов?

```
1. function нетНет(любойАргумент) {
       if (!!любойАргумент) {
           console.log('любойАргумент — "ложка"');
       } else if (!любойАргумент) {
```

```
            console.log(' любойАргумент — "ложка"');
        } else {
            console.log(' любойАргумент — "правка"');
        }
    }
2. if (25 > 14) {
        'Да';
    } else if {
        'нет';
    }
3. let !т = 'вот что кузнечики стрекочут нам в ответ'
4. true && false = false;
5. let x || 5 = !true;
```

КОМПЛЕКСНЫЙ ОБЗОР

1. Является ли следующее предложение правильным (и если нет — почему)?

   ```
   let мояМамаГотовитЛучшеВсех = !!'ну ещё бы!';
   ```

2. Перечислите (по порядку) все типы данных для следующих «ложек»:

   `false; " "; null; 0; ' '; undefined;`

3. Какое действие осуществляет оператор `%`?
4. Строка в JavaScript состоит из отдельных _____.
5. Истина/Ложь: когда интерпретатор JavaScript доходит до однострочного комментария, то он игнорирует всё с самого начала комментария и до конца строчки.
6. Всего в JavaScript существует три _____ оператора (`&&`, `||` и `!`).
7. Вот операторы _____, которые можно и нужно применять: `===`, `!==`, `>`, `>=`, `<` и `<=`.
8. Какой тип предложений используется для запуска блоков кода на основании данных условий?
9. Если вы вызываете функцию, которая находится внутри другой функции, то такую функцию называют _____.
10. Правильно ли написан следующий код (если хотите, попробуйте в консоли)? Если нет — почему?

    ```
    function высотаБаскетбольногоКольца(высота) {
        let минВысота= 60; // см
        let максВысота = 120;
    ```

```
    let вердикт;
    if (высота > минВысота && высота < максВысота) {
        вердикт = 'Порядок! Можно играть!';
    } else if (высота > максВысота) {
        вердикт = 'Это слишком высоко!';
    } else {
        вердикт = 'Это слишком низко!';
    }
    console.log(вердикт);
}
высотаБаскетбольногоКольца(53);
высотаБаскетбольногоКольца(89);
высотаБаскетбольногоКольца(134);
высотаБаскетбольногоКольца(120);
```

11. Истина/Ложь: логические операторы всегда возвращают булевы значения.
12. При помощи какого символа можно было бы упростить следующее присваивание:

    `счётВМатче = счётВМатче + 10;`

13. Каким оператором стоит воспользоваться, чтобы узнать, делится ли некое число нацело на 52?
14. Укажите какие из следующих значений — «правки», а также перечислите типы данных, к которым они принадлежат:

    `" "; -1; true; 15.3; 'false'; '0';`

15. Правильно ли написан следующий код (если нет — почему)?

```
function поздоровайсяСоМной(моёИмя) {
    return моёИмя + '!';
    console.log('Привет, ' + моёИмя);
}
поздоровайсяСоМной('Джереми');
```

16. Какой оператор сравнения вернёт в ответе `false`, если обнаружит, что значение справа больше или равно значению слева?
17. Где разработчик может легко и просто проверить работу написанного кода и получить немедленный ответ интерпретатора?
18. Что идёт после вторника?
19. При помощи какой клавиши можно выделить код отступом, чтобы он легче читался (например, внутри функции)?
20. Истина/Ложь: когда одни скобки заключены в другие, всегда нужно сперва выяснить значение во внешних скобках, прежде чем переходить ко внутренним.

21. Истина/Ложь: операторы сравнения всегда возвращают булевы значения.
22. Правильно ли написан следующий код (если нет — почему?):

```
function дайПять(5) {
    return 5;
}
дайПять(5);
```

23. При помощи какой встроенной функции можно округлить десятичную дробь до ближайшего целого?
24. Истина/Ложь: `31 >= -31 && !(38 <= 38) || !!(-39 < 39);`.
25. Истина/Ложь: код, заключённый в комментарии, будет проигнорирован интерпретатором JavaScript.
26. Какой комбинацией клавиш делается перенос строки в консоли?

Сделай сам: приключения Ланка!

Слыхали? Скоро выходит новая часть знаменитой саги «*Легенды Зебры*»[1]! Игра уже почти готова, но вот незадача: программистам всё не удаётся чётко проработать логику прохождения последнего — Полосатого Храма. По сюжету дверь Храма должна вести себя следующим образом:

- Если у Ланка нет с собой всех шести частицСилы, то храмоваяДверь отсылает его на поиски недостающих Частиц.
- Если же у него есть все Частицы, тогда он должен сразу показать, что имеет также *и* посохРотовойВони, *и* главныйКлюч (*или* же 10 обычныхКлючей, если главногоКлюча у него нет).
- Если у Ланка есть искомые посох *и* ключ(и), то Дверь объявляет ему, что теперь он может попасть в Полосатый Храм.
- Если же у него нет посоха и/или ключа, то Дверь должна отказать ему, выдав какое-нибудь туманное и бесполезное сообщение, заключив его пожеланием, чтобы он возвращался, когда будет получше готов (похоже, это единственный способ дать понять Ланку, что нельзя просто взять с собой случайные предметы и ожидать, что они сработают! Можно, скажем, послать его отнести блюдо тушёного мяса с луком верховномуМагу в тёмныйГрад… после дождичка в четверг!).

Итак, вы понимаете, что вам нужно написать функцию с блоками `if`…`else`, которая бы следовала указанному выше сюжету (проверяла наличие или отсутствие необходимых артефактов, которые должны предлагаться в качестве аргументов

[1] Отсылка к популярной игровой серии Legends of Zelda и её главному герою Ланку.

вроде `главногоКлюча` и так далее). Не забудьте про логические операторы — они будут вам здесь большим подспорьем! Когда закончите, не забудьте проверить функцию с разными аргументами для всех сюжетных линий (надеюсь, вы понимаете, что рагу для верховногоМага не является среди них такой уж обязательной! Это так, смеха ради).

По завершении (ну или до него, если возникнут непреодолимые трудности) обязательно загляните в рекомендации по выполнению в конце книжки. Помните, что выполнять проекты «Сделай сам» можно несколькими правильными способами. Единственное требование — это чтобы ваш способ учитывал все необходимые комбинации артефактов, которые имеет или не имеет с собой Ланк.

7 Проекты ПроектыПроекты

Поздравляю! Вы зашли уже очень далеко! Надеюсь, вы и сами понимаете, насколько многому успели научиться за эти шесть глав. Здорово, правда?

Полагаю, эта глава покажется вам самой весёлой из всех, что были до сих пор. Мы, конечно, разучим пару маленьких новых трюков, но в основном будем упражняться в том, что вы уже узнали из предыдущих глав. Вот ваш шанс размяться и вдоволь «поиграть мускулами» на множестве самых разнообразных проектов!

Идея этой главы предельно проста: я вам что-то показываю, мы делаем проект на эту тему, а затем вы делаете ещё один самостоятельно. Потом я ещё что-то показываю, мы делаем ещё один проект вместе, а затем вы делаете ещё один самостоятельный проект. И так далее! Один за другим, без лишних остановок и проволочек! Говоря «без лишних», я имею в виду, что если вы и так знаете, что и как нужно будет делать, — не ждите моих разъяснений, сразу же беритесь за дело! Я хочу, чтобы вы продумали каждый проект и попытались выполнить его по возможности самостоятельно; по крайней мере, мне бы очень хотелось, чтобы вы потратили какое-то время на размышление, прежде чем ознакомиться с моим вариантом. Ведь это (ну, знаете, всякие там размышления и всё такое) — львиная доля того, чем занимаются настоящие разработчики; так что чем больше и лучше вы думаете, тем более крутым разработчиком будете.

Прежде чем приступить к нашим проектам, пожалуйста, закройте-откройте Chrome, откройте пустую вкладку и запустите консоль. Итак, если вы готовы, мы начинаем!

Всякому безумию — своя логика!

В некоторых проектах нам понадобятся классные штуки под названием «свойства и методы». Вот простой пример свойства:

```
'абвгдеёж'.length; // .length = то есть "длина" — отвечает на вопрос
                   // "сколько символов в строке?"
```

Ответ: `8`

В общем, если видите точку между строкой и каким-то словом — знайте, что это самое «какое-то» слово и есть свойство. **Свойство** — это некоторое именованное значение, прикреплённое к объекту. Что-что? «Что такое объект?» Хм-м… Вот надо было вам спросить, а?

Ну… Значит… Наверное, следующее покажется вам непонятным и странным. Штука в том, что для того чтобы объяснить всё чётко и полностью, потребовалось бы несколько страниц; я же постараюсь втиснуть объяснение лишь в один абзац, чтобы сильно не удаляться от наших проектов. В общем, так: **объекты** — это одни из важнейших единиц в JavaScript. JavaScript просто доверху набит объектами! Объекты в JavaScript — это такие наборы свойств (именованных значений). Ну, скажем, помните `Math.random()`? Так вот: `Math` является объектом, а точка (`.`) указывает на то, что следующее слово (`random`) будет свойством этого объекта. Ну а поскольку после `random` идут скобки, то это указывает ещё и на то, что это — функция (ведь у всех функций есть скобки). Если свойство является функцией, то его называют **методом**.

Но мы уже начинаем забегать сильно вперёд. С объектами мы будем разбираться чуть позже. Пока нам достаточно того, что мы знаем, — строка является одним из типов объектов и может иметь свойства вроде `length`, которые могут нам что-либо рассказать о ней.

У объектов, как мы сказали, также есть и *методы* (прикреплённые функции), которые позволяют выполнять разные манипуляции с этими объектами. Продолжая пример со строками, рассмотрим методы `.contact()` и `.repeat()`:

```
'Какое небо'.concat(' голубое.'); // делает то же, что и + при объединении
'А почему? '.repeat(50);          // угадайте, что делает эта штука?
                                  // Попробуйте у себя в консоли!
```

Теперь вы можете доставать своих друзей при помощи всего лишь одной строчки кода! И это ещё не всё: скоро вы выучите ещё кучу всяких новых методов и всего такого!

Уф! Простите, что пришлось отвлечься на всё это. Мне подчас нелегко обойти стороной что-то, что интересно и важно, но не совсем своевременно — вроде объектов, свойств и методов. Но всё же нужно соблюдать некоторую систему, так ведь?! Так о чём это я…

Ах, да! Давайте-ка разучим парочку новых штук для нашего первого проекта!

Смена регистра: toUpperCase() и toLowerCase()

Итак, НАЧНЁМ с ПРОСТОГО! Вы КОГДА-НИБУДЬ замечали, ЧТО в НЕКОТОРЫХ сообщениях ЧЕРЕСЧУР много СЛОВ, выделенных ПРОПИСНЫМИ буквами? Иногда ВООБЩЕ КАЖЕТСЯ, будто НА ВАС ОРУТ, когда выделено СТОЛЬКО СЛОВ СРАЗУ! А вы НИКОГДА не ХОТЕЛИ просто КАК-ТО УГОМОНИТЬ этих людей СО ВЗБЕСИВШИМСЯ КАПС_ЛОКОМ? Что ж, именно этому мы сейчас и научимся.

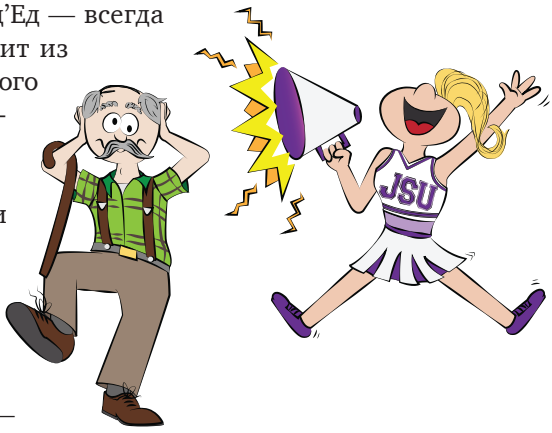
Вы можете запросто взять любую строку и вернуть её новую версию через прописные буквы (то есть в верхнем регистре), используя встроенный метод `.toUpperCase()`. И, соответственно, можно вернуть любую строку в строчных буквах (то есть в нижнем регистре) при помощи метода `.toLowerCase()`. Помните, что метод — это всего лишь функция, прикреплённая к объекту, в данном случае к строке. Работает это так:

```
'Я люблю пиццу!'.toUpperCase();
"СЛУШАЙ, НУ ХВАТИТ УЖЕ ТАК ОРАТЬ.".toLowerCase();

let имяПользователя = 'Фрэнк';
let приветствие = "Как я рад тебя видеть, " + имяПользователя + '!';
console.log(приветствие.toUpperCase() + ' Я так скучал!');
```

Есть вопросы? Нет? Отлично! Тогда едем дальше!

Повторяй за мной: крикоглушилка!

Один из крупнейших городских меценатов и глава городского совета — сэр Дитый д'Ед — всегда шибко нервничает и даже выходит из себя, когда на веб-странице городского форума видит «ОРУЩИЕ» сообщения. Очень важно, чтобы влиятельный сэр Дитый был спокоен и всем доволен. Поэтому нам поручили написать для него крикоглушилку. Она будет переводить в нижний регистр любое новое сообщение на форуме, так что почтенному старейшине не придётся больше нервничать (ну да-да, вы правы — это какая-то ерунда, но всё же давайте напишем такую штуку, ладно?).



```
function крикоглушилка(сообщение) {
     return сообщение.toLowerCase();
}
// проверим
крикоглушилка("НЕСНОСНОЕ СООБЩЕНИЕ, НАПИСАННОЕ КАПС_ЛОКОМ!!!");
крикоглушилка("emAil-AdrESSa@oByCHno-RaboTAyuT-S-LyuBiMI-BUkVaMi.cOM");
```

Всё очень и очень просто, правда? А теперь самостоятельный проект!

Сделай сам: настраиваем спаморассылку

Итак, вас наняла компания по рассылке спама из Гдетодалеконии. В этой компании приняты чёткие стандарты качества для своей продукции: их спам обязательно должен обещать огромные суммы денег, содержать орфографические и пунктуационные ошибки, призывать срочно действовать, но самое главное — всё письмо должно быть строго в верхнем регистре. Большинство пунктов этого списка уже выполнено квалифицированными специалистами (так что вам не стоит о них переживать), однако с прописными буквами вам придётся справиться самостоятельно, написав соответствующую функцию.

Создание переменных с const

Тема, которую мы сейчас обсудим, — одна из важнейших в этой главе, так что будьте предельно внимательны! Помните, как в главе 1 мы научились создавать переменные при помощи ключевого слова `var`? А потом помните, как я вам сказал, что мы лучше будем использовать `let` вместо `var`, хотя, в общем-то, они делают почти одно и то же (хотя я так и не удосужился пояснить, в чём же там разница)? Помните, да? Эх… Хорошие были времена…

Ну так что же? Выучим ещё одно ключевое слово для создания переменных! Ведь вам всегда хотелось узнать *третий* способ для того, чтобы выполнять одно и то же самое простое действие, верно? Итак, прошу любить и жаловать: `const` (сокращение от «constant», то есть **константа** — постоянная, неизменяемая величина). Используется она следующим образом:

```
const днейВНеделе = 7;
днейВНеделе;
```

Ответ = `7`

Пожалуй, вы бы сказали, что `const` — довольно бесполезная штука, так? Разве нельзя было сделать то же при помощи `let` или `var`? Что ж, и да и нет. Вы, конечно, могли бы написать схожий код, используя `let` или же `var`, получив затем тот же ответ, но тут есть одна большая разница. Сейчас увидите какая:

```
днейВНеделе = 8;
```

Ответ: `Uncaught TypeError: Assignment to constant variable.`

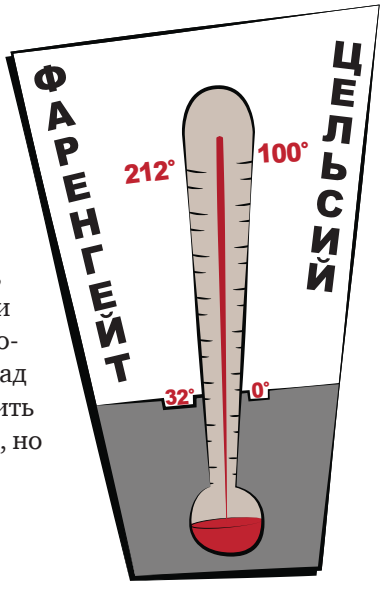
Ну как вам? Вы понимаете, что создай мы нашу переменную с `var` или `let` — никаких проблем для интерпретатора не составило бы присвоить ей новое значение, изменив `7` на `8`. Но именно потому, что мы использовали `const`, интерпретатор был предупреждён, что эта переменная — константа, то есть своего значения она *не изменит*. Так что если вы теперь наберёте `днейВНеделе;` в консоль, то получите тот же ответ — `7`.

Таким образом, `const` мы будем использовать тогда, когда нам не нужно, чтобы значение переменной изменялось. Мой личный опыт показывает, что я применяю `const` раза в 3–4 чаще, чем `let`; от использования же `var` я уже довольно давно и вовсе отказался. Давайте для закрепления напечатаем в консоли несколько примеров с `const`:

```
const месяцевВГоду = 12;
const мойДеньРождения = '1981-06-30';
const длинаЭкватора = 40075;
const секундВМинуте = 60;
const столицаПольши = 'Варшава';
```

Все эти вещи не изменятся (по крайней мере, за время написания и запуска вашего кода!). Вот вам золотое правило создания переменных: если вы точно знаете, что ваше значение не изменится, используйте `const`. В любом противном случае — `let`. В использовании `var` нынче нет никакой надобности: `var` — это пережиток далёкого прошлого и попал в эту книжку лишь для того, чтобы вы знали, что это такое (поскольку его всё ещё можно встретить во многих и многих кодах). Итак, из этих трёх чаще всего стоит использовать `const`.

Повторяй за мной: конвертер температур (из Фаренгейта в Цельсия)



На очереди ещё один небольшой проект! Как и со всеми прочими проектами здесь, если вы знаете, что и как нужно делать, не ждите и дерзайте! Если нет, тогда сейчас всё сделаем вместе. Я бы хотел, чтобы перед завершающим этапом вы вернулись назад и пробежались по коду с самого начала, чтобы оценить свежим взглядом, сможете ли вы проделать всё то же, но теперь уже самостоятельно.

Наша задача: для школьного проекта по физике вам необходимо разогревать жидкости до разной температуры. Проблема в том, что все значения указаны в градусах Фаренгейта, а ваш термометр снабжён только шкалой Цельсия. Нам нужна функция, которая бы переводила данное ей значение из градусов Фаренгейта в градусы Цельсия.

Вы, наверное, нисколько не удивитесь, если я скажу вам, что Google уже проделал всю работу за нас; простой поисковый запрос выдаст вам формулу перевода одной температурной шкалы в другую:

```
(градусыФаренгейта - 32) / 1.8;
```

Так что мы просто можем вставить эту формулу в нашу функцию:

```
function фаренгейтыВЦельсии(градусыФаренгейта) {
    return ( градусыФаренгейта - 32) / 1.8;
}
```

Проверим:

```
фаренгейтыВЦельсии(113);
фаренгейтыВЦельсии(32);
```

Ну что? Пожалуй, сделано? Так-то так, но… есть небольшая проблема. Если посмотреть на нашу функцию беспристрастно, то можно подумать, что значения в формуле взяты… как бы сказать… словом, чуть ли не «с потолка». В самом деле, почему именно `1.8` и `32`? Почему не `2.9` и `42`, например? Наверное, тут должно быть какое-то объяснение, как считаете?

Такие сомнения — дело довольно частое. Порой, когда встречаются такие «странные» числа вроде тех, что выше, бывает очень полезно составить некоторый поясняющий текст вроде инструкции или мануала. Подобную «инструкцию» для кода принято называть **документированием**.

Пожалуй, самое время представить вам ещё одно «золотое правило»: когда вы имеете дело с числами, почти всегда стоит сразу присваивать их переменным. Таким образом, кто бы ни изучал ваш код, ему сразу же будет понятно, почему были выбраны те или иные значения. Ваш код просто становится более читабельным и понятным, а у вас отпадает необходимость писать многословные комментарии и программные документации. Если код написан с использованием понятно названных переменных и функций, то, в общем-то, он может объяснить себя сам; это называется **самодокументированием**, и я очень вам рекомендую по возможности всегда писать свой код именно так.

Так что же означают эти `1.8` и `32`? Что касается `1.8` — это так называемое «конверсионное соотношение» температурных шкал Цельсия и Фаренгейта, то есть это то число, на которое нужно умножить или разделить значение при переводе его в другую систему. `32` же — это температура замерзания воды по шкале Фа-

ренгейта, по шкале Цельсия вода замерзает при 0 градусов. Для нашего задания не столь важно, почему всё так, а не иначе. Нам сейчас важнее просто понять, что стоит за этими числовыми значениями, а также то, что они — *константны*.

Теперь, разобравшись с этим, нам будет проще простого сделать наш код самодокументированным. Вернём его нажатием стрелки вверх и немного видоизменим:

```
function фаренгейтыВЦельсии(градусыФаренгейта) {
    const конверСоотношение = 1.8;
    const замерзаниеВодыУФаренгейта = 32;
    return ( градусыФаренгейта - замерзаниеВодыУФаренгейта) /
             конверСоотношение;
}
```

Мы знаем, что `1.8` и `32` никогда в этой формуле не изменятся, поэтому можем назвать их константами (при помощи `const`) и использовать для них «говорящие» имена, что делает наш код красивым и самодокументированным.

```
фаренгейтыВЦельсии(212); // 100
фаренгейтыВЦельсии(59);  // 15
```

Всё получилось? Надеюсь, вам было понятно про «самодокументирование». Если нет, просто читайте дальше. Это не такое важное дело, чтобы на нём лишний раз останавливаться (но очень хорошо и полезно, если вы его уже поняли с первого раза!).

Сделай сам: ещё один температурный конвертер

Теперь, когда у нас готова функция для перевода градусов Фаренгейта в градусы Цельсия, давайте посмотрим, сможете ли вы сделать всё то же, только наоборот! То есть функцию, которая будет переводить температуру по Цельсию в температуру по Фаренгейту. Подсказка: нужно будет умножить наше температурное значение на `конверСоотношение` (то есть на `1.8`) и прибавить к нему `замерзаниеВодыУФаренгейта` (то есть `32`).

Если всё проделано верно, то вы сможете взять сконвертированные первой функцией значения и расконвертировать их обратно! В результате у вас должны будут получиться те же (или же очень-очень близкие к ним) значения!

confirm('Позвольте спросить?');

Помните, в главе 4 я вам показывал встроенную функцию `alert()`? А ещё помните, я тогда сказал, что применять её в коде *не* следует, так как она нервирует пользователей? Не, ну правда, что за дела такие? Зачем учить чему-то, чтобы потом говорить, что этим не нужно пользоваться?!

Слушайте! Я тут подумал… И правда! Вот вам *ещё две* встроенные функции! Мы их сейчас разучим, а потом не будем использовать! Первая из них — `confirm()`:

```
confirm('Пойдём нырять в озеро в последний весенний день?');
```

Круто, правда? Но вы, вероятно, спрашиваете себя: «А зачем мне вообще учить что-то, что я потом не буду использовать?» Если так, то я замечу, что вы ошибаетесь. И тут вы, вероятно, думаете: «Да ну? Ошибаюсь? Ерунда какая-то!» И тут я вам замечу, что вы будете совершенно правы.

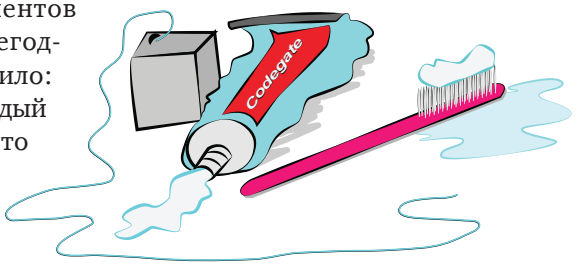
Вот ещё один способ использования функции `confirm()`:

```
if (confirm('А мама разрешила?')) {
       console.log("Отлично! Хватай плавки и бежим!");
} else {
       console.log('Что ж, наверное, она решила, что так будет
                    спокойнее.');
}
```

Словом, `confirm()` замораживает всё, чтобы получить немедленное подтверждение от пользователя. Значение, возвращаемое этой функцией, всегда булево. При нажатии OK возвращается `true`, а при нажатии «Отмена» (Cancel) — `false`. Это может быть полезно, скажем, если пользователь нажимает клавишу «Удалить». Тогда можно запросить у него подтверждение — `confirm('Вы точно хотите это удалить?')`, и таким образом помочь ему избежать возможной ошибки. Попробуйте удалить какой-нибудь файл на собственном компьютере, и вы увидите нечто похожее.

Повторяй за мной: ты уже почистил зубы?

Местный дантист доктор Ханс-Рвач фон Клещефф весьма устал от пациентов с плохим запахом изо рта и с сегодняшнего дня вводит новое правило: прежде чем попасть на приём, каждый пациент должен *подтвердить*, что он почистил зубы. Поможете доктору написать такую функцию?



Всё, как вы понимаете, очень просто и быстро. Если знаете, как это сделать, — вперёд! Если нет, давайте вместе:

```
function сегодняЧистилЗубы() {
    if (confirm('А вы уже почистили зубы?')) {
        return 'Можете пройти в кабинет доктора фон Клещеффа.';
    } else {
        return 'Нет уж! Сперва почистите зубы! Раковина — вон там.';
    }
}
сегодняЧистилЗубы();
сегодняЧистилЗубы();
```

Вот и всё!

Сделай сам: а как насчёт зубной нити?

Хм-м… Теперь доктор фон Клещефф хочет, чтобы каждый пациент перед приёмом не только почистил зубы, но и прошёлся зубной нитью по ротовой полости. А что дальше? Может, они ещё и каналы себе сами прочистят и запломбируют, а? Во дела…

Как бы то ни было, наша задача: во-первых, получить от пациента подтверждение, что он почистил зубы (и если нет — отправить его чистить их); во-вторых, если зубы почищены — спросить, прошёлся ли он после этого зубной нитью (и если нет — отправить сделать это). Если всё сделано, то можно проходить в кабинет к доктору. Если у вас возникнут какие-то трудности, обязательно загляните в ответы в конце.

Запросим prompt(ответ)

А вот и *вторая* встроенная функция, которую мы выучим, а потом я скажу, что использовать её не стоит (обещаю, скоро всё объясню)! Она называется `prompt()` и используется следующим образом:

```
prompt('Как тебя зовут?');
'Привет, ' + prompt('Как тебя зовут?') + '!';
'Тебе ' + prompt('Сколько тебе лет?') + ' лет.';
```

Функция `prompt()`, как и `confirm()`, замораживает всё, но просит не ответить «да»/«нет» на вопрос, а ввести свой ответ или же нажать кнопку отмены. Отсюда и различие в ответе: `prompt()` вернёт лишь строку (с ответом пользователя), а `confirm()` всегда ответит булевым значением. Строка вернётся лишь в том случае, если после ввода ответа пользователь нажал кнопку подтверждения; если же была нажата кнопка отмены, то `prompt()` вернёт в ответе `null`.

Повторяй за мной: поварёнок

Закроем-откроем браузер и запустим консоль в пустой вкладке.

Папин лучший друг устроился на работу в какую-то забегаловку! Ему там нравится, но чтобы не потерять место, ему нужна практика. И в этом ему можете помочь вы! Он будет готовить всё, что пожелаете! Вам лишь нужно написать функцию с запросом (то есть `prompt()`), чтобы можно было в ответе указать желаемое блюдо и выводом на экран получить сообщение, что папин друг берётся его приготовить.

Как обычно, если уверены (или не уверены, но хотите!), что справитесь сами, — пожалуйста, вперёд! Если же нет, тогда вот наша функция:

```
function приготовьтеМне() {
    const любимоеБлюдо = prompt ("А какое ваше любимое блюдо?");
    console.log(любимоеБлюдо + "? Превосходно! " + "Уже готовлю!");
}
приготовьтеМне();
приготовьтеМне();
приготовьтеМне();
```

Быть может, хоть раз из этих трёх вы попробовали нажать «Отмену» (Cancel)? Ну или нажали OK, не вводя никакого ответа? Если так, то вы уже оценили прекрасный ответ: `"null? Превосходно! Уже готовлю!"`… Да, ничто в мире не идеально!

Впрочем, при помощи обычной конструкции `if`…`else` мы можем это исправить. Вернём наш код стрелкой вверх:

```
function приготовьтеМне() {
    const любимоеБлюдо = prompt ("А какое ваше любимое блюдо?");

    if (любимоеБлюдо) {
        console.log(любимоеБлюдо + " ? Превосходно! " + "Уже готовлю!");
    } else {
        console.log("Разве у вас нет любимого кушанья? Ну, раз вы такой
                    вредный, приготовлю вам какую-нибудь дрянь!");
    }
}
приготовьтеМне(); // попробуйте "Отмену" (Cancel)
приготовьтеМне(); // попробуйте ничего не отвечать
```

Сработало? Всё ли было понятно? Надеюсь, что так!

Но всё же! Вопрос остаётся: так почему же не стоит использовать `confirm()` и `prompt()`? Пожалуй, наиболее простым ответом будет: «Потому что они всё

замораживают и никакой код не работает, пока вы не отреагируете». Не думаю, что вам покажется это достаточной причиной на столь ранней стадии обучения, просто имейте в виду, что потом — в будущем — вы поймёте, о чём шла речь.

Стоит признать, что порой использование `confirm()` (но не `prompt()`) оправданно — ведь эта функция может стать «последним рубежом обороны», когда пользователь собирается сделать что-то необратимое (к примеру, навсегда удалить свой аккаунт в социальной сети). В то же время функция `prompt()` в этом смысле не делает ничего особо полезного, а скорее даже раздражает и мешает пользователям.

К тому же существует множество других, менее навязчивых способов взаимодействия пользователя с интерфейсом программы, нежели чем постоянно тыкать ему в лицо всплывающим окном `prompt()` и дожидаться, пока он введёт ответ.

Тем не менее эти «менее навязчивые» и вообще более приятные способы являются одновременно и несколько более сложными, чем те, к которым мы пока готовы. Именно поэтому мы с вами пока разобрали, как работают `prompt()` и `confirm()`. И, пока мы не разучили чего-нибудь получше, так уж и быть — используйте `prompt()`! Пожалуй даже, вам придётся это сделать как минимум для того, чтобы выполнить проекты «Сделай сам» в этой главе.

Сделай сам: варим самый экзотический суп

Напишите функцию, которая будет трижды спрашивать пользователя о том, какой ингредиент еще добавить в суп. Как только все три (разные) ингредиента добавлены, выведите на экран или в консоль название экзотического супа, которое должно включать в себя все составляющие. К примеру, вы можете приготовить «Картофельный суп с луком и морковью».

Округляем десятичные дроби с toFixed()

Ещё одна полезная встроенная функция, которая поможет вам округлить любую десятичную дробь с точностью до нужного количества знаков после запятой[1]:

```
(8.111111).toFixed();  // если не указан аргумент, то функция округлит
                       // до ближайшего целого числа
(8.111111).toFixed(2); // в аргументе — 2, значит, после запятой будет
                       // 2 знака
```

[1] В англоязычных странах при записи чисел целая часть отделяется от дробной точкой, и тогда мы говорим о «плавающей точке» (*англ.* floating point). В российской традиции целая часть числа от дробной отделяется запятой и используется термин «плавающая запятая». В примерах кода используется запись через точку. — *Примеч. ред.*

```
(8.8899).toFixed();     // без аргумента, значит — до ближайшего целого
(5.666667).toFixed(1);
let точнаяВеличина = 2 / 3; // 0.6666666666666666
точнаяВеличина.toFixed(3)   // до 3 знаков после запятой = "0.667"
```

Вы заметили, что `toFixed()` всегда возвращает в ответе строку? Присмотритесь — все ответы заключены в кавычки! Так-то!

Эта функция может быть полезна, например, при каких-либо финансовых операциях. Скажем, вам необходимо подсчитать сумму налогового сбора при какой-нибудь внушительной покупке. Если с каждого доллара вы будете должны уплатить 0,09 центов, то можно представить это в виде следующего кода:

```
const налог = 0.09;
const знаковПослеЗапятой = 2;
(2.35 * налог).toFixed(знаковПослеЗапятой); // налог взимается с $2.35
(15.59 * налог).toFixed(знаковПослеЗапятой);
(9.99 * налог).toFixed(знаковПослеЗапятой);
```

В общем, с `toFixed()` всё довольно просто и понятно. Вы вряд ли будете пользоваться ею часто, но если вам придётся продираться сквозь толщу математических операций, вы ей очень обрадуетесь.

Повторяй за мной: калькулятор (предлагатор) чаевых

Закроем и заново откроем Chrome, а затем запустим консоль в `about:blank`.

Управляющий одного из местных ресторанчиков тайской кухни под названием «Ти-тай-ник» очень обеспокоен тем, что клиенты его заведения оставляют недостаточно чаевых. Он обратился к вам с просьбой написать «калькулятор чаевых», который бы, скажем так, помог посетителям ресторана быть чуть менее финансово скромными. Калькулятор должен вести себя следующим образом: после вызова функции посетитель получает через *запрос* на ввод стоимость всех заказанных блюд; после этого должно появиться сообщение, объявляющее рекомендованный размер чаевых (18 % от общей суммы).

Управляющий настаивает, чтобы рекомендуемые чаевые всегда указывались в размере 18 %; то есть мы можем сказать, что это значение — *постоянная* величина, так как изменению она не подлежит. Если бы эта величина менялась в зависимости от суммы заказа, то мы бы назвали такую переменную динамиче-

ской. Таковой, кстати, является как раз сумма заказа, которая каждый раз иная и зависит от данных, которые введёт пользователь в ответ на запрос функции.

Обратите внимание: любое значение в процентах — это дробь с делителем 100. Например, 18 % — это 18/100. В свою очередь, 18/100 можно записать в виде десятичной дроби: 0,18. Таким образом, если вы хотите узнать, сколько будет 18 % от некоего числа, то всё, что требуется, это просто умножить это число на 0,18.

Прежде чем пуститься вместе в конструирование этого «предлагатора» чаевых, задайте себе вопрос: «А смогу ли я сам сделать эту штуку?» И если ответ будет утвердительным, то не читайте дальше, а сразу отправляйтесь за подвигами в консоль! Если же отрицательный, то я бы хотел, чтобы вы медленно (с толком, с чувством и так далее) перечитали предыдущий абзац! Постарайтесь вникнуть в то, что там сказано. Если вы решите, что хотя бы часть проекта вы таки сможете сделать, то обязательно попытайтесь до того, как продолжить далее!

Ну а теперь приступим! Надеюсь, вы уже попробовали сделать хотя бы какую-то часть самостоятельно, так что теперь нам представится шанс сравнить вашу и мою версии требуемой программы! Итак, начнём, конечно же, с того, что создадим функцию, которая будет через `prompt()` запрашивать у клиента стоимость заказа:

```
function рекомендуемыеЧаевые() {
    const процентЧаевых = 18/100; // то же, что и 0.18
    const суммаЗаказа = prompt('Пожалуйста, укажите стоимость вашего
                               заказа.');
    const знаковПослеЗапятой = 2; // обычно в денежных операциях обходятся
                                  // двумя знаками после запятой
    const чаевые = (суммаЗаказа * процентЧаевых).
          toFixed(знаковПослеЗапятой);

    return 'Рекомендуемые чаевые для вашего заказа - $' + чаевые;
}
рекомендуемыеЧаевые();
рекомендуемыеЧаевые();
рекомендуемыеЧаевые();
```

Вот проект из «Повторяй за мной» и выполнен! Теперь попробуйте свои силы с проектом «Сделай сам».

Сделай сам: калькулятор чаевых (переменных и непостоянных!)

Возьмите за основу функцию для калькулятора из «Повторяй за мной». Теперь ваша задача попробовать написать также калькулятор чаевых, но без жёстко фиксированных чаевых. Ваш усовершенствованный калькулятор будет предлагать клиентам указать как стоимость их заказа, так и размер чаевых, которые они

готовы оставить, в процентах от общей суммы; после этого калькулятор выведет результат на экран. Ну же — смелее, и вперёд!

Надеюсь, вы немного повеселились, выполняя все эти проекты. На самом деле рабочие будни профессионального программиста очень похожи на то, чем мы занимались в этой главе; конечно, всё более сложно, но часто и более весело. Но так или иначе, сам подход к решению проблем примерно такой же. Кроме того, разработчики зачастую трудятся в командах, что делает всё это ещё более увлекательным и весёлым делом… а ещё кроме этого — программистам ну очень хорошо за это платят! А теперь — викторина!

ВИКТОРИНА

В этой главе викторина будет довольно короткой, так как нового у нас было немного. Но всё же: не забывайте, пожалуйста, заносить свои ответы в Рабочую тетрадь, а после того, как закончите, — проверить их.

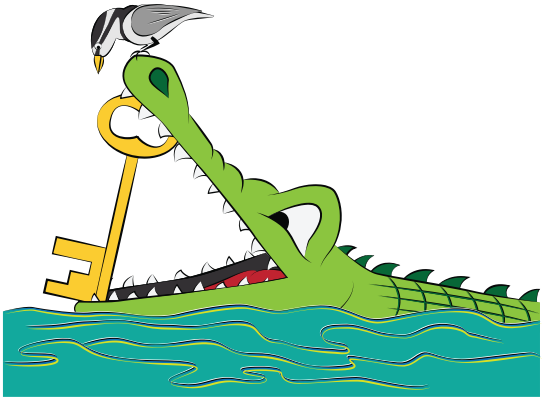
1. Истина/Ложь: объект в JavaScript — это набор свойств (то есть именованных значений).
2. При помощи какого метода (функции) можно изменить все буквы в строке на прописные?
3. При помощи какого ключевого слова вы можете создать переменную, которая никогда не будет изменяться?
4. Назовите термин, которым обозначают параллельные пояснения и комментарии по использованию и действию кода.
5. Когда стоит использовать в коде ключевое слово `let`?
6. Какие встроенные функции будут ожидать от пользователя нажатия на кнопку, прежде чем позволят интерпретатору продолжить работу с остальной частью кода?
7. Назовите термин, обозначающий код, в котором все функции и переменные обладают «говорящими» названиями (благодаря чему разработчику предельно просто взаимодействовать с написанным таким образом кодом).
8. При помощи какого метода (функции) можно изменить все буквы в строке на строчные?
9. Когда стоит использовать ключевое слово `var`?
10. Истина/Ложь: когда свойством объекта является функция, её называют «дочерней».
11. Какая встроенная функция будет ожидать от пользователя ввода информации, прежде чем позволит интерпретатору продолжить работу с остальной частью кода?

12. Какое ключевое слово вы будете использовать наиболее часто для создания переменных?
13. Истина/Ложь: функция `prompt()` всегда возвращает булево значение.
14. Истина/Ложь: при использовании в коде чисел для облегчения понимания зачастую лучше присваивать их к переменным с «говорящими» названиями.
15. Истина/Ложь: функции `alert()`, `confirm()` и `prompt()` применяются профессиональными разработчиками весьма широко, так как помогают создавать и улучшать комфортный и красивый пользовательский интерфейс.

КЛЮЧЕВЫЕ ЭПИЗОДЫ

Пробегитесь по списку, и если что-либо в нём вызовет у вас сомнения, обязательно вернитесь назад и повторите!

- Объекты.
- Свойства.
- Методы.
- `toUpperCase()`.
- `toLowercase()`.
- `const`.
- Документирование.
- Самодокументированный код.
- `confirm()`.
- `prompt()`.
- `toFixed()`.



УПРАЖНЕНИЯ

I. Введите следующие правильные фрагменты кода в консоль.

На этот раз никаких особых сюрпризов с ответами интерпретатора не будет. Это просто примеры кода для закрепления материала, чтобы вы его прописали, подумали над ним, поняли и двигали дальше.

```
1. const литровВГаллоне = 3.7854; // это — постоянная величина (то есть
                                  // неизменяемая)
   литровВГаллоне.toFixed(1);

2. "иМяПользОваТеляНапиСанНоЕВотТАкВоТКоЕкАк".toLowerCase();

3. let номинальнаяСтоимость = 1.5999; // здесь используется let, так как
                                      // значение может меняться
   if (confirm('Округлить значение?')) {
       номинальнаяСтоимость = номинальнаяСтоимость.toFixed(2); // видите?
                                                               // вот и поменялось.
   }
   console.log('Номинальная стоимость - $' + номинальнаяСтоимость);

4. const примерноеРасстояниеДоЛуны = 384400; // километров

5. const вашеИмя = prompt("Простите, не расслышал, как вас зовут?");
   console.log("Рад снова вас видеть, " + вашеИмя + '!');

6. const любаяНеизменяемаяВеличина = 'должна объявляться при помощи
                                     const!';

7. (18.5).toFixed();

8. 'патентное бюро сша'.toUpperCase();

9. let любаяИзменяемаяВеличина = 'должна объявляться при помощи ключевого
                                 слова';
   любаяИзменяемаяВеличина += ' let!';
   любаяИзменяемаяВеличина;

10. const полнаяОчистка = confirm('Провести ПОЛНУЮ очистку?');
    if (полнаяОчистка) {
        console.log('Всё вычищено и блестит!');
    } else {
        console.log('Нет, оставим всё как есть.');
    }

11. /* обратите внимание на самодокументирование:
    имена всех переменных говорят сами за себя  */
    const стоимостьБезУчётаНалога = 17.99;
    const налоговаяСтавка = 0.08;
    const налог = стоимостьБезУчётаНалога * налоговаяСтавка;
    const стоимостьВместеСНалогом =  стоимостьБезУчётаНалога + налог;
    const округлённаяСтоимость = стоимостьВместеСНалогом.toFixed(2);
    if (confirm('Вы готовы заплатить указанную стоимость? ($' +
                округлённаяСтоимость + ')')) {
        console.log('Благодарим за покупку!');
    } else {
        console.log("Что ж, тогда мы отменим ваш заказ.");
    }

12. function можноЛиПожатьВамРуку() {
        let можноЛиПожать = false; // let — потому что всё может
                                   // измениться
```

```
        if (confirm('Я вам доверяю?')) {
            if (confirm('Вы себя хорошо чувствуете?')) {
                можноЛиПожать = true;
        } else {
                const болезнь = prompt('Чем вы болеете?');
                /* обратите внимание на НЕ (!) далее...
                   благодаря чему подтверждается ОБРАТНОЕ
                   (НЕ заразность). */
                if (!confirm('А ' + болезнь + ' — это заразно?')) {
                    можноЛиПожать = true;
                } // здесь не требуется else, поскольку можноЛиПожать уже
                  // ложно (false)
            }
        }
        if (можноЛиПожать) {
            return "Конечно! Вот вам моя рука!";
        } else {
            return "Пожалуй, пока нам лучше быть более сдержанными.";
        }
    }

13. можноЛиПожатьВамРуку();

14. можноЛиПожатьВамРуку(); // проиграйте разные сценарии

15. можноЛиПожатьВамРуку(); // проиграйте разные сценарии

16. можноЛиПожатьВамРуку(); // проиграйте разные сценарии
```

II. Что не так с каждым из фрагментов?

```
1. const времяДня = '1:30';
   времяДня = '2:15'z;

2. let деньРожденияАвраамаЛинкольна  = '1809-02-12';
   var деньРожденияДжорджаВашингтона = '1732-02-22';

3. const годРождения = confirm('В каком году вы родились?');

4. const всеЗаглавныеБуквы = 'все с капс'.toCapitalized();

5. const точныйГрадусУглаДляЗапускаРакетыВКосмос = 88.726484293724;
   const ещёБолееТочныйУгол = точныйГрадусУглаДляЗапускаРакетыВКосмос.
         toFixed();

6. const x = 365; // под "x" подразумевается количество дней в году;
   const y = 7;   // теперь "y" мы будем принимать за количество дней
                  // в неделе;
   const z = 24;  // давайте под "z" понимать количество часов в сутках;

7. if (prompt('Вы уверены?')) {
       console.log('Отлично!');
   }
```

КОМПЛЕКСНЫЙ ОБЗОР

1. Если для работы условного выражения нам необходимы небулевы данные, которые мы всё же считаем `false` или `true`, то такие данные мы называем соответственно _____ и _____.
2. Какой символ следует добавить в следующее предложение, чтобы избежать синтаксической ошибки?

   ```
   const старая песня Эрика Клэптона = 'I Can't Stand It';
   ```

3. Что означает DRY (принцип сухости) применительно к программированию?
4. Является ли следующее предложение правильным (и если нет — почему)?

   ```
   const ниндзя всегда = 'ходит' !== 'шумно';
   ```

5. При помощи какой функции вы бы могли написать код, который бы выбирал случайное число в диапазоне от 0 до 20 и округлял бы его до двух знаков после запятой?
6. Выполняет ли следующий код замысел разработчика (и если нет — почему)?

   ```
   const секунд В Минуте = 60;
   ```

7. Истина/Ложь: блок комментариев указывает интерпретатору JavaScript, что нужно игнорировать всё, начиная с начала блока и до конца строчки.
8. Какой это тип данных?

   ```
   'false'
   ```

9. Назовите примитивный тип данных, который указывает, что никакого значения присвоено не было.
10. В каком стиле следует прописывать названия функций в JavaScript?
11. Истина/Ложь: чтобы строка, в которой есть апострофы, не выдала ошибки, следует заключить её в двойные кавычки.
12. Является ли следующий код правильным (и если нет — почему)?

    ```
    const любимая Буква = 'К';
    любимая Буква = "Р"; // передумал
    ```

13. Какого результата ожидает разработчик, использующий оператор modulo?
14. Назовите значение переменной `чем Бы Заняться` после запуска следующего предложения:

    ```
    const чем Бы Заняться = 'сделать домашку' || 'поесть чипсов' ||
    'поиграть в футбол';
    ```

15. Истина/Ложь: сообщения об ошибках предназначены для людей, чтобы они смогли придумать решение возникшей проблемы.

16. Что означает один знак равенства в предложении?

17. Истина/Ложь: разработчики часто пренебрегают функциями `alert()`, `confirm()` и `prompt()`, так как они блокируют работу всего остального кода, пока не получат реакцию от пользователя.

18. Истина/Ложь: пользователи зачастую находят поведение функций `alert()`, `confirm()` и `prompt()` навязчивым и раздражающим, так что разработчики предпочитают применять другие способы для достижения тех же результатов.

19. Назовите три ключевых слова, которыми можно объявить новую переменную.

20. Как можно создать новую строчку в ответе консоли (а также в `alert()`, `confirm()` и `prompt()`)?

21. Как и операторы сравнения, _____ операторы (`&&`, `||` и `!`) прекрасно работают с условными выражениями.

22. Является ли следующий код правильным (если нужно, попробуйте в консоли!) и если нет — почему?

```
function можноЛиУжеВДетскийСадик(возрастРебёнка) {
    const минВозраст = 5;
    if (возраст > минВозраст) {
        console.log('Уже можно!');
    } else {
        console.log('Ещё рано!');
    }
}
можноЛиУжеВДетскийСадик(5);
```

23. Является ли следующий код правильным (и если нет — почему)?

```
function вДенежномИсчислении(число) {
    return число.toFixed(2);
    console.log('В деньгах это будет $' + число.toFixed(2));
}
вДенежномИсчислении(10 / 3);
```

24. Что нужно ввести в адресной строке браузера, чтобы открыть пустую вкладку?

25. Chrome, Internet Explorer, Edge, Firefox и Safari — это _____.

26. Истина/Ложь: `122 >= 122 && !(67 <= 68);`

27. Назовите термин, обозначающий код, в котором все функции и переменные обладают «говорящими» названиями (благодаря чему разработчику предельно просто взаимодействовать с написанным таким образом кодом).

28. Каким символом обозначается логический оператор, который всегда возвращает булево значение (причём если он стоит перед «ложкой», то вернёт правду, а если же перед «правкой», то — ложь!)?
29. Значение `null`, `false`, `' '`, `0` и `undefined` — _____; а значение `true`, `'string'` и `1` — _____.
30. Истина/Ложь: операторы сравнения всегда возвращают булевы значения.

Сделай сам: «переводчик» оценок!

Напишите функцию, которая бы запрашивала у пользователя оценку по 100-балльной шкале и возвращала бы в ответ букву (соответствующую 5-балльной шкале). Пользователь должен ввести числовое значение от 0 до 100; если значение больше или равно 90, то следует вернуть «A»; если больше или равно 80, то «B»; если больше или равно 70, то «C»; больше или равно 60 — «D». Если введено любое другое значение, тогда следует вернуть оценку «F» (то есть «двойку»). Уверен, вы сможете справиться самостоятельно, но если у вас всё же возникли трудности — обязательно обратитесь к моему совету в конце книжки. Удачи!

8 Массив, массив! Да здравствует массив!

Ну как, весело было выполнять все эти проекты? Конечно, ведь проекты — отличный способ проверить и закрепить полученные навыки.

В этой главе мы будем учиться работать со списками всяких всякостей (это должен уметь каждый уважающий себя разработчик!). Вперёд!

Ах, да! Чуть не забыл: закройте все программы, откройте Chrome, пустую вкладку и запустите консоль (если забыли, как и что, — загляните в главу 1).

Готово? Отлично! Вот теперь — вперёд!

Массивы: основная информация

Что же такое «массив»?

Так, слушайте внимательно — штука очень важная! **Массив** в JavaScript (да и в любом другом языке программирования) — это такой список значений, указанных в определённом порядке. Массив умеет хранить множество переменных… всего в одной.

Вот пример массива (набирайте вместе со мной):

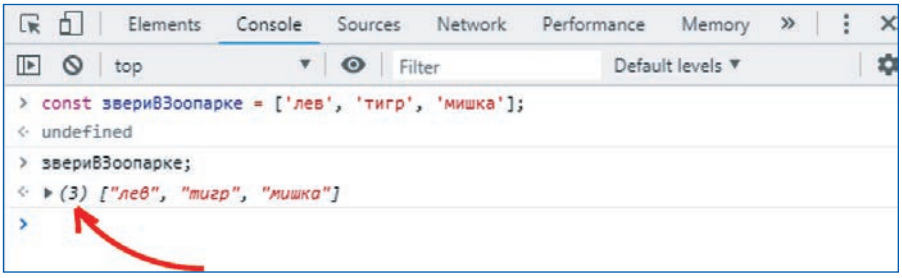
```
const звериВЗоопарке = ['лев', 'тигр', 'мишка'];
```

Имя переменной для массива зачастую стоит во множественном числе, поскольку массив содержит несколько значений. Теперь, создав массив, его можно вызвать, набрав в консоли его имя:

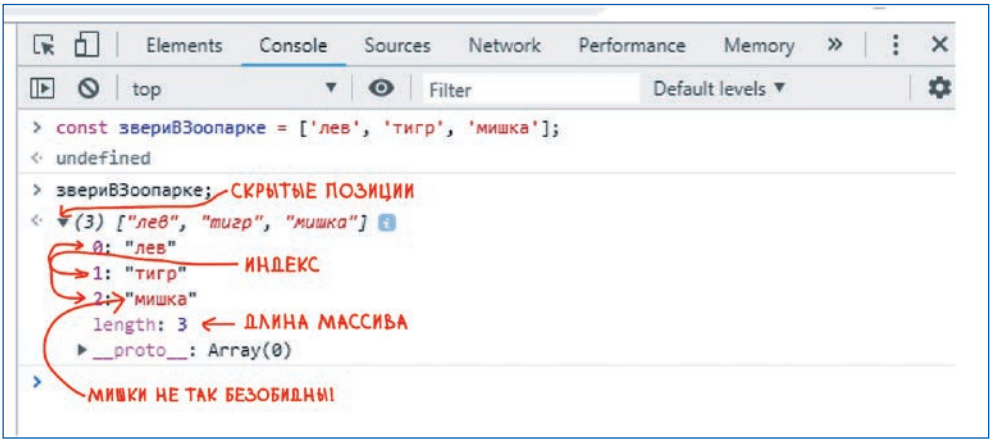
```
звериВЗоопарке;
```

Ответ: `(3) ['лев', 'тигр', 'мишка']`

Остановимся на этом ответе чуть подробнее и проанализируем его. Сперва в нём стоит `3`, да ещё и в скобках — `(3)`. Угадайте, почему `3`? Догадались? Ну, замечательно! И почему же? Потому что у трёхколёсного велосипеда три колеса?! Э-э-э… Ну, *технически* вы, конечно, правы — колёс там и правда три, но всё же это неверный ответ. `(3)` *в нашем примере указывает на длину* (параметр `length`), то есть на количество пунктов (значений) в массиве. Звучит более правдоподобно, чем трёхколёсный велосипед, да?



Присмотритесь! Видите маленький треугольничек рядом с `(3)`? Он похож на кнопку PLAY на DVD-проигрывателе… В смысле «что такое DVD-проигрыватель»? Ну-у… это такой древний аппарат, куда вставлялся диск, на котором помещался только один фильм… Словом, спросите родителей как-нибудь, они вам расскажут.

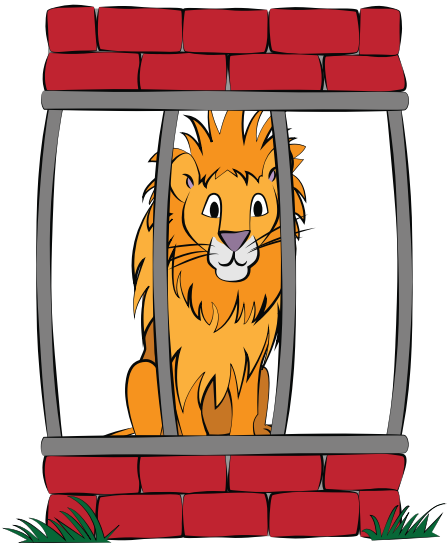


Нажмите на этот треугольник. Он теперь развернулся вниз! И весь массив также выстроился вертикально. Как видите, все элементы теперь пронумерованы — это так называемый индекс массива. **Индекс** указывает на позицию того или иного элемента в массиве. И обратите внимание — список начинается с 0, а не с 1! Так что то, что мы бы назвали первым в этом списке, на самом деле расположено на 0-й позиции. Вы можете видеть, что в качестве свойства здесь также указана длина массива — `length`. Пожалуйста, запомните всё это, поскольку мы к этому ещё не раз вернёмся.

В случае с нашими `зверямиВЗоопарке` все элементы массива были строками, однако это вовсе не обязательно. Элементами массива могут быть данные любого типа или даже целый микс из типов. Вот, к примеру, такой массив:

```
const булевыЗначения = [true, false];
const простыеЧисла = [2, 3, 5, 7, 11, 13];
const всеЛожки = [ // "сборная солянка" из типов (вдобавок тут ещё
                   // и несколько строчек)
false,     // булев тип
0,         // число
null,      // null
" ",       // строка (пустая)
undefined // undefined
];
const массивСМассивом = [ // массив, внутри которого… другой массив!
                          // Круто, да?
['первый', 'внутренний', 'массив'],
['второй', 'внутренний', 'массив'],
['внутренний', 'массив', ['который', 'ещё','глубже']]
];
```

Как обратиться к элементу массива?



Как только вы создали нужный вам массив, можете обращаться к любому из его элементов. Для этого вам понадобятся квадратные скобки с индексом нужного элемента (помните, я показывал на скриншоте?). Попробуйте:

```
звериВЗоопарке[0]; // лев
звериВЗоопарке[1]; // тигр
звериВЗоопарке[2]; // мишка
```

Какое там у нас было третье `простоеЧисло` (помните, что «третье» будет «2» по индексу, потому что всё начинается с `0`)?

```
простыеЧисла[2]; // третье число в массиве имеет индекс 2
простыеЧисла[5]; // под индексом 5 идёт шестое число в ряду
```

В общем-то, в квадратные скобки массива можно заключить даже переменную, если её значение — это целое число (без всяких дробных частей), которое больше или равно нулю:

```
let индекс = 2; // let — потому что мы собираемся вскоре изменить значение
                // переменной
звериВЗоопарке[индекс]; // мишка
индекс = 0;
звериВЗоопарке[индекс]; // лев
```

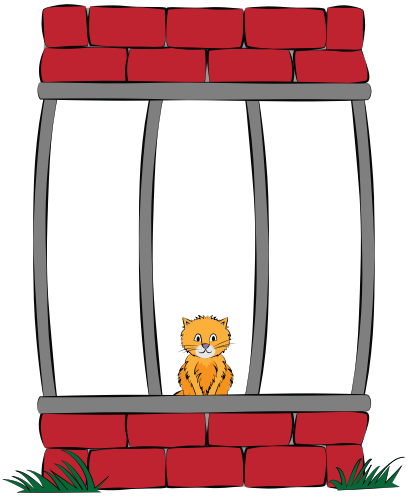
Вот таким образом мы и обращаемся к любому нужному элементу массива. Надеюсь, вы поняли, как это делается. Если нет, — пожалуйста, перечитайте последние пару страниц, прежде чем двигаться далее. Это и правда важно.

Как изменить элементы в массиве?

Если вы внимательно следили за всем происходящим, то, вероятно, должны были заметить, что наш массив со `зверямиВЗоопарке` был создан при помощи `const`. Значит, этот массив останется навсегда неизменным, правильно? Конечно, правильно. Как же иначе?

И кстати. Хотите покажу забавный трюк? Давайте набирайте в консоль:

```
звериВЗоопарке[0] = 'котёнок';
звериВЗоопарке;
```



Что?! Как же так? Массив изменился! Но мы же были уверены, что наш массив `звериВЗоопарке` неизменный, константный и всякий там постоянный… Мы же даже использовали `const` и всё такое… Да что вообще происходит? Как это понимать надо? Как теперь быть?! Всё с ног на голову и кверху тормашками!

Так, во-первых, возьмите себя в руки. Не нужно так нервничать! Слушайте внимательно: константа `звериВЗоопарке` прикреплена к некоторому массиву. Изменить значение константы невозможно. Попробуйте это сделать — и получите именно ту ошибку, которую ожидали от меня в прошлый раз:

```
звериВЗоопарке = ['другой','массив'];
```

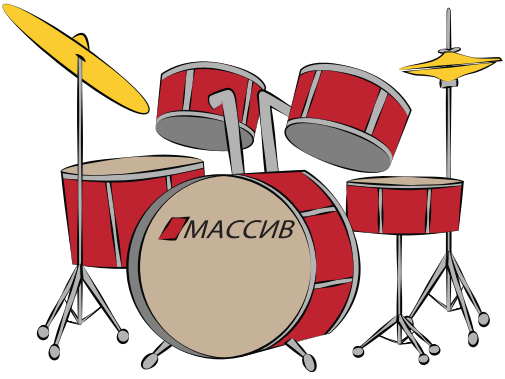
Ответ: `Uncaught TypeError: Assignment to constant variable.`

`звериВЗоопарке` накрепко и навсегда связаны с этим массивом. Всё, что мы сделали, — это изменили один из *элементов* этого массива. Словом, пока вы не пытаетесь присвоить иное, чем уже есть, значение переменной `звериВЗоопарке` — всё было в порядке. На самом деле можно всё завернуть ещё круче, но об этом — чуть позже!

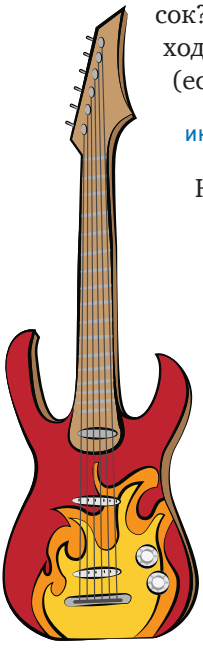
Как посчитать элементы в массиве?

Давайте создадим новый массив, чтобы поделать с ним всякие забавные штуки!

```
const инструментыДляРокГруппы = [
'гитара',
'барабаны',
'бас'
];
```



Таким образом, у нас в массиве три элемента, верно? Верно. И их можно очень легко посчитать. А что, если бы мы не смогли столь же легко посчитать элементы в другом массиве? Скажем, если бы в нём было несколько сотен или тысяч элементов? Откуда нам знать, насколько длинным должен быть такой список? Именно здесь и выходит на сцену маленькое, но крутое встроенное свойство `length` (если не помните, взгляните ещё раз на тот скриншот):



```
инструментыДляРокГруппы.length // 3
```

Как вы, я надеюсь, прекрасно помните из предыдущей главы, эта маленькая точка, разделяющая объект (в данном случае массив) и какое-то слово, указывает, что это слово является свойством (то есть «именованным значением, прикреплённым к объекту»). А ещё вы можете помнить и само свойство `length` (иногда пишут `.length`) — при его содействии мы узнавали количество символов в строке. Ну а теперь, при работе с массивами, он подсказывает нам количество элементов в них.

Кстати, у массивов, так же как и у строк, есть ряд встроенных свойств (то есть функций). Вызываются они, как обычно, со скобками`()`. Когда свойство является функцией, его называют… Помните, как его называют? Нет? Ну ладно. На этот раз я подскажу, но впредь — пожалуйста, не забывайте таких вещей! Это важно! Функция, которая

«прикреплена» к объекту (строке, массиву и так далее), называется *методом*.

Как добавить элементы в массив?

Мы уже видели, каким образом можно изменить какой-нибудь из элементов массива:

```
инструментыДляРокГруппы[2] = 'бас-гитара';
```

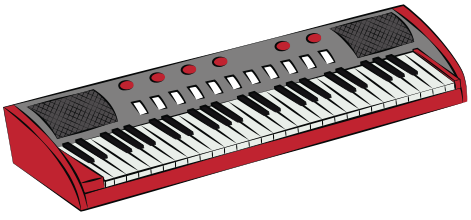
А вот так можно легко и непринуждённо добавлять элементы в список:

```
инструментыДляРокГруппы[3] = 'клавиши';
инструментыДляРокГруппы.length;
инструментыДляРокГруппы;
```

Как видите, мы добавили в наш массив четвёртый элемент, просто-напросто присвоив новую строку позиции 3 в списке!

А вот ещё один отличный способ добавить в массив новый элемент; для этого нам потребуется *метод* `.push()` — как вы помните, мы знаем, что это *метод* (то есть функция), благодаря наличию скобок.

```
инструментыДляРокГруппы.push('микрофон');
                        инструментыДляРокГруппы;
```



Этот метод хорош ещё и тем, что вам необязательно знать, какой индекс следует присвоить новому элементу; `.push()` просто добавит его в конец списка!

А как удалить элемент из массива?

Схожим образом, как мы добавляли элементы в список при помощи метода `.push()`, их можно оттуда и удалить — при помощи метода `.pop()`. Этот метод запросто удалит последний элемент из списка (соответственно, уменьшая его длину — `length` — на один пункт):

```
инструментыДляРокГруппы.length; // 5 элементов
инструментыДляРокГруппы.pop();  // вернётся последний элемент — микрофон
инструментыДляРокГруппы.length; // теперь в списке только 4 элемента
```

О методе `.pop()` также стоит знать и то, что у него есть *возвращаемое* значение. Это означает, что этот метод не только поможет вам уменьшить массив на его последний элемент, но также и *вернёт* в консоль этот последний элемент. Когда какая-либо функция обладает возвращаемым значением, вы можете присвоить это значение переменной:

```
инструментыДляРокГруппы.length; // в нашем списке 4 инструмента
const анджелаБудетИгратьНа = инструментыДляРокГруппы.pop(); // присвоение
                                                            // клавиш
const нэяБудетИгратьНа = инструментыДляРокГруппы.pop(); // бас-гитары
const тониБудетИгратьНа = инструментыДляРокГруппы.pop(); // барабанов
const аЯбудуИгратьНа = инструментыДляРокГруппы.pop(); // и наконец
                                                      // соло-гитары
инструментыДляРокГруппы.length; // в массиве осталось 0 инструментов
console.log("У нас в группе четыре человека: Анджела(" +
анджелаБудетИгратьНа + "), "
    + "Нэя (" + нэяБудетИгратьНа + "), "
    + "Тони (" + тониБудетИгратьНа + "), "
    + "и я (" + аЯбудуИгратьНа + "). Ну что, вдарим рок в этой дыре?!"); [1]
```

Массивы: всякие классные штуки

А с массивами можно ещё что-нибудь делать?

Шутите? Да с массивами можно делать *уйму* всего! Не хочу надолго увязнуть в объяснениях и всевозможных деталях, так что просто приведу пару примеров.

Мы уже разобрали функцию `.pop()`, которая может убирать элемент из *конца* списка. А с помощью `.shift()` можно убрать элемент и из его *начала*.

```
// здесь используется let, потому что мы планируем изменить значение
// переменной
let корзинаСБельём = ['брюки', 'футболка', 'трусы', 'носки'];
const можноЕщёПоносить = корзинаСБельём.shift(); //  чистые!
console.log('Как эти ' + можноЕщёПоносить + " сюда попали? Ведь они
             чистые!");
корзинаСБельём; // обратите внимание, что первого элемента в списке
                // больше нет
```

Или, скажем, помните, мы при помощи `.push()` добавляли в *конец* списка ещё один элемент? Так вот, используя `.unshift()`, можно добавить элемент и в *начало* списка:

```
корзинаСБельём.unshift('полотенце');
корзинаСБельём;
```

[1] Участники группы — дети Джереми (автора книги). Если вас интересует полный список имен, то его можно найти в разделе «Ответы», в рекомендациях для «Сделай сам» главы 1.

Итак:

`.push(новыйЭлемент)` — *добавит* новый элемент в *конец* списка;

`.pop()` — *уберёт* его оттуда (из *конца*), а также вернёт ответом в консоль;

`.unshift(новыйЭлемент)` — *добавит* новый элемент в *начало* списка;

`shift()` — *уберёт* его оттуда (из *начала*), а также вернёт ответом в консоль.

Очень полезна бывает функция `.toString()`: с её помощью можно легко и просто превратить массив в строку, в которой через запятую перечислены все его элементы:

```
console.log("В моей корзине с бельём находятся " +
корзинаСБельём.toString());
```

Если же вам нужно сделать из массива строку, но вы не хотите, чтобы разделителем (строкой, которая разделяет все элементы) была запятая, то можно использовать `.join()` вместе с нужным вам разделителем в качестве аргумента:

```
console.log("В моей корзине с бельём находятся " +
корзинаСБельём.join(" и ") + ".");
```

Как только ваш массив превратился в строку, к нему, понятное дело, можно применить весь известный набор инструментов обращения со строками: `.repeat()`, `.toUpperCase()`, `.toLowerCase()` и так далее.

Попутно проясню один момент. Когда я говорю, что при помощи метода `.join()` массив превратился в строку, всё же нужно себя поправить: `.join()` ничего не «превращает», а переменная `корзинаСБельём` всё так же хранит массив. Что этот метод делает, так это создаёт и возвращает новую строку. И эту новую строку мы уже можем присвоить к переменной или вообще использовать, как нам заблагорассудится. Звучит, быть может, излишне занудно и педантично, но порой очень важно понимать такие мелочи, когда пишешь собственный код.

Отлично! Что-нибудь ещё?

Можно «склеить» между собой два массива при помощи метода `.concat()`:

```
const вчерашняяСтирка = ['толстовка', 'вчерашние трусы', 'полосатые
                         носки', 'бандана'];
корзинаСБельём.concat(вчерашняяСтирка); // какой длинный получился массив!
корзинаСБельём; // Хмм… всё осталось на своих местах?!
```

Ага, попались! Ведь `корзинаСБельём` всё-таки не *изменилась* из-за метода `.concat()`! Он всего лишь *вернул* новый массив, слепленный из двух, но мы с ним так ничего и не сделали, так что они оба остались без изменений. Помните, мы объявляли нашу `корзинуСБельём` через `let`, а не через `const`? Вот зачем. Давайте на самом деле добавим белья в нашу корзину при помощи метода `.concat()`:

```
корзинаСБельём = корзинаСБельём.concat(вчерашняяСтирка); // а вот и новое
                                                          // значение!
корзинаСБельём; // весомое прибавление, правда?
```

Всё понятно? Надеюсь, вы не запутались. Но если всё же что-то упустили, то обязательно перечитайте этот абзац ещё раз.

Кстати говоря, тут приходила ваша матушка, и выяснилось, что у неё есть особый… эээ… пунктик (так это называется, да?) по поводу корзины для стирки: всё бельё должно быть обязательно уложено в алфавитном порядке! В этом нам поможет метод `.sort()`. В отличие от `.concat()`, он таки на самом деле изменяет массив (и вдобавок возвращает в ответе изменение):

```
корзинаСБельём.sort();
console.log("Мам, я всё разложил. Вот: " + корзинаСБельём.join(', '));
```

Кстати, стоит заметить, что отсортированный массив ведёт себя ровно так же, как и любой другой. Поэтому вы можете, например, обратиться к какому-нибудь элементу по индексу и так далее:

```
корзинаСБельём.sort()[0]; // вернётся 1-й элемент из отсортированного
                          // массива
корзинаСБельём.sort()[3]; // вернётся 4-й элемент из отсортированного
                          // массива
```

Ну что, всё?

Нет, не всё! Можно ещё сделать новый массив из любой строки при помощи метода `.split()`. Например, если у вас есть некоторая строка, слова в которой отделены друг от друга звёздочками (`'моя*звёздочная*строка'`), и вам требуется сделать из всех этих слов массив (`['моя','звёздочная','строка']`), то вы просто можете применить `.split('*')` — со звёздочкой в аргументе. Интерпретатор JavaScript «сделает» (и вернёт в консоль) из этой строки массив, основываясь на том, что там, где есть звёздочка, кончается один элемент и начинается другой.

Попробуйте ввести в консоль следующие примеры (и обратите внимание на ответы интерпретатора!):

```
"моя*звёздочная*строка".split('*');
"строка, всегда желавшая быть массивом".split(', ');   // запятая означает
                                                        // новый элемент
"сейчас сделаем все эти слова элементами массива".split(' '); // в скобках
                 // указан пробел, а значит, после пробела — новый элемент
"Леонардо | Донателло | Рафаэль | Микеланджело".split(' | ');
```

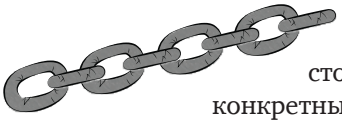
А вот, пожалуй, самое полезное применение метода `.split()`: задайте в аргумент пустую строку — (`''`) и тогда интерпретатор сам разобьёт вашу строку на отдельные символы! Вот смотрите:

```
'АБВГДЕЁЖЗИК'.split(''); // и вот вам массив из отдельных символов!
```

К слову о полезном! Покажу вам ещё один очень полезный приём под названием «**цепочка**». Цепочкой называют последовательность методов (функций), которые находятся все в одной строчке кода и вызываются один за другим. Попробуйте сами:

```
/* переведём строку 'взагжбдкеи' в верхний регистр // 'ВЗАГЖБДКЕИ'
* сделаем из строки массив с символами// ['В', 'З', 'А', 'Г', 'Ж', 'Б',
                                          'Д', 'К', 'Е', 'И',]
* отсортируем всё это в алфавитном порядке // ['А', 'Б', 'В', 'Г', 'Д',
                                               'Е', 'Ё', 'Ж', 'З', 'И', 'К']
* теперь слепим обратно через дефис // "А-Б-В-Г-Д-Е-Ж-З-И-К"
* и это всё — за одну строчку кода! Вот это да!
*/
'взагёжбдкеи'.toUpperCase().split('').sort().join('-'); // "А-Б-В-Г-Д-Е-Ё-
Ж-З-И-К"
```

Ну что, круто? А теперь, пожалуй, самое время спросить себя: «А почему это вообще так работает?» Это отличный вопрос, который вам сослужит весьма добрую службу в будущем. Понимание, почему что-то вроде этих цепочек работает так, а не иначе, очень важно, потому что неизменно влечёт за собой творческий и в целом — профессиональный рост. Однако для того, чтобы достичь общего понимания принципов работы некой системы, необходимо сперва понять принципы работы её составляющих (в нашем случае — методов, составляющих нашу цепочку).



Предположим, я никогда не упоминал ранее о цепочках; но вы всё равно могли бы прийти к этой идее, просто попутно схватывая происходящее! Давайте разберём конкретный пример. Следите внимательно!

Учительница просит вас помочь ей с оценками вашего класса за последний тест. Она хочет получить общую и структурированную картину результатов класса. Сейчас у неё есть цепочка вида `"c,a,f,b,c,d,f,c,b,a,f,d,f,c,d,a,f,d"`, но ей бы хотелось, чтобы она приняла вид `"A | A | A | B | B | C`[1] `… и так далее"`.

Такая работа обещает быть весьма муторной, не правда ли? Но что делать. Потратим пару минут, возьмём ручку, листок бумаги и всё сделаем… Просто нужно быть предельно аккуратным, когда считаешь все эти оценки, группируешь их и всё такое… Но постойте-ка! Оказывается, что это не разовая работа! Учительница говорит, что теперь каждую пятницу вам предстоит сортировать классную

[1] В школах США принята буквенная система оценки успеваемости учащихся. — *Примеч. пер.*

успеваемость за неделю. Помимо нее с точно таким же заданием к вам обратились ещё пятеро учителей из параллельных классов. И очень-очень важно, чтобы вся работа была проделана безошибочно! Э-эх, нудная работа на глазах превращается в прямо-таки каторгу!

К счастью (если вы внимательно следили за тем, чему я тут учил), вы уже вполне можете написать простенькую строчку кода с цепочкой методов, которые сделают всю эту работу! Что ж, давайте так и поступим. Значит, сперва мы хотим все прописные буквы сделать строчными, так? С этим всё просто:

```
const оценки = 'c,a,f,b,c,d,f,c,b,a,f,d,f,c,d,a,f,d';
оценки.toUpperCase(); // "C,A,F,B,C,D,F,C,B,A,F,D,F,C,D,A,F,D"
```

Давайте тут остановимся на секунду. Подумайте, что именно сейчас сделал метод `.toUpperCase()`? Вы бы, пожалуй, ответили: «Он изменил регистр оценок… что тут думать-то? Ответили бы и оказались неправы. Ведь оценки — константа (помните? — `const`), и они *не могут* быть изменены! Так что вместо того, чтобы изменить что-то в оценках, метод `.toUpperCase()` просто *вернул новую* строку!

Ну и почему же это так важно? А потому, что как только у нас есть новая строка, мы можем на ней запустить любые, какие нам только в голову придут, методы. Так что оба следующих метода должны произвести одинаковый эффект:

```
'C,A,F,B,C,D,F,C,B,A,F,D,F,C,D,A,F,D'.split(','); // вернёт то же самое,
                                                  // что и...
оценки.toUpperCase().split(',');
```

Ответ: `["C", "A", "F", "B", "C", "D", "F", "C", "B", "A", "F", "D", "F", "C", "D", "A", "F", "D"]`

И снова подумайте: «Если `.toUpperCase()` *всегда возвращает новую строку*, то что же делает `.split()`?»

Если вы подумали: «Он всегда возвращает новый *массив*!», то отныне вы — гордость всего класса (…и должны будете неловко стоять у всех на виду, пока учительница не сочтёт, что на вас уже достаточно насмотрелись, и не отправит вас на место). Вы совершенно правы! Метод `.split()` всегда возвращает новый массив. А раз у вас новый массив, то к нему можно применить любые известные вам методы или свойства. Например, такие:

```
оценки.toUpperCase().split(',').length; // вернёт количество элементов
                                        // в массиве
оценки.toUpperCase().split(',')[0];     // покажет самый первый
оценки.toUpperCase().split(',')[9];     // десятый
оценки.toUpperCase().split(',').toString(); // обратит в строку
оценки.toUpperCase().split(',').push('K');  // добавит новый элемент
оценки.toUpperCase().split(',').pop(); // удалит и вернёт в консоль
                                       // последний элемент
```

Словом, можно много чего делать с массивами… а что там от нас хотела учительница? А, чтобы оценки были сгруппированы! Помните, каким образом можно отсортировать элементы в массиве?

```
оценки.toUpperCase().split(',').sort(); // в алфавитном порядке
```

Поскольку `.sort()` также возвращает новый массив, у нас появился ещё один массив, элементы которого расставлены в алфавитном порядке. Всё, что нам осталось сделать, это сделать из этого массива вновь строку, а также сделать разделителем « | ». Помните, как это делается?

```
оценки.toUpperCase().split(',').sort().join(' | ');
```

Ответ: `"A | A | A | B | B | C | C | C | C | D | D | D | D | F | F | F | F | F"`

Ну вот! Так легко и просто мы решили все проблемы с оценками! Можно даже завернуть всё это в функцию, чтобы все учителя, когда им только нужно будет, могли бы ею воспользоваться:

```
function форматОценок(всеОценки) {
    return всеОценки.toUpperCase().split(',').sort().join(' | ');
}
форматОценок('d,b,b,a,a,c,f,d,b,f,a,b,b,c,a,d'); // класс И. Нитонисёнова
форматОценок('a,b,a,a,c,a,a,a,b,a,b,a,a,b,b,a,a'); // класс А. Лодырникова
форматОценок('c,d,f,b,f,d,f,c,f,d,d,f,f,c,d,b,f'); // класс док. мат. наук
                                                    // Д. Гранитнаукова
```

Благодаря цепочкам методов можно сделать уйму полезных вещей. В нашем коротком примере мы: использовали базовую строку для возврата новой строки (в верхнем регистре), из которой сделали массив, из которого сделали другой (отсортированный) массив, из которого сделали ещё одну строку! И всё это одной строчкой кода! Попробуйте сами в консоли следующие занятные цепочки:

```
'Сколько слов в этом предложении?'.split(' ').length
'1-2-3-'.repeat(6).split('-').sort().toString();
const предложения = 'Здесь два предложения. Сколько слов в каждом?';
предложения.split('. ')[0].split(' ').length; // количество слов
                                              // в 1 предложении
предложения.split('. ')[1].split(' ').length; // количество слов
                                              // во 2 предложении
'Она меня не любит... :-('.split('...')[0].concat(', а очень любит!').
                           toUpperCase();
```

Круто? Дальше — больше! С массивами ещё столько всего можно делать!

Ладно-ладно! Критическая масса массивов достигнута!

Хм-м… Ну что ж, мы недурно потрудились. Хотите верьте, хотите нет, но с массивами можно проделать ещё очень и очень *много* всяких полезных трюков. В настоящей главе нам едва показалась лишь верхушка айсберга под названием «массивы». Если честно, я даже не успел показать вам свой любимый способ работы с массивами, но, пожалуй, его стоит приберечь для следующей главы.

Мой одиннадцатилетний сын, помогавший мне тестировать код для этой книги, резонно заметил, что за последние пару страниц мы и так уже прошли ну очень много. Думаю, он совершенно прав, и это отличный шанс оглянуться назад, чтобы с уверенностью сказать, что мы всё помним и знаем, и пойти дальше с новыми силами. Пробегитесь по списку (пройденных нами) методов и спросите себя, все ли вам известны: `.push()`, `.pop()`, `.shift()`, `.unshift()`, `.toString()`, `.join()`, `.concat()`, `.sort()` и `.split()`?

Если всё так, то давайте сделаем вместе небольшой проект при помощи некоторых из них! Но прежде чем мы начнём — откройте новое окно консоли в `about:blank`!

Повторяй за мной: тренировки легкоатлетов

Настало время соревнований, и школьные легкоатлетические команды приступают к усиленным тренировкам! И догадайтесь, кому же был доверен учёт результатов пробных забегов? Ну конечно же, вам! Школьное руководство поручило вам вести табели забегов и заносить в них участников в порядке пересечения финишной черты. Как только кто-то окончил забег, он должен тут же быть отмечен в итоговом списке.

Не знаю, как вы, но чуть только я слышу слово «список», как сразу думаю о массивах. Так что с этого давайте и начнём — создадим пустой массив, в который будут попадать результаты забегов (обратите внимание: в этом проекте мы не будем создавать пользовательских функций):

```
let результатыЗабега = []; // let — потому что результаты будут постоянно
                           // меняться
```

Глядите-ка! Похоже, у нас есть первый финишировавший! Это Алан! Давайте отметим это в нашем списке!

```
результатыЗабега[0] = 'Алан';
```

О! А за ним целая четвёрка преследователей! И все финишировали одновременно! Давайте добавим их всех в одну графу (как здорово, что мы использовали `let`, верно?):

```
результатыЗабега = результатыЗабега.concat(['Бернардо', 'Сэсил', 'Деррек',
'Эмилио']);
```

И наконец, финишную черту пересекают и последние два участника — чуть живые Фредерик с Гордоном. Давайте добавим этих двух в конец нашего списка:

```
результатыЗабега.push ('Фредерик');
результатыЗабега.push ('Гордон');
```

Что? Сколько? Только что тренер объявил, что в команде есть место только для пятерых лучших бегунов. Сколько там у нас в списке?

```
console.log('В нашем списке сейчас ' + результатыЗабега.length + '
бегунов.');
```

Семь? Вот досада. Это разобьёт Фредерику сердце! Ведь он весь месяц тренировался! Но ничего личного, мы должны просто выполнить свою работу. Уберём наших бегунов с конца списка:

```
результатыЗабега.pop() // всё, Гордон больше не в команде (а ему какое
                       // дело? он всё равно бегать терпеть не может!)
                       // переходим к самому трудному...
```

Стойте! Погодите! Последние новости: похоже, Алан попался на допинге! Чёрт возьми, Алан, ну как же так? Алан теперь дисквалифицирован! А значит, надо убрать его из *начала* списка!

```
const удалённыйЧитер = результатыЗабега.shift();
```

Так, ну и сколько теперь у нас бегунов в команде?

```
console.log('В нашем списке сейчас ' + результатыЗабега.length +
            ' бегунов.');
```

Ровно пять! Фредерик прошёл! Ура-ура! Ну что, можно вывешивать список?

Упс… Похоже, ещё рано. В наш список закралась орфографическая ошибка: как выяснилось, одного из наших бегунов зовут вовсе не «Деррек», а «Деррик»! Хм. Как же нам это поправить? Ну, во-первых, нам необходимо выяснить, на каком месте в списке он находится. Раз Алан дисквалифицирован, то «Деррек» теперь становится третьим (раз массив начинается с 0, то третье место в нём — это 2-я позиция). Используя этот индекс, теперь мы можем просто исправить ошибку:

```
результатыЗабега[2] = 'Деррик';
```

Отлично! Всё готово! Можно вывешивать список:

```
console.log('Вот наша новая команда по бегу: ' + результатыЗабега.join
(', ') + '. Желаем удачного старта!');
```

Ну как вам? Всё ли было понятно? При изучении любого языка программирования есть некоторые базовые принципы и идеи, которые необходимо просто хорошо понять и уметь применить. Именно такой «несущей» идеей и являются массивы. Если вам кажется, что вы недостаточно хорошо уловили тот или иной момент, обязательно вернитесь на пару страниц назад и повторите материал (всё равно при повторном разборе времени уйдёт куда меньше, ведь все долгие штуки мы уже сделали!). Также весьма полезным будет просто «поиграть» в консоли: посоздавать свои массивы, проделать с ними какие-нибудь манипуляции, которым мы тут научились. Как будете готовы — переходите к нашей викторине.

ВИКТОРИНА

Выполняйте задания в Рабочей тетради, не подглядывая в ответы. Когда закончите — попросите папу или маму (ну или самостоятельно, если они заняты) проверить вас по ответам в конце. После этого пробегитесь по ошибкам и попробуйте их исправить.

1. Как в JavaScript называется список значений, расположенных в определённом порядке?
2. Какое действие осуществляет `.length()`?
3. _____ — это номер, под которым в массиве значится тот или иной элемент.
4. Какое значение нужно будет указать в квадратных скобках, чтобы получить в ответе первый элемент массива?

   ```
   мойМассив[   ];
   ```

5. Истина/Ложь: в один массив могут входить самые разные типы данных.
6. При помощи какого встроенного метода можно добавить новый элемент в конец массива?
7. Истина/Ложь: если к `моемуМассиву` применить метод `.toString()`, то в ответ я получу строку, включающую все элементы массива, перечисленные через точку с запятой.

8. Метод `.pop()` _____ и _____ последний элемент массива.

9. Истина/Ложь: после запуска следующего кода массив `учебники` будет иметь длину (`.length()`) 4.

```
const учебники = ['английский', 'биология'];
учебники.concat(['математика', 'история']);
```

10. Каким термином называют последовательное выполнение одного метода за другим, указанных в одной и той же строчке кода?

11. Дайте ответ, прежде чем проверить в консоли: какой будет результат запуска следующего кода:

```
const швейныеПринадлежности = ['нитка', 'иголка', 'напёрсток']
швейныеПринадлежности.sort()[1];
```

12. Истина/Ложь: если применить `мойМассив.join("\n")`, то строка `"\n"` здесь станет разделителем, то есть строкой, которая разделяет между собой элементы массива.

13. Метод `.unshift()` используется для добавления, а метод _____ — для удаления элемента из начала списка.

14. Какое значение вернёт следующий код:

```
const дляВолос = ['шампунь', 'лак', 'гель'];
const мнеБольшеВсегоНравится = 2;
дляВолос [мнеБольшеВсегоНравится];
```

15. Истина/Ложь: если запустить `мойМассив.join(' или ')`, то в консоль вернётся новый массив, включающий в себя все элементы старого, разделённые между собой строкой `' или '`.

16. Дайте ответ, прежде чем проверить в консоли: какой будет результат запуска следующего кода:

```
['Г', 'В', 'А', 'Б'].toString().toLowerCase().split(',').sort()[3];
```

17. Истина/Ложь: технически метод `.join()` не конвертирует массив в строку, но скорее создаёт новую строку, основываясь на порядке элементов в массиве.

18. Дайте ответ, прежде чем проверить в консоли: какой будет результат запуска следующего кода:

```
const мойБитбокс = 'сдоба , стоп , сдоба , стоп , сдоба , стоп';
мойБитбокс.split(' , ').sort()[2].length;
```

КЛЮЧЕВЫЕ ЭПИЗОДЫ

Пробегитесь по списку и обязательно повторите темы, которые покажутся вам непонятными или неизвестными.

- Массивы.
- `.length()`.
- Индекс.
- `.push()`.
- `.pop()`.
- `.shift()`.
- `.unshift()`.
- `.toString()`.
- `.join()`.
- Разделитель.
- `.concat()`.
- `.sort()`.
- `.split()`.
- Цепочки.

УПРАЖНЕНИЯ

I. Введите следующие правильные фрагменты кода в консоль.

Набирайте фрагменты в консоли, внимательно следя за синтаксисом (если позабыли, что это такое, загляните в глоссарий) выражений. Ваша задача — угадать ответы интерпретатора.

```
1. const моёУтро = [
       'просыпаюсь',
       'иду в душ',
       'одеваюсь',
       'чищу зубы'
   ];
2. моёУтро.shift(); // ну с этим и так всё понятно...
```

```
3. моёУтро.sort(); // эээм… а чего это у меня вся одежда мокрая
   alert("Вот что я делаю с утра: " + моёУтро.join(', потом ') + '.');

4. моёУтро.push('готовлю завтрак');

5. моёУтро[0] = 'иду в душ'; // давайте-ка поменяем эти два пункта местами
   моёУтро[2] = 'чищу зубы';

6. моёУтро.pop(); // мама говорит, что она с радостью приготовит!
                  // Спасиб, мам!

7. моёУтро.unshift('завтракаю'); // ага! до того, как зубы почистить

8. 'Значит, каждое утро я совершаю ' + моёУтро.length + ' действия.';

9. const пунктКоторыйЯВсегдаЗабываю = 3;
   "Не забудь про пункт "" + моёУтро[пунктКоторыйЯВсегдаЗабываю] + '" !';

10. const чтоЯЗабыл = ['собираю учебники', 'заправляю постель',
                       'причёсываюсь'];

11. console.log('Раньше моё утро всегда было таким: "' + моёУтро.
                toString()
       + '" но теперь этот список расширился: "'
       + моёУтро.concat(чтоЯЗабыл).join(', ') + '".');


12. 'Камень, ножницы, бумага'.split(', ').sort().join(', ');

13. считалка = 'Эне, бене, раба, квинтер, финтер, жаба !';
    считалка.split(' ')[5].toUpperCase().split('').join('-');

14. 'День добрый '.split(', ')[1].concat('Добрый вечер!'.split(' ')[1]);
```

II. Что не так с каждым из фрагментов?

```
1. const орешки = ['кешью', 'миндаль', 'арахис'];
   console.log('В список входят: ' + орешки.size + ' и ещё множество
                других!');

2. // получите в ответе третий по счёту орех из массива
   'Третьим видом орехов в списке является ' + орешки[3];

3. орешки.pull('лесной орех'); // добавить в конец массива

4. console.log('В массиве ' + орешки.pop().length + ' видов орехов.');

5. орешки.unpop('бразильский орех'); // в начало списка

6. let полныйСписок = орешки.toString();
   полныйСписок.shift(); // убрать первый вид из списка

7. const ещёОрешки = ['грецкий орех', 'пекан', 'кедровый орех'];
   орешки.concat(ещёОрешки);
   console.log('Мы добавили ещё больше орешков!: ' + орешки.join(', '));

8. let орешкиВАлфавитномПорядке = орешки.order();

9. орешки = орешки.concat(ещёОрешки).sort().toString();
```

КОМПЛЕКСНЫЙ ОБЗОР

1. Является ли следующее предложение правильным (и если нет — почему)?

```
const заДвумяЗайцамиПогонишься = !! '– одного поймаешь!';
```

2. Перечислите типы данных следующих «ложек»:

```
0; null; undefined; ''; false;
```

3. Строка в JavaScript состоит из _____.
4. При помощи какой функции можно изменить регистр в строке на высокий?
5. Существуют всего три _____ оператора (`&&`, `||` и `!`).
6. Вот операторы _____, которыми следует пользоваться: `===`, `!==`, `>`, `>=`, `<` и `<=`.
7. При помощи какого ключевого слова можно создавать переменные, которые не будут изменяться?
8. Какой тип предложений используется для запуска блоков кода на основании данных *условий*?
9. Если вы вызываете функцию, которая находится внутри другой функции, то такую функцию называют _____.
10. Является ли следующий код правильным? Если нет — почему (попробуйте в консоли)?

```
function надетьЛиЛёгкуюКуртку(температура) {
    const минТемпература = 8;
    const максТемпература = 20;
    if (температура >= минТемпература && температура
        <= максТемпература) {
        return 'Отличная погода, чтобы прогуляться налегке!';
    }
    if (температура > максТемпература) {
        return 'Слишком жарко! Лучше иди в футболке';
    }
    return 'На улице сильный мороз! Лучше надень пуховик!';
}
надетьЛиЛёгкуюКуртку(22);
надетьЛиЛёгкуюКуртку(13);
надетьЛиЛёгкуюКуртку(4);
надетьЛиЛёгкуюКуртку(20);
```

11. Истина/Ложь: логические операторы всегда возвращают булевы значения.
12. Какие встроенные функции будут ожидать от пользователя нажатия на кнопку, прежде чем позволят интерпретатору продолжить работу с остальной частью кода?

13. При помощи какого символа можно было бы упростить следующее присваивание?

```
текущийЦикл = текущийЦикл + 1;
```

14. Каким оператором стоит воспользоваться, чтобы узнать, делится ли некое число нацело на 14?

15. В каких случаях стоит использовать `var`?

16. Какие из представленных ниже значений — «правки»? Перечислите типы данных, к которым они принадлежат:

```
-1; ' '; true; 5; '0'; 'false';
```

17. Является ли следующий код правильным (и если нет — почему?):

```
function поиграемВИмена(имя) {
    return "Сыграем в игру! Вот имя: " + имя + '!';
    return имя + ', ' + имя + ', Бо- ' + имя + '...';
}
поиграемВИмена ('Бенджамин МакГилликатти III');
```
[1]

18. Какой оператор сравнения вернёт в ответе `false`, если обнаружит, что значение слева больше или равно значению справа?

19. Где разработчик может легко и просто проверить работу написанного кода и получить немедленный ответ интерпретатора?

20. Что вы используете для того, чтобы видеть всякие вещи?

21. Истина/Ложь: когда в коде встречаются числа, лучше им присваивать переменные с «говорящими» именами, чтобы сразу можно было понять, что это за число и зачем оно здесь. К примеру, вместо `поездкаНаАвтобусе(5)` лучше было бы так: `const учебныеДни = 5; поездкаНаАвтобусе (учебныеДни);`.

22. Последовательное применение для объекта одного метода за другим в одной и той же строчке кода называют _____.

23. При помощи какой клавиши можно выделить код отступом, чтобы он легче читался (например, внутри функции)?

24. При помощи какого встроенного свойства можно выяснить, сколько элементов содержит в себе массив?

25. Истина/Ложь: когда внутри скобок находятся другие скобки, то всегда следует сперва вычислять значение внутренних, затем переходить ко внешним.

26. Истина/Ложь: операторы сравнения всегда возвращают булевы значения.

---

[1] Популярная песня Ширли Эллис. Песня-игра задаёт несколько правил изменения имён (например, имя Билли, согласно правилу, станет «Билли-Билли-бо-илли».

27. Является ли следующий код правильным (и если нет — почему)?

```
function дайПять(число) {
    return 5;
}
дайПять(8);
```

28. При помощи какой встроенной функции можно округлить десятичную дробь до ближайшего целого?

29. При помощи какого встроенного метода можно добавить элемент в конец массива?

30. Истина/Ложь: `215 >= -622 && !(812 <= 812) || !!(-389 < 389);`.

31. Истина/Ложь: интерпретатор JavaScript игнорирует код, заключённый в комментарии.

32. Назовите значение переменной `встречныйПоперечный` после запуска следующей строчки кода:

```
const встречныйПоперечный ='Иванов' || 'Петров' || 'Сидоров';
```

33. При помощи какой комбинации клавиш делается перенос строки в консоли?

34. Что нужно прописать в консоли, чтобы получить в ответ «Наоми»?

```
const популярныеЖенщины = ['Дженнифер', 'Ванесса', 'Наоми',
'Присцилла'];
```

35. Истина/Ложь: функция `confirm()` всегда возвращает булево значение.

Сделай сам: кошелёк или жизнь!

В этом проекте вы напишете несколько разных блоков кода. В каждом из них так или иначе вы должны будете применить изученные навыки работы с массивами. Если вы застряли на каком-либо этапе работы — скорее обращайтесь к коду, который мы писали в разделе «Повторяй за мной» (они с «Сделай сам», как обычно, весьма похожи). Задание на пять с плюсом: когда закончите с кодом, упакуйте его в функцию и вызовите!

Так вот! Настал Хэллоуин! Вам просто не терпится продемонстрировать всем свой суперский костюм, а потом пойти с друзьями собирать сладости по домам! Но такое дело: в прошлом году часть ваших сладостей таинственным образом куда-то делась, поэтому на сей раз вы собираетесь зорко следить за своей корзинкой! Для этого нам понадобится *список* того, что происходит с вашими сладостями (каждое событие должно быть отражено в коде):

- Прежде чем уйти из дому, вы кладёте в корзинку Kit-Kat и пачку Skittles.
- В первом доме вы получаете Snickers.
- В следующем доме — пару батончиков Nuts и Bounty.

- M&M's! Ура! Ваши любимые! Их вы кладёте на самый *верх*, чтобы всё время их держать на виду и помнить, какой вы счастливчик!
- Вы решаете, что пора съесть батончик Bounty (последний в списке ваших сладостей).
- Вы встречаете своего приятеля Флетчера, который предлагает вам за ваши Skittles целую пачку Starburst. Вы, конечно, с радостью соглашаетесь, поскольку в итоге остаётесь в выигрыше! (Следует просто заменить название соответствующего элемента.)
- А ещё Флетчер только что вспомнил, что у него аллергия на шоколад, так что Milky Way, арахисовые M&M's, шоколадку Mars и батончики Hershey's он решает отдать вам! Ну разве не здорово иметь такого друга, как Флетчер, на Хэллоуин?! (Все сладости Флетчера добавьте к своим при помощи одного-единственного метода.)
- Больше вы вытерпеть не в состоянии, так что вы съедаете все M&M's.
- Вернувшись домой, вы принимаетесь сортировать все сладости в алфавитном порядке.
- Когда всё разложено по алфавиту, вы подсчитываете общий улов и объявляете результат маме.
- Мама интересуется, какие именно сладости у вас есть, так что вы начинаете рассказывать (console.log) ей, перечисляя их все по порядку (в одну строку через «и»).

Когда закончите (ну или если застрянете), обязательно загляните в конец книги и сравните ваше решение с моим. Помните, что правильных решений вполне может быть несколько. Всё что нужно, так это чтобы ваш способ также успешно изменял массивы и выдал в ответе верное количество сладостей, расположенных в верном порядке.

9 Эй, loop-оглазый!

В предыдущей главе мы выучили пять крутых штук для работы с массивами (ну плюс ещё парочку не столь крутых)! В этой главе мы выучим самую наиполезнейшую штуку, которую только можно делать с массивами. Настало время больших открытий. Настало время изучить циклы!

Блистательный цикл while

Не существует ни одного языка программирования, который бы так или иначе не предусматривал **циклов**. Цикл в JavaScript (да и во всех прочих языках) — это блок кода, который повторяет сам себя, пока известное **условие** остаётся истинным. Условие — это всё, что программа проверяет, перед каждым перезапуском цикла. Всякий раз, пока условие остается «правкой», код будет циклично перезапускаться. Как только условие «правкой» быть перестанет (то есть станет «ложкой») — прекратится и цикл.

Самый обыкновенный while

Перед вами простейший вид цикла `while` (откройте пустую вкладку и новое окно консоли):

```
let счётчик = 0;
while (счётчик < 10) {
    console.log('счётчик показывает ' + счётчик + ". Давайте добавим
                ещё 1.");
```

[1] Loop — *англ.* цикл, петля (читается – «луп»).

```
    счётчик++; // инкремент ++ это то же самое, что и `счётчик =
                  счётчик + 1;`
}
console.log('Да хватит уже считать!');
```

Так что тут у нас происходит? Ну, сперва я бы хотел это у вас спросить — вдруг вы смогли бы догадаться об этом, просто глядя на код? Если так, то просто замечательно! Если нет, тоже не беда, я сейчас всё объясню.

- `счётчик` в нашем примере сперва имеет значение `0`.
- *Условием* нашего цикла (оно прописано в скобках) задано, что блок с кодом будет повторять себя снова и снова до тех пор, пока (то есть собственно «`while`») счётчик меньше `10`.
- Поскольку начальное значение `0` меньше `10`, код запускается.
- Каждый раз интерпретатор заново проверяет условие; если условие всё ещё «правка», то код будет запущен вновь (поэтому он и называется «цикл»).
- Каждый раз мы увеличиваем значение `счётчика` на 1. Это очень важно, так как в противном случае значение бы оставалось на `0`, отчего наше условие (`счётчик < 10`) также оставалось бы *всегда* истинным, а наш код бы зациклился навечно! (Обратите внимание: если вы случайно запустите «вечную петлю», то вам придётся перезапустить Chrome, чтобы продолжить работу.)
- Поскольку `счётчик` с каждым циклом увеличивается на 1, то в итоге (после 10 циклов) он станет равным `10`.
- После этого интерпретатор вновь проверит условие (`счётчик < 10`) и, убедившись, что оно теперь ложно, завершит цикл и перейдёт к финальным строчкам нашего кода.

Просто «так принято»

А теперь я вновь напишу тот же код, но назову переменную `i`, а не `счётчик`:

```
let i = 0;
while (i < 10) {
    console.log('i равняется ' + i);
    i++;
}
console.log('Да хватит уже считать!');
```

Так, ну и в чём же тут смысл? Неужели необходима была другая переменная? Да нет. А зачем тогда всё это? Дело в том, что в названии `i` самом по себе, конечно, ничего особенного нет, но по условленной договорённости, если хотите, по традиции, переменную, которая циклично увеличивается или уменьшается, принято называть именно так. Когда подобного рода решения принимаются не

из соображений правильности или неправильности кода, но «по традиции», мы называем это **соглашением** о стандартах оформления кода.

На самом деле множество из того, чему я вас тут учил, является не предписанием верного способа действия, но именно способом исполнения *соглашения*. Возьмём в качестве примера верблюжийРегистр: ведь на работе кода, по сути, никак не скажется, если ваши переменные будут ВОТТАКИМИ, или воттакими, или вовсе вот_такими. Однако, следуя *соглашению* об использовании верблюжьегоРегистра, мы также его используем; таким образом, ваш код с самого начала визуально походил на код профессиональных разработчиков, а им самим будет гораздо легче читать и понимать его.

Так вот, как я уже сказал, использование строчной `i` для циклично увеличивающейся (при помощи инкремента `++`) или уменьшающейся (при помощи декремента `--`) переменной — это также стандарт оформления кода. Если вы решите продолжить карьеру в качестве профессионального разработчика, то переменную `i` в таких обстоятельствах вам предстоит лицезреть сотни и тысячи раз. И чтобы вы не впали в замешательство, а были в курсе таких вещей, я всё это вам и рассказываю.

Жизненный цикл

Рассмотрим пример цикла `while` из реальной жизни.

Предположим, вы — трёхлетний малыш (вам же когда-то было три, так ведь?). Всё утро вы с упоением играли в конструктор, но тут пришла мама и сказала, что пора заканчивать, а все кубики нужно собрать в коробку. А их тут на полу несметные сотни! Вы быстро прикидываете в уме и понимаете, что за раз вам с этим никак не управиться. Так что вы приходите к единственному разумному решению в такой ситуации: начинаете плакать. И тут ваша мама поражает вас до глубины души, предлагая простое решение, казалось бы, неразрешимой проблемы: она берёт сначала часть кубиков, затем кладёт их в коробку, затем возвращается и… повторяет то же самое! И этот удивительный трюк нужно будет проделать столько раз, сколько потребуется, чтобы все кубики с пола перекочевали в коробку.

Вы прикидываете в уме… Ну да, вроде всё верно! Должно сработать!

Давайте шаг за шагом разберём мамин план:

- Предположим, вы выяснили, что ваша `кубикоПодъёмность` (то есть то, сколько кубиков за раз вы сможете унести) — постоянная величина (вы понимаете — `const`анта) и равняется `10`.
- Давайте также предположим, что `кубиковНаПолу` разбросано `230` штук.
- Инструкции таковы: взять наибольшее возможное количество кубиков с пола и перенести их в коробку (то есть вычесть нашу `кубикоПодъёмность` из `кубиковНаПолу`).
- Всё время, пока (то есть `while`) `кубикиНаПолу` ещё остаются (то есть пока наше условие остаётся истинным), нам предстоит возвращаться (то есть совершать цикл) к выполнению инструкции.

Вот вариант этого плана на языке JavaScript:

```
const кубикоПодъёмность = 10;
let кубиковНаПолу = 230;
console.log('На полу ' + кубиковНаПолу + ' кубиков.');
while (кубиковНаПолу > 0) {
    // комбинированное присваивание (см. Гл. 2): `x -= y;` то же,
    // что и `x = x - y;`
    кубиковНаПолу -= кубикоПодъёмность;
    console.log('Я убрал в коробку ' + кубикоПодъёмность
        + ' кубиков, и теперь на полу осталось только ' +
            кубиковНаПолу + '.');
}
console.log('Ура-а-а! Все кубики уложены в коробку!');
```

Надеюсь, тут всё понятно? И смотрите, какая занятная штука: всё то же самое вы можете проделать с любым количеством `кубиковНаПолу`, а количество необходимого для этого кода останется примерно тем же! Чтобы показать вам, насколько гибкими могут быть циклы, мы завернём наш код в функцию с одним параметром. Если считаете, что сможете сделать это сами, призываю вас попробовать, не дожидаясь моего решения! Если же нет, оставайтесь с нами.

Во-первых, вернём наш код нажатием стрелки вверх, а затем допишем к нему сверху нашу функцию:

```
function собираемКубики() {
    //  … (здесь наш код)
}
```

Теперь уберём строчку с количеством `кубиковНаПолу` и сделаем эту переменную параметром нашей функции, поместив её в скобки. Итак, у нас должно выйти следующее:

```
function собираемКубики(кубиковНаПолу) {
    const кубикоПодъёмность = 10;
```

```
    console.log('На полу ' + кубиковНаПолу + ' кубиков.');
    while (кубиковНаПолу > 0) {
        кубиковНаПолу -= кубикоПодъёмность;
        console.log('Я убрал в коробку ' + кубикоПодъёмность
        + ' кубиков, и теперь на полу осталось только ' + кубиковНаПолу +
'.');
    }
    console.log('Ура-а-а! Все кубики уложены в коробку!');
}
```

Ну а теперь можно просто вызвать нашу функцию и, «предложив» ей аргумент (помните? *аргумент*, который мы предлагаем функции, является значением *параметра* этой функции; в нашем случае им будет количество `кубиковНаПолу`), — собрать все кубики!

```
собираемКубики(400);
собираемКубики(30);
собираемКубики(1020);
```

Всё понятно? Отлично. Прежде чем дальше рассказывать вам о циклах, у меня есть небольшая задачка для особо прилежных: а что, если ваша `кубикоПодъёмность` не была бы постоянной величиной? Смогли бы вы немного переписать нашу функцию, чтобы добавить туда ещё один параметр (чтобы она, соответственно, принимала *два* значения в качестве аргументов — через запятую, конечно же)? Двумя этими параметрами должны быть, естественно, `кубикиНаПолу` и `кубикоПодъёмность`. Я уверен, что вам это под силу! (Если что, пример подобной работы можно подсмотреть в главе 4.)

Знаменитый цикл for

Помните, когда мы писали наш первый цикл `while`, что происходило с нашим `счётчиком`? Я сказал тогда, что очень важно, чтобы он постоянно изменялся (в этом нам помог **инкремент** `++`). Потому как, если забыть про это, то код попадёт в бесконечную петлю циклов. Всё так?

Что ж, подобные ситуации случаются довольно регулярно с `while`-циклами; именно поэтому создатели JavaScript сделали ещё один оператор цикла — близкого родственника `while` — `for`. Он представляет собой нечто вроде `while`, но уже со встроенным `счётчиком`!

Синтаксис цикла for

Впервые видя перед собой синтаксис цикла `for`, можно подумать, что это что-то весьма странное. Но изучить его всё равно следует, поскольку вы встретите подобное ещё тысячу и тысячу раз! Откроем новую вкладку с консолью и начнём:

```
for (let i = 0; i < 10; i++) {
    console.log('i = ' + i);
}
console.log('Да хватит уже считать!');
```

Ну как, узнали? А должны были! Ведь это практически тот же код, что мы использовали для `while`-цикла в начале главы! Три части цикла отделены друг от друга точкой с запятой; все они были и в том цикле `while`, но в других местах. Давайте разберём подробнее:

1. Первым делом мы объявляем переменную.
2. Далее мы задаём условие, которое интерпретатор будет проверять перед каждой **итерацией** цикла (чтобы определить, а стоит ли вообще запускать код по новой). Итерацией называют один виток цикла (например, если у нас было 10 циклов запуска, то можно сказать, что имели место 10 итераций кода).
3. Третьей частью нашего кода является инкремент (или же декремент). Он запускается в *конце* каждой итерации, чтобы всякий раз менять нашу переменную.

Практика циклов for

Практически всегда, когда известно количество необходимых итераций, вы скорее будете работать с `for`, нежели с `while`. В связи с тем, что циклы `for` столь популярны (намного более популярны, чем `while`), я и хочу, чтобы вы напрактиковались в синтаксисе таких циклов. Напишите код для всех следующих сценариев (выдуманных, конечно же). Для каждого из них открывайте новое окно консоли. Очень хорошо, если вы выполните задание самостоятельно, но в случае чего — пожалуйста, возвращайтесь сюда за помощью.

Сценарий цикла: бег по кругу

Ваш физрук майор в отставке Пэйн задал вам пробежать 5 кругов по залу, громко отсчитывая каждый круг.

```
function бегПоКругу(всегоКругов) {
    for (let i = 0; i < всегоКругов; i++) {
        console.log( i + ' круг' + '!');
    }
}
бегПоКругу(5); // 5 — аргумент для параметра всегоКругов
```

Хм… «0 круг», да? Что-то тут не так… Наверное, стоит починить эту штуку. Но как? Так, давайте рассуждать вместе. Значит, заместо «0 круг» во время первой итерации мы бы хотели видеть «1 круг», верно? А во время второй — «2 круг» (а не «1 круг»).

Тут определённо есть логика! Ведь в каждом случае мы хотим, чтобы отображаемый номер круга был ровно на 1 больше, чем значение переменной `i`. Что ж, это очень просто: `i + 1`. Удивительно! Стоит только чётко описать проблему, как тут же найдётся её решение! Давайте обновим нашу функцию:

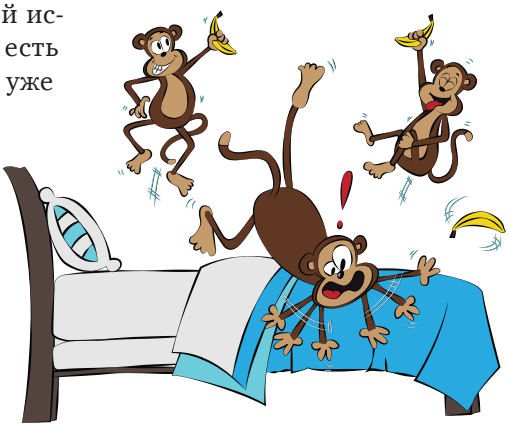
```
    for (let i = 0; i < всегоКругов; i++) {
        console.log( (i + 1) + ' круг' + '!');
    }
}
```

Вот невезуха! На следующий день майор Пэйн пришёл не в духе… Теперь весь класс мотает 8 кругов! Как здорово, что у нас готова функция: теперь только надо дать ей 8 в аргументы вместо 5!

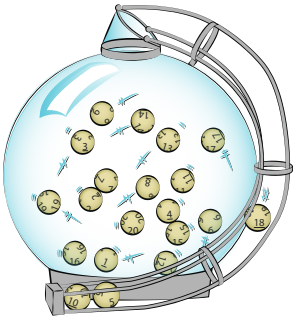
```
бегПоКругу(8);  // следующий день
бегПоКругу(10); // после-следующий
бегПоКругу(15); // после-после-следующий (не физрук, а садист у вас
                // какой-то!)
```

Сценарий цикла: десять обезьянок скакали по кровати

Есть одна детская считалка, в которой используется цикл с декрементом (то есть счётом в обратном порядке). Сейчас уже не припомню всех деталей, но основной сюжет таков: 10 обезьянок скачут по кровати и, несмотря на то что постоянно кто-то один падает и бьётся головой, они продолжают «резвиться»… Неудивительно, что в конце концов упали все 10! И куда только смотрела их маменька?



```
function десятьОбезьянокСкакалиПоКровати (всегоОбезьянок) {
    for (let i = всегоОбезьянок; i > 0; i--) {
        console.log(i + " обезьянок скакали по кровати. "
            + 'Одна упала "ой" - ушиблась головой.');
    }
    console.log('Угомонившиеся обезьянки отказались от комментариев. ' );
}
десятьОбезьянокСкакалиПоКровати (10);
```



Сценарий цикла: лотерея

В вашем городе устраивают новую лотерею. Для проведения розыгрыша нужно 6 случайных чисел в диапазоне от 1 до 20. Обратите внимание: здесь понадобятся встроенные функции `Math.random()` и `Math.floor()`; если вы не помните, что это и как это работает, вернитесь к главе 4 и повторите.

```
function случайныеНомераДляЛотереи(всегоНомеров, минЗначение,
         максЗначение) {
    for (let i = 0; i < всегоНомеров; i++) {
        console.log(Math.floor(Math.random() * максЗначение) +
                    минЗначение);
    }
}
случайныеНомераДляЛотереи(6, 1, 20);
случайныеНомераДляЛотереи(6, 1, 20); // должен получиться другой набор
                                     // из 6 чисел
случайныеНомераДляЛотереи(7, 1, 20); // новый набор из 7 чисел
```

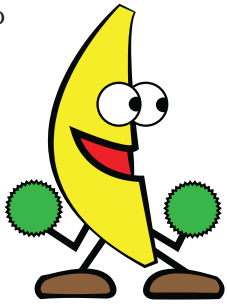
Сценарий цикла: считаем до больших чисел

Папа немного утомился от постоянных просьб «купить мороженое»; в качестве отвлекающей тактики он придумал, что до следующей просьбы вам необходимо досчитать от 1 до 300 (обратите внимание: здесь будет массив!).

```
function посчитаемКа(крайнееЧисло) {
    const всеЧисла = []; // создаём пустой массив
    for (let i = 0; i < крайнееЧисло; i++) {
        всеЧисла.push(i + 1); // добавляем 1, 2, 3... и так далее —
                              // в массив
    }
    console.log(всеЧисла.join(', '));
    console.log('... Ну! А ТЕПЕРЬ-ТО мне можно мороженое?');
}
```

```
посчитаемКа (300);
посчитаемКа (500);
посчитаемКа (1000); // Ну ладно-ладно! Так уж и быть! Купим тебе
                    // мороженое! (ну даёшь)
```

Петляем в массивы!



В начале этой главы я пообещал показать вам самую крутую штуку (или я, может, сказал «самую полезную» или ещё что-нибудь в этом духе…), которую можно делать с массивами. Но сперва нам нужно было обсудить циклы… Что ж, циклы мы успешно разобрали, с массивами *тоже* успели вдоволь поиграть. Теперь пришло время узнать, что эти двое просто созданы друг для друга! Представьте, что я сначала учил вас намазывать сэндвич арахисовой пастой, потом учил намазывать его джемом, ну а теперь… ТЕПЕРЬ МОЖНО СДЕЛАТЬ НАИВКУСНЕЙШИЙ СЭНДВИЧ С ДЖЕМОМ И АРАХИСОВОЙ ПАСТОЙ!

А ещё знаете, что здорово? Что вплоть до конца этой главы мы не будем учить больше ничего нового! Мы будем использовать только уже знакомые нам инструменты, комбинировать их, чтобы получилось что-нибудь весьма полезное — я покажу. Приступим!

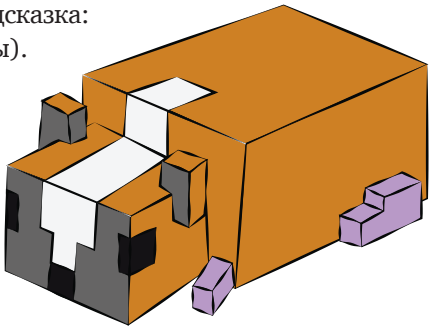
Массив с while-циклом

Сперва коснёмся менее популярного цикла — `while`. Мы создадим массив, элементами которого выступят действия, необходимые для того, чтобы сыграть в новую компьютерную игру — Мойкрафт (теперь и с сюжетным режимом!); затем при помощи `while` мы запустим цикл, чтобы убедиться, что каждый шаг выполнен. Закройте и заново откройте браузер, откройте пустую вкладку, запустите окно консоли. Итак:

```
let чтобыСыграть = [
    'Повесь на стенку телевизор',
    'Чуть ниже поставь топовую консоль ИхБохИх 720',
    'Соедини HDMI-кабелем ИхБох и телевизор',
    'Включи телек',
    'Включи ИхБох',
    'Дважды кликни на иконке Мойкрафта',
    'Можно играть (берегись взрывающихся скримеров!)'
];
```

Теперь нам нужен `while`-цикл, который последовательно выведет в консоль каждый шаг. В общем-то, учитывая всё, что мы за последние пару глав изучили, вы

вполне можете справиться самостоятельно (подсказка: не забудьте о методе `.shift()` из прошлой главы). Если вы согласны, что сможете, тогда смело закрывайте книжку прямо сейчас и — удачи! Если же нет, тогда оставайтесь тут, и мы напишем код вместе.



Давайте для начала выведем в консоль первый элемент массива `чтобыСыграть`. Мы можем это сделать двумя способами (пока не печатайте):

```
чтобыСыграть[0]; // возвращает первый элемент (массив остаётся в целости)
чтобыСыграть.shift(); // возвращает первый элемент (и УДАЛЯЕТ его
                      // из массива!)
```

Так как второй вариант действительно *изменяет* состав элементов в массиве, значит, если запустить с ним цикл, то каждую итерацию будет возвращаться новый элемент.

Но помните: всегда, когда работаете с циклами, будьте супервнимательны, чтобы не образовалось вечного цикла! Для этого необходимо такое условие, которое не *всегда* останется истинным. А этого мы можем добиться, просто проверяя длину массива. Ведь если каждую итерацию из массива удаляется по одному элементу, то рано или поздно ответ `.length()` — длины массива достигнет 0 (то есть перестанет быть истинным и станет ложным). Попробуем:

```
while (чтобыСыграть.length > 0) {
    const следующийШаг = чтобыСыграть.shift();
    console.log(следующийШаг);
}
```

Если всё сделано правильно, то в консоли отобразятся все действия в нужном порядке! И ещё кое-что: давайте узнаем, сколько действий в нашем списке осталось:

```
чтобыСыграть;
```

Ага! Пустой массив: ведь цикл с каждой итерацией убирал из массива элемент за элементом!

А теперь я бы хотел проделать всё то же, но с небольшими добавлениями. Так что раза три-четыре нажмите стрелку вверх, чтобы вернуться к нашему исходному массиву (который начинался с `let чтобыСыграть = [`). Когда массив вернётся — уберите из начала `let`. Как вы помните, это означает, что теперь мы собираемся *изменять* переменную, а не создавать её по-новой. А теперь запустите код, чтобы получить назад наш массив.

Давайте несколько упростим наш цикл. Попробуйте следующий цикл (он должен вести себя точно так же) и сравните его работу с предыдущим:

```
while (чтобыСыграть.length) { // в том цикле у нас было
                              // (чтобыСыграть.length > 0)
    console.log(чтобыСыграть.shift()); // а там была ещё одна переменная
}
```

Не кажется ли вам странным, что мы убрали «`> 0`» из условия? Дело вот в чём: `чтобыСыграть.length` возвращает количество элементов в массиве; количество элементов — величина всегда положительная (и целая), если оно не равно нулю, так? Что ж, любое возможное положительное (и, конечно, целое) число является «правкой»… за исключением… ну конечно! Опять же — нуля!

Вы понимаете, что пока количество элементов в массиве не равняется нулю, то условие будет оставаться «правкой» и `while` будет продолжать итерации. Но как только элементы в массиве кончатся, то условие станет «ложкой» (ведь вы помните? нуль — «ложка») и мы «вырвемся» из цикла!

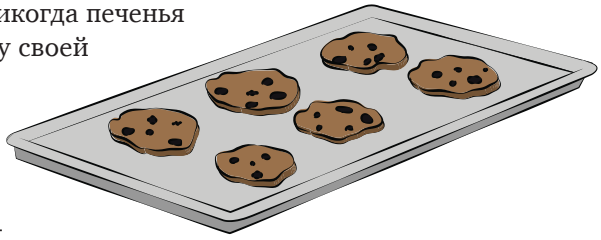
Что до средней части нашего кода (внутри `while`), то тут мы просто «слепили» две строчки в одну; так как `чтобыСыграть.shift()` возвращает строку, то мы сразу же можем её и вывести в консоль (`console.log`). Не забывайте, что если у вас есть *вложенная* функция, то интерпретатор всегда будет начинать вычислять значение изнутри, то есть будет запускать сперва внутреннюю, а затем внешнюю функцию.

Массив с for-циклом

Как я уже говорил, циклы `for` много популярнее `while`. В следующем примере мы как раз «запетляем» массив `for`-циклом. И, в отличие от предыдущего цикла `while`, мы при этом нисколько не изменим исходного массива.

```
const готовимПеченьки = [
    'Разогрейте духовку до 175 градусов',
    'Смешайте маргарин, белый сахар, тёмный сахар, ваниль и яйца',
    'Засыпьте муку, крахмал, соль, соду и шоколадную крошку',
    'Выложите на противень',
    'Оставьте в духовке на 12 минут',
    'Достаньте противень',
    'Приятного аппетита!'
];

for (let i = 0; i < готовимПеченьки.length; i++) {
    const шагНомер = i + 1; // наконец-то у нас сразу будет "1 шаг",
                            // а не "0 шаг";
    const шаг = готовимПеченьки[i];
    console.log(шагНомер + '. ' + шаг);
}
```

Ладно-ладно, каюсь: в жизни никогда печенья не пёк. Но рецепт я выпросил у своей дочери, а она готовит просто восхитительное шоколадное печенье!



Удалось ли вам понять, почему код работает? Всё просто. С каждой итерацией значение `i` увеличивается на 1. В первой итерации значение `i` равняется `0`, а значит, `готовимПеченьки[0]` вернёт сообщение `'Разогрейте духовку до 175 градусов'`, так? Так. И раз `i` в первой итерации было `0`, то и `готовимПеченьки[i]` вернёт то же, что и `готовимПеченьки[0]`.

Далее, на втором витке цикла значение `i` становится уже `1`, на третьем — `2` и так далее. Следовательно, `готовимПеченьки[i]` становится сначала `готовимПеченьки[1]`, потом `готовимПеченьки[2]` и так далее с каждой итерацией. Уловили?

А поскольку в нашем `for`-цикле не указано конкретное количество необходимых итераций, то мы можем спокойно убирать или, наоборот, добавлять новые элементы в наш массив!

```
готовимПеченьки.push('Не забудьте запивать всё это парой литров молока!');
готовимПеченьки.push('Чтобы до последней крошечки всё съели!');
готовимПеченьки.push('А теперь — испытайте чувство вины, что не оставили
                      ничего сестре');
готовимПеченьки.push('Теперь у вас болит живот; прилягте');

for (let i = 0; i < готовимПеченьки.length; i++) { // более простой вари-
                                                   ант; тоже работает
    console.log((i + 1) + '. ' + готовимПеченьки[i]);
}
```

Но! Мы можем пойти ещё дальше и ещё более упростить наш код без потери в результате! Я бы, пожалуй, не стал собственный код писать таким образом, но в целях демонстрации работы цикла `for` сделаю это здесь. Сравните следующий код с тем, что у нас был прежде, и попробуйте найти все изменения (не забывайте, что на этот раз нам нет нужды «перезагружать» наш исходный массив, так как мы его и не меняли):

```
for (let k = 1; k <= готовимПеченьки.length; k++) {
    console.log(k + '. ' + готовимПеченьки[k - 1]);
}
```

Ну как? Нашли? Вот те, на которые я особо хочу обратить ваше внимание:

1. Вместо `i` здесь мы использовали `k`. Причина проста: показать, что ничего уникального в `i` нет, и можно использовать абсолютно любую переменную.

2. Мы сменили начальную позицию на `k = 1;` вместо `k = 0;` (или `i = 0;`). Таким образом, значение `k` теперь пойдёт от `1` до `7`, а не от `0` до `6`. Опять же, я бы так делать не стал (я бы начинал с `0`). Просто показываю, что можно начинать и с `1` (или любого другого значение), если так нужно.

3. Мы поменяли `<` на `<=`. Это важно, раз мы начали с `1`. В противном случае наш цикл окончится прежде, чем мы достигнем последнего элемента в массиве.

4. Мы использовали `готовимПеченьки[k - 1]`. Вы, быть может, уже догадались зачем? Затем, что `k` начинает с `1` и нам необходимо как раз таки отнять `1`, чтобы не начать с `готовимПеченьки[1]` (`Смешайте маргарин, белый сахар, тёмный сахар, ваниль и яйца`). Ведь если не сделать этого, то мы так никогда и не разогреем духовку, и в итоге нам придётся съесть целый противень сырого теста… Хмм… а вообще-то звучит не так уж плохо!

Мораль такова: массивы и циклы (особенно с `for`) постоянно используются вместе. Выучите синтаксические правила и всякий раз, когда у вас есть *список* чего-либо, сразу представьте себе соответствующий массив; всякий раз, когда нужно будет сделать что-то с элементом списка, используйте цикл `for`. Если вы грамотно построили `for`-цикл, то вы (или пользователи) сможете как угодно изменять список, при том что код будет работать безукоризненно.

Повторяй за мной: спойлератор

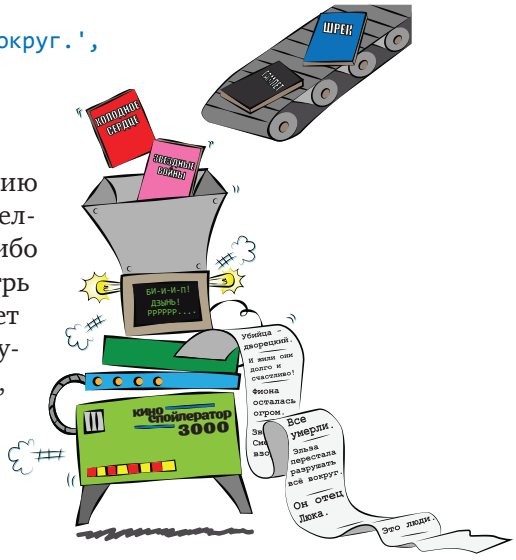
У каждого, наверное, есть такой странный приятель, который смотрит кучу фильмов, а потом путает их концовки. Такова ваша подруга Стелла. И эта её привычка подчас приводит к весьма неловким ситуациям, так что друзьям приходится краснеть за неё. К счастью, у Стеллы нашёлся отважный друг-программист, который вызвался написать для неё генератор спойлеров, чтобы она всегда могла вспомнить, о каком именно фильме она говорит, и, как вы уже, должно быть, догадались, этим отважным другом являетесь вы.

Функция, которую вы собираетесь смастерить для Стеллы, будет на предложенное название фильма отвечать его концовкой… (печальный вздох)… и как только вы умудрились подписаться на такое?

Да ладно. Что бы я был за друг, если бы не попытался помочь вам? Что, дать совет? Хорошо, тогда слушайте: вам нужно составить два списка (подсказка: когда слышите слово «список», можете смело читать «массив»); в первый поместить названия известных фильмов, а во второй — ёмкое описание их концовок. Очень важно, чтобы оба эти списка шли параллельно (то есть, скажем, третий фильм в первом списке должен соответствовать третьей концовке во втором).

```
const фильмы = [
    'Властелин колец',
    'Шрек',
```

```
    'История игрушек',
    'Звёздные войны',
    'Холодное сердце',
    'Гамлет',
    'Золушка'
];

const спойлеры = [
    'Кольцо Всевластья уничтожено.',
    'Фиона осталась огром.',
    "Базз Лайтер всё-таки осознал, что он — игрушка.",
    'Звезда Смерти взорвана.',
    'Эльза перестала разрушать всё вокруг.',
    'Все умерли.',
    'И жили они долго и счастливо.'
];
```



Теперь упакуем оба массива в функцию (подсказка: либо воспользуйтесь стрелкой вверх и допишите код функции, либо же скопируйте и вставьте массивы внутрь уже написанной). У этой функции будет всего один параметр (то есть она будет ожидать лишь одного аргумента), а именно — название фильма, который нужно будет заспойлерить. Давайте попробуем (не забудьте про `// план:`):

```
function заспойлерить(мойФильм) {
    // … (сюда мы вставим оба массива)

    // план: здесь будет правило сопоставления фильма со спойлером
    // план: а здесь будет ответ (с return)
}
```

Отлично. Теперь, просто чтобы убедиться, что всё работает, давайте поместим какое-нибудь `return`-сообщение вместо последнего `//плана:`. Допустим, функцию вызвали, но не предложили ей фильм или предложили, но не из списка. Что в таком случае должна будет вернуть наша функция? Пожалуй, что-нибудь в таком духе:

```
    return "К сожалению, этого фильма нет в нашем списке.";
```

Давайте проверим:

```
заспойлерить('Какое-нибудь левое кино'); // Ничего удивительного.
заспойлерить('Властелин Колец'); // Как так? Но я уверен, что...
заспойлерить('История игрушек'); // А! Постойте-ка...
```

Так… теперь наша функция работает? Ну да. Но, вообще-то, не совсем. Проблема в том, что она всегда возвращает одно и то же. Давайте вернём функцию стрелкой вверх и добавим в неё всё недостающее. В первую очередь — логику сопоставления названия фильма с нужной концовкой. Кстати, вы же помните, что при помощи SHIFT+ENTER можно запросто перепрыгнуть со строчки на строчку, так?

Важное замечание: код с сопоставлением обязательно должен быть расположен *выше* возвращаемого сообщения. Всё очень просто: как только интерпретатор выполняет предложение с `return`, он понимает, что его работа с функцией окончена, и больше никакого кода из неё не читает. Короче, вместо первого `// плана:` наберём следующий код:

```
for (let i = 0; i < фильмы.length; i++) {
    if (фильмы[i] === мойФильм) {
        return спойлеры[i];
    }
}
```

Пожалуйста, внимательно посмотрите на этот код. Это, конечно, непросто (но вы ведь понимаете, что стадию «простого» программирования мы уже давно миновали), но подумайте и постарайтесь вникнуть и понять логику. Надеюсь, что вам это удалось. Итак, если всё сделано правильно, то у вас в консоли должно быть нечто подобное:

```
> function заспойлерить(мойФильм) {
    const фильмы = [
    'Властелин колец',
    'Шрек',
    'История игрушек',
    'Звёздные войны',
    'Холодное сердце',
    'Гамлет',
    'Золушка'
];

const спойлеры = [
    'Кольцо Всевластья уничтожено.',
    'Фиона осталась огром.',
    "Базз Лайтер всё таки осознал, что он — игрушка.",
    'Звезда Смерти взорвана.',
    'Эльза перестала разрушать всё вокруг.',
    'Все умерли.',
    'И жили они долго и счастливо.'
];
        for (let i = 0; i < фильмы.length; i++) {
        if (фильмы[i] === мойФильм) {
            return спойлеры[i];
        }
    }

    return "К сожалению, этого фильма нет в нашем списке.";
}
```

Теперь давайте попробуем нашу функцию:

```
заспойлерить('Властелин колец'); // так-то лучше
заспойлерить('Гамлет'); // какая жалость! :-(
заспойлерить('Золушка'); // Ура! :-)
```

Функция просматривает наш массив с фильмами в поисках совпадений со значением параметра `фильм`. Если таковое находится, то функция возвращает в `return`-сообщении спойлер с тем же индексом, что и фильм (именно поэтому для них использована одна и та же переменная `i`). После выполнения `return` никакой другой код не запускается, поэтому, запрашивая концовку фильма из массива, вы никогда не увидите сообщения `"К сожалению, этого фильма нет в нашем списке."`. Всё понятно?

На этом можно, пожалуй, и закончить… Можно — «было бы»! Ведь мы, разработчики, частенько любим всё ещё пару раз доделать, сверх законченного…

А что, если Стелле было бы удобнее не по одному «спрашивать» у функции спойлеры, но дать сразу *список* фильмов и получить назад спойлеры к каждому? Давайте не будем исправлять для этого старую функцию, а просто напишем *новую*, которая будет вызывать старую (возможно даже циклично). Если вы считаете, что справитесь самостоятельно — бросьте читать и вперёд на подвиги!

Если вы всё ещё здесь, тогда давайте посмотрим на одно из возможных решений. Наберите у себя в консоли и проверьте:

```
function многоСпойлеров(списокФильмов) {
    console.log('ОСТОРОЖНО СПОЙЛЕРЫ! Эти фильмы заканчиваются так:');
    for (let i = 0; i < списокФильмов.length; i++) {
        const название = списокФильмов[i];
        console.log(название + ': ' + заспойлерить(название));
    }
}

многоСпойлеров(['История игрушек', 'Холодное сердце', 'Отель "Руанда"',
                'Властелин колец']);
многоСпойлеров(['Звёздные войны', 'Шрек', 'Взрослая скукотища с кучей
                 умной болтовни']);
```

Надеюсь, у вас всё получилось! Но не останавливайтесь на достигнутом; подробно изучите код, поиграйте с ним, поизменяйте туда и сюда. Попробуйте разобраться, почему и как всё в нём работает именно таким образом. Не копируйте вслепую! Лучше постарайтесь проанализировать и понять принципы работы. Навык чтения и понимания кода является одним из важнейших навыков профессионального разработчика. Другим же важнейшим навыком является умение грамотно загуглить то, что осталось непонятым (думаете, я шучу? нисколько! спросите любого знакомого программиста, и он подтвердит мои слова!).

ВИКТОРИНА

Как обычно, отвечайте на вопросы в Рабочей тетради, никуда не подглядывая; когда закончите — попросите родителей или проверьте себя сами по ответам в конце.

1. Как называется блок кода, повторяющий себя снова и снова, пока заданное условие не перестанет быть истинным?
2. Что следует добавить на место прочерка в следующем коде?

```
let i = 0;
_____ (i < 5) {
    console.log(i);
    i++;
}
```

3. Какой из двух операторов цикла, которые обсуждались в главе, является более популярным?
4. Если _____ всегда будет оставаться истинным, то код может застрять в бесконечном _____.
5. Сколько сообщений выдаст в консоль следующий код?

```
const номера = [1, 2, 3, 5, 6];
for (let j = 0; j < номера.length; j++) {
    console.log(j + ' => ' + номера[j]);
}
```

6. Когда при написании кода некоторое решение обусловлено не практическими или техническими соображениями, но условленными нормами, мы называем это _____.
7. Сколько сообщений выдаст в консоль следующий код?

```
let k = 0;
while (k < 7) {
    console.log('ещё строчка!');
}
```

8. При помощи какого символа отделяются друг от друга находящиеся в скобках части цикла `for`?
9. Как называется средняя из трёх заключённых в скобки частей цикла `for`?
10. Будет ли в результате запуска следующего кода выведено в консоль какое-нибудь сообщение? Если нет — почему? Если да — какое? (Дайте ответ до проверки в консоли!)

```
const топ40Хитов = ['Hey Jude', 'Billie Jean', 'Imagine', 'Hotel
Calif.'];[1]
for (let m = 1; m <= топ40Хитов.length; m++) {
    if (топ40Хитов[m] === 'Танец маленьких утят') {
        console.log('Достойный выбор!');
    } else if (топ40Хитов[m] === undefined) {
        console.log('Какая ошибка!');
    } else if (топ40Хитов[m] === 'Hey Jude') {
        console.log("О! Слона-то я и не приметил!");
    }
}
```

11. Сколько сообщений выдаст в консоль следующий код?

```
const максЗначение = 17;
let n = 0;
while (n <= максЗначение) {
    console.log('Посчитаем! ' + n);
    n++;
}
```

12. Назовите термин, которым обозначается единичное прохождение цикла?

13. Что следует добавить на место прочерка в следующем коде?

```
const шляпы = ['ковбойская', 'котелок', 'колпак', 'цилиндр'];
for (let p = 0; p < ___________________; p++) {
    console.log(шляпы[p] + ' — вот что я бы хотел носить!');
}
```

КЛЮЧЕВЫЕ ЭПИЗОДЫ

Пробегитесь по списку и обязательно вернитесь, чтобы освежить ту или иную тему, если она покажется вам не до конца понятной!

- Цикл.
- `while`-цикл.
- Условие.
- Соглашение о стандартах оформления кода.
- `for`-цикл.
- Итерации.

[1] Известные песни The Beatles, Майкла Джексона, Джона Леннона и The Eagles соответственно.

- Инкремент в циклах `for`.
- Массивы с `while`-циклом.
- Массивы с `for`-циклом.
- Вызов функции с циклом.

УПРАЖНЕНИЯ

I. Введите следующие правильные фрагменты кода в консоль.

Обязательно, прежде чем проверить фрагменты в консоли, попробуйте решить их сами — какой ожидаемый результат для каждого из них?

```
1. let i = 0;
   while (i < 6) {
       console.log(i);
       i++;
   }
```

```
2. const семёрки = [];
   for (let i = 0; i < 10; i++) {
       семёрки.push(i * 7);
   }
   семёрки.join('|');
```

```
3. while (семёрки.length) {
       console.log(семёрки.shift());
   }
```

```
4. for (let j = 0; j < 20; j++) {
       if (j % 2) {
           console.log(j + ' нечётное!');
       } else {
           console.log(j + ' чётное!');
       }
   }
```

```
5. const буквыАлфавита = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'.
                         toUpperCase().split('');
   const алфавитВОбратнуюСторону = []; // создадим пустой массив
   for (let k = буквыАлфавита.length; k > 0; k--) { // отсчитываем "вниз"
                                                    // (не "вверх")
       алфавитВОбратнуюСторону.push(буквыАлфавита[k - 1]); // добавим бук-
                                                           // вы в массив
   }
   алфавитВОбратнуюСторону;
```

```
6. const гласные = 'ауоыиэяюёе '.toUpperCase().split(''); // создадим
                                                        // массив с гласными
   while (алфавитВОбратнуюСторону.length) { // продолжим цикл, пока массив
                                            // не опустеет
       const этаБуква = алфавитВОбратнуюСторону.pop(); // убираем буквы
                                                       // из массива
       for (let m = 0; m < гласные.length; m++) { // цикл по всем гласным
           if (гласные[m] === этаБуква) { // проверим, является ли буква
                                          // гласной
           console.log(этаБуква + ' — гласная!');
   }
       }
   }

7. for (let i = 0; i < 30; i+= 3) {
       console.log(i);
   }

8. function какоеЧислоДелитсяНа(максЧисло, делитель) {
       console.log('Какое число, меньше чем ' + максЧисло
          + ' делится на ' + делитель + '?');
       for (let n = 0; n < максЧисло; n++) {
          if (!(n % делитель)) {
              console.log(n + ' делится на ' + делитель + '!');
          }
       }
   }

9. какоеЧислоДелитсяНа(100, 10);

10. какоеЧислоДелитсяНа(60, 5);

11. какоеЧислоДелитсяНа(90, 3);

12. let p = 0;
    while (p < 52) {
        console.log(p);
        p += 5;
    }

13. function числовойРяд(основноеЧисло) {
       const ряд = [];
       for (let i = 1; i <= 10; i++) {
           ряд.push(основноеЧисло * i);
       }
       return ряд.join(' | ');
    }

14. числовойРяд(3);

15. числовойРяд(5);

16. числовойРяд(7);

17. function полнаяТаблица(основноеЧисло) {
```

```
        const ряды = [];
        for (let i = 1; i <= основноеЧисло; i++) {
            ряды.push(числовойРяд(i));
        }
        return ряды.join('\n');
    }
18. полнаяТаблица(2);
19. полнаяТаблица(6);
20. полнаяТаблица(10);
```

II. Что не так с каждым из фрагментов?

```
1. while (let i = 0; i < 6; i++) {
       console.log(i);
   }
2. let q = 10;
   for (q > 0) {
       console.log(q--);
   }
3. let сигналыСветофора = 'красный-жёлтый-зелёный'.split('-');
   for (const r = 0; r < сигналыСветофора.length; r++) {
       console.log(сигналыСветофора[r]);
   }
4. function всеЦветаРадуга() {
       const цвета = 'красный,оранжевый,жёлтый,зелёный,голубой,синий,
                      фиолетовый'.split();
       for (let i = 0; i < цвета.length; i++) {
           console.log('Один из цветов: ' + цвета[i]);
       }
           console.log("Вот всё цвет радуги!");
   }
   всеЦветаРадуга();
5. function считаемДвойками() {
       for (let r = 0; r <= 10; r+= 2) {
           return console.log(r);
       }
   }
6. function считаемДвойками() {
       for (let s = 0; s <= 10; s++) {
           if (s % 2) {
              console.log(s);
           }
       }
   }
```

МЕГАКОМПЛЕКСНЫЙ ОБЗОР

Итак, мы подошли к последнему «Комплексному обзору»! Он будет супердлинным и довольно сложным, но как иначе? Ведь он должен охватить целиком всё то (громко сказано: конечно же, удалось упомянуть далеко не всё), что мы успели изучить! Взгляните, как много мы успели узнать за это время! Разве не поразительно? Ну, а раз вы зашли уже столь далеко, то и вы, и ваши способности — тоже поразительные, разве нет?! Вы уже на самой-самой финишной прямой; остался лишь последний рывок!

1. Вот операторы _____, которые мы используем: `===`, `!==`, `>`, `>=`, `<` и `<=`.
2. Если в качестве условия используется не строго булево значение, то в случаях, когда оно расценивается как `true`, мы называем такое значение _____, а когда `false` — _____.
3. Истина/Ложь: логические операторы всегда возвращают булевы значения.
4. Что означает DRY (принцип сухости) применительно к программированию?
5. Истина/Ложь: когда происходит вычисление значения вложенной функции, интерпретатор сперва вычисляет значение внутренней, а затем внешней функции.
6. При помощи какого кода вы можете получить случайное число в диапазоне от `0` до `40`?
7. Что показывает длина (`.length`) массива? А `.length` строки?
8. Выполняет ли следующий код замысел разработчика (и если нет — почему)?

   ```
   const часовВСутках = 24;
   ```

9. _____ элемента в массиве — это номер, под которым данный элемент «записан» в этом массиве.
10. Истина/Ложь: при работе с блоком комментариев интерпретатор JavaScript будет игнорировать всё находящееся от `/*` и до `/*`.
11. При помощи какого встроенного метода можно добавлять элементы в конец массива?
12. Назовите примитивный тип данных, значением которого является то, что никакого значения присвоено не было.
13. Каким термином называют последовательное выполнение одного метода за другим, указанных в одной и той же строчке кода?
14. Каким общепринятым стилем пользуется большинство разработчиков для написания имён переменных?

15. Истина/Ложь: при запуске строчки `мойМассив.join('\n')` будет выдана строка, перечисляющая все элементы массива, разделённые переносом.

16. Является ли следующий код правильным (и если нет — почему)?

```
const сегодняВыступают = 'Трёхколёсный Сыр';
сегодняВыступают = "Сал Монелла и Тан Дыр"; // "Трёхколёсные"
                                            // не смогли приехать
```

17. Чем следует заполнить пропуски, чтобы код генерировал случайные значения из элементов массива?

```
const пожитьБыВХолоде = ['Антарктике', 'Северном полюсе', 'Канаде'];
const случайныйИндекс = Math._____(Math._____() *
                        пожитьБыВХолоде._____);
const холодноеМестечко = пожитьБыВХолоде[случайныйИндекс];
console.log('Если уж решил поселиться в ' + холодноеМестечко + ',
             то тебе стоит запастись хорошим пуховиком!');
```

18. Какой результат рассчитывает получить разработчик, используя оператор modulo?

19. Дайте ответ, прежде чем проверите его в консоли: каков будет результат запуска следующего кода?

```
['Я', 'Ю', 'Ь', 'Э'].toString().toLowerCase().split(',').sort()[1];
```

20. Назовите значение переменной `мобильноеУстройство` после запуска следующей строчки:

```
const мобильноеУстройство = 'смартфон' || ('планшет' && 'карманный
компьютер (КПК)');
```

21. Назовите значение переменной `немобильноеУстройство` после запуска следующей строчки:

```
const немобильноеУстройство = 'ТВ' && ('ноутбук' || 'стационарный
комп');
```

22. Назовите два способа, которыми можно было бы сократить (то есть упростить) следующую строчку:

```
счётКомандыПротивника = счётКомандыПротивника + 1;
```

23. Каким оператором стоит воспользоваться, чтобы узнать, делится ли некое число нацело на 5?

24. Истина/Ложь: сообщения об ошибках предназначены для людей, чтобы, ознакомившись с таковым, разработчики смогли найти решение проблемы.

25. Какой оператор сравнения вернёт в ответе `false`, если обнаружит, что значение справа меньше или равно значению слева?

26. Истина/Ложь: после запуска следующего кода длина (`.length`) массива `вагоныПоезда` будет 5.

```
const вагоныПоезда = ['головной', 'грузовой'];
вагоныПоезда.push(['пассажирский', 'багажный', 'тормозной']);
```

27. При помощи какой встроенной функции можно округлить десятичную дробь до ближайшего целого?

28. Какое значение имеет одиночный знак равенства?

29. При помощи какого сочетания клавиш можно выполнить перенос строки в консоли?

30. Истина/Ложь: разработчики зачастую не используют функции `alert()`, `confirm()` и `prompt()`, поскольку они замораживают код в ожидании действия пользователя.

31. Истина/Ложь: функция `confirm()` всегда возвращает булево значение.

32. Назовите три ключевых слова, при помощи которых можно объявить новую переменную; какое из них вы, вероятно, будете использовать наиболее часто, а какое, возможно, никогда?

33. Если _____ всегда будет оставаться истинным, то код может застрять в бесконечном _____.

34. Каким образом можно создать новую строчку в консоли (а также в сообщениях для функций `alert()`, `confirm()` и `prompt()`)?

35. Когда при написании кода некоторое решение оправдано не практическими или техническими соображениями, но условленными нормами, мы называем это _____.

36. Так же как и операторы сравнения, _____ операторы (`&&`, `||` и `!`) прекрасно работают с условными выражениями.

37. Назовите термин, которым обозначается единичное прохождение цикла.

38. Является ли следующий код правильным (проверьте в консоли!), и если нет — почему?

```
function достаточноЛиДенег(вКошельке) {
     const цена = 10.30;
    if ( вКошельке > цена) {
      console.log('Вам хватит денег!');
    } else {
      console.log("Боюсь, вы не можете себе этого позволить!");
   }
}
достаточноЛиДенег(10.30);
```

39. Сколько сообщений вернётся в консоль после запуска следующего кода?

```
let x = 10;
while (x > 0) {
    console.log('значение x равно ' + x);
    x++;
}
```

40. Является ли следующий код правильным (и если нет — почему?):

```
function сократитьДо3Знаков(число) {
    return число;
    const точноЧисло = число.toFixed(3);
    console.log('Более точная сумма = $' + точноеЧисло);
    return точноеЧисло;
}
сократитьДо3Знаков(25.98765);
```

41. Сколько сообщений вернётся в консоль после запуска следующего кода?

```
const цвета = ['красный', 'жёлтый', 'синий', 'оранжевый', 'зелёный',
'фиолетовый'];
for (let i = 0; i < цвета.length; i++) {
    const цвет = цвета [i];
    if (цвет.length >= 7) {
        console.log(цвет);
    }
}
```

42. Истина/Ложь: `!!('умный' && 'красивый');` // выдержка из моего резюме.

43. Назовите термин, обозначающий код, в котором все функции и переменные обладают «говорящими» названиями (благодаря чему разработчику предельно просто взаимодействовать с написанным таким образом кодом).

44. Перед «правкой» он вернёт `false`, а перед «ложкой» — `true`?

45. Истина/Ложь: операторы сравнения всегда возвращают булевы значения.

46. `null`, `false`, `' '`, `0` и `undefined` являются _____; а `true`, `'строка'` и `1` — _____.

Сделай сам: странная аллергия

Звонил ваш лечащий врач — сказал, что пришли результаты анализов. Не хочу вас расстраивать, но у вас обнаружилась крайне редкая форма пищевой аллергии. Судя по всему, у вас аллергия на все сладости, оканчивающиеся на «s»… Какая досада, а вы ведь только собирались посидеть-расслабиться после теста в компании груды сладостей, заработанных честным трудом на Хэллоуин (вы так ждали этого момента со времён проекта «Сделай сам» из прошлой главы)!

Но тем не менее. В этом проекте я бы хотел, чтобы вы взяли список сладостей, полученных на Хэллоуин, и при помощи цикла вывели бы для каждой сообщение в консоль, относительно её опасности или же безопасности для вас. Чтобы далеко не ходить, вот вам список: Nuts, Hershey's, Kit-Kat, Milky Way, M&M's, Snickers, Starburst и Bounty. Напишите функцию, которая поочерёдно выдаст 8 сообщений (по одному на каждый пункт списка), гласящих, сможете или не сможете вы съесть ту или иную сладость. И, как обычно, если у вас возникнут трудности — обращайтесь к моим советам в конце книжки.

10 Играем в «виселицу»

Неблизкий путь вы проделали, чтобы добраться сюда! Всё, что вы почерпнули из этой книжки, относится в первую очередь к JavaScript; однако оно также приложимо и ко многим другим языкам программирования. Параллельно с тем, чтобы учиться на начальном уровне программировать на JavaScript, вы также получили базовые навыки работы с Ruby, Python, PHP, Java, C# и многими другими языками, на которых написаны многие популярные веб-сайты и которые используются в крупных международных компаниях. Если не сбавлять оборотов, впереди вас ожидает весьма и весьма светлое карьерное будущее!

До сих пор мы с вами напряжённо работали, в каждой главе учили что-то новое. Теперь учёба закончилась, и… вы заслужили переменку! Что ещё делать на перемене, как не играть? Давайте же повеселимся! Словом, в этой последней главе, клянусь, ничему новому я вас учить не буду. Вместо этого мы применим уже имеющиеся у нас навыки для того, чтобы сыграть в «виселицу»!

Вы знаете, как играть в «виселицу»? Для тех, кто никогда не играл, я расскажу. В неё частенько играют, например, в дороге — в автобусе там или в поезде. Правила игры такие: загадывается случайное слово, а затем его пытаются угадать, одну за другой предлагая буквы. Каждая неверная догадка приближает проигрыш (а также повешение невинного человека!). Сможете ли вы отгадать слово, прежде чем случится непоправимое?!

Я буду рассуждать вслух, так что, думаю, вам не составит труда следовать за мной и самостоятельно написать игру. На всякий случай полный код моей версии игры находится в секции ответов в конце, так что если вы застряли, то можете обратиться к моему коду и с его помощью выловить «жучков» в своём. Если какая-то часть останется неясной, то просто прилежно скопируйте её, чтобы потом, увидев конечный вариант, вернуться к ней и, быть может, тогда понять её лучше.

И помните: вы уже знакомы с идеей каждой самой крошечной строчки в коде этой игры! Без исключений, даже не сомневайтесь, мы всё это проходили. Надеюсь, как только мы приступим к созданию игры, вы ощутите новые суперспособности, которыми теперь обладают ваши пальцы. Подумайте сами: сколькие из ваших приятелей смогли бы с нуля написать рабочую игру, да так, чтобы ещё и понимать каждую строчку написанного кода?! Это же реально круто!

Подготовка

До сих пор по большей части все задания мы выполняли в консоли. В общем-то, и сейчас вы вполне можете, если желаете, всю игру писать также в консоли. Конечный вариант, конечно же, мы будем запускать в консоли, но в нём будет порядка 85 строчек кода. Поэтому я рекомендовал бы вам воспользоваться Рабочей тетрадью или просто каким-нибудь текстовым редактором, поскольку там вам будет проще вносить изменения и исправления.

Несколько раз я попрошу вас проверить наш код в консоли: всё что нужно — просто скопировать и вставить код в консоль. К настоящему моменту, я полагаю, вы, вероятно, уже успели познакомиться с техникой копирования/вставки (привет главе 1!); однако, если вдруг подзабыли, то я напомню:

1. Выделите мышкой нужный код.
2. При помощи CTRL+C (или же COMMAND+C на Mac) скопируйте его.
3. Откройте консоль и при помощи CTRL+V (или же COMMAND+V на Mac) вставьте его туда.

Важное замечание: при копировании кода из Рабочей тетради или текстового редактора очень важно, чтобы были ОТКЛЮЧЕНЫ «умные кавычки» в настройках Google Docs, Microsoft Word и так далее (см. главу 1). *По умолчанию* автозамена («умные») кавычек зачастую включена во всех программах. Если не *отключить* эту функцию, то ваш код попросту не запустится.

Что ж, довольно приготовлений! Давайте уже повеселимся!

Начальная функция: ответыДляВиселицы()

Для нашей игры мы создадим три отдельные функции (с названиями, как обычно, в верблюжьемРегистре). Мы проверим их все индивидуально, а затем

соединим вместе. Первая из них будет генерировать слова. Переписывайте код в Рабочую тетрадь или перепечатывайте в текстовый редактор, а затем проверяйте в консоли:

```
function ответыДляВиселицы() {
    const ответы = [
        'взаимность',
        'претендент',
        'секундомер',
        'автомобиль',
        'авантюрист',
        'облачность',
        'наблюдение',
        'штукатурка',
        'укротитель',
        'виолончель',
        'заговорщик',
        'клубничник',
        'антарктика'
    ];
    const случайныйИндекс = Math.floor(Math.random() * ответы.length);

    return ответы[случайныйИндекс].toUpperCase();
}
```

Если код перенесён верно, то функция должна выбирать из списка случайное слово и возвращать его прописными буквами, вроде `"ЗАГОВОРЩИК"`, `"ВИОЛОНЧЕЛЬ"`, `"АНТАРКТИКА"` и так далее. Проверьте сами:

```
ответыДляВиселицы();
ответыДляВиселицы();
ответыДляВиселицы();
```

Надеюсь, в этой функции вам не встретилось ничего непонятного, но на всякий случай я её поясню. Но перед этим я бы хотел, чтобы вы сами ещё раз пробежались по коду, чтобы уж точно знать, что всё ясно. Если так, то прекрасно! Если же нет — не расстраивайтесь, сейчас во всём разберёмся. Ведь мы же просто хотим чуть-чуть повеселиться, в конце концов. Итак, следите внимательно.

- В первой части нашей функции (начиная с `const ответы`) мы создали обычный массив, состоящий из списка слов для игры. В списке 13 слов, в каждом из которых по 10 букв; в них нет ничего особенного, так что если хотите — придумайте свои. Я бы даже советовал вам сделать это, а может, даже и расширить список раза в полтора-два. Вы можете использовать какие угодно слова, но игра будет интереснее, если они будут простыми, но при этом не очень короткими.
- Во второй части (с `const случайныйИндекс`) генерируется случайное целое число в диапазоне от 0 до 12 (то есть все варианты индексов в нашем массиве). Как вы помните из главы 8, если любое из этих чисел поместить в квадратные

скобки, то вы получите в ответе соответствующий элемент массива (например, `ответ[1]` ответит вам `'претендент'` и так далее).

- Наконец, в последней части (`return`) берётся (при помощи `случайногоИндекса`) случайный элемент из массива, переводится в прописные буквы (`toUpperCase()`) и возвращается в консоль. Проще простого!

Всё кончено: конецИгры()

А теперь мы перескакиваем сразу в конец! Напишем функцию для `концаИгры`, которая будет выдавать нашему игроку (пользователю) сообщение об этом. Мне кажется такой подход очень продуктивным, когда создаёшь код для игры: сразу написать код для её окончания, прежде чем начинать копаться во всех игровых внутренностях. Причина проста: когда вы пишете игру, чётко понимая, чем всё заканчивается, вам намного проще писать и проверять код для всех промежуточных стадий. Когда же такого понимания нет, то в какой-то момент можно обнаружить, что уйма сил и времени была потрачена не на то, что надо; а когда в конце что-то вдруг пойдёт не так, то придётся возвращаться далеко назад, чтобы всё поправить (и ещё потратить опять же силы и время на поиски этого «всего»).

Так что мы сразу напишем завершающую игру функцию. Она будет очень простая. Но я всё же поясню пару-тройку моментов; начну, пожалуй, с `asciiВиселицы`. Это просто небольшая ASCII-картинка[1] (то есть картинка, нарисованная при помощи символов) вроде того мишки и главы 4, если помните. Запоминать ничего не нужно, просто аккуратно перепечатайте все символы, а затем вставьте код в консоль:

```
function конецИгры(ответ, выигрыш) {
    const asciiВиселица = '____\n|/ |\n| @\n| /|\\\n| / \\\n|\n=====';
    let сообщение = '';
    if (выигрыш) { // если параметр 'выигрыш' отсутствует, то будет
                   // "ложка" (undefined)
        сообщение = 'ТЫ ВЫИГРАЛ!';
    } else {
        сообщение = 'ВСЁ КОНЧЕНО\n\n' + asciiВиселица;
    }
    сообщение += '\n\nПравильный ответ: ' + ответ + '!';
```

[1] ASCII — *American standard code for information interchange*, таблица сопоставлений символов и числовых кодов. Была разработана в США в 1963 году.

```
    alert(сообщение);
    return сообщение;
}
```

Функция имеет два параметра. Первый — `ответ` — является, очевидно, правильным ответом. Второй же параметр — `выигрыш` — ожидает булева значения, сообщающего, удалось ли игроку выиграть или нет. Как вы помните из главы 4, если вызвать функцию, не указывая какой-либо из аргументов (или оба), то этому неуказанному аргументу (или же вообще всем) будет автоматически присвоено значение `undifend`. А поскольку `undifend` является «ложкой», то он и будет вести себя, как и подобает обычным ложным (`false`) булевым значениям! Давайте проверим наш код:

```
конецИгры('АВАНТЮРИСТ', true); // ТЫ ВЫИГРАЛ!
конецИгры('ОБЛАЧНОСТЬ', false); // ВСЁ КОНЧЕНО
конецИгры('ПУШИСТЫЙКРОЛИК'); // ВСЁ КОНЧЕНО
const временноеРешение = ответыДляВиселицы();
конецИгры(временноеРешение, true); // ТЫ ВЫИГРАЛ!
конецИгры(временноеРешение); // ВСЁ КОНЧЕНО
конецИгры(ответыДляВиселицы(), true); // ТЫ ВЫИГРАЛ!
конецИгры(ответыДляВиселицы()); // ВСЁ КОНЧЕНО
```

Разберёмся!

- С ASCII мы уже разобрались.
- Следующие 7 строчек создают `сообщение`, которое получит ваш пользователь. Если функция вызвана с аргументом `true` для параметра `выигрыш`, то последует `"ТЫ ВЫИГРАЛ!"`, если же `false` — то пользователь увидит грустное `"ВСЁ КОНЧЕНО"` и картинку.
- Поскольку вторая часть `сообщения` остаётся неизменной, вне зависимости от выигрыша или проигрыша, мы используем *комбинированное присваивание* (помните? если нет — освежите в памяти главу 2!), чтобы присоединить её к первой. Обратите также внимание на `\n`, при помощи которой мы очень ловко создали новую строчку.
- Оставшиеся строчки просто выводят наше `сообщение` в `alert()`, а затем возвращают (`return`) его в консоль. Быть может, вы в недоумении? «Разве Джереми не говорил нам, что использовать `alert()` — это ужасно и дурно?! Разве он не говорил, что пользователей раздражает эта функция?!» И всё такое. Хм... По правде сказать, мы собираемся использовать здесь все три «раздражающие» функции — и `alert()`, и `confirm()`, и `prompt()`. Я бы, конечно, не стал их применять в собственном коде, но ведь моей целью было помочь вам написать игру, используя лишь те навыки, которые нам уже известны. А эти три функции являются пока единственным известным нам способом взаимодействия пользователя и программы. В следующей книжке я, конечно, покажу вам несколько куда более элегантных способов сделать это.

Главная функция: сыграемВВиселицу()

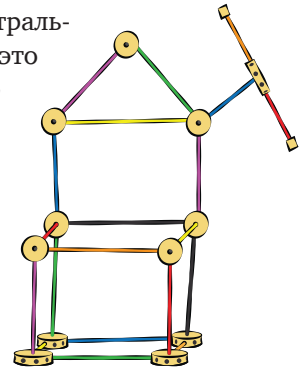
Вот наконец мы и подобрались к нашему главному блюду! Мы напишем главную функцию в несколько этапов, и в конце каждого из них будем весь написанный код проверять на работоспособность (несмотря на то, что сперва игра будет вести себя так, будто выиграть невозможно). Внимательно следите за тем, как будет устроен процесс нашей работы над этой функцией, так как примерно таким образом вы будете подходить к любой мало-мальски серьёзной программистской задаче в будущем.

Структура

Как обычно, набирайте код в Рабочей тетради или редакторе, а затем вставляйте его в консоль. Не забывайте о `// планах:`, потому что они будут указывать нам, какие и где блоки кода мы должны будем разместить.

```
function сыграемВВиселицу() {
    const ответ = ответыДляВиселицы();

    //план: сюда пойдут наши главные переменные

    //план: здесь будет подтверждающее сообщение

    //план: цикл игры должен быть тут

return конецИгры(ответ, false); // если игрок видит это, значит, игра
                                // окончена
}
```

Итак, вот структура, можно даже сказать — скелет, центральной функции нашей игры. Впрочем, на данный момент это «скелет» почти без «мяса»; всё, что она пока делает, — это то же, что мы делали ранее вручную:



- Генерирует ответы.
- А после вызывает функцию `конецИгры`, предлагая ей ответ в качестве первого аргумента.

Попробуйте сами в консоли:

```
сыграемВВиселицу();
```

Обратите внимание, что пока никаких ошибок в коде у нас нет. Если потом вы что-то добавите и код будет выдавать ошибки или вообще странно работать, попробуйте вернуться к этому этапу, когда всё гладко работало. Поверьте, это

избавит вас от многочасовой головной боли выискивания пары крошечных ошибок!

Переменные

Заменим наш `// план: сюда пойдут наши главные переменные` на настоящие переменные!

```
const буквыОтвета = ответ.split('');
const неверная Догадка = [];
const максНеверныхДогадок = 7;
const прогрессИгры = '_'.repeat(ответ.length).split('');
```

Если закончили, тогда выделите целиком функцию `сыграемВВиселицу` в Рабочей тетради или редакторе (начиная со слова `function` и заканчивая закрывающей фигурной скобкой), скопируйте и вставьте в консоль. Итак, теперь наша функция может:

- Генерировать `ответ` и присваивать его константной переменной.
- Создавать массив из отдельных символов (к этому мы вернёмся позже, когда дадим возможность игроку угадывать буквы).
- Создавать пустой массив, чтобы сохранять в него неправильно угаданные буквы.
- Определять количество `неверныхДогадок`, которое есть у игрока, прежде чем он проиграет. Этот показатель можно менять, чтобы усложнить или, наоборот, облегчить игру.
- Создавать массив из нижних подчёркиваний (`_`), равный по количеству элементов количеству букв в ответе (то есть, если вы используете мои слова, — 10, поскольку все возможные ответы состоят из 10 букв). С этого будет начинаться игра. Далее по ходу игры подчёркивания будут заменяться на правильно угаданные буквы.

Чтобы уж точно убедиться, что всё по-прежнему работает, давайте проверим нашу функцию:

```
сыграемВВиселицу();
```

Устранение ошибок

Надеюсь, наша функция пока работает безошибочно… А что, если бы она работала не столь успешно? Что, если бы вдруг выскочила какая-то неожиданная ошибка? Или ещё хуже — что, если бы никаких ошибок даже и не было, но написанный код просто-напросто работал бы не так, как ожидалось? В подобном случае вам необходимо было бы провести **устранение ошибок** (то есть найти причину неполадки и исправить её).

Ну, во-первых, если ошибка всё же выскочила, то нужно использовать информацию из сообщения, чтобы поправить дело (зачастую это — кратчайший путь). Если не получается, то стоит сравнить каждую строчку своего кода с кодом в книжке. Также не забудьте проверить, отключена ли опция автозамены кавычек («умных» кавычек), и вообще проверить все кавычки, скобки, апострофы и так далее.

Ещё можно попробовать такой способ: заключите какой-нибудь блок кода в комментарии, чтобы проверить, как будет работать остальная часть кода после этого. Постарайтесь до максимума упростить ваш код, добавляя (то есть «раскоменчивая») одну за другой более сложные части. В конце концов, обнаружится одна или две строчки, после которых ваш код не запустится, — в них-то и нужно будет очень внимательно искать причины всех ваших бед! Надеюсь, этот небольшой «ликбез» по диагностике поможет вам в случае возникновения проблем. Очень хорошо, если сейчас он вам вовсе не понадобится (если ваш код и так работает отлично), но если что — теперь вы знаете, что делать и куда вернуться, чтобы узнать, если вдруг забудете.

Подтверждение

Едем дальше! Вместо комментария `//план: здесь будет подтверждающее сообщение` вставьте следующий блок кода:

```
const подтверждение = confirm("Давай сыграем в "виселицу"!\n\n"
    + "Я задумал какое-то слово; можешь начинать отгадывать буквы!\n"
    + "Это самое обычное слово из " + ответ.length + ' букв.\n'
    + 'Ну что, начнём?');
if (!начнёмИгру) {
    return конецИгры(ответ, false);
}
```

Чего мы добились:

- При помощи `confirm()` сделали диалоговое окно, через которое пользователь должен подтвердить начало игры. Как вы помните из главы 7, `confirm()` выдаёт всплывающее окно, в котором пользователь должен нажать либо OK , либо Отмена. После этого в консоль вернётся булево значение (`true`, если OK, или же `false`, если Отмена). Это возвращаемое значение мы затем присваиваем нашей переменной, чтобы потом запросить её.
- Блок с `if` проверяет, подтвердил ли (`true` или `false`) пользователь начало игры. Как вы помните из главы 6, когда перед чем-либо стоит логический оператор `!` — оно обратится в противоположное. Таким образом, если (`if`) пользователь нажал на Отмена, то будет запущен код, заключённый в фигурные скобки.

- Раз пользователь отменил начало игры, ему можно сразу же выдать сообщение `конецИгры` и послать `false` в качестве второго аргумента функции. То есть если пользователь не выиграл, то он в любом случае увидит `"ВСЁ КОНЧЕНО"`. (Кстати, это сработает, даже если вовсе не упоминать второй аргумент — `return конецИгры(ответ);`.)

И снова: скопируйте обновлённую функцию `сыграемВВиселицу()` из Рабочей тетради или редактора и вставьте в консоль. Проверьте, всё ли правильно работает, и если нет — попытайтесь исправить ошибки. В появившемся диалоговом окне выберите Отмена:

```
сыграемВВиселицу();
сыграемВВиселицу();
```

Если вы любопытный человек, то, вероятно, вы попробовали нажать кнопку OK в диалоге `confirm()`. Если так, то вы уже видели, что интерпретатор «перескочил» через первый блок `if`, а вы всё равно получили безрадостное сообщение `конецИгры`. Это произошло просто потому, что мы ещё не написали остальную часть игры. Не волнуйтесь, всё в порядке.

Раз мы чётко определили то, что должно произойти, если (`if`) пользователь выбирает Отмена, то вы, должно быть, ожидаете появления блока `else`, чтобы определить то, что должно происходить, если он этого *не* сделает, так? Ну, раз так, тогда я огорошу вас: блок `else` здесь не очень-то и необходим. В чём причина? Она кроется в строчке с `return`! Если условие при `if` окажется истинным (`true`), тогда будет запущен `return`, а *после него (как мы уже упоминали выше) больше никакая часть кода запущена быть уже не может*. Таким образом, всё, что мы поместим после этого, может быть запущено, только если условие при `if` окажется ложным (`false`)! Вот и получается, что в блоке `else` действительно нет нужды! Функция и ведёт себя так, будто бы остальная часть кода уже заключена в гигантский блок `else`. В общем-то, тут нет ничего нового, но, быть может, поможет взглянуть на старое с новой точки зрения.

Цикл игры

Что интересно: все компьютерные игры построены на циклах! Смотрите сами: игроку показывают некоторую ситуацию, затем берут результат этой ситуации (нажатие на какие-нибудь клавиши, сделанный выбор и тому подобное), изменяют ситуацию согласно этому результату (порой едва заметным образом), вновь предлагают её игроку, вновь берут результат и так далее. Циклы в игре запускаются тысячи, быть может даже миллионы раз, вплоть до

ВНИМАНИЕ:
БЕСКОНЕЧНЫЙ ЦИКЛ

момента, когда необходимое для выхода из цикла будет исполнено. Скажем, у игрока кончились жизни (и игра на том и закончилась) или, напротив, — игрок собрал все звёздочки и выиграл. А может, он просто вышел из игры. Так или иначе, модель одна и та же: цикл-за-циклом-за-циклом-за… пока что-нибудь не прекратит цикл.

То, что я сейчас расскажу, поможет вам лучше понять, как именно подступиться к созданию этой или любой другой игры. Вот что нам необходимо сделать в нашем игровом цикле:

1. Построить цикл, который бы продолжался до того момента, пока пользователь не совершит критическое количество ошибок (`невернаяДогадка.length < максНеверныхДогадок`). При каждой итерации количество ошибок будет заново проверяться, и в случае перебора игра будет закончена.
2. Наш цикл должен показывать `прогрессИгры` (например, «_ _ _ _ _ _ _ _ _ _» или «A_TO_O_ИЛЬ» и так далее).
3. Также цикл будет показывать игроку его `неверныеДогадки` (чтобы он не повторял ошибок).
4. И наконец, цикл будет просить игрока выбрать букву:
   а) если просьба отклонена, то цикл закончен и происходит `конецИгры`;
   б) если игрок выбирает букву, то всё продолжается по новой…

С остальным мы ещё разберёмся далее; пока остановимся на следующем блоке кода (поместите его вместо комментария `//план: цикл игры должен быть тут`):

```
while (невернаяДогадка.length < максНеверныхДогадок ) {
    const сообщениеПрогресса = 'Пока ты добился: \n'
        + прогрессИгры.join(' ') + '\n'
        + 'Ошибки: [' + невернаяДогадка.toString() + ']\n\n'
        + 'Выбирай следующую букву!';
    const ответИгрока = prompt(сообщениеПрогресса);

    if (!ответИгрока) {
        return конецИгры(ответ);
    }

    //план: здесь будут догадки игрока
}
```

Вновь скопируйте нашу функцию целиком и проверьте, всё ли работает. Пока что выиграть невозможно, но и бесконечного цикла получиться нигде не должно. В любой момент должен быть возможен выход из игры по нажатию на Отмена или же пустому полю в окне `prompt()`.

```
сыграемВВиселицу();
```

Итак, объединив (как мы делали ещё в главе 3) несколько строк в одну, мы создали `сообщениеПрогресса` для игрока. Затем строку `сообщениеПрогресса` мы предложили встроенной функции `prompt()`, которая будет показывать его игроку во время каждой новой итерации. Если пользователь решит оставить поле ввода пустым или же нажмёт Отмена, то наступит `конецИгры`.

Всё ли пока понятно? Если не всё, то я бы советовал вам продержаться до конца, чтобы потом вы смогли попробовать в работе готовый код. Подчас охватить взглядом картину целиком, со всем обилием деталей — дело вовсе не лёгкое. Поэтому, увидев готовый код, попробовав саму игру, просто вернитесь вновь к этому месту и, быть может, вы поймёте то, что ускользало от вас прежде.

Итак, что у нас дальше на очереди?

5. Нам нужно взять предложенную игроком букву и присвоить её переменной `догадка`.
6. А затем проверить — есть ли она (`догадка`) в ответе:
   а) если найдётся как минимум одно совпадение — нужно считать догадку игрока верной (то есть не добавлять её к `неверныхДогадкам`);
   б) после каждой отгаданной буквы нужно обновлять `прогрессИгры`.

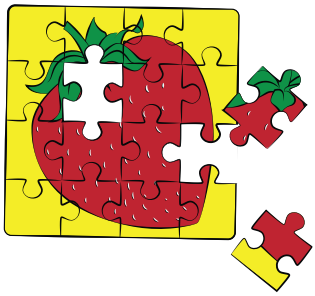
Опять же, остальное — чуть позже. Пока займёмся перечисленными пунктами. Следующим кодом замените комментарий `//план: здесь будут догадки игрока`:

```
const догадка = ответИгрока.toUpperCase();
let верная Догадка = false;

for (let i = 0; i < буквыОтвета .length; i++) {
    if (буквыОтвета [i] === догадка) {
        верная Догадка = true;
        прогрессИгры[i] = догадка;
    }
}
//план: здесь будет условие выбора верной/неверной догадки
```

Уверен, что вы и без меня тут всё поняли, но я всё же бегло поясню:

- Мы перевели ответ игрока в верхний регистр при помощи `toUpperCase()`.
- Создали новую переменную `верная Догадка` (при помощи `let`, поскольку она должна быть изменяемой) с булевым значением и присвоили ей значение по умолчанию — `false` (ниже мы зададим параметры смены значения на `true`, в случае если в догадке игрока будет буква, содержащаяся в ответе).
- Вслед за этим идёт цикл с `for` (как мы делали в главе 9), который как раз и будет искать букву из `догадки` в ответе, «пробегая» по всему слову в её поисках.

- Если найдётся как минимум одно совпадение, то значение `верноДогадки` станет `true`, а также будет внесено изменение в массив `прогрессИгры`, чтобы нижнее подчёркивание (`"_"`) сменилось на отгаданную букву!



Ну, например, правильный ответ — «КЛУБНИЧНИК», и на данный момент массив `прогрессИгры` имеет следующий вид: ['К','Л', '_', '_', 'Н', '_', '_', '_', '_', 'К']. Если игрок сейчас предложит в догадке букву «И», то цикл «пробежит» все буквы в «КЛУБНИЧНИКЕ» (то есть в ответе), чтобы сравнить каждую с `догадкой`. Когда цикл обнаружит совпадение на 5-й позиции (значение переменной `i` будет `5` на тот момент), то он запустит строчку `прогрессИгры[2] = "И"`. То же произойдёт и с элементом `8` — `прогрессИгры[8] = "И"`. Когда цикл завершит работу, то окончательный вид массива `прогрессИгры` будет таким: ['К','Л', '_', '_', 'Н', 'И', '_', '_', 'И', 'К'].

Давайте проверим имеющийся у нас код (хотя наша игра всё ещё не даёт ни единого шанса выиграть!) — всё должно работать без ошибок, показывая `прогрессИгры` и всё прочее:

```
сыграемВВиселицу();
```

Ну как? Надеюсь, всё хорошо. Что же дальше? А дальше нужно доработать алгоритм выбора `верноДогадки` (и, соответственно, не верной). Эта переменная, вы помните, по умолчанию имеет значение `false`, а при совпадении догадки с ответом меняется на `true`. Приступим.

7. Нам нужно, чтобы код игры проверял булево значение `верноДогадки` на предмет совпадения `догадки` с ответом:
   а) если значение `true`, то:
      i) следует проверить, не дал ли игрок правильный ответ целиком;
         - если так, то следует выдать `return` сообщение `конецИгры` `"ТЫ ВЫИГРАЛ!"`;
         - в противном случае игра продолжается...
      ii) выдать окно `alert()` с сообщением о правильной `догадке`;
   б) если значение `false`, то:
      i) следует добавить `догадку` в массив `неверныхДогадок`;
      ii) выдать окно `alert()` с сообщением о неправильной `догадке` и о том, сколько ещё раз игроку дозволено ошибаться.

Теперь давайте выразим всё то же в виде кода. Заменим следующим кодом комментарий `// план: здесь будет условие выбора верной/неверной догадки`:

```
if (вернаяДогадка) {
    if (прогрессИгры.join('') === ответ) {
        return конецИгры(ответ, true);
    }

    alert('Верно!');
} else {
    неверныхДогадка.push(догадка);
    alert('Извини, буквы ' + догадка + ' тут нет.\nТы можешь ошибиться
           ещё '
        + (максНеверныхДогадок  - неверныхДогадка.length) + ' раз,
           прежде чем\n этот бедняга будет повешен.');
}
```

Попробуйте сами внимательно вчитаться и разобраться в этом коде. Уверен, что у вас получилось! Но на всякий пожарный случай я оставлю тут пояснения (вдруг всё же что-то из них окажется полезным?). Надеюсь, вы уже чувствуете ритм работы с программным кодом, да? Эти повторяющиеся раз за разом этапы и всё прочее… Так оно и происходит на самом деле!

- Если переменная `верныхДогадка` получает значение `true`, то:
  - Пользуясь условием при `if`, сравнить `ответ` с `прогрессомИгры`. Так как мы сравниваем строку с массивом, нам здесь помогает метод `.join()` с пустой строкой в скобках. Она служит для того, чтобы при собирании элементов массива в строку никакая другая строка не «вклинилась» между ними:
    - если был дан ответ целиком, то следует сразу перейти к строчке `return` из функции `конецИгры` и выдать версию сообщения `"Ты выиграл!"` (если не помните, о чём речь, — просмотрите заново код функции `конецИгры`);
    - если ответ был дан не целиком, но буква угадана верно — то игра продолжается…
  - Следует вывести `alert()` с сообщением, что `догадка` игрока была `верной-Догадкой`.
- Если же `верныхДогадка` — `false`, то:
  - При помощи `.push()` добавить `догадку` к массиву `неверныхДогадок` игрока. Перед каждой итерацией `while`-цикла длину массива `неверныхДогадок.length` нужно будет сравнить с `максНеверныхДогадок` при помощи оператора сравнения «меньше чем» из главы 5; если максимальное значение (`.length`) будет превышено, то цикл будет прерван, наступит `конецИгры` и появится конечное `return`-сообщение о проигрыше.
  - Следует вывести `alert()` с сообщением, что `догадка` была неверной, и количеством оставшихся шансов на ошибку, прежде чем [предположительно] ни в чём не повинный бедняга будет жестоко повешен.

И вот, друзья мои, ЭТО И ЕСТЬ ФИНАЛЬНЫЙ и распоследний блок нашего кода для игры! Скопируйте и вставьте его и, наконец, попробуйте с удовольствием сыграть в виселицу!

```
сыграемВВиселицу(); // попробуйте все-все возможные сценарии, испытайте
                    // код!
сыграемВВиселицу(); // попробуйте просто отгадать слово
сыграемВВиселицу(); // попробуйте отказаться от игры в самом начале
сыграемВВиселицу(); // попробуйте ответить неверно максимальное число раз
сыграемВВиселицу(); // попробуйте прервать игру на середине
сыграемВВиселицу(); // дайте сыграть папе/маме/брату!
```

Возможные улучшения

Итак, наша «виселица» готова! Если код отказывается работать как положено, то попробуйте воспользоваться моими советами по устранению неполадок. Если и это не сработает, загляните в раздел ответов в конце — там я разместил полный вариант кода игры. Строчку за строчкой сравните ваш код с моим. Думаю, вам удастся найти недостающий или неверно прописанный элемент кода.

Но если у вас всё получилось, то примите мои поздравления! Я бы хотел, чтобы вы теперь ещё раз пробежались по коду и убедились, что вам всё в нём понятно. Это важно, поскольку эта игра охватывает очень много (почти все) тем, которые мы разбирали в предыдущих главах. Честно сказать, я даже и не планировал писать в конце игру, она как-то сама написалась из желания скомпоновать изученные темы в один проект. Но ведь именно так и работают языки программирования! Они вообще весьма похожи в этом на человеческие языки; подумайте сами: только сегодня вы уже множество раз использовали слова, которые выучили в самые разные периоды жизни!

Так вот, если игра безошибочно работает и вы всё понимаете, можете считать дело оконченным! Ну а если вам понравилось и в вас проснулся этот прекрасный голод до новых свершений, то я набросал небольшой список улучшений, которые вы (если захотите) можете внести в код игры. Вдруг бы вам захотелось воспользоваться моими советами? Словом, вот список того, что можно добавить или изменить:

- добавить десяток-другой новых слов в ответы;
- изменить текст сообщений, выдаваемых игроку;

- добавить полезные комментарии к коду;
- добавить ASCII-картинку в разные места кода;
- создать собственную ASCII-картинку на случай выигрыша;
- улучшить метод восприятия догадок игрока на случай, если было введено более одного символа (например, принимать только первый из них?);
- и вообще всё, что только вам в голову придёт!

В общем, эта «виселица» теперь полностью ваша, так что распоряжайтесь ею как заблагорассудится! Желаю вам от души повеселиться!

Послесловие

Надеюсь, вы с удовольствием попробовали в деле свои новые суперспособности. Но я также надеюсь, что на этом вы не остановитесь и продолжите расти и развиваться. Эта книжка познакомила вас с основами JavaScript, но грамотный программист обязан хоть немного, но знать и другие языки программирования. Словом, это только начало — начало долгого пути, полного захватывающих приключений и замечательных открытий!

А вы, кстати, заметили, что на протяжении всей книжки не было ни единого нормального изображения? Мы перелопатили целую тонну кода, наделали кучу всевозможных функций, но не делали ничего, так сказать, «в цвете»! Мы не создавали веб-страниц, никуда не встраивали видео, короче — ничего, что можно было бы просто отправить друзьям по e-mail. Думаете, оттого, что это слишком трудные темы для начала? Ничуть не бывало!

С вашими новыми суперспособностями вы с лёгкостью освоите всё это и куда больше в мгновение ока! Мы не касались подобных тем просто потому, что они более связаны с написанием кода на HTML (Hypertext Markup Language) и CSS (Cascading Style Sheets). Обладая навыками программирования на JavaScript, вы, несомненно, сможете самостоятельно освоить и их! Все разработчики в один голос соглашаются, что они намного проще в изучении, чем JavaScript.

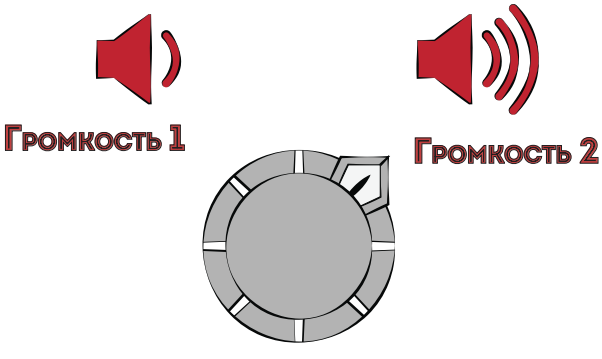
«А почему же, — спросите вы, — если они проще, мы с них не начали?» Ну, пожалуй, я бы мог начать и с них, ведь на начальных этапах изучения сами принципы работы и идеи у всех трёх в общем-то схожи. Кто-то, наоборот, начинает с HTML или CSS. Мне был ближе подход, в котором я сперва преподам основы JavaScript, так что мы с него и начали.

Основной моей целью стояло научить вас думать и действовать как настоящий программист. Мне хотелось, чтобы вы умели на практике применить полученные знания и, я уверен, что лучше, чем язык JavaScript, с этим не справился бы никакой другой.

Но изучить HTML и CSS вам всё равно, скорее всего, придётся, чтобы вы смогли писать целые веб-сайты. Но не забывайте и о JavaScript: применяя его в связке с другими языками, вы непременно сможете сделать что-нибудь небывалое и интересное!

Советую вам ознакомиться со второй частью этой книжки, в которой я как раз буду объяснять основы HTML и CSS. Там я наглядно покажу, насколько круто JavaScript работает в связке с этими языками программирования. Мы будем писать веб-сайты, работать с изображениями, и вы сможете сохранять и делиться своей работой с семьёй и друзьями! В общем, сможете с чистой совестью проститься с этим прискорбным ощущением: «Ну да, был неплохой проект, но ведь я всё делал в консоли, и оно уже куда-то пропало...»

Короче, если эта книжка вам пришлась по душе, то понравится и продолжение! Но даже если и нет, всё равно — обязательно продолжайте обучение и развивайте навыки программирования. Ни в коем случае не останавливайтесь на достигнутом! Надеюсь, вы продолжите начатое увлекательное путешествие, в котором, как я уже упоминал, вас ожидает ещё множество всего поразительного и удивительного!



Ответы

Глава 1

Викторина

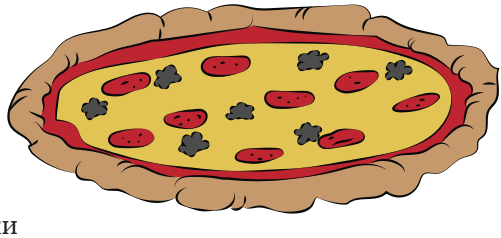
1. `about:blank`.
2. CTRL+SHIFT+J (COMMAND+OPTION+J для Mac).
3. `let` и `var`.
4. `;`.
5. `/`.
6. Скобки ходят парами: всегда есть и открывающая, и закрывающая!
7. SyntaxError (точка с запятой (`;`) стала для интерпретатора неожиданностью. Она должна быть справа от закрывающей скобки).
8. верблюжийРегистр.
9. Скобки (ещё подошёл бы ответ «кавычки»!).
10. Ошибка в использовании `let` во второй строчке кода.
11. `=` означает присваивание: когда переменной присваивается некое значение (к примеру — `let мойВозраст = 10;`).

Упражнения

I. Введите следующие правильные фрагменты кода в консоль.

1. `"пепперони и сосиски"`.
2. `21`.
3. `4`.
4. `12`.

5. `14`.
6. `"6 футов 7 футов"`.
7. `"Работаю с 9 до 5"`.
8. `"73"` `//` ведь «3» — строка! Так что 7 с ней просто «сцепилась» (то есть объединилась), а не математически прибавилась к ней.
9. `Infinity` (то есть бесконечность).



II. Что не так с каждым из фрагментов?

1. Нужно сперва объявить переменную с ключевым словом `var` или `let`; только после этого её можно будет использовать.
2. Здесь не хватает закрывающей кавычки (`"`).
3. Одиночный знак равенства означает *присваивание*; для присваивания необходимо, чтобы слева от знака равенства была лишь одна переменная (например, `let x = 3 + 5;`). Также необходимо объявить переменную при помощи `let` или `var`, прежде чем её можно будет использовать.
4. SyntaxError. Точка с запятой должна быть расположена справа от закрывающей скобки.
5. Отсутствует закрывающая скобка (или поставлена лишняя открывающая).
6. Отсутствует открывающая скобка (или поставлена лишняя закрывающая).
7. Одиночный знак равенства означает *присваивание*; для присваивания необходимо, чтобы слева от знака равенства была лишь одна переменная (например, `let x = 15 - 12;`). Также необходимо объявить переменную при помощи `let` или `var`, прежде чем её можно будет использовать.
8. Здесь недостаёт точки с запятой (`;`) в конце (кстати, ошибку эта строчка на самом деле не выдаст).
9. Название переменной `аэтоневерблюжийрегистр` прописано **строчными** буквами; вместо этого нужно было использовать верблюжийРегистр — `аЭтоНеВерблюжийРегистр` (обратите внимание: это просто соглашение по оформлению кода; ошибки в консоли вы не увидите).
10. Название переменной `АЭТОЧТОЛУЧШЕ` прописано целиком в **ВЕРХНЕМ РЕГИСТРЕ** (прописными буквами); вместо этого нужно было использовать верблюжийРегистр — `аЭтоЧтоЛучше`.
11. Название переменной `а_может_быть_так` прописано в так называемом **змеином_регистре**. Но лучше всё же использовать верблюжийРегистр — `аМожетБытьТак`.

12. Название переменной `ВотВродеТакПохоже` прописано в так называемом **ПаскалевскомСтиле** (PascalCase). Вместо этого лучше использовать верблюжийРегистр: `вотВродеТакПохоже` (начиная со строчной буквы).
13. Название переменной `ну-вот-так-уж-наверное-точно` прописано в так называемом **шашлычном-регистре** (kebab-case). Это ГРУБАЯ ошибка, поскольку интерпретатор JavaScript будет расценивать дефис как знак минус! Вместо этого нужно было использовать верблюжийРегистр — `нуВотТакУжНаверноеТочно`.
14. Здесь не хватает кавычек — `"Вот так - хорошо!"` (а ещё лучше бы поставить в конце точку с запятой).

Рекомендации для «Сделай сам»

Вот возможный код для этого проекта (скопируйте и вставьте код в консоль):

```
let возрастПапы = 36;
let возрастМамы = 36;
let возрастАнджелы = 13;
let возрастТони = 11;
let возратсХармони = 8;
let возрастЧарити = 7;
let возрастЧейза = 5;
let возрастСимфонии = 0;
let суммаВозрастов = возрастМамы + возрастМамы + возрастАнджелы
+ возрастТони + возрастХармони + возрастЧарити + возрастЧейза +
возрастСимфонии;
let вНашейСемье = 8;
let среднийВозраст = суммаВозрастов / вНашейСемье;
среднийВозраст; [1]
```

Глава 2

Викторина

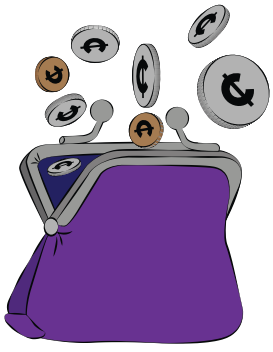
1. Плюс обозначает действие сложения.
2. `*`.
3. Число.

[1] Джереми, конечно, написал код для расчета среднего возраста своей семьи; все переменные здесь — члены его семьи (жена и дети).

4. Косая черта обозначает деление.
5. Число.
6. Скобки: `(` вот так `)`
7. `*=` (`мояПеременная *= 2;`)
8. `%` (знак процента).
9. `мояПеременная++;` (инкремент).
10. `мояПеременная--;` (декремент).
11. `3 * (4 + 1);`.
12. `2`.
13. `-=` (`моёЗначение -= 8;`).
14. Пять различных значений (`0`, `1`, `2`, `3` или `4`).
15. Комбинированное присваивание.
16. Оператор modulo (`%`). В нашем случае это будет выглядеть как `% 3`, поскольку у нас всего три возможных значения (`красный`, `зелёный` и `синий`).

Упражнения

I. Введите следующие правильные фрагменты кода в консоль.



1. `16`.
2. `5`.
3. `4`.
4. `6`.
5. `20`.
6. `7`.
7. `2`.
8. `2`.
9. `0`.
10. `18 // кстати: перед вами один из способов сократить любую дату до двузначной.`
11. `3`.
12. `1`.

II. Что не так с каждым из фрагментов?

1. Математические операции никогда не должны стоять слева от знака равенства (`=`), поскольку в JavaScript всё, что будет находиться слева, расценивается как присваиваемое значение.
2. Название переменной никогда не может быть числом (и даже начинаться с него не может!).
3. Технически предложение не является неправильным, но, по всей видимости, само рассуждение разработчика содержит в себе ошибку. Вероятнее всего, разработчик пытался написать примерно следующее:

   ```
   моёЛюбимоеЧисло = моёЛюбимоеЧисло + 6;
   ```

   или ещё лучше так:

   ```
   моёЛюбимоеЧисло += 6; (с тем же результатом)
   ```

4. Отсутствует закрывающая скобка.
5. Для присваивания значения используется одиночный знак равенства (`=`), а не двойной (`==`).
6. Переменные `делимое` и `делитель` нужно сперва объявить с присвоенным значением.
7. Отсутствует открывающая скобка.
8. `+*` не является оператором (для интерпретатора это просто два неких символа, написанные подряд и не имеющие никакого смысла).
9. Математические операции никогда не должны стоять слева от знака равенства (`=`), поскольку в JavaScript всё, что будет там находиться, будет расцениваться как присваиваемое значение. Кроме того, даже если поменять стороны местами, перед переменной `конечныйРезультат` всё равно недостаёт `let` или `var`.
10. `+=` нельзя использовать одновременно с ключевым словом `let`, которое означает, что новая переменная ещё только создаётся; таким образом, `+=` невозможно здесь использовать, поскольку переменная, которая только создаётся (но ещё не создана!), не может прибавить себя к себе!
11. Слева от одиночного знака равенства не должно быть никаких чисел и вообще действий (помните, что `+=`, `-=` и так далее также присваивают переменным значения).
12. Это, в общем-то, не ошибка, но более практично использовать `%` только с целыми числами (положительными или отрицательными — неважно) или с нулём. Впрочем, в некоторых случаях эти рекомендации вполне можно нарушить, но такие случаи крайне редки.

13. `+/` не является оператором (для интерпретатора это просто два неких символа, написанные подряд и не имеющие никакого смысла).

Комплексный обзор

1. верблюжийРегистр.
2. CTRL+SHIFT+J (COMMAND+OPTION+J на Mac).
3. Нет. Название переменной не может начинаться числом.
4. Да.
5. Истина. Сообщения об ошибках призваны помочь вам понять, что пошло не так, и исправить дело. Именно поэтому никогда не отмахивайтесь, но внимательно читайте эти ошибки!
6. SyntaxError.
7. `%` (modulo).
8. `about:blank`.
9. `+=` (`матемЗначение += 7;`).
10. Да. Несколько предложений, отделённых точкой с запятой (`;`), могут помещаться в одной строчке.
11. Нет, не является, поскольку во второй строчке не нужен `let`.

Рекомендации для «Сделай сам»

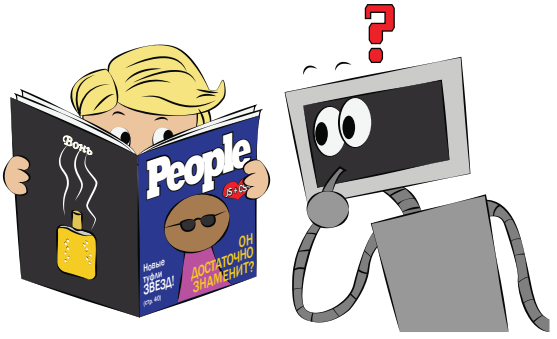
Вот мой вариант этого проекта (попробуйте вставлять по одной строчке кода в консоль, наблюдая за результатами):

```
let номерКвадрата = 4; // всего 4 квадрата
let номерId = 1; // первый ID (ID будет меняться в каждой следующей
строчке)
номерId % номерКвадрата; // оператор % будет выдавать один из 4 вариантов
(числа между 0 и 3). В данном случае — 1 (то есть — квадрат № 1)
номерId = 2; номерId % номерКвадрата;  // 2 (квадрат № 2)
номерId = 3; номерId % номерКвадрата;  // 3 (квадрат № 3)
номерId = 4; номерId % номерКвадрата;  // 0 (квадрат № 0)
номерId = 5; номерId % номерКвадрата;  // 1
номерId = 6; номерId % номерКвадрата;  // 2
номерId = 7; номерId % номерКвадрата;  // 3
номерId = 8; номерId % номерКвадрата;  // 0
номерId = 9; номерId % номерКвадрата;  // 1
номерId = 10; номерId % номерКвадрата; // 2
номерId = 11; номерId % номерКвадрата; // 3
номерId = 12; номерId % номерКвадрата; // 0
```

```
номерId = 13; номерId % номерКвадрата; // 1
номерId = 14; номерId % номерКвадрата; // 2
номерId = 15; номерId % номерКвадрата; // 3
номерId = 16; номерId % номерКвадрата; // 0
```

Глава 3

Викторина



1. b. человек.
2. Строка.
3. c. проигнорирует.
4. `//`.
5. SHIFT+ENTER.
6. Кавычки (одинарные или двойные).
7. `/*`.
8. `*/`.
9. Ложь. Выбор кавычек не имеет никакого значения.
10. Символов (хотя если бы вы хитро ответили «строк», то технически и этот ответ не был бы неправильным).
11. Объединение.
12. Необходимо добавить `\` перед апострофом: `'The name\'s Bond. James Bond.'`.
13. Измените одинарные кавычки на двойные: `"Here's looking at you, kid."`.

Упражнения

I. Введите следующие правильные фрагменты кода в консоль.

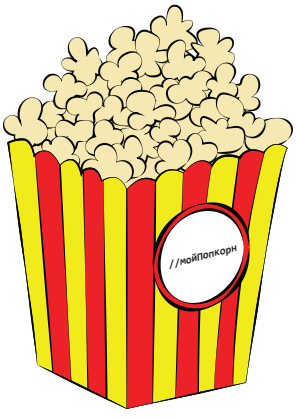
1. (просто наберите код в консоли).
2. (просто наберите код в консоли).
3. (просто наберите код в консоли).
4. `"Snickers двойной, с орехами"`.
5. `"Мы уже приехали?"`

6. `"Нет."`
7. `"Мы уже приехали? Мы уже приехали? Мы уже приехали?"`
8. `"Мы приедем тогда, когда приедем."`
9. `"Но мама сказала: "Ты и опомниться не успеешь, как мы приедем"!"`
10. `"А ещё я тебе говорила: "Хватит дёргать папу, когда он за рулём"!"`
11. `"Но яйаа таааак устаааала, а мы всё под'езжаем да под'езжаем, а никак не приедем, а уснуть, прислонившись к окну, я не могу."`
12. `"Если не прекратишь канючить и коверкать 'подъезжаем' в 'под'езжаем', я сейчас же остановлю машину, и мы пойдём пешком!"`

II. Что не так с каждым из фрагментов?

1. Строчка с объявлением переменной вынесена в комментарии (так что интерпретатор её просто проигнорирует). Таким образом, мы получим ReferenceError, так как пытаемся обратиться к необъявленной переменной.
2. Нет закрытия блока комментариев.
3. В Д'Арк нужен обратный слэш (или же заключить всю строку в двойные кавычки).
4. Это не блок комментариев, так как из-за `//` в комментарии попадёт лишь одна строчка.
5. Лишний `/` в конце.
6. Вместо `*` там нужен `+`.
7. В названии песни `i` должна быть заключена в кавычки — `'i'`.
8. Обратный слэш находится не там, где должен; его следует переместить на один символ правее — `"выпустил " + "\"собак?\"`.

Комплексный обзор



1. CTRL+SHIFT+J (COMMAND+OPTION+J на Mac)
2. Да.
3. Да.
4. `about:blank`.
5. Нет. Кроме переменных, слева от знака равенства (`=`) ничего не должно стоять, так как интерпретатор расценивает этот знак как присваивание значения.

6. Истина. Сообщения об ошибках — ваши верные друзья. Они призваны помочь вам исправить то, что работает неверно. Всегда внимательно читайте их!
7.
```
let возраст = 12;
возраст += 1;
```
8. В консоли.
9. SyntaxError (потому что ключевое слово `let` использовано к одной переменной более одного раза).
10. Да.
11. Всё это — интернет-браузеры (или просто браузеры).
12. верблюжийРегистр.
13. Скобки `(` вот так `)`.
14. Строка (не забывайте обращать внимание на кавычки!).
15. Строка (не забывайте обращать внимание на кавычки!).
16. Присваивание.
17.
```
let любимоеБлюдо = "Курочка";
любимоеБлюдо += " Буррито";
```
18. Modulo (или просто mod).
19. Да (хотя смысла большого в нём не будет, поскольку присваиваемые значения не используются).
20. Нет. Переменная `любимыйПопкорн` так и не была объявлена (ведь её объявление вынесено в комментарии).

Рекомендации для «Сделай сам»

Вот возможный код для этого проекта (попробуйте перепечатать код в консоль или же просто скопировать его туда, если пользуетесь цифровой версией этой книги):

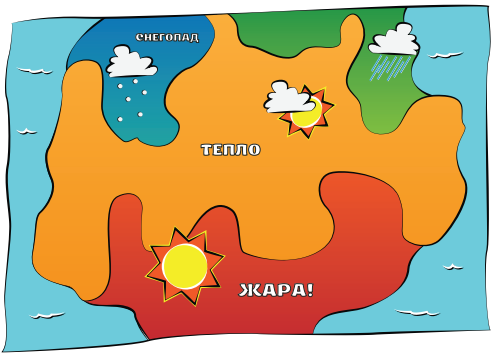
```
/* Переменные */
let имя = 'Джереми';
let хобби = 'ходить на мюзиклы';
let домашниеЖивотные = 'не имею (не считая детей)';
let хорошоУмею = 'поиграть в шахматы';
let класс = 31; // (как здорово, что их не продолжают учитывать
                // не каждый год )
let любимыйМультик = 'Рапунцель';
let местоимение = 'он'; // в противоположность 'она'!
let притяжательноеМестоимение = 'его'; // в противоположность 'её'!
```

```
/* Теперь соберём всё в одном абзаце */
let bio = 'Несмотря на то, что ' + имя + ' — мегаизвестный и популярный, '
    + местоимение + ' всегда рад ' + хорошоУмею + ' и повеселиться с '
    + притяжательноеМестоимение + ' прекрасными домашними питомцами,
    под названием "' + домашниеЖивотные + '". В ' + ' свободное время '
    + местоимение + ' любит ' + хобби + ', пересмотреть "'
    + любимыйМультик + '" или же вдоволь погрызть гранит науки, чтобы
    попасть на доску почёта в '+ ' школе, где ' + местоимение
    + ' до сих пор учится в ' + класс + ' классе.';
bio; // должен выйти связный текст
```

Глава 4

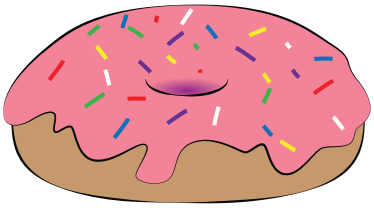
Викторина

1. Функция.
2. Фигурные скобки `{` вот так `}`.
3. После объявления функцию можно вызвать.
4. В верблюжьемРегистре.
5. `return`.
6. Параметры/аргументы.
7. D.R.Y. (Принцип сухости).
8. `alert()`.
9. Запятая `,`.
10. `console.log()`.
11. Скобки `()`.
12. `0` и `1` (имеется в виду, конечно, функция `Math.random()`).
13. `Math.floor()`.
14. Истина.
15. Истина.
16. Ложь. Интерпретатор вычислит возвращаемое значение (также называемое просто «результатом») внутренней функции, а затем передаст его в качестве аргумента внешней.
17. TAB.
18. `\n`.



Упражнения

I. Введите следующие правильные фрагменты кода в консоль.



1. (просто наберите код в консоли) .
2. `"Дикий зной со снегопадом"`.
3. (просто наберите код в консоли) .
4. `"Погода на сегодня: облачно, возможны осадки в виде фрикаделек"`.
5. (просто наберите код в консоли) .
6. `"Тебя зовут Джереми"` // ваш вариант может отличаться от моего!
7. (просто наберите код в консоли) .
8. `"Тебе уже 36, а тебе всё ещё нравится Xbox One?"` // ваш вариант может отличаться от моего!
9. (просто наберите код в консоли) .
10. `3` // любое целое число от 1 до 6; ваш результат может отличаться от моего.
11. (просто наберите код в консоли) .
12. `У вас выпало 11! (5 и 6)` // ваш результат может отличаться от моего.

II. Что не так с каждым из фрагментов?

1. Функция не объявлена.
2. Не хватает фигурных скобок `{ }`.
3. Квадратные скобки `[ ]` следует заменить на фигурные `{ }`.
4. Функция `console.log()` должна быть снаружи, а `аПончикиЭтоВкусно()` — заключена в её скобки в качестве аргумента (так как она вернёт строку, которую можно будет вывести в консоль).
5. Функция `Math.random()` должна быть вложенной в `Math.floor()`, поскольку возвращаемое значение `Math.random()` будет затем предложено в качестве аргумента функции `Math.floor()`. Кстати, значение в данном случае гарантировано будет нулем, поскольку все возвращаемые значения будут округляться в меньшую сторону до ближайшего целого, то есть нуля.
6. Ну а это… это просто чушь какая-то. Люди терпеть не могут все эти всплывающие `alert()` и так далее.

7. Точка с запятой не может быть внутри функции `console.log()`; она должна быть в конце предложения.
8. После `какойтоЦвет` должны следовать скобки `()`.
9. В скобки функции нужно добавить параметр `ресторан`, чтобы потом можно было его использовать в возвращаемом значении.

Комплексный обзор

1. Строка.
2. `let` и `var`.
3. Истина.
4. Да.
5. Перед апострофом в `Can't` нужен обратный слэш.
6. Ложь. Это называется «принципом сухости» — D.R.Y. (Don't Repeat Yourself).
7. Нет. Судя по всему, разработчик оставил однострочный комментарий, так что следовало использовать `//` вместо `/*`, который означает начало блока комментариев, который, соответственно, должен быть закрыт при помощи `*/`.

8. Число.
9. `//`.
10. Символов.
11. `%`.
12. Ложь. Вполне допустимо использовать двойные кавычки, если разработчик не забывает экранировать внутренние кавычки. Например: `"Мама сказала \"Пора в школу!\""`.
13. Он рассчитывает получить остаток деления [от евклидового деления].
14. Истина.
15. Зелёного.
16. В консоли.
17. Да.
18. Modulo `%`.

19. `+=` (`огромноеЧисло += 19;`).
20. Строка.
21. Нет. В параметре функции не может быть указано число; там должна быть указана переменная (а имена переменных, как вы помните, не могут начинаться с чисел).
22. Для переноса строки (то есть создания новой строчки).
23. TAB.
24. Нет. В функции заявлено два параметра — `мин` и `максЧисло`. При вызове функции ей было предложено аргументом лишь одно число, которое будет соответствовать первому параметру (`мин`), который в теле функции никак не задействован. Второй же параметр (`максЧисло`), использованный в функции, получит значение `undefined`, которое не позволит функции выполнить поставленную задачу — найти целое число меньше 20.
25. Вложенной.

Рекомендации для «Сделай сам»

Вот мой вариант этого проекта (попробуйте в консоли):

```
/*
Городская лотерея!
*/
function случайноеДвузначноеЧисло() {
    let максЧисло = 100;
    let числоСДесятичнымиЗнаками = Math.random() * максЧисло;
    let округлитьДоЦелого = Math.floor(числоСДесятичнымиЗнаками);

    return округлитьДоЦелого;
}
function лотерейныеНомера() {
    let часть1 = случайноеДвузначноеЧисло();
    let часть2 = случайноеДвузначноеЧисло();
    let часть3 = случайноеДвузначноеЧисло();
    let выигрышныеНомер = часть1 + '-' + часть2 + '-' + часть3;

    console.log('Вот номера, принёсшие вам выигрыш: ' +
                выигрышныеНомер + '!');
}
// Должно каждый раз выдавать новые номера
лотерейныеНомера();
лотерейныеНомера();
лотерейныеНомера();
```

Глава 5

Викторина

1. Булев (или логический) тип.
2. `<` («меньше чем»).
3. Ложь.
4. `!==` («строго не равно»).
5. . `==` и `!=`.
6. `>=` // удалось мне вас запутать формулировкой?
7. Условные.
8. «Правка» / «ложка».
9. `else`.
10. Истина.
11. Истина.
12. Ложь (при блоке `else` нет скобок).
13. Ложь.

Упражнения

I. Введите следующие правильные фрагменты кода в консоль.

1. `false`.
2. `undefined` // код при блоке if не запускается.
3. `true`.
4. `true`.
5. `true` («правка»).
6. `true`.
7. `хм, чот странное`.
8. `false`.
9. `"Теперь понимаете, почему не стоит эти операторы использовать? Странные штуки!"`.

10. `false`.
11. `false`.
12. `false`.
13. `undefined` // код при блоке if не запускается.
14. Просто наберите код в консоли.
15. `301 больше 212`.
16. `155 больше -800`.
17. `Да они же равны!`.
18. `деёж больше абвг`.
19. `Да они же равны!`.

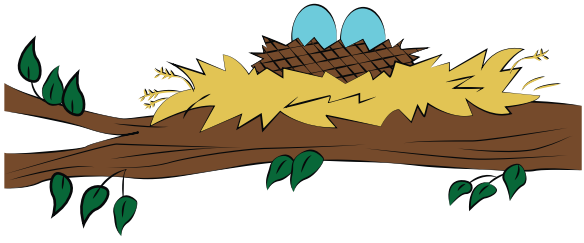
II. Что не так с каждым из фрагментов?

1. В этом коде не предусмотрен вариант на случай равенства двух чисел. В своём нынешнем виде код будет отвечать, что одно из чисел меньше, даже когда они равны.
2. При блоке `if` не хватает скобок.
3. При блоке `else` скобок быть не должно.
4. Когда в одном выражении используются два оператора сравнения за раз, такое сравнение вряд ли выдаст какой-то осмысленный результат.
5. Здесь должно быть `===`.
6. Это присваивание, а значит, должно быть `=`.
7. Это присваивание, а значит, должно быть `=`.

Комплексный обзор

1. Ложь. Интерпретатор будет игнорировать всё, начиная с `/*` и до закрывающих блок комментариев `*/`.
2. Да.
3. Перед кавычками в слове `"Goodbye"` нужно добавить по обратному слэшу: `\"Goodbye\"`.
4. Принцип означает, что нужно избегать повторений, стараясь как можно «суше» и лаконичнее писать код.
5. Условные.

6. Да.
7. Строка.
8. `/*` и `*/`.
9. «Правкой» / «Ложкой».
10. modulo (взятие остатка).
11. верблюжьемРегистре.
12. Истина.
13. SyntaxError.
14. Он ожидает остатка от деления.
15. Нет. Здесь лишняя закрывающая скобка.
16. Ложь. Сообщения об ошибках как раз и нужны, чтобы было проще понять, что пошло не так.
17. Присваивание.
18. CTRL+SHIFT+J (COMMAND+OPTION+J на Mac).
19. Да, но ожидаемого результата он не даст. Оператор `>` следует изменить на `>=`.
20. `+=` // вот так: `приветствие += ', ' + имя;`.
21. Булев (или логический) тип.
22. Да. Но ожидаемый разработчиком результат он не даст. После `return` никакой код запущен быть уже не может, поэтому его следует переместить в конец функции.
23. `about:blank`.
24. Браузеры.
25. `==` и `!=`.
26. `<=` // вопрос с подвохом. Если с первого раза не поняли, попробуйте ещё раз!
27. TAB.
28. Да.
29. Вложенной.
30. Истина.
31. `Math.random()`.



Рекомендации для «Сделай сам»

Вот мой вариант этого проекта (попробуйте в консоли):

```
/*
Детская церковь!
*/
function подходитЛиПоВозрасту(возраст) {
    let минВозраст = 6;
    let максВозраст = 13;
    if (возраст < минВозраст) {
        console.log("Ты ещё слишком мал.");
    } else {
        if (возраст <= максВозраст) {
            console.log("Добро пожаловать в Детскую церковь!");
        } else {
            console.log("Ты уже достаточно большой, чтобы ходить
                        во Взрослую церковь.");
        }
    }
}

// Проверим
подходитЛиПоВозрасту(5);  // маловат
подходитЛиПоВозрасту(15); // староват
подходитЛиПоВозрасту(9);  // самое то!
подходитЛиПоВозрасту(13); // подходит!
```

Глава 6

Викторина

1. `undefined`.
2. Логические.
3. `null`.
4. `undefined`, `undefined`.
5. Логическое `ИЛИ`, `||`.
6. Всего одно возможное значение — `null`.
7. Ложь.
8. `else if`.

9. Логическое `НЕ`, `!`.
10. Истина.
11. Логическое `ИЛИ`, `||`.
12. Истина.
13. Ложь.
14. Истина.
15. Логическое `И`, `&&`.
16. Истина.
17. `undefined`.
18. Всего одно возможное значение — `undefined`.
19. Истина.
20. `true` (помните, что `!` всегда возвращает булево значение).
21. Логическое И, `&&`.
22. «Ложками» / «правками».
23. `"любовь"`.

Упражнения

I. Введите следующие правильные фрагменты кода в консоль.

1. `false`.
2. `true`.
3. `B`.
4. `0`.
5. `C`.
6. `C`.
7. (просто наберите код в консоли).
8. `1-й человек старше, а значит, ходит первым.`
9. `2-й человек старше, а значит, ходит первым.`
10. `1-й человек старше, а значит, ходит первым.`
11. `Нужно предложить возрасты этих двоих в качестве аргументов!`

12. `Да они же ровесники! Давайте выберем случайным образом? 2-й ходит первым!`
13. `Нужно предложить возрасты этих двоих в качестве аргументов!`
14. `2-й человек старше, а значит, ходит первым.`
15. `Да они же ровесники! Давайте выберем случайным образом? 1-й ходит первым!`
16. (просто наберите код в консоли).
17. `Его зовут Толька С.-Корей.`
18. `Его зовут Иван Иваныч.`
19. `Его зовут Большеголо, Вова.`
20. `Его зовут Иван Иваныч.`

II. Что не так с каждым из фрагментов?

1. Код не является неправильным, хотя финальный блок `else` так никогда и не запустится. Кроме того, строчка с `console.log()` в первом блоке `if` вводит в заблуждение, поскольку переменная `любойАргумент` должна быть правкой, чтобы блок запустился.
2. При блоке `else if` не хватает скобок.
3. Раз это присваивание (а именно об этом говорит `=`), то в левой части (перед `=`) должна находиться лишь одна переменная; никаких прочих операторов, чисел и так далее там быть не должно.
4. Раз это присваивание (а именно об этом говорит `=`), то в левой части (перед `=`) должна находиться лишь одна переменная; никаких прочих операторов, чисел и так далее там быть не должно.
5. Раз это присваивание (а именно об этом говорит `=`), то в левой части (перед `=`) должна находиться лишь одна переменная; никаких прочих операторов, чисел и так далее там быть не должно.

Комплексный обзор

1. Да.
2. Булев, строка, null (или объект, если угодно), число, строка, undefined.
3. Взятие остатка.
4. Символов.
5. Истина.

6. Логических.
7. Сравнения.
8. Условный.
9. Вложенной.
10. Да, но ожидаемого разработчиком результата он не даёт. В код блока `if` следует включить `>=` и `<=`, чтобы функция не исключала эти конкретные параметры, но нормально могла работать с ними.

11. Ложь. `!` всегда возвращает булево значение, но операторы `&&` и `||` — далеко не всегда.
12. `+=` // вот так: `счётВМатче += 10;`.
13. Modulo (`%`).
14. Они все «правки». Вот типы их данных: строка, число, булев тип, число, строка, строка.
15. Да, но ожидаемого разработчиком результата он не даёт. Никакой код не запускается после `return` — это слово всегда должно идти в самом конце функции.
16. `<` // это был вопрос с подвохом; если не вышло с первого раза — попробуйте ещё!
17. В консоли.
18. Среда.
19. TAB.
20. Ложь. Сперва всегда вычисляется значение внутренних скобок и функций.
21. Истина.
22. Нет. Параметр не может быть выражен числом; он всегда выражается переменной (а название переменной не может начинаться с числа).
23. `Math.floor()`.
24. Истина.
25. Истина.
26. SHIFT+ENTER.

Рекомендации для «Сделай сам»

Вот мой вариант этого проекта (попробуйте в консоли):

```
/*
Полосатый Храм
*/
function ответХрамовойДвери(частицыСилы, посохРотовойВони, главныйКлюч,
                            обычныеКлючи) {
    let всегоЧастицСилы = 6;
    let минОбычныхКлючей = 10;
    let сбщн;

    if (!( частицыСилы >= всегоЧастицСилы)) {
        let неХватает = всегоЧастицСилы - частицыСилы;
        сбщн = 'Тебе не хватает ещё ' + неХватает + ' Частиц Силы,
                чтобы попасть в Храм.';
    } else if ( посохРотовойВони && (главныйКлюч || обычныеКлючи >=
                минОбычныхКлючей)) {
        сбщн = 'Теперь можешь войти в Полосатый Храм!';
    } else {
         // let дельноеСообщение = 'Сперва отнеси большое блюдо тушёного
         // мяса верховномуМагу тёмногоГрада.';
        let загадочноеСообщение = "Возвращайся, когда соберёшь всё
                                   необходимое.";
        сбщн = 'Ты ещё не готов. ' + загадочноеСообщение;
    }

    console.log(сбщн);
}

// Проверим
ответХрамовойДвери(4, true, true);      // не хватает частиц
ответХрамовойДвери(6, true, false, 9);  // не готов
ответХрамовойДвери(6, false, true);     // опять не готов!
ответХрамовойДвери(6, true, false, 12); // можно заходить!
ответХрамовойДвери(6, true, true);      // прямо в храм!
```

Глава 7

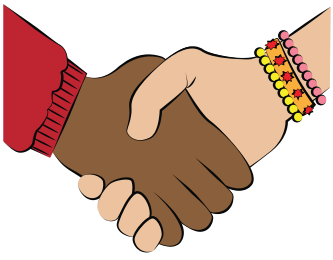
Обратите особое внимание на рекомендации по DIY к этой главе в конце.

Викторина

1. Истина.
2. `toUpperCase()`.

3. `const`.
4. Документирование.
5. Когда вы предполагаете, что значение переменной будет меняться.
6. `confirm()`, `prompt()` и `alert()`.
7. Самодокументированный.
8. `toLowerCase()`.
9. Никогда.
10. Ложь. Её называют «методом».
11. `prompt()`.
12. `const`.
13. Ложь. Она всегда возвращает строку (или же `null`).
14. Истина.
15. Ложь. Они раздражают пользователей и блокируют весь остальной код, ожидая действия пользователя. Поэтому эти функции сейчас практически не используются.

Упражнения



I. Введите следующие правильные фрагменты кода в консоль.

1. `3.8`.
2. `имяпользователянаписанноевоттаквоткоекак`.
3. `Номинальная стоимость — $1.60`.
4. `undefined`.
5. `Рад снова вас видеть, Джереми! // ваш ответ может отличаться`.
6. `undefined`.
7. `19`.
8. `ПАТЕНТНОЕ БЮРО США`.
9. `должна объявляться при помощи ключевого слова let!`
10. `Всё вычищено и блестит! // ваш результат может отличаться`.
11. `Благодарим за покупку! // ваш результат может отличаться`.
12. `undefined`.

13. `Конечно! Вот вам моя рука! // ваш результат может отличаться.`
14. `Пожалуй, пока нам лучше быть более сдержанными. // ваш результат может // отличаться.`
15. `Пожалуй, пока нам лучше быть более сдержанными. // ваш результат может // отличаться.`
16. `Конечно! Вот вам моя рука! // ваш результат может отличаться.`

II. Что не так с каждым из фрагментов?

1. Нельзя использовать `const` для меняющихся величин; в таких случаях следует использовать `let`.
2. Лучше во всех подобных случаях, когда значение очевидно неизменяемо, использовать `const` (вместо `let` или `var`).
3. Следовало использовать `prompt()` вместо `confirm()`, раз требовался ввод данных.
4. Да ну вас! `toCapitalized()` вообще ни разу не функция! Здесь нужен `toUpperCase()`.
5. Технически здесь нет ошибки, но вряд ли здесь очень кстати была использована функция `toFixed()`, которая округлила точное значение и заявила, что обнаружила `ещёБолееТочныйУгол`. Пожалуй, запуская ракету в космос, не лучшая идея высчитывать углы полёта таким приблизительным образом!
6. Здесь нет ошибок, однако его документирование вызывает недоумение; почему бы просто не назвать переменные `днейВГоду`, `днейВНеделе`, `часовВсутках` или что-нибудь в таком духе? Таким образом, код был бы самодокументированным и не нуждался бы во всех этих комментариях.
7. Здесь нужно использовать обычный `confirm()`, а не `prompt()`, поскольку требуется лишь выбрать положительный или отрицательный ответ на вопрос.

Комплексный обзор

1. «Ложкой»/«Правкой».
2. Перед апострофом в `Can't` нужен обратный слэш: `Can\'t`.
3. Избегать повторений (Don't Repeat Yourself), писать как можно более «сухой» и лаконичный код.
4. Да.
5. `(Math.random() * 20).toFixed(2)`;
6. Да (а вдобавок он ещё и самодокументирован!).

7. Ложь. Блок комментариев открывается `/*` и продолжается (порой довольно большое количество строчек), пока не закроется `*/`.
8. Строка.
9. `undefined`.
10. верблюжьемРегистре.
11. Истина.
12. Нет. Если переменная может меняться, то следует использовать `let`.
13. Остаток.
14. `"сделать домашку"` (при использовании `||` всегда берётся лишь первая «правка», а остальное игнорируется).
15. Истина.
16. Присваивание `//` загляните в глоссарий, если не совсем понимаете значение.
17. Истина.
18. Истина.
19. `const`, `let` и `var` (не используйте `var`).
20. `\n`.
21. Логические.
22. Нет, не правильный. Параметр, ожидающий аргумента, называется `возрастРебёнка`, но далее так никогда и не используется. Взамен используется ещё одна переменная `возраст`, которая никак не определена. Если же изменить `возраст` на `возрастРебёнка`, то код заработает. Впрочем, заработает он всё же не так, как от него ожидается; для правильной работы необходимо заменить `>` на `>=`, чтобы научить функцию не прогонять из садика пятилетних детей (`минВозраст`).
23. Да, но ожидаемого разработчиком результата он не даст. После `return` уже не запустится никакой код, поэтому его следует помещать в самый конец функции.
24. `about:blank`.
25. Браузеры.
26. Ложь.
27. Самодокументированный код.
28. `!`.
29. « Ложка» / «правка».
30. Истина.

Рекомендации для «Сделай сам»

Здесь представлены мои варианты выполнения всех проектов в этой главе; помните, что правильных вариантов может быть несколько.

Сделай сам: настраиваем спаморассылку

```
function калибровкаСпамомёта(рассылка) {
    return рассылка.toUpperCase();
}
// Проверим:
калибровкаСпамомёта('Срочно! Внимание!');
калибровкаСпамомёта('Вы были выбраны победителем в конкурсе на грант
в размере 42 миллионов долларов!');
калибровкаСпамомёта('Нужно всего лишь сделать взнос в размере 2000$
срочным переводом Western Union — и деньги ваши!');
```

Сделай сам: ещё один температурный конвертер (Цельсии в Фаренгейты):

```
function цельсииВФаренгейты(градусыЦельсия) {
    const конверСоотношение = 1.8;
    const замерзаниеВодыУФаренгейта = 32;

    return (градусыЦельсия * конверСоотношение) +
замерзаниеВодыУФаренгейта;
}

// Проверим:
цельсииВФаренгейты(37.7);
цельсииВФаренгейты(0);
цельсииВФаренгейты(100);
```

Сделай сам: а как насчёт зубной нити?

```
function сегодняЧистилЗубыСНитью() {
    if (confirm('А вы уже почистили зубы?')) {
        if (confirm('А как насчёт зубной нити?')) {
            return 'Можете пройти в кабинет доктора фон Клещеффа.';
        } else {
            return "Сперва воспользуйтесь зубной нитью; раковина —
                    вон там!";
        }
    } else {
        return 'Нет уж! Сперва почистите зубы! Раковина — вон там.';
    }
}

// проверим:
сегодняЧистилЗубыСНитью();
сегодняЧистилЗубыСНитью();
```

Сделай сам: варим самый экзотический суп

```
function варимЭкзоСуп() {
    const ингредиент1 = prompt("Сперва мы положим много ");
    const ингредиент2 = prompt("Потом щедро добавим ");

    return 'Ну вот, и готов наш суп из ' + ингредиент1 + ', ' +
            ингредиент2  + ' и '
        + prompt("И под конец добавим щепотку ") + '!';
}

// Давайте проверим
варимЭкзоСуп();
варимЭкзоСуп();
```

Сделай сам: калькулятор чаевых (переменных и непостоянных!):

```
function расчётЧаевых() {
    const суммаЗаказа = prompt('Пожалуйста, укажите стоимость вашего
                                заказа.');
    const процентЧаевых = prompt('Сколько бы вы желали оставить на чай?')
                                / 100;
    const знаковПослеЗапятой = 2; // обычно в денежных операциях обходятся
                                // двумя знаками после запятой
    const чаевые = (суммаЗаказа * процентЧаевых).
toFixed(знаковПослеЗапятой);

    return 'Вы решили оставить официанту на чай $' +чаевые + '. Спасибо!';
}

// Проверим
расчётЧаевых();
расчётЧаевых();
```

Сделай сам: «переводчик» оценок

```
function оценкаЗаСочинение() {
    const оценка = prompt('Сколько ты получил?');
    if (оценка >= 90) {
        return 'A';
    } else if (оценка >= 80) {
        return 'B';
    } else if (оценка >= 70) {
        return 'C';
    } else if (оценка >= 60) {
        return 'D';
    }

    return 'F'; // если не будет выдан ни один из перечисленных ответов
}
```

```
// Проверим
оценкаЗаСочинение(); // попробуйте 80
оценкаЗаСочинение(); // попробуйте 53
оценкаЗаСочинение(); // попробуйте 62
```

Глава 8

Викторина

1. Массив.
2. Выдаёт количество элементов в массиве.
3. Индекс.
4. `0`.
5. Истина.
6. `.push()`.
7. Истина. В ответ будет выдана строка, перечисляющая элементы массива через запятую.
8. Удалит/вернёт в консоль.
9. Ложь. Массив `учебники` ничуть не изменится: его длина (`.length()`) останется 2.
10. Цепочка.
11. `напёрсток`.
12. Истина.
13. `.shift()`.
14. `гель`.
15. Ложь. В консоль будет возвращена строка, перечисляющая все элементы массива, разделяя их при помощи `' или '`.
16. `г`.
17. Истина.
18. `5` (потому что в слове «сдоба» — 5 букв).

Упражнения

I. Введите следующие правильные фрагменты кода в консоль.

1. `undefined`.
2. `просыпаюсь`.
3. `Вот что я делаю с утра: иду в душ, потом одеваюсь, потом чищу зубы.`
4. `4` // длина массива.
5. `иду в душ`

   `чищу зубы`.
6. `готовлю завтрак`.
7. `4` // длина массива.
8. `Значит, каждое утро я совершаю 4 действия.`
9. `Не забудь про пункт "чищу зубы"!`
10. `undefined`.
11. `Раньше моё утро всегда было таким: "завтракаю, иду в душ, одеваюсь, чищу зубы", но теперь этот список расширился: "завтракаю, иду в душ, одеваюсь, чищу зубы, собираю учебники, заправляю постель, причёсываюсь"`.
12. `Камень, бумага, ножницы`.
13. `Ж-А-Б-А`.
14. `добрый вечер!`

II. Что не так с каждым из фрагментов?

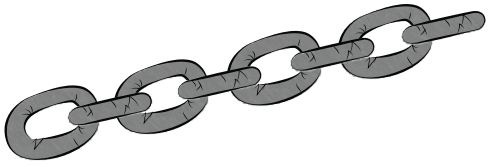
1. Используйте `орешки.length` вместо `орешки.size`.
2. Чтобы получить третий элемент массива, используйте `орешки[2]` (ведь индексы начинаются с нуля).
3. Должно быть: `орешки.push('лесной орех');`.
4. `орешки.pop()` удалит и вернёт в консоль последний элемент массива — `'арахис'`, который является строкой, а не массивом (таким образом, `.length` не сможет показать никаких элементов массива, а покажет лишь количество символов в строке «арахис»).
5. Должно быть: `.unshift('бразильский орех');`.
6. `полныйСписок` является строкой, так что метод `.shift()` не произведёт никакого эффекта.

7. `орешки.concat()` вернут в консоль НОВЫЙ массив. Этот массив не был ни к чему прикреплён, так что он просто «потерялся». В то же время массив `орешки` не был изменён и длиннее не стал.
8. Должно быть: `nuts.sort();`.
9. Всё было бы хорошо, если бы `орешки` были созданы при помощи `let`; но поскольку они были объявлены с `const` — ничего не выйдет.

Комплексный обзор

1. Да.
2. Число, null (или, если угодно, объект), undefined, строка, булев тип.
3. Символов.
4. `toUpperCase()`.
5. Логических.
6. Сравнения.
7. `const`.
8. Условный.
9. Вложенной.
10. Да, является.
11. Ложь. `!` Всегда возвращает булево значение, но `&&` с `||` — далеко не всегда.
12. `confirm()`, `prompt()` и `alert()`.
13. `++` или `+= 1;` // вот так: `текущийЦикл++; или текущийЦикл += 1;`.
14. Modulo (`%`).
15. Ни в каких.
16. Они все — «правки». Вот типы их данных: число, строка, булев тип, число, строка, строка.
17. Да, код правильный, но ожидаемого результата он не даст. Никакой код после `return` запускаться не будет. Поэтому `return` зачастую лучше ставить в самый конец функции (если речь не идёт о `return` при блоке `if`).
18. `<` // вопрос с подвохом! Попробуйте ещё подумать, если сразу не вышло!
19. В консоли.
20. Глаза.

21. Истина.
22. Цепочкой.
23. TAB.
24. `.length` // вот так: `['просто', 'массив'].length;`
25. Истина.
26. Истина.
27. Да. Однако в объявлении функции значится параметр, который далее никак в коде не задействован. Лучше избегать использования элементов кода, которые затем никак не используются.
28. `Math.floor()`.
29. `.push()`.
30. Истина.
31. Истина.
32. `Иванов`.
33. SHIFT+ENTER.
34. `популярныеЖенщины[2];`
35. Истина.



Рекомендации для «Сделай сам»

Вот мой вариант проекта (попробуйте в консоли):

```
/*
* Это Хэллоуин!
*/

// кстати: в общем-то необязательно всё это делать в виде функции!
function уловНаХэллоуин() {
    let сласти = ['Kit-Kat', ' Skittles '];
    сласти[2] = 'Snickers';
    сласти.push('Nuts'); сласти.push('Bounty');
    сласти.unshift("M&M's");
    console.log("Всё! Не могу больше! Съем этот вкуснейший батончик " +
                сласти.pop() + '!');
    сласти[2] = 'Starburst';
    const шоколадФлетчера = [
        'Milky Way',
        "M&M's",
```

```
        'Mars',
        "Hershey's"
    ];
    сласти = сласти.concat(шоколадФлетчера);
    сласти.shift(); // Ммммм… вкуснотища!
    сласти.sort();
    console.log('Теперь мой улов насчитывает ' + сласти.length +
                ' сластей!\n'
        + 'Вот, что у меня есть: ' + сласти.join(', ') + '!');
}
уловНаХэллоуин();
```

Вот ответы (в смысле только лог-сообщения консоли), которые я получил:

```
Всё! Не могу больше! Съем этот вкуснейший батончик Bounty!
Теперь мой улов насчитывает 8 сластей!
Вот что у меня есть: Hershey's, Kit-Kat, M&M's, Mars, Milky Way, Nuts,
Snickers, Starburst!
```

Глава 9

Викторина

1. Цикл.
2. `while`.
3. `for`.
4. Условие / цикл.
5. 5.
6. Соглашением о стандартах оформления кода.
7. Бесконечное количество! Здесь отсутствует инкремент, так что код застрянет в бесконечной петле!
8. `;` (точки с запятой).
9. Условие.
10. Да. Будет выведено сообщение: `'Какая ошибка!'` (дело в том, что начальное значение `m` (1) возрастает с каждой итерацией, и наконец сравнивается с длиной массива. После этого `топ40Хитов[m]` будет иметь то же значение, что и `топ40Хитов[4]`, то есть — `undefined`).
11. 18 (не забывайте, что всё начинается с `0`!).

12. Итерация.
13. `шляпы.length`.

Упражнения

I. Введите следующие правильные фрагменты кода в консоль.

1. (слишком длинно; просто проверьте в консоли).
2. `"0|7|14|21|28|35|42|49|56|63"`.
3. (слишком длинно; просто проверьте в консоли).
4. (слишком длинно; просто проверьте в консоли).
5. `["Я", "Ю", "Э", ... "В", "Б", "А"]` // (в сокращённом виде).
   `"почистить зубы"`
6. `А — гласная!` // и так далее с прочими гласными.
7. (слишком длинно; просто проверьте в консоли).
8. (просто наберите код в консоли).
9. `0 делится на 10!` // 10, 20, 30 и так далее.
10. `0 делится на 5!` // 5, 10, 15, 20 и так далее.
11. `0 делится на 3!` // 3, 6, 9, 12, 15 и так далее.
12. (слишком длинно; просто проверьте в консоли).
13. (просто наберите код в консоли).
14. `"3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30"`.
15. `"5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50"`.
16. `"7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70"`.
17. `"10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100"`.
18. (просто наберите код в консоли).
19. `"1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10`
    `2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20"`.
20. (слишком длинно; просто проверьте в консоли).
21. (слишком длинно; просто проверьте в консоли).

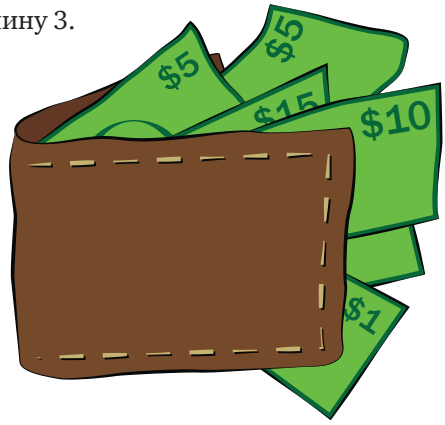
II. Что не так с каждым из фрагментов?

1. Должно быть `for` вместо `while`.
2. Должно быть `while` вместо `for`.
3. Должно быть `const` вместо `let`.
4. Методу `.split()` нужно предложить строку (например, `.split(',')`).
5. Будет только одна итерация. `return` указывает интерпретатору завершить функцию, так что цикл будет прерван.
6. Результатом будут все нечётные числа вместо чётных. Если разработчику нужны были чётные, то стоило использовать `if (!(s % 2))`, который вернул бы `true`, если число нацело делится на 2.

Мегакомплексный обзор

1. Сравнения.
2. «Правкой»/«ложкой».
3. Ложь.
4. Избегать повторений (Don't Repeat Yourself), писать как можно более «сухой» и лаконичный код.
5. Истина.
6. `Math.floor(Math.random() * 40);` // обратите внимание, что именно «40» вы так не получите.
7. Количество элементов в массиве `/` количество символов в строке.
8. Да.
9. Индекс.
10. Ложь. Блок комментариев начинается с `/*` и заканчивается `*/`.
11. `.push('новый_элемент');`
12. `undefined`.
13. Цепочка.
14. верблюжьимРегистром.
15. Истина.
16. Нет. Нельзя сменить значение у переменной, объявленной с `const`.

17. `floor` / `random` / `length`.
18. Остаток (при целочисленном делении).
19. `э` // обратите внимание на нижний регистр!
20. `смартфон`.
21. `ноутбук`.
22. `счётКомандыПротивника++;` или `счётКомандыПротивника += 1;`
23. `%` (modulo).
24. Истина.
25. `<` // вот этот вопрос был совсем с подвохом; перечитайте очень внимательно, если ошиблись!
26. Ложь. `вагоныПоезда.length` покажет длину 3.
27. `Math.floor()`.
28. Присваивание значения переменной (например, `let хорьки = 2;`).
29. SHIFT+ENTER.
30. Истина.
31. Истина.
32. `var`, `let`, `const` / `const` , `var`.
33. Условие / цикл.
34. `\n`.
35. Соглашением о стандартах оформления кода.
36. Логические.
37. Итерация.
38. Код правильный, но ожидаемого результата, судя по всему, не выдаст. Следует использовать оператор `>=`, который вернёт сообщение о том, что `"Вам хватит денег!"`, если будет указана конкретная сумма.
39. Бесконечное количество! Код войдёт в бесконечную петлю циклов, поскольку значение `x` будет продолжать увеличиваться, а значит, условие (`x > 0`) будет всегда оставаться истинным!
40. Да, является, но в нём есть баг: после первого же `return` ничего больше не запустится.
41. 4 сообщения.



42. Истина.
43. Самодокументированный код.
44. `!`.
45. Истина.
46. «Ложками» / «правками».

Рекомендации для «Сделай сам»

Вот мой вариант этого проекта (попробуйте в консоли):

```
function можноЛиМне() {
    const сласти = [
        'Nuts',
        "Hershey's",
        'Kit-Kat',
        'Milky Way',
        "M&M's",
        'Snickers',
        'Starburst',
        'Mars'
    ];
    for (let i = 0; i < сласти.length; i++) {
        const вкусность = сласти[i];
        const последняяБукваНазванияСласти = вкусность.split('').pop();
        if ( последняяБукваНазванияСласти === 's') {
            console.log("Ни в коем случае не ешь " + вкусность + "! У тебя
                        же аллергия!");
        } else {
            console.log(вкусность + ' можно с удовольствием съесть!');
        }
    }
}
можноЛиМне();
```

Вот что мне ответила консоль:

```
Ни в коем случае не ешь Nuts! У тебя же аллергия!
Ни в коем случае не ешь Hershey's! У тебя же аллергия!
Kit-Kat можно с удовольствием съесть!
Milky Way можно с удовольствием съесть!
Ни в коем случае не ешь M&M's! У тебя же аллергия!
Ни в коем случае не ешь Snickers! У тебя же аллергия!
Starburst можно с удовольствием съесть!
Ни в коем случае не ешь Mars! У тебя же аллергия!
```

Глава 10

Вот полный код «виселицы»! Играйте с удовольствием!

```
function ответыДляВиселицы() {
    const возможныйОтвет = [
        'взаимность',
        'претендент',
        'секундомер',
        'автомобиль',
        'авантюрист',
        'облачность',
        'наблюдение',
        'штукатурка',
        'укротитель',
        'виолончель',
        'заговорщик',
        'клубничник',
        'антарктика'
    ];
    const случайныйИндекс = Math.floor(Math.random() *
                            возможныйОтвет.length);

    return возможныйОтвет[случайныйИндекс].toUpperCase();
}

function конецИгры(ответ, выигрыш) {
    const asciiВиселица = '____\n|/ |\n| @\n| /|\\\n| / \\\n|\n=====';
    let сообщение = '';
    if (выигрыш) { // если параметр `выигрыш` отсутствует, то будет
                   // "ложка" (undefined)
        сообщение = 'ТЫ ВЫИГРАЛ!';
    } else {
        сообщение = 'ВСЁ КОНЧЕНО\n\n' + asciiВиселица;
    }
    сообщение += '\n\n ответ: ' + ответ + '!';
    alert(сообщение);
    return сообщение;
}
/*******************************************
* Мясо нашей "виселичной" функции! *
*******************************************/
function сыграемВВиселицу() {
    const ответ = ответыДляВиселицы();
    const буквыОтвета = ответ.split('');
    const неверная Догадка = [];
    const максНеверныхДогадок = 7;
    const прогрессИгры = '_'.repeat(ответ.length).split('');
```

```
    const подтверждение = confirm("Давай сыграем в виселицу!\n\n"
        + "Я задумал некое слово. Можешь начинать отгадывать буквы!\n"
        + "Это самое обычное слово из " + ответ.length + ' букв.\n'
        + 'Ну что, начнём?');
    if (!подтверждение) {
        return конецИгры(ответ, false);
    }

    while (неверныеДогадка.length < максНеверныхДогадок ) {
        const сообщениеПрогресса = 'Пока ты добился: \n'
            + прогрессИгры.join(' ') + '\n'
            + 'Ошибки: [' + неверныеДогадка.toString() + ']\n\n'
            + 'Выбирай следующую букву!';
        const ответИгрока = prompt(сообщениеПрогресса);

        if (!ответИгрока)  {
            return конецИгры(ответ);
        }

        const догадка = ответИгрока.toUpperCase();
        let верныеДогадка = false;

        for (let i = 0; i < буквыОтвета.length; i++) {
            if (буквыОтвета [i] === догадка) {
                верныеДогадка = true;
                прогрессИгры[i] = догадка;
            }
        }

        if (верныеДогадка) {
            if (прогрессИгры.join('') === ответ) {
                return конецИгры(ответ, true);
            }

            alert('Верно!');
        } else {
            неверныеДогадка.push(догадка);
            alert('Извини, буквы ' + догадка + ' тут нет.Ты можешь
                   ошибиться ещё '
                + (максНеверныхДогадок - неверныеДогадка.length) + ' раз,
                   прежде чем этот бедняга будет повешен.');
        }
    }
    return конецИгры(ответ, false); // полный провал
}

сыграемВВиселицу();
сыграемВВиселицу();
сыграемВВиселицу();
```

ГЛОССАРИЙ

(**Термин** *[глава, в которой он впервые появился]* — определение)

Аббревиатура *[гл. 2]* — сокращение, зачастую сконструированное из начальных букв слов. Вроде LOL или BTW.

Аргумент *[гл. 4]* — значение, предлагаемое функции для присваивания его к параметру. Когда вызывается функция и ей «предложено» такое значение — оно и называется аргументом.

Баг *[гл. 1]* — ошибки или опечатки, допущенные при написании кода.

Блок комментариев *[гл. 3]* — позволяет «растянуть» комментарий на нужное количество строчек. Интерпретатор будет игнорировать всё заключённое в блок, открывающийся `/*`, пока не «увидит» его закрытие — `*/`.

Булев(ый)/логический тип *[гл. 5]* — один из пяти примитивных типов данных («булевый», кстати, рифмуется с «всё разруливай»). Всегда имеет строгое значение: либо `true` (истина), либо `false` (ложь).

верблюжийРегистр *[гл. 1]* — наиболее удобная форма написания имён переменных и функций. Название должно начинаться со строчной буквы, а каждое последующее слово (или акроним) — с прописной. Своё название этот стиль получил за схожесть прописных букв в середине «слова» (имени функции или переменной) с верблюжьими горбами.

Взятие остатка/modulo *[гл. 2]* — оператор взятия остатка при целочисленном делении; обозначается символом `%`. Пример использования: `5 % 3 === 2;`

Вложенная(-ый) *[гл. 4]* — функция, находящаяся внутри другой функции; `for`-цикл, находящийся внутри другого `for`-цикла; элемент HTML, встроенный в другой.

Встроенная(-ый) *[гл. 4]* — что-то (скажем, функция), существующее на уровне самого языка программирования (в противоположность «пользовательскому» — созданному тем, кто писал код).

Вызов функции *[гл. 4]* — запуск функции.

Вычисление *[гл. 4]* — обработка интерпретатором функции или переменной.

Глоссарий *[гл. 1]* — всё равно что словарь, только для одной этой книжки.

Дебаггинг/отладка *[гл. 1]* — процесс избавления от жуков — багов.

Декремент *[гл. 2]* — уменьшение числа на единицу: `моёЧисло--`

Документирование *[гл. 7]* — развёрнутый комментарий, поясняющий работу программного кода.

Евклидово деление *[гл. 2]* — заумное название для целочисленного деления (см. определение ниже).

Индекс *[гл. 8]* — номер, отражающий положение элемента в массиве. Нумерация элементов начинается с нуля; таким образом, первый элемент будет иметь индекс `0`, второй — `1` и так далее.

Инженер *[гл. 1]* — то же, что программист или разработчик.

Инкремент *[гл. 2]* — увеличение числа на единицу: `моёЧисло++`

Интернет-браузер *[гл. 1]* — программы, через которые вы можете просматривать веб-страницы. Мы пользуемся Chrome. Ещё есть Firefox, Safari, Edge и Internet Explorer.

Интерпретатор *[гл. 1]* — встроенная программа, «читающая» и запускающая код.

Итерация *[гл. 9]* — единичный проход блока цикла.

Код *[гл. 1]* — инструкции, указывающие компьютеру, что и когда следует делать.

Комбинированное присваивание *[гл. 2]* — краткий способ записи, совмещающий действие и присваивание нового значения переменной. Сперва производится действие, а затем происходит присваивание. Например: `+=`, `-=`, `*=`, `/=`.

Комментарий *[гл. 3]* — может быть однострочным, а может быть и целым блоком. Интерпретатор будет игнорировать всё вынесенное

в комментарии. Используется для пояснений или для сокрытия от интерпретатора тех блоков кода, которые вы не хотите в данный момент запускать.

Компьютер *[гл. 1]* — то, на чём вы работаете, — стационарный компьютер, ноутбук, Chromebook и так далее (но пока вы здесь — не смартфон или планшет!).

Консоль *[гл. 1]* — сверхсекретная функция браузера Chrome для разработчиков. Позволяет с лёгкостью протестировать код или даже проинспектировать нужную веб-страницу.

Константа *[гл. 7]* — неизменяемая переменная.

«Ложка» (falsy) *[гл. 5]* — не строго булево значение, однако для простоты построения выражений вроде `if…else` принимается в качестве `false`. Например, «ложкой» является пустая строка (`""`).

Лог *[гл. 4]* — записанное сообщение (например, в консоли).

Логический оператор *[гл. 6]* — определяет, является ли значение «правкой» или «ложкой». Не всегда возвращает строго булевы значения. Вот логические операторы в JavaScript: `&&`, `||` и `!`.

Логическое И *[гл. 6]* — логический оператор, определяющий, являются ли значения слева и справа от `&&` оба «правками». Оператор `&&` возвращает последнее значение, если все они оказались «правками». Если хоть один из них — «ложка», то `&&` возвратит первое ложное значение. Например: `(5 > 2) && 12;` вернёт `12`.

Логическое ИЛИ *[гл. 6]* — логический оператор, определяющий, является ли хоть одно из значений по обе стороны от `||` «правкой». Оператор возвращает первую же «правку»; если истинных значений нет, то последнее — ложное. Например, `5 || (1 === 0);` вернёт `5`.

Логическое НЕ *[гл. 6]* — логический оператор, всегда возвращающий булево значение. Стоя перед «ложкой», он вернёт истину, а стоя перед «правкой» — ложь. Примеры: `!true;` вернёт `false`, а `!(3 > 5);` вернёт `true`.

Массив *[гл. 8]* — список пронумерованных значений. Используется для хранения нескольких значений в одной переменной.

Метод *[гл. 7]* — функция, принадлежащая объекту или классу. Любое свойство объекта, являющееся функцией, называется методом.

Объединение *[гл. 3]* — процесс слияния двух и более строк (или чисел и строк) в одну.

Объект *[гл. 7]* — набор свойств (то есть именованных значений).

Объявление *[гл. 1]* — создание новой переменной или функции. Для каждой переменной или функции может иметь место лишь одно объявление.

Однострочный комментарий *[гл. 3]* — интерпретатор проигнорирует всё, от начала комментария с `//` и до конца строчки.

Оператор *[гл. 2]* — осуществляет арифметические действия (`+`, `-`, `*`, `/`, `%`), а также служит для объединения строк.

Оператор сравнения *[гл. 5]* — специальный символ, используемый для сравнения двух значений; всегда возвращает булево значение. Вот все нужные для работы операторы сравнения: `===`, `!==`, `<`, `<=`, `>` и `>=`.

Операционная система *[гл. 1]* — среда, в которой запускаются все программы на вашем компьютере. Если у вас ПК, то у вас, вероятно, установлена ОС Windows. Если Mac — MacOS. Если же вы обладатель Chromebook, то он работает под управлением ChromeOS.

Определение *[гл. 1]* — присвоение значения переменной или функции.

Остаток *[гл. 2]* — остаток при целочисленном делении.

Отрицательное *[гл. 2]* — число меньше нуля.

Параметр *[гл. 4]* — переменная, заключённая в скобки при функции. При вызове функции её параметрам будут предлагаться значения — аргументы.

Переменная *[гл. 1]* — имя и выделенная область в памяти для хранения значений. Бывает изменяемой, а бывает постоянной (или константной).

Перенос строки *[гл. 3]* — техническое обозначение перехода на новую строчку.

Положительное *[гл. 2]* — число больше нуля.

Пользовательская(-ий) *[гл. 4]* — нечто созданное самим пользователем, а не встроенное в сам язык программирования.

«Правка» (truthy) *[гл. 5]* — не строго булево значение, однако для простоты построения выражений вроде `if …else` принимается в качестве `true`. Например, «правкой» является любое целое число, кроме нуля.

Предложение *[гл. 1]* — «единица» кода JavaScript. Часто бывает длиной в одну строчку.

Примитивный *[гл. 5]* — любой тип данных, кроме объекта, который не имеет встроенных методов (функций).

Присваивание *[гл. 1]* — предложение, означающее, что переменной слева принадлежит значение справа.

Программист *[гл. 1]* — тот, кто программирует, то есть пишет код.

Рабочая тетрадь *[гл. 1]* — так я на протяжении книги называю созданный вами документ (например, в Google Docs) для записи ответов на викторины, упражнения и для записи проектов «Сделай сам».

Разделитель *[гл. 8]* — маленькая строка, разделяющая элементы при конвертации массива в строку.

Разработчик *[гл. 1]* — тот, кто разрабатывает, то есть пишет код.

Самодокументирование *[гл. 7]* — оформление кода «говорящими» названиями переменных и функций; облегчает понимание кода без необходимости читать длинные поясняющие комментарии или сопроводительную документацию.

Свойство *[гл. 7]* — именованное значение, привязанное к объекту.

Символ *[гл. 3]* — то, из чего состоит строка (любая буква, цифра, знак и так далее).

Синтаксис *[гл. 1]* — правила «правописания» программного кода на JavaScript, чтобы интерпретатор смог понять, что же вы имеете в виду.

Скобки *[гл. 1]* — всегда ходят парами. Используются для разделения, группировки и определения порядка выполнения операций.

Соглашение о стандартах оформления кода *[гл. 9]* — набор общепринятых договорённостей о том, как следует оформлять код. Примером следования такому соглашению является использование верблюжьего-Регистра при наименовании переменных и функций.

Сообщение об ошибке *[гл. 1]* — ответ консоли, сообщающий о том, что и где работает неверно.

Строка *[гл. 1]* — один из пяти примитивных типов данных; состоит из отдельных символов. Зачастую отображается в кавычках.

Тип данных *[гл. 2]* — определяет типы значений, которые используются в языке программирования. В этой книжке мы имеем дело с пятью: число, строка, булев тип, null и undefined.

Условие *[гл. 5 и 9]* — применительно к операторам сравнения: возвращает булево значение и определяет, запустится код или нет; применительно к циклам: то, что определяет, будет продолжаться цикл или же нет. Как только условие перестаёт быть истинным, цикл прекращается.

Условная конструкция *[гл. 5]* — используется для запуска блока кода согласно известному условию. Условие (скажем, `x === y`) возвращает булево значение (то есть `true` или `false`). Соответственно, если условие вернуло `true` — код будет запущен, если же нет — не будет.

Устранение ошибок *[гл. 10]* — поиск причины неполадки и попытка её устранения.

Функция *[гл. 4]* — блок кода, предназначенный для выполнения определённой задачи.

Целочисленное деление *[гл. 2]* — деление, при котором любые знаки после запятой (если таковые есть) округляются, а остаток деления выносится отдельно от частного (то есть результата действия деления).

Цепочка *[гл. 8]* — последовательное применение одного метода за другим на объект в одной и той же строчке кода.

Цикл *[гл. 9]* — блок кода, последовательно перезапускающий сам себя до тех пор, пока условие не перестанет быть истинным.

Число *[гл. 2]* — один из типов данных. По сути, любое число или даже бесконечность. Может быть отрицательным, положительным, нулём; иметь или же не иметь знаки после запятой. Всегда отображается без кавычек.

Экранирование *[гл. 3]* — применение обратного слэша (`\`) для «скрытия» кавычек, апострофов и так далее, чтобы не допустить ошибку. Например: `let имяСАпострофом = 'д\'Артаньян';`.

Язык программирования *[гл. 1]* — набор слов и символов, «понимаемых» и интерпретируемых компьютером определённым образом. Примеры популярных языков программирования: JavaScript, Python, Java, PHP.

Chrome *[гл. 1]* — интернет-браузер, который нам помогает на протяжении всей книги.

DIY *[гл. 1]* — Do It Yourself, сделай сам. Можете сперва попытаться выполнить задание самостоятельно, а потом свериться с рекомендациями в ответах.

D.R.Y. *[гл. 4]* — Don't Repeat Yourself, принцип сухости. Подразумевает отсутствие ненужных повторений и стремится как можно «суше» и лаконичнее писать код.

JavaScript *[гл. 1]* — широко распространённый язык программирования, использующийся в большинстве веб-сайтов. Язык, изучению которого и посвящена эта книга.

Null *[гл. 6]* — один из пяти примитивных типов данных в JavaScript. Означает попросту «ничего». Является «ложкой».

Undefined *[гл. 6]* — один из пяти примитивных типов данных в JavaScript. Является значением по умолчанию любой переменной, подразумевая, что пока никакого другого значения ей присвоено не было. Является «ложкой».

Об авторе

Джереми Мориц профессионально программирует с 2004 года; в настоящее время работает ведущим программистом крупной отраслевой компании в Канзасе. Помимо компьютерных дел Джереми ещё и успешный шахматный тренер. Впрочем, неудивительно: ведь Джереми начинал вообще как хореограф, театральный режиссёр и даже актёр! Кстати говоря, на его счету постановки более 100 мюзиклов с участием взрослых и детей. Джереми уже много лет преподаёт основы программирования, а также обучает на дому собственных детей (уже шестерых!), так что его вполне можно назвать экспертом по части детской психологии и методики преподавания. Имея обширные познания в программировании вкупе с богатым театральным опытом, Джереми создаёт поразительно увлекательную атмосферу, способствующую обучению, богатую полезной информацией и весёлыми шутками. В настоящее время Джереми и Кристин, автор замечательных иллюстраций в этой книжке и его любимая жена, проживают в Канзасе. Связаться с ними можно через сайт www.CodeForTeens.com!

Об иллюстраторе

Кристин Мориц начала рисовать с тех самых пор, когда её руки достаточно окрепли, чтобы суметь удержать мелок. Она обожает рисование и вообще любое творческое предприятие. В свободное от живописи время Кристин любит поиграть на синтезаторе, приготовить что-нибудь вкусненькое, а порой даже поработать дрелью или молотком!

«Учимся кодить на JavaScript» — первая её серьёзная работа в качестве иллюстратора, но далеко не первая в длинном списке совместных проектов с её мужем Джереми.

Оглавление

Джереми Мориц

Учимся кодить на JavaScript

Перевел с английского А. Чёрный

Заведующая редакцией	*Ю. Сергиенко*
Руководитель проекта	*Н. Римицан*
Ведущий редактор	*К. Тульцева*
Литературный редактор	*А. Руденко*
Художественный редактор	*В. Мостипан*
Корректоры	*С. Беляева, Г. Шкатова*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 04.2019. Наименование: детская литература. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 25.03.19. Формат 70×100/16. Бумага офсетная. Усл. п. л. 20,640. Тираж 2000. Заказ 0000.