ЧТО ТАКОЕ JAVASCRIPT?

Компьютеры — необычайно мощные машины, способные делать потрясающие вещи. Например, они могут играть в шахматы, обслуживать тысячи интернет-страничек и менее чем за несколько секунд выполнять миллионы сложных вычислений. Однако сами по себе компьютеры неразумны, и делают они лишь то, что прикажут люди. Мы сообщаем компьютерам, что нам от них нужно, с помощью наборов инструкций, которые называются программами. Без программ компьютеры вообще ничего не умеют! Знакомьтесь: JavaScript Более того, компьютеры не знают ни английского, ни русского, ни других естественных языков; и компьютерные программы создают на специальных языках программирования. Одним из таких языков является JavaScript. Даже если вы слышите про JavaScript впервые, вы определенно заходили на сайты, которые его используют. Например, JavaScript может управлять внешним видом странички или делать так, чтобы страница реагировала на нажатие клавиши или перемещение мышки. Такие сайты, как Gmail, Facebook и Twitter, используют JavaScript для облегчения работы с почтой, отправки комментариев или улучшения навигации. К примеру, когда вы, читая в Twitter сообщения от @nostarch, проматываете страничку вниз и видите все больше и больше сообщений, это происходит благодаря JavaScript. Чтобы понять, чем же так хорош JavaScript, достаточно посетить несколько сайтов: 18 Часть I. Основы • JavaScript позволяет проигрывать музыку и создавать яркие визуальные эффекты. Например, вы можете полетать в интерактивном видеоклипе от студии HelloEnjoy на песню Элли Голдинг Lights (http://lights.helloenjoy.com/), рис. 1.1.

С помощью JavaScript можно создавать инструменты для творчества. Patatap (http://www.patatap.com/) — это нечто вроде виртуальной драм-машины, которая издает всевозможные шумы и звуки, а также проигрывает забавные анимации, рис. 1.2.

JavaScript дает нам возможность играть в увлекательные игры. CubeSlam (https://www.cubeslam.com/) — это трехмерное подобие классической игры «Понг», похожее на аэрохоккей. Посоревнуйтесь с кем-нибудь из друзей или с медведем, за которого играет компьютер. См. рис. 1.3.

Зачем изучать JavaScript? JavaScript — далеко не единственный язык программирования. В сущности, языков очень много, счет идет на сотни, однако есть немало причин выбрать именно JavaScript. Например, изучать его гораздо проще (и интереснее), чем многие другие языки. Но, пожалуй, самая веская причина такова: чтобы писать и выполнять JavaScript-программы, достаточно интернет-браузера — такого, как Internet Explorer, Mozilla Firefox или Google Chrome. В каждый из этих браузеров встроен интерпретатор JavaScript, который сможет выполнять JavaScript-программы. И никакого специального программного обеспечения вам не понадобится. Написав программу на JavaScript, отправьте ссылку на нее другим людям, и они тоже смогут ее запустить — у себя на компьютере, в браузере (см. «Обмен кодом с помощью JSFiddle» на с. 280). Пробуем JavaScript Давайте напишем простую JavaScript-программку с помощью браузера Google Chrome (www.google.com/chrome). Установите Chrome на свой компьютер (если он еще не установлен), запустите его и введите слова about:blank в адресной строке. Теперь нажмите ENTER — откроется пустая страничка, как на рис. 1.4. Рис. 1.3. Игра CubeSlam написана целиком на JavaScript! 20 Часть I. Основы Начнем с программирования в JavaScript-консоли Chrome (это секретный инструмент для тестирования коротких программ на JavaScript). Если ваш компьютер работает под управлением Microsoft Windows или Linux, нажмите и не отпускайте клавиши CTRL и SHIFT, а затем нажмите J. Если же вы пользуетесь системой MacOS, нажмите и удерживайте COMMAND и OPTION, а затем нажмите J. Если вы все сделали правильно, то увидите пустую веб-страницу, под которой стоит значок угловой скобки (>), а после него мигает курсор (|). Здесь нам и предстоит писать код на языке JavaScript! ! Текст в консоли Chrome подсвечивается разными цветами в зависимости от типа данных. В этой книге код для ввода в консоль напечатан такими же цветами там, где это имеет значение. Но там, где разноцветный код будет вас только отвлекать, синим мы будем выделять то, что сами вводим в консоль, а данные, которые автоматически выдаст программа, будут цветными. В отличие от простого сложения, с которого мы начали, эта программа занимает несколько строк. Чтобы ввести ее в консоли, нужно будет в конце каждой строки переходить на новую строку нажатием SHIFT-ENTER. (Если нажать просто ENTER, Chrome попытается выполнить те команды, которые вы уже ввели, и программа не будет работать правильно. Сами по себе компьютеры ничего не соображают — я предупреждал!) Введите в консоли браузера:

// Рисуем столько котиков, сколько захотим! var drawCats = function (howManyTimes) { for (var i = 0; i < howManyTimes; i++) { console.log(i + " =^.^="); } }; drawCats(10); // Вместо 10 тут может быть другое число

Когда вы введете код и нажмете ENTER, JavaScript должен запустить (иначе говоря, выполнить) ваш код, показав на следующей строке результат (когда он есть). Например, введите в консоли:

В конце последней строки нажмите ENTER, а не SHIFT-ENTER. Программа должна напечатать следующее:

0 =^.^= 1 =^.^= 2 =^.^= 3 =^.^= 4 =^.^= 5 =^.^= 6 =^.^= 7 =^.^= 8 =^.^= 9 =^.^=

Если при вводе программы вы где-то ошиб лись, результат может оказаться другим — возможно, вы даже получите сообщение об ошибке. Это я и имел в виду, говоря, что компьютеры неразумны, — даже простейшая программа должна быть написана идеально, чтобы компьютер понял, что от него требуется! Я не буду сейчас вдаваться в подробности, объясняя, как работает этот код (мы еще вернемся к нему в восьмой главе), однако давайте рассмотрим некоторые особенности этой программы, да и JavaScript-программ в целом. Синтаксис В нашей программе встречается много символов, таких как скобки (), точки с запятой ;, фигурные скобки {}, знаки плюс +, а также некоторые таинственные на первый взгляд слова (например, var и console.log). Все это является частью синтаксиса JavaScript — то есть правил, указывающих, как объединять символы и слова, чтобы составить работающую программу. Одна из главных сложностей при освоении нового языка программирования — запомнить правила написания команд. Поначалу легко пропустить какие-нибудь скобки или запутаться в очередности записи значений. Не волнуйтесь, с опытом вы привыкнете писать код правильно. В этой книге мы будем изучать материал медленно, постепенно знакомясь с новыми командами языка, чтобы вы могли писать все более и более мощные программы. Комментарии В первой строке нашей программы написано: // Рисуем столько котиков, сколько захотим! 1. Что такое JavaScript? 23 Это называется комментарий. Программисты пишут комментарии, чтобы другим программистам было легче читать и понимать их код. Компьютер же комментарии игнорирует. В JavaScript комментарии начинаются с двух символов наклонной черты (//). Все, что идет следом за ними (в той же строке), интерпретатор JavaScript пропускает, поэтому комментарии не оказывают влияния на выполнение программы — это всего лишь пояснение. В примерах кода, которые встретятся вам в этой книге, комментарии описывают, что и как там происходит. При написании своего кода тоже добавляйте комментарии — когда вы заглянете в программу некоторое время спустя, они напомнят вам, как работает код и что происходит на том или ином этапе. В конце нашей программы-примера есть еще один комментарий. Напоминаю: все, что записано после символов //, компьютер игнорирует! drawCats(10); // Вместо 10 тут может быть другое число Комментарии могут занимать отдельную строку или следовать сразу после кода. Но если вы поставите // перед кодом, вот так: // drawCats(10); ...то не произойдет вообще ничего! Chrome решит, что вся эта строка — комментарий, хоть там и записаны инструкции на языке JavaScript. Когда вы, помимо примеров в этой книге, начнете изучать чужой JavaScript-код, вам будут попадаться комментарии, которые выглядят иначе: /\* Рисуем столько котиков, сколько захотим! \*/ Это другая разновидность комментариев; их обычно используют, когда текст примечания не помещается на одной строке. Однако принцип здесь тот же: текст, записанный между /\* и \*/, — это комментарий, и выполнять его компьютер не будет. Что мы узнали В этой главе мы познакомились с языком JavaScript и узнали, что можно делать с его помощью. Кроме того, мы научились запускать JavaScript-код в браузере Google Chrome и ввели несложную программу-пример. Все примеры из этой книги можно (и нужно!) запускать в JavaScript-консоли Google Chrome (если только я не скажу, что этого делать не надо). Просто читать код недостаточно — проверяйте, как он работает! Это единственный способ научиться программировать. В следующей главе мы приступим к изучению основ языка JavaScript, начиная с трех основных типов данных, с которыми вам предстоит работать: чисел, строк и булевых значений.

ТИПЫ ДАННЫХ И ПЕРЕМЕННЫЕ Программирование — это работа с данными, но что такое данные? Данные — это информация, которая хранится в наших компьютерных программах. Например, ваше имя — это элемент данных, и ваш возраст тоже. Цвет волос, количество братьев и сестер, ваш адрес и пол — все это данные. В JavaScript есть три основных типа данных: числа, строки и булевы значения. Числа — они и есть числа, тут все понятно. Например, числом можно выразить возраст или рост. В JavaScript числа записываются так: 5; Любые текстовые данные записываются в строки. В JavaScript ваше имя можно выразить строкой (так же как и адрес вашей электронной почты). Строки выглядят так: "Привет, я строка"; Булевы значения могут хранить одну из двух величин — либо это true («истина»), либо false («ложь»). Например, таким способом можно показать, носите ли вы очки или любите ли вы брокколи. 26 Часть I. Основы Пример булева значения: true; С данными разных типов и обращаться следует по-разному. Например, перемножить два числа можно, а перемножить две строки — нет. Зато, имея строку, можно выделить пять ее первых символов. Взяв два булевых значения, можно проверить, являются ли они оба «истиной» (true). Вот все эти действия на примере: 99 \* 123; 12177 "Вот длинная строка".slice(0, 3); "Вот" true && false; false Любые данные в JavaScript — не более чем сочетание этих основных типов. Далее мы по очереди рассмотрим каждый тип данных и изучим различные способы работы с ними. ! Наверное, вы заметили, что все эти команды оканчиваются на точку с запятой (;). Этим символом обозначают конец каждой отдельной команды или инструкции языка JavaScript — примерно так же, как точка отмечает конец предложения. Числа и операторы JavaScript позволяет выполнять основные математические операции, такие как сложение, вычитание, умножение и деление. Для их записи используются символы +, −, \* и /, которые называют опе раторами. Консоль JavaScript можно использовать как калькулятор. Один из примеров — сложение 3 и 4 — нам уже знаком. Давайте вычислим что-нибудь посложнее: сколько будет 12345 плюс 56789? 12345 + 56789; 69134 Посчитать это в уме не так уж просто, а JavaScript мгновенно справился с задачей. 2. Типы данных и переменные 27 Можно сложить несколько чисел с помощью нескольких знаков «плюс»: 22 + 33 + 44; 99 Также JavaScript умеет вычитать... 1000 - 17; 983 умножать (с помощью символа «звездочка»)... 123 \* 456; 56088 и делить (с помощью косой черты — слэша)... 12345 / 250; 49.38 Кроме того, можно объединять эти простые операции, составляя более сложные выражения, вроде такого: 1234 + 57 \* 3 - 31 / 4; 1397.25 Есть один нюанс — результат вычислений зависит от порядка, в котором JavaScript выполняет отдельные операции. В математике существует правило, по которому умножение и деление выполняются прежде, чем сложение и вычитание, и JavaScript ему следует. Порядок, в котором интерпретатор JavaScript выполняет эти операции, показан на рис. 2.1. Сначала он умножает 57 \* 3, получая 171 (выделено красным). Затем делит 31 / 4, получая 7.75 (выделено синим). Затем складывает 1234 + 171, получая 1405 (выделено зеленым). И наконец, вычитает 1405 − 7.75, что дает 1397.25 — окончательный результат. Но как быть, если вы хотите выполнить сложение и вычитание до умножения и деления? Для примера предположим, что у вас есть 1 брат, 3 сестры и 8 карамелек, которые 1397.25 1234 + 171 - 7.75 1234 + 171 - 31 / 4 1234 + 57 \* 3 - 31 / 4 1405 - 7.75 Рис. 2.1. Очередность выполнения операций: умножение, деление, сложение, вычитание 28 Часть I. Основы вы решили поровну разделить между ними (свою долю вы уже прикарманили). Нужно разделить 8 на общее количество братьев и сестер. Попытаемся это сделать: 8 / 1 + 3; 11 Это не может быть верным ответом! Не получится дать каждому родственнику по 11 карамелек, если у вас всего-то 8 конфет! Проблема в том, что JavaScript выполняет деление прежде, чем сложение, то есть он делит 8 на 1 (что равно 8) и затем прибавляет 3, получая в результате 11. Чтобы исправить эту ошибку, заставим JavaScript сначала выполнить сложение, воспользовавшись скобками: 8 / (1 + 3); 2 Так гораздо лучше — вышло по две карамельки каждому из родственников. Скобки вынудили JavaScript сложить 1 и 3 до деления 8 на 4. ПОПРОБУЙТЕ! Предположим, ваша подруга пытается подсчитать с помощью JavaScript, сколько ей нужно купить воздушных шаров. Она устраивает вечеринку и хочет, чтобы каждый из гостей смог надуть по 2 шарика. Сначала было приглашено 15 человек, но потом ваша подруга позвала еще 9. Она написала такой код: 15 + 9 \* 2; 33 Однако ответ, судя по всему, неверен. Где надо поставить скобки, чтобы JavaScript сначала складывал, а потом умножал, и сколько шариков нужно вашей подруге на самом деле? Переменные Значениям в JavaScript можно давать имена, используя переменные. Переменная похожа на ящичек, в который помещается лишь один предмет. Чтобы положить туда что-то еще, прежнее содержимое придется заменить. Чтобы создать новую переменную, используйте ключевое слово var, после которого укажите имя переменной. Ключевое слово — это слово, 2. Типы данных и переменные 29 обладающее для JavaScript особым значением. В данном случае, когда JavaScript встречает слово var, он понимает, что следом указано имя новой переменной. Например, вот как создать переменную с именем nick: var nick; undefined Мы создали новую переменную под названием nick. В ответ консоль выдала undefined — «значение не определено». Однако это не ошибка! JavaScript всегда так делает, если команда не возвращает какого-либо значения. Вы спросите, а что такое «возвращать значение»? Вот пример: когда вы ввели 12345 + 56789;, консоль вернула значение 69134. Однако в JavaScript команда создания переменной никакого значения не возвращает, поэтому интерпретатор печатает undefined. В этом примере и дальше мы будем давать переменным англоязычные имена, потому что английский — основной язык всей IT-области и программы принято писать только латиницей (кроме комментариев и строковых значений). Использовать русскоязычные имена переменных — это как если при составлении математических уравнений вместо x и y вы использовали бы русские буквы. Можно, но не принято. Итак, чтобы задать переменной значение, используйте знак «равно»: var age = 12; undefined Задание значения переменной называют присваиванием (здесь мы присваиваем значение 12 переменной age). И опять в консоли появляется undefined, поскольку мы только что создали новую переменную. (В дальнейших примерах я буду пропускать это undefined.) Теперь в интерпретаторе есть переменная age, которой присвоено значение 12. И если ввести в консоли имя age, интерпретатор выдаст значение этой переменной: age; 12 Здорово! При этом значение переменной не высечено в камне (переменные потому так и зовутся, что могут менять значения), и, если вам вздумается его обновить, просто используйте знак «равно» еще раз. age = 13; 13 Undefined — значение не определено Аge — возраст 30 Часть I. Основы На этот раз я не использовал ключевое слово var, поскольку переменная age уже существует. Писать var нужно только при создании переменной, а не при ее использовании. И обратите внимание: поскольку мы не создавали новой переменной, команда присваивания вернула значение 13, которое и было напечатано в следующей строке. Вот чуть более сложный пример — решение задачи про карамельки без помощи скобок: var numberOfSiblings = 1 + 3; var numberOfCandies = 8; numberOfCandies / numberOfSiblings; 2 Сначала мы создали переменную с именем numberOfSiblings (количество братьев и сестер) и присвоили ей значение выражения 1 + 3 (которое JavaScript вычислил, получив 4). Потом мы создали переменную numberOfCandies (количество карамелек) и присвоили ей значение 8. И наконец, мы ввели: numberOfCandies / numberOfSiblings. Поскольку переменная numberOfCandies содержит значение 8, а numberOfSiblings — 4, JavaScript вычислил, сколько будет 8 / 4, вернув в результате 2. Имена переменных Вводя имена переменных, будьте внимательны и не допускайте опечаток. Даже если вы перепутаете строчные и заглавные буквы, интерпретатор JavaScript не поймет, чего вы от него хотите! Например, если вы случайно введете имя numberOfCandies со строчной буквой c, возникнет ошибка: numberOfcandies / numberOfSiblings; ReferenceError: numberOfcandies is not defined Увы, JavaScript следует вашим указаниям буквально. Если вы неправильно ввели имя переменной, JavaScript не поймет, что вы имели в виду, и выдаст сообщение об ошибке. Еще один нюанс именования переменных в JavaScript — в именах не должно быть пробелов, из-за чего они могут оказаться сложными для чтения. Если бы я назвал переменную numberofcandies, без заглавных букв, читать программу стало бы труднее, поскольку неясно, где в этом имени заканчиваются отдельные слова. Один из обычных способов решения этой проблемы — писать каждое слово с заглавной буквы: NumberOfCandies. Такую манеру именования Number of siblings — число братьев и сестер Number of candies — число конфет Reference error — ошибка данных 2. Типы данных и переменные 31 называют верблюжьей записью, поскольку выпирающие заглавные буквы напоминают верблюжьи горбы. Имена переменных принято начинать со строчной буквы, поэтому с заглавной буквы обычно пишут все слова имени, кроме самого первого: numberOfCandies. В этой книге я также использую эту форму верблюжьей записи; впрочем, вы можете называть свои переменные как вам угодно! Создание новых переменных на основе вычислений Можно создавать новые переменные, выполняя математические действия с переменными, созданными ранее. Давайте с помощью переменных выясним, сколько секунд в году и каков ваш возраст в секундах! Но для начала разберемся, сколько секунд в одном часе. Сколько секунд в часе Сначала создадим две новые переменные — secondsInAMinute (количество секунд в минуте) и minutesInAHour (количество минут в часе) — и присвоим им обеим значение 60 (поскольку, как мы знаем, в минуте 60 секунд, а в часе 60 минут). Теперь создадим переменную secondsInAHour (количество секунд в часе), и пусть ее значение равняется secondsInAMinute умножить на minutesInAHour. И наконец в строке υ введем secondsInAHour, что означает «покажи мне содержимое переменной secondsInAHour», и JavaScript тут же выдаст ответ: 3600. var secondsInAMinute = 60; var minutesInAnHour = 60; var secondsInAnHour = secondsInAMinute \* minutesInAnHour; υ secondsInAnHour; 3600 Сколько секунд в сутках Теперь создадим переменную hoursInADay (количество часов в сутках) и присвоим ей значение 24. Затем создадим переменную secondsInADay (количество секунд в сутках), и пусть она равняется secondsInAHour умножить на hoursInADay. Запросив в строке υ значение secondsInADay, получим 86 400 — именно столько секунд в сутках. Seconds in a minute — секунд в минуте Minutes in a hour — минут в часе Hours in a day — часов в день 32 Часть I. Основы var hoursInADay = 24; var secondsInADay = secondsInAnHour \* hoursInADay; υ secondsInADay; 86400 Сколько секунд в году И наконец, создадим переменные daysInAYear (количество дней в году) и secondsInAYear (количество секунд в году): daysInAYear присвоим значение 365, а secondsInAYear пусть равняется secondsInADay умножить на daysInAYear. Запрашиваем значение secondsInAYear и видим, что это число 31 536 000 (более 31 миллиона секунд!). var daysInAYear = 365; var secondsInAYear = secondsInADay \* daysInAYear; secondsInAYear; 31536000 Возраст в секундах Теперь, зная, сколько секунд в году, вы можете запросто узнать свой возраст в секундах (с точностью до последнего дня рождения). К примеру, когда я пишу эти строки, мне 29 лет: var age = 29; age \* secondsInAYear; 914544000 Чтобы вычислить свой возраст в секундах, введите тот же самый код, но замените значение переменной age на ваш возраст. Или просто замените эту переменную на число, соответствующее вашему возрасту: 29 \* secondsInAYear; 914544000 Смотрите-ка, мне исполнилось больше 900 миллионов секунд! А вам? Инкремент и декремент Вам как программисту понадобится увеличивать или уменьшать значения числовых переменных на единицу. Например, у вас в программе может быть переменная для подсчета, сколько раз за день вам сказали Days in a year — дней в году 2. Типы данных и переменные 33 «Дай пять!». И при каждом новом приветствии эту переменную надо будет увеличить на 1. Увеличение на 1 называют инкрементом, а уменьшение на 1 — декрементом. Выполняются инкремент и декремент с помощью операторов ++ и --. var highFives = 0; ++highFives; 1 ++highFives; 2 --highFives; 1 После выполнения оператора ++ значение highFives (количество приветствий) увеличится на 1, а после выполнения оператора -- уменьшится на 1. Также эти операторы можно писать после имени переменной — эффект будет прежним, однако после выполнения такой команды JavaScript вернет первоначальное значение переменной, каким оно было до инкремента или декремента. highFives = 0; highFives++; 0 highFives++; 1 highFives; 2 В этом примере мы сначала обнулили значение highFives. Команда highFives++ увеличивает переменную на 1, но число, которое печатает после этого JavaScript, является значением до инкремента. Однако, запрашивая значение highFives в самом конце (после двух инкрементов), мы получаем 2. += (плюс-равно) и –= (минус-равно) Чтобы увеличить значение переменной на заданное число, можно написать такой код: var x = 10; x = x + 5; x; 15 High fives — дай пять! 34 Часть I. Основы Сначала мы создаем переменную x и даем ей значение 10. Затем присваиваем x значение x + 5 — то есть используем старое значение x, чтобы получить новое значение. Таким образом, выражение x = x + 5 по сути означает «увеличить x на 5». В арсенале JavaScript есть более простой способ увеличения или уменьшения переменной на заданную величину: это операторы += и −=. Пусть у нас есть переменная x, тогда команда x += 5 означает то же самое, что и x = x + 5. Оператор −= работает аналогично, то есть x −= 9 соответствует x = x – 9 (уменьшить x на 9). С помощью этих операторов можно, например, управлять подсчетом очков в игре: var score = 10; score += 7; 17 score -= 3; 14 В этом примере мы сначала присваиваем переменной score (счет игры) начальное количество очков (10). Потом, победив монстра, мы увеличиваем счет на 7 очков с помощью оператора += (score += 7 соответствует score = score + 7). Поскольку изначально в score было число 10, а 10 + 7 = 17, этой командой мы установили счет в 17 очков. После победы над монстром мы столкнулись с метеоритом, и счет уменьшился на 3 очка. Опять же, score -= 3 это то же самое, что и score = score - 3. Поскольку перед этим в score было 17, score - 3 равняется 14; это число и будет новым значением score. ПОПРОБУЙТЕ! Есть и другие операторы, похожие на += и -=. Например, \*= и /=. Как вы думаете, для чего они? Опробуйте их в деле с воздушными шариками: var balloons = 100; balloons \*= 2; ??? Что делает команда balloons \*= 2? А теперь попробуйте такой код: var balloons = 100; balloons /= 4; ??? Что делает команда balloons /= 4? Score — счет Balloon — воздушный шар 2. Типы данных и переменные 35 Строки До сих пор мы имели дело только с числами. Пора познакомиться с еще одним типом данных — со строками. В JavaScript (как и в большинстве других языков программирования) строка является набором символов — букв, цифр, знаков пунктуации и пробелов. Чтобы JavaScript знал, где начинается и заканчивается строка, ее берут в кавычки. Вот классический пример с фразой «Привет, мир!»: "Привет, мир!"; "Привет, мир!" Чтобы создать строку, поставьте знак двойной кавычки ("), затем введите какой-нибудь текст и закройте строку еще одной двойной кавычкой. Можно пользоваться и одинарными кавычками ('), однако, чтобы не путаться, все строки в этой книге будут в двойных кавычках. Строки можно хранить в переменных, так же как числа: var myAwesomeString = "Что-то ОЧЕНЬ крутое!!!"; Также ничто не мешает присвоить строковое значение переменной, где раньше хранилось число: var myThing = 5; myThing = "это строка"; "это строка" А что если записать в кавычках число? Строка это будет или число? В JavaScript строка остается строкой, даже если там хранятся цифровые символы. Например: var numberNine = 9; var stringNine = "9"; В переменной numberNine (число девять) хранится число, а в переменной stringNine (строка девять) — строка. Чтобы выяснить, в чем их различие, посмотрим, как они реагируют на сложение: My awesome string — Моя крутая строка My thing — моя штука Number nine — номер девять String nine — строка девять 36 Часть I. Основы numberNine + numberNine; 18 stringNine + stringNine; "99" Сложив числовые значения 9 и 9, мы получили 18. Однако при использовании оператора + со строками "9" и "9" эти строки просто склеиваются воедино, образуя "99". Объединение строк Как мы только что убедились, оператор + можно использовать и со строками, однако действует он при этом совсем иначе, чем с числами. С помощью оператора + строки можно объединять: результатом будет новая строка, состоящая из первой строки, к концу которой присоединена вторая: var greeting = "Привет"; var myName = "Ник"; greeting + myName; "ПриветНик" Здесь мы создали две переменные (greeting и myName) и присвоили каждой из них строковое значение ("Привет" и "Ник" соответственно). При сложении этих переменных строки объединяются, образуя новую строку — "ПриветНик". Впрочем, не все тут идеально — между "Привет" и "Ник" должен стоять пробел и запятая. JavaScript не ставит пробелов по собственной инициативе, зато его можно попросить его об этом, добавив пробел к одной из первоначальных строк: υ var greeting = "Привет, "; var myName = "Ник"; greeting + myName; "Привет, Ник" Дополнительный пробел перед закрывающей кавычкой в строке υ дает пробел в середине результирующей строки. Помимо их объединения, со строками можно выполнять множество разных действий. Вот несколько примеров. Greeting — приветствие My name — мое имя 2. Типы данных и переменные 37 Как узнать длину строки Чтобы узнать длину строки, достаточно добавить к ее концу .length: "Суперпупердлиннаястрока".length; 23 Можно добавлять .length к концу как самой строки, так и переменной, содержащей строку: var java = "Java"; java.length; 4 var script = "Script"; script.length; 6 var javascript = java + script; javascript.length; 10 Здесь мы присвоили строковое значение "Java" переменной java, а значение "Script" — переменной script. Затем мы добавили .length к концу каждой из переменных, узнав таким образом длины отдельных строк, а также длину составленной из них новой строки. Обратите внимание: я говорил «можно добавлять .length к концу как самой строки, так и переменной, содержащей строку». Это касается очень важного свойства переменных: в любом месте программы, где допустимо использовать число или строку, можно также использовать переменную, в которой хранится число или строка. Получение отдельного символа строки Иногда требуется получить из строки одиночный символ. Например, вы можете зашифровать в наборе слов тайное послание, состоящее из вторых символов каждого слова. Тогда, чтобы узнать это послание, нужно получить все вторые символы и объединить их в новую строку. Чтобы получить символ, стоящий в определенной позиции строки, используйте квадратные скобки — []. Возьмите строку (или переменную, в которой хранится строка) и поставьте сразу после нее квадратные скобки, в которых указана позиция нужного символа. Например, чтобы получить первый символ строковой переменной myName, используйте запись myName[0]: Length — длина 38 Часть I. Основы var myName = "Ник"; myName[0]; "Н" myName[1]; "и" myName[2]; "к" Обратите внимание — чтобы получить первый символ, мы указали в скобках позицию 0, а не 1. Дело в том, что JavaScript (как и многие другие языки программирования) ведет отсчет символов с нуля. Таким образом, для получения первого символа строки указывайте позицию 0, второго — 1 и т. д. Попробуем разгадать наш тайный шифр, где во вторых буквах некоторого набора слов скрыто послание. Вот как это сделать: var codeWord1 = "обернись"; var codeWord2 = "неужели"; var codeWord3 = "огурцы"; var codeWord4 = "липкие"; var codeWord5 = "?!"; codeWord1[1] + codeWord2[1] + codeWord3[1] + codeWord4[1] + 8 codeWord5[1]; "беги!" ! Страницы этой книги недостаточно широки, чтобы напечатать выражение, в котором мы объединяем буквы, одной строкой. Значками 8 помечены места, где код пришлось перенести на следующую строку. Однако вы, вводя этот код в компьютер, можете напечатать его без переносов. И снова обращаю внимание — второй символ каждой строки мы получаем, указав позицию 1. Получение среза строки Чтобы получить часть, или «срез», строки, используйте slice. Например, представьте, что вам нужен отрывок из длинного описания фильма для анонса на вашем сайте. Чтобы воспользоваться slice, поставьте в конце строки (или переменной, содержащей строку) точку, а после нее слово slice и круглые скобки. В скобках укажите позицию первого символа той части строки, которую вы хотите получить, затем запятую, а затем позицию последнего символа. На рис. 2.2 показано, как использовать slice. Сode word — кодовое слово Slice — часть 2. Типы данных и переменные 39 Например: var longString = "Эта длинная строка такая длинная"; longString.slice(4, 18); "длинная строка" Первое число в скобках — позиция символа, с которого начинается срез, а второе число — позиция символа, который следует за последним символом среза. На рис. 2.3 показано, каким символам соответствуют эти значения. Начальная (4) и конечная (18) позиции выделены синим цветом. Эта длинная строка такая длинная 0123 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 Рис. 2.3. В нашем примере slice возвращает символы, обведенные серой рамкой По сути, мы попросили JavaScript: «Вырежи из этой длинной строки часть, которая начинается с символа в позиции 4 и продолжается до позиции 18». Если указать в скобках после slice только одно число, мы получим строку-срез, которая начинается с данной позиции и длится до конца строки: var longString = "Эта длинная строка такая длинная"; longString.slice(4); "длинная строка такая длинная" Перевод строки в заглавный или строчный регистр Если нужно вывести какой-нибудь текст заглавными буквами, воспользуйтесь toUpperCase. "Эй, как дела?".toUpperCase(); "ЭЙ, КАК ДЕЛА?" Long string — длинная строка строка Эти два числа задают начало и конец среза. Рис. 2.2. Использование slice для получения среза строки To upper case — в верхний регистр 40 Часть I. Основы .toUpperCase() возвращает новую строку, все буквы в ко то рой — заглавные. Можно произвести и обратную операцию, использовав toLowerCase: "эЙ, кАК деЛа?".toLowerCase(); "эй, как дела?" .toLowerCase() делает все символы строчными. Но ведь по правилам предложение должно начинаться с заглавной буквы? Как сделать первый символ строки заглавным, а остальные — строчными? ! Попробуйте сами разобраться, как с помощью только что изученных команд превратить строку "эЙ, кАК деЛа?" в "Эй, как дела?". Если ничего не выходит, сверьтесь с разделами, где рассказывается о получении символа строки и использовании slice. Когда закончите, вернитесь к этому месту и сравните свое решение с моим. Вот один из вариантов решения: υ var sillyString = "эЙ, кАК деЛа?"; ϖ var lowerString = sillyString.toLowerCase(); ω var firstCharacter = lowerString[0]; ξ var firstCharacterUpper = firstCharacter.toUpperCase(); ψ var restOfString = lowerString.slice(1); ζ firstCharacterUpper + restOfString; "Эй, как дела?" Давайте разберем этот код построчно. В строке υ мы создаем новую переменную sillyString и кладем в нее строку, которую собираемся изменить. В строке ϖ мы с помощью .toLowerCase() получаем версию sillyString, где все буквы строчные ("эй, как дела?"), и кладем ее в новую переменную lowerString. В строке ω мы с помощью операции [0] получаем первый символ lowerString (это «э») и сохраняем это значение в переменной firstCharacter (напоминаю, позиция 0 соответствует первому символу). Затем в строке ξ мы переводим firstCharacter в верхний регистр и сохраняем в переменной firstCharacterUpper. В строке ψ мы с помощью slice получаем все символы lowerString, начиная со второго ("й, как дела?"), и сохраняем их в переменной restOfString. И наконец, в строке ζ мы объединяем Silly string — буквально «глупая строка» Lower string — здесь «строка в нижнем регистре» First character — здесь «первая буква» First character upper — здесь «первая буква в верхнем регистре» Rest of string — оставшаяся часть строки To lower case — в нижний регистр 2. Типы данных и переменные 41 firstCharacterUpper («Э») и restOfString, что дает нам искомую строку "Эй, как дела?". Поскольку значения и переменные взаимозаменяемы, можно заменить строки со ϖ по ζ единственной строкой: var sillyString = "эЙ, кАК деЛа?"; sillyString[0].toUpperCase() + sillyString.slice(1). 8 toLowerCase(); "Эй, как дела?" Однако понять такой код сложнее, так что имеет смысл решать сложные задачи вроде этой пошагово, с помощью переменных — по крайней мере до тех пор, пока вы не привыкнете читать сложный код. Булевы значения Теперь поговорим о булевых значениях. В сущности, есть лишь два варианта таких значений — это либо true (истина), либо false (ложь). Например, вот простое выражение с булевым значением: var javascriptIsCool = true; javascriptIsCool; true Здесь мы создали новую переменную с именем javascriptIsCool и присвоили ей булево значение true. Следующей строкой мы запросили содержимое javascriptIsCool и, разумеется, получили true. Логические операции Подобно тому как числа можно объединять с помощью математических операторов (+, −, \*, / и других), булевы значения можно объединять посредством булевых (логических) операторов. Результатом выражения, составленного из булевых значений и булевых операторов, всегда будет другое булево значение (либо true, либо false). Три основных булевых оператора — это &&, || и !. Выглядят они странновато, однако после небольшой практики пользоваться ими будет несложно. Давайте познакомимся с ними поближе. && (И) Оператор && означает «и». Вслух его называют «и», либо «и-и», либо «амперсанд-амперсанд» (амперсандом называется символ &). Используйте оператор && с двумя булевыми значениями, когда нужно узнать, равны ли они оба true. Javascript is cool — Javascript — это круто 42 Часть I. Основы Например, перед тем как пойти в школу, вы хотите убедиться, что приняли душ, а также взяли рюкзак. Если оба эти условия истинны (true), можно идти в школу, но если хоть одно ложно (false), вы еще не готовы. var hadShower = true; var hasBackpack = false; hadShower && hasBackpack; false Здесь мы устанавливаем переменную hadShower («вы приняли душ?») в true, а переменную hasBackpack («вы взяли рюкзак?») в false. Далее, вводя hadShower && hasBackpack, мы спрашиваем JavaScript: «равны ли оба этих значения true»? Поскольку это не так (рюкзак не в руках), JavaScript возвращает false (то есть вы не готовы идти в школу). Давайте повторим попытку, установив на этот раз обе переменные в true: var hadShower = true; var hasBackpack = true; hadShower && hasBackpack; true Теперь JavaScript сообщает нам, что hadShower && hasBackpack равняется true. Можно идти в школу! || (ИЛИ) Булев оператор || означает «или». Так его и следует называть — «или», или даже «или-или», хотя некоторые называют его «пайпс», поскольку среди англоязычных программистов символ | зовется «пайп» («труба»). Используйте оператор || с двумя булевыми значениями для проверки, что как минимум одно из них равняется true. Предположим, вы снова готовитесь идти в школу и хотите взять с собой к обеду фрукты, причем вам неважно, будет это яблоко, или апельсин, или и то и другое. С помощью JavaScript можно проверить, есть ли у вас хотя бы один из этих плодов: var hasApple = true; var hasOrange = false; hasApple || hasOrange; true Has apple — есть яблоко Has orange — есть апельсин Had shower — принял душ Has backback — есть рюкзак 2. Типы данных и переменные 43 Выражение hasApple || hasOrange даст true, если либо hasApple («взяли яблоко?»), либо hasOrange («взяли апельсин?»), либо обе эти переменные имеют значение true. Однако если обе они равны false, выражение даст false (то есть у вас с собой нет ни одного фрукта). ! (НЕ) Оператор ! означает «не» — так его и называйте. Используйте этот оператор, чтобы превратить false в true или, наоборот, true в false. Это полезно для работы со значениями-противоположностями. Например: var isWeekend = true; var needToShowerToday = !isWeekend; needToShowerToday; false В этом примере мы установили переменную isWeekend («сейчас выходной?») в true. Затем мы дали переменной needToShowerToday («сегодня нужно принять душ?») значение !isWeekend. Оператор ! преобразует значение в противоположное — то есть, если isWeekend равно true, !isWeekend даст нам не true (то есть false). Соответственно, запрашивая значение needToShowerToday, мы получаем false (сегодня выходной, так что мыться совсем не обязательно). Поскольку needToShowerToday равно false, !needToShowerToday даст true: needToShowerToday; false !needToShowerToday; true Иными словами, то, что вам не обязательно принимать сегодня душ, — истина (true). Совмещение логических операторов Операторы дают больше возможностей, если использовать их совместно. Допустим, вам нужно идти в школу, если сегодня не выходной, и вы приняли душ, и у вас с собой есть яблоко или апельсин. Вот как с помощью JavaScript проверить, выполняются ли все эти условия: var isWeekend = false; var hadShower = true; var hasApple = false; Is weekend — выходной Need to shower today — нужно принять душ сегодня 44 Часть I. Основы var hasOrange = true; var shouldGoToSchool = !isWeekend && hadShower && (hasApple || 8 hasOrange); shouldGoToSchool; true В данном случае сегодня не выходной, вы приняли душ, у вас нет с собой яблока, зато есть апельсин — значит, нужно идти в школу. Выражение hasApple || hasOrange записано в скобках, поскольку нам важно убедиться, что эта проверка выполнена в первую очередь. Точно так же как JavaScript выполняет умножение прежде сложения, в логических выражениях он выполняет && прежде ||. Сравнение чисел с помощью булевых значений Булевы значения можно использовать для проверки чисел, если эта проверка подразумевает простой ответ: да или нет. Например, представьте, что вы работаете в парке развлечений, где один из аттракционов имеет ограничение: туда допускаются посетители ростом не менее 150 см (иначе они могут вывалиться из кабинки!). Когда кто-нибудь хочет прокатиться, он сообщает свой рост, и вам нужно понять, больше названное число или меньше. Больше Чтобы узнать, больше ли одно число, чем другое, нужно использовать оператор «больше» (>). Например, для проверки, что рост посетителя (155 см) больше, чем ограничение по росту (150 см), мы можем задать переменной height (рост посетителя) значение 155, а переменной heightRestriction (ограничение по росту) значение 150, а затем использовать оператор > для сравнения двух переменных: var height = 155; var heightRestriction = 150; height > heightRestriction; true Введя height > heightRestriction, мы просим JavaScript показать нам, больше ли первое значение, чем второе, или нет. В данном случае посетитель достаточно высок! Но что если рост посетителя в точности равен 150 см? var height = 150; var heightRestriction = 150; height > heightRestriction; false Should go to school — нужно идти в школу Height — высота Height restriction — ограничение по росту 2. Типы данных и переменные 45 Нет, посетитель недостаточно высок! Хотя если ограничение по росту — 150 см, наверное, стоит пускать и тех, чей рост в точности равен 150 см? Это нужно исправить. К счастью, в JavaScript есть еще один оператор, >=, что означает «больше или равно». var height = 150; var heightRestriction = 150; height >= heightRestriction; true Ну вот, теперь лучше — 150 удовлетворяет условию «больше или равно 150». Меньше Оператор, противоположный «больше» (>), зовется оператором «меньше» (< heightRestriction; false Мы хотим убедиться, что рост посетителя меньше ограничения, и поэтому используем <=, что означает «меньше или равно». var height = 120; var heightRestriction = 120; height <= heightRestriction; true Посетителю, рост которого равен 120 см, вход все еще разрешен. Равно Чтобы проверить два числа на точное равенство, используйте тройной знак равенства (===) — это оператор «равно». Будьте осторожны, не путайте === с одиночным знаком равенства (=), поскольку === означает «равны ли эти два числа?», а = означает «положить значение справа в переменную слева». Иначе говоря, === задает вопрос, а = присваивает переменной значение. 46 Часть I. Основы При использовании = имя переменной должно стоять слева, а значение, которое вы хотите в эту переменную положить, справа. Однако === служит лишь для проверки двух значений на равенство, поэтому неважно, какое значение с какой стороны стоит. Представьте, что вы загадали своим друзьям Чико, Харпо и Граучо число, а именно число 5. Вы облегчили им задачу, сообщив, что это число от 1 до 9, и ваши друзья начали угадывать. Сначала присвоим переменной mySecretNumber значение 5. Первый из играющих, Чико, загадывает ответ 3, который мы кладем в переменную chicoGuess. Поглядим, что будет дальше: var mySecretNumber = 5; var chicoGuess = 3; mySecretNumber === chicoGuess; false var harpoGuess = 7; mySecretNumber === harpoGuess; false var grouchoGuess = 5; mySecretNumber === grouchoGuess; true Число, которое вы загадали, находится в переменной mySecretNumber. Переменные chicoGuess, harpoGuess и grouchoGuess соответствуют предположениям ваших друзей. Далее с помощью оператора === можно проверить, равен ли какой-нибудь ответ вашему числу. Третий друг, Граучо, назвал 5 и победил. Сравнивая два числа с помощью ===, вы получаете true, только когда оба числа совпадают. Поскольку в grouchoGuess находится значение 5, а mySecretNumber также равно 5, выражение mySecretNumber === grouchoGuess вернет true. Другие варианты ответа не совпадают с mySecretNumber, поэтому сравнение с ними даст false. Также с помощью === можно сравнить две строки или два булевых значения. Если же сравнивать так значения разных типов, ответом всегда будет false. Двойной знак равенства Еще немного запутаю вас: в JavaScript есть еще один оператор сравнения (двойное равно, ==), который означает «практически равно». Используйте его для проверки двух значений на соответствие друг другу, даже если одно из них строка, а другое — число. Все значения принадлежат к тому или иному типу, так что число 5 отличается от строки «5», хоть они и выглядят похоже. Если сравнить их с помощью ===, JavaScript ответит, что значения не равны. Однако при сравнении через == они окажутся равными: Чико, Харпо и Граучо — псевдонимы троих из братьев Маркс, американских звезд немой комедии. My secret number — мое загаданное число Chico guess — догадка Чико (Харпо, Граучо) 2. Типы данных и переменные 47 var stringNumber = "5"; var actualNumber = 5; stringNumber === actualNumber; false stringNumber == actualNumber; true Возможно, тут вы подумаете: «Похоже, двойное равно удобнее, чем тройное!» Однако будьте очень осторожны: двойное равно может ввести вас в заблуждение. Например, как считаете, 0 равен false? А строка "false" значению false? При сравнении через двойное равно 0 оказывается равным false, а строка "false" не равна false: 0 == false; true "false" == false; false Дело в том, что, сравнивая значения через двойное равно, JavaScript первым делом пытается преобразовать их к одному типу. В данном случае булево значение он преобразует в числовое — при этом false становится нулем, а true — единицей. Поэтому, сравнивая 0 == false, вы получите true! Из-за всех этих странностей лучше пока пользуйтесь только оператором ===.

ПОПРОБУЙТЕ! Вас попросили написать JavaScript-код для автоматической системы управления кинотеатром. Задача состоит в том, чтобы опре делить, пускать ли зрителя на фильм «с 12 лет и старше» или нет. Правила таковы: если посетителю 12 лет или больше, он может проходить. Если ему еще не исполнилось 12, но его сопровождает взрослый, пусть тоже проходит. Во всех остальных случаях вход запрещен. var age = 11; var accompanied = true; ??? Допишите этот код, чтобы он определял, можно ли 11-летнему посетителю посмотреть фильм (возраст задается в переменной age, а переменная accompanied равна true, если посетитель пришел со взрослым). Попробуйте поменять эти значения (например, пусть в age будет число 12, а в accompanied — true) и убедитесь, что код по-прежнему находит верное решение.

То есть highFives++ это highFives = highFives + 1

0 + 1 = 1

1 + 1 = 2

Мы же каждый раз highFives переписываем. Объяснения Даши по инкрементам и декрементам.

Ответ:

var age = 12;

var accompanied = true;

var passed = age >= 12 || (age <= 12 && accompanied);

passed;

true

undefined и null И наконец, в JavaScript есть два особых значения, они называются undefined и null. Оба они означают «пусто», но смысл этого в обоих случаях немного различается. JavaScript использует значение undefined, когда не может найти иного значения. Например, если, создав новую переменную, вы не присвоите ей значение с помощью оператора =, ее значением будет undefined: var myVariable; myVariable; undefined А значение null обычно используется, чтобы явно обозначить — «тут пусто». var myNullVariable = null; myNullVariable; null Пока вы будете нечасто использовать undefined и null. Вы получите undefined, если создадите переменную и не присвоите ей значения, — JavaScript всегда возвращает undefined, когда значение не определено. Однако специально undefined обычно ничему не присваивают; если вам захочется обозначить, что в переменной «пусто», используйте для этого null. Иначе говоря, null нужен, чтобы явно показать отсутствие значения, и порой это бывает полезно. Например, есть переменная, обозначающая ваш любимый овощ. Если вы терпеть не можете все без исключения овощи, имеет смысл дать переменной «любимый овощ» значение null. Этим вы явно покажете любому, кто увидит ваш код, что у вас нет любимого овоща. Однако если в переменной будет undefined, ктонибудь может подумать, что вы просто еще не приписали ей значения. Что мы узнали Теперь вы знаете все базовые типы данных JavaScript — это числа, строки и булевы значения, — а также специальные значения null и undefined. Числа нужны для всего, что связано с математикой, строки — для работы с текстом, а булевы значения — для разрешения вопросов, на которые можно ответить «да» или «нет». Значения же null и undefined дают нам способ обозначать то, чего не существует. В следующих двух главах мы поговорим о массивах и объектах — и то и другое представляет собой способ объединения простых типов данных в более сложные наборы значений.

МАССИВЫ

Мы уже изучили числа и строки — типы данных, которые можно хранить и использовать в своих программах. Но одни лишь числа и строки — это как-то скучновато; не столь уж многое можно сделать со строкой как таковой. С помощью массивов JavaScript позволяет создавать и группировать данные более любопытными способами. А по сути своей массив — всего лишь список, где хранятся другие значения. Например, если вашему другу интересно, какие три вида динозавров вам нравятся больше всего, вы можете создать массив и расположить там по порядку названия этих динозавров: var myTopThreeDinosaurs = ["Тираннозавр", "Велоцираптор", 8 "Стегозавр"]; Теперь вместо того, чтобы показывать своему другу три отдельные строки, вы можете воспользоваться единственным массивом myTopThreeDinosaurs. Зачем нужны массивы? Вернемся к нашим динозаврам. Положим, вы решили написать программу для учета всех видов динозавров, которые вам известны. Вы можете создать для каждого вида отдельную переменную: My top three dinosaurs — три моих любимых динозавра 50 Часть I. Основы var dinosaur1 = "Тираннозавр"; var dinosaur2 = "Велоцираптор"; var dinosaur3 = "Стегозавр"; var dinosaur4 = "Трицератопс"; var dinosaur5 = "Брахиозавр"; var dinosaur6 = "Птеранодон"; var dinosaur7 = "Апатозавр"; var dinosaur8 = "Диплодок"; var dinosaur9 = "Компсогнат"; Однако пользоваться этим списком не слишком удобно — у вас есть девять переменных там, где можно обойтись лишь одной. А теперь представьте, что динозавров в программе не девять, а 1000! Пришлось бы создать 1000 отдельных переменных, работать с которыми было бы решительно невозможно. Это похоже на список покупок, составленный так, что каждая покупка указана на отдельном листе бумаги. На одном листке написано «яйца», на другом — «хлеб», на следующем — «апельсины». Большинство людей предпочли бы видеть весь список на одном листе бумаги. Так не проще ли сгруппировать всех динозавров в один список? Вот для этого и нужны массивы. Создание массива Чтобы создать массив, используйте квадратные скобки []. Фактически для задания пустого массива достаточно лишь пары квадратных скобок: []; [] Dinosaur — динозавр 3. Массивы 51 Но кому нужен пустой массив? Давайте-ка заполним его ди но - заврами! Чтобы создать массив со значениями, нужно перечислить эти значения внутри квадратных скобок, разделяя их запятыми. Отдельные значения, хранящиеся в массиве, называют элементами. В данном примере все элементы будут строковыми (это названия любимых динозавров), поэтому запишем их в кавычках. Сохраним наш массив в переменной с именем dinosaurs: var dinosaurs = ["Тираннозавр", "Велоцираптор", "Стегозавр", 8 "Трицератопс", "Брахиозавр", "Птеранодон", "Апатозавр", 8 "Диплодок", "Компсогнат"]; Длинный список сложно читать, когда он записан одной строкой, но, к счастью, это не единственный способ форматирования кода при создании массива. Вы можете поставить открывающую квадратную скобку на одной строке, каждый элемент писать с новой строки и последней строкой поставить закрывающую квадратную скобку: var dinosaurs = [ "Тираннозавр", "Велоцираптор", "Стегозавр", "Трицератопс", "Брахиозавр", "Птеранодон", "Апатозавр", "Диплодок", "Компсогнат" ]; Чтобы ввести такой код в консоли, вам придется нажимать одновременно с ENTER клавишу SHIFT каждый раз, когда нужно перейти к новой строке. Иначе JavaScript попытается выполнить то, что вы уже ввели, даже если команда еще не завершена. Поэтому при работе в консоли проще записывать массивы одной строкой. Для JavaScript неважно, отформатируете вы код, расположив весь массив на одной строке или на нескольких строках по частям. Сколько бы ни стояло переносов, JavaScript увидит один и тот же массив — в нашем случае состоящий из девяти строк. 52 Часть I. Основы Доступ к элементам массива Чтобы получить доступ к элементам массива, используйте квадратные скобки с индексом нужного вам элемента, как в этом примере: dinosaurs[0]; "Тираннозавр" dinosaurs[3]; "Трицератопс" Индекс — это номер элемента, в котором хранится значение. Аналогично символам в строке, первому элементу массива соответствует индекс 0, второму — 1, третьему — 2 и т. д. Поэтому, запросив индекс 0 в массиве dinosaurs, мы получили "Тираннозавр" (это первый элемент), а запросив индекс 3 — "Трицератопс" (четвертый элемент). Возможность доступа к отдельным элементам массива очень полезна. Например, если вы хотите показать кому-то самого-самого любимого своего динозавра, ни к чему показывать весь массив. Вместо этого просто возьмите первый элемент: dinosaurs[0]; "Тираннозавр" 3. Массивы 53 Создание и изменение элементов Используя индекс в квадратных скобках, можно задавать или изменять значения элементов и даже добавлять новые элементы. Например, чтобы заменить содержимое первого элемента массива dinosaurs ("Тираннозавр") на "Тираннозавр рекс", можно написать: dinosaurs[0] = "Тираннозавр рекс"; После этого массив dinosaurs станет таким: ["Тираннозавр рекс", "Велоцираптор", "Стегозавр", "Трицератопс", 8 "Брахиозавр", "Птеранодон", "Апатозавр", "Диплодок", 8 "Компсогнат”] С помощью индексов также можно добавлять в массив элементы. Например, вот как создать массив dinosaurs, задавая каждый элемент через квадратные скобки: var dinosaurs = []; dinosaurs[0] = "Тираннозавр"; dinosaurs[1] = "Велоцираптор"; dinosaurs[2] = "Стегозавр"; dinosaurs[3] = "Трицератопс"; dinosaurs[4] = "Брахиозавр"; dinosaurs[5] = "Птеранодон"; dinosaurs[6] = "Апатозавр"; dinosaurs[7] = "Диплодок"; dinosaurs[8] = "Компсогнат"; dinosaurs; ["Тираннозавр", "Велоцираптор", "Стегозавр", "Трицератопс", 8 "Брахиозавр", "Птеранодон", "Апатозавр", "Диплодок", 8 "Компсогнат”] Сначала создаем пустой массив: var dinosaurs = []. Затем в каждой из следующих строк добавляем по одному элементу командами dinosaurs[] с индексом от 0 до 8. Закончив наполнение массива, можно посмотреть его содержимое (набрав dinosaurs;) и убедиться, что JavaScript расположил значения по порядку, в соответствии с индексами. 54 Часть I. Основы На самом деле в массив можно добавить элемент с любым индексом. Например, чтобы добавить нового (выдуманного) динозавра с индексом 33, введем: dinosaurs[33] = "Филосораптор"; dinosaurs; ["Тираннозавр", "Велоцираптор", "Стегозавр", "Трицератопс", 8 "Брахиозавр", "Птеранодон", "Апатозавр", "Диплодок", 8 "Компсогнат", undefined × 24 "Филосораптор"] Элементы между индексами 8 и 33 получат значение undefined. При печати массива Chrome сообщает количество этих undefi ned-элементов, а не выводит каждый из них по отдельности. Разные типы данных в одном массиве Не обязательно, чтобы все элементы массива были одного типа. Например, вот массив, в котором хранится число (3), строка ("динозавры"), массив (["трицератопс", "стегозавр", 3627.5]) и еще одно число (10): var dinosaursAndNumbers = [3, "динозавры", ["трицератопс", 8 "стегозавр", 3627.5], 10]; Чтобы обратиться к элементам массива, вложенного в другой массив, нужно использовать вторую пару квадратных скобок. Например, если команда dinosaursAndNumbers[2]; вернет весь вложенный массив, то dinosaursAndNumbers[2][0]; — лишь первый элемент этого вложенного массива ("трицератопс"). dinosaursAndNumbers[2]; ["трицератопс", "стегозавр", 3627.5] dinosaursAndNumbers[2][0]; "трицератопс" Dinosaurs and numbers — динозавры и числа [3, "динозавры", ["трицератопс", "стегозавр", 3627.5], 10]; индекс [0] индекс [1] индекс [2] индекс [3] индекс [2][0] индекс [2][1] индекс [2][2] Рис. 3.1. Индексы основного массива показаны красным цветом, а индексы вложенного массива — синим 3. Массивы 55 Вводя dinosaursAndNumbers[2][0];, мы просим JavaScript обратиться к индексу 2 массива dinosaursAndNumbers, где находится массив ["трицератопс", "стегозавр", 3627.5], и вернуть значение с индексом 0 из этого вложенного массива — это первый элемент, "трицератопс". На рис. 3.1 показаны индексы для этих массивов. Работаем с массивами Работать с массивами вам помогут свойства и методы. Свойства хранят различные сведения о массиве, а методы обычно либо изменяют его, либо возвращают новый массив. Давайте разберемся. Длина массива Порой нужно знать, сколько в массиве элементов. Например, если снова и снова добавлять динозавров в массив dinosaurs, вы можете забыть, сколько их теперь всего. Для этого есть свойство length (длина), хранящее количество элементов в массиве. Чтобы узнать длину массива, просто добавьте .length после его имени. Давайте посмотрим, как это работает. Но сначала создадим новый массив с тремя элементами: var maniacs = ["Якко", "Вакко", "Дот"]; maniacs[0]; "Якко" maniacs[1]; "Вакко" maniacs[2]; "Дот" Чтобы узнать длину этого массива, добавим .length к maniacs: maniacs.length; 3 JavaScript сообщает, что в массиве 3 элемента, и мы знаем, что их индексы — 0, 1 и 2. Отсюда следует полезное наблюдение: последний индекс массива всегда на единицу меньше длины этого массива. Это значит, что есть простой способ получить последний элемент массива, какой бы ни была его длина: maniacs[maniacs.length - 1]; "Дот" Якко, Вакко и Дот — герои мультсериала «Озорные анимашки» (англ. Animaniacs) о семейке Уорнер, созданного компанией Warner Bros. 56 Часть I. Основы Мы попросили JavaScript вернуть элемент из нашего массива, но вместо числового индекса ввели в квадратных скобках выражение: длина массива минус 1. JavaScript нашел свойство maniacs.length со значением 3, вычел 1, получив 2, и наконец вернул элемент с индексом 2 — это и есть последний элемент, "Дот". Добавление элементов в массив Чтобы добавить элемент к концу массива, можно воспользоваться методом push. Введите .push после имени массива, а после в круглых скобках укажите элемент, который нужно добавить: var animals = []; animals.push("Кот"); 1 animals.push("Пес"); 2 animals.push("Лама"); 3 animals; ["Кот", "Пес", "Лама"] animals.length; 3 Командой var animals = []; мы создали пустой массив animals, а затем методом push добавили туда элемент "Кот". Потом снова использовали push, добавив "Пес", а затем "Лама". Запросив теперь содержимое массива animals, мы видим, что "Кот", "Пес" и "Лама" стоят там в том же порядке, в каком мы их добавляли. Запуск метода в программировании называется вызовом метода. При вызове метода push происходят две вещи. Во-первых, в массив добавляется элемент, указанный в скобках. Во-вторых, метод задает новую длину массива. Именно эти значения длины появляются в консоли после каждого вызова push. Чтобы добавить элемент в начало массива, используйте метод .unshift(элемент): animals; ["Кот", "Пес", "Лама"] υ animals[0]; "Кот" animals.unshift("Мартышка"); 4 animals; Push — буквально «протолкнуть, добавить» Animals — животные 3. Массивы 57 ["Мартышка", "Кот", "Пес", "Лама"] animals.unshift("Белый медведь"); 5 animals; ["Белый медведь", "Мартышка", "Кот", "Пес", "Лама"] animals[0]; "Белый медведь" ϖ animals[2]; "Кот" Мы начали с массива, созданного раньше, — ["Кот", "Пес", "Лама"]. Затем добавили в его начало элементы "Мартышка" и "Белый медведь", отчего остальные элементы сдвинулись вперед — при каждом добавлении их индексы увеличивались на 1. В результате элемент "Кот", у которого раньше был индекс 0 υ, оказался под индексом 2 ϖ. Как и push, метод unshift при каждом вызове задает новую длину массива. Удаление элементов массива Убрать из массива последний элемент можно, добавив к его имени .pop(). Метод pop делает сразу два дела: удаляет последний элемент из массива и возвращает этот элемент в виде значения. Для примера начнем с нашего массива animals ["Белый медведь", "Мартышка", "Кот", "Пес", "Лама"]. Далее создадим новую переменную lastAnimal и сохраним в ней последний элемент, вызвав animals.pop(). animals; ["Белый медведь", "Мартышка", "Кот", "Пес", "Лама"] υ var lastAnimal = animals.pop(); lastAnimal; "Лама" animals; ["Белый медведь", "Мартышка", "Кот", "Пес"] ϖ animals.pop(); "Пес" animals; ["Белый медведь", "Мартышка", "Кот"] ω animals.unshift(lastAnimal); 4 animals; ["Лама", "Белый медведь", "Мартышка", "Кот"] При вызове animals.pop() в строке υ последний элемент массива animals, "Лама", был возвращен и сохранен в переменной lastAnimal. Pop — буквально «выдавить» Last animal — последнее животное 58 Часть I. Основы Кроме того, элемент "Лама" был удален из массива, в котором после этого осталось четыре элемента. При следующем вызове animals.pop() в строке ϖ был удален из массива и возвращен элемент "Пес", а элементов в массиве осталось всего три. Вызвав animals.pop() для элемента "Пес", мы не сохранили это значение в переменной, и оно пропало. С другой стороны, элемент "Лама" был сохранен в переменной lastAnimal, чтобы при случае им можно было снова воспользоваться. В строке ω мы с помощью unshift(lastAnimal) добавили "Лама" обратно, в начало массива. В итоге получился массив ["Лама", "Белый медведь", "Мартышка", "Кот"]. Методы push и pop хорошо друг друга дополняют, поскольку порой нужно работать только с концом массива. Вы можете добавить элемент в конец вызовом push, а потом, когда это понадобится, забрать его оттуда вызовом pop. Мы рассмотрим это на примере чуть позже в этой главе. ["Белый медведь", "Мартышка", "Кот", "Пес", ] "Лама" pop ["Кот", "Пес", ] "Лама" push Чтобы удалить из массива первый элемент, вернув его значение, используйте .shift(): animals; ["Лама", "Белый медведь", "Мартышка", "Кот"] var firstAnimal = animals.shift(); firstAnimal; "Лама" animals; ["Белый медведь", "Мартышка", "Кот"] Метод animals.shift() работает аналогично animals.pop(), но элемент берется из начала массива. В начале этого примера массив animals имел вид ["Лама", "Белый медведь", "Мартышка", "Кот"]. Вызов .shift() вернул первый элемент, "Лама", который мы сохранили в переменной firstAnimal. Поскольку .shift() не только возвращает элемент, но и удаляет его, в массиве animals осталось лишь ["Белый медведь", "Мартышка", "Кот"]. First animal — первое животное 3. Массивы 59 Методы unshift и shift добавляют и удаляют элементы с начала массива — так же как push и pop добавляют и удаляют элементы с конца. [ , "Белый медведь", "Мартышка", "Кот"] "Лама" shift [ , "Белый медведь", "Мартышка", "Кот"] "Лама" unshift Объединение массивов Чтобы «склеить» два массива, создав таким образом новый массив, используйте команду firstArray. concat(otherArray). Метод concat создает массив, в котором элементы из firstArray будут расположены перед элементами из otherArray. Пускай у нас есть два списка — список пушистых животных и список чешуйчатых животных — и мы хотим их объединить. Если поместить наших пушистых животных в массив furryAnimals, а чешуйчатых — в массив scalyAnimals, команда furryAnimals. concat(scalyAnimals) создаст новый массив, в начале которого будут элементы из первого массива, а в конце — из второго. var furryAnimals = ["Альпака", "Кольцехвостый лемур", "Йети"]; var scalyAnimals = ["Удав", "Годзилла"]; var furryAndScalyAnimals = furryAnimals.concat(scalyAnimals); furryAndScalyAnimals; ["Альпака", "Кольцехвостый лемур", "Йети", "Удав", "Годзилла"] furryAnimals; ["Альпака", "Кольцехвостый лемур", "Йети"] scalyAnimals; ["Удав", "Годзилла"] First Array — первый массив Other Array — другой массив Furry animals — пушистые животные Scaly animals — чешуйчатые животные Furry and scaly animals — пушистые и чешуйчатые животные 60 Часть I. Основы Хоть команда firstArray.concat(otherArray) и возвращает массив, содержащий все элементы из firstArray и otherArray, сами эти массивы остаются прежними. Запросив содержимое furryAnimals и scalyAnimals, мы видим, что массивы не изменились. Объединение нескольких массивов С помощью concat можно объединить больше чем два массива. Для этого укажите дополнительные массивы в скобках, разделив их запятыми: var furryAnimals = ["Альпака", "Кольцехвостый лемур", "Йети"]; var scalyAnimals = ["Удав", "Годзилла"]; var featheredAnimals = ["Ара", "Додо"]; var allAnimals = furryAnimals.concat(scalyAnimals, 8 featheredAnimals); allAnimals; ["Альпака", "Кольцехвостый лемур", "Йети", "Удав", "Годзилла", "Ара", "Додо"] Мы видим, что пернатые животные из массива featheredAnimals оказались в самом конце нового массива, поскольку featheredAnimals был указан последним в скобках метода concat. Метод concat удобен, когда нужно объединить несколько массивов в один. Скажем, пусть у вас есть список любимых книг и у вашего друга тоже есть свой список, и вы решили выяснить, можно ли купить все эти книги в магазине по соседству. Тогда будет проще, если у вас будет не два списка, а один. Просто объедините ваш список со списком вашего друга методом concat, и пожалуйста — у вас один список на двоих! Поиск индекса элемента в массиве Чтобы выяснить, какой у определенного элемента индекс в массиве, используйте .indexOf("элемент"). Создадим массив colors с названиями цветов, а затем получим индексы элементов «синий» и «зеленый» с помощью команд colors.indexOf("синий") и colors. indexOf("зеленый"). Поскольку "синий" располагается по индексу 2, colors.indexOf("синий") вернет 2. А "зеленый" находится по индексу 1, так что colors.indexOf("зеленый") вернет 1. var colors = ["красный", "зеленый", "синий"]; colors.indexOf("синий"); 2 colors.indexOf("зеленый"); 1 Feathered Animals — животные с перьями All animals — все животные Color — цвет 3. Массивы 61 Метод indexOf похож на квадратные скобки, только здесь все наоборот: команда colors[2] вернет "синий", а colors.indexOf("синий") вернет 2. colors[2]; "синий" colors.indexOf("синий"); 2 Хотя элемент "синий" стоит третьим по порядку, его индекс равен 2, ведь мы всегда считаем с 0. Разумеется, то же относится к "зеленый" с индексом 1. Если элемента, индекс которого вы запрашиваете, в массиве нет, JavaScript вернет значение −1. colors.indexOf("фиолетовый"); -1 Таким образом JavaScript сообщает: «элемент не найден», так или иначе возвращая из метода число. Если элемент встречается в массиве больше чем один раз, indexOf вернет индекс того элемента, который находится ближе к началу массива. var insects = ["Пчела", "Муравей", "Пчела", "Пчела", "Муравей"]; insects.indexOf("Пчела"); 0 Превращаем массив в строку Воспользовавшись методом .join(), можно соединить все элементы массива в одну большую строку. var boringAnimals = ["Мартышка", "Кот", "Рыба", "Ящерица"]; boringAnimals.join(); "Мартышка,Кот,Рыба,Ящерица" Метод join возвращает строку, в которой через запятую перечислены все элементы массива boringAnimals. Но что если мы не хотим использовать в качестве разделителя запятую? Нам поможет метод .join ("разделитель"), который делает все то же самое, но вместо запятых ставит между элементами выбранный Join — соединить Insects — насекомые Boring animals — скучные животные 62 Часть I. Основы разделитель. Давайте попробуем три разных разделителя: дефис с пробелами по сторонам, звездочку \* и союз «и» с пробелами по сторонам. Обратите внимание: разделитель нужно записывать в кавычках — ведь это строка. var boringAnimals = ["Мартышка", "Кот", "Рыба", "Ящерица"]; boringAnimals.join(" - "); "Мартышка - Кот - Рыба - Ящерица" boringAnimals.join("\*") "Мартышка\*Кот\*Рыба\*Ящерица" boringAnimals.join(" и ") "Мартышка и Кот и Рыба и Ящерица" Этот вариант join удобен, когда у вас есть массив, из которого нужно сделать строку. Предположим, у вас много средних имен и вы решили хранить их все в массиве вместе со своим личным именем и фамилией. И вдруг кому-то понадобилось ваше полное имя в виде строки. Тогда метод join с разделителем-пробелом преобразует все имена в искомую строку: var myNames = ["Николас", "Эндрю", "Максвелл", "Морган"]; myNames.join(" "); "Николас Эндрю Максвелл Морган” Не будь метода join, пришлось бы соединять элементы вручную, что очень утомительно: myNames[0] + " " + myNames[1] + " " + myNames[2] + " " + myNames[3]; "Николас Эндрю Максвелл Морган" Кроме того, этот код сработает, только если у вас ровно два средних имени. Если же их три или одно, программу придется менять. А c join ничего менять не надо — этот метод соединит в строку все элементы массива независимо от его длины. Если же в массиве хранятся нестроковые значения, JavaScript преобразует их в строки перед тем, как соединить: var ages = [11, 14, 79]; ages.join(" "); "11 14 79" Среднее имя — имя, обычно расположенное между личным именем и фамилией. Используется как элемент полного имени, в основном в Европе и западных странах. Ages — возрасты 3. Массивы 63 Что полезного можно сделать с массивами Теперь вы умеете разными способами создавать массивы и знаете немало действий с ними. Но как все это может вам пригодиться в жизни? В этом разделе мы разберем несколько коротких программ, посвященных практическому использованию массивов. Поиск дороги домой Представьте, что ваша подруга побывала у вас в гостях, а теперь хочет показать вам свой дом. Но вот незадача — вы никогда не бывали у нее раньше, а путь назад вам предстоит проделать в одиночку. К счастью, вам в голову приходит хитрый способ решения этой проблемы: по дороге к дому подруги вы будете записывать возможные ориентиры (телефонную будку, вывеску магазина или аптеки, школу и т. д.). А по дороге назад, двигаясь по списку с конца, вычеркивать каждый встреченный ориентир — так вы всегда будете знать, куда идти дальше. Построение массива с помощью push Давайте напишем код для выполнения этих действий. Начнем с создания массива — пустого, поскольку, пока вы еще не отправились в гости, неизвестно, какие ориентиры вам повстречаются. Затем, по дороге к дому вашей подруги, мы будем добавлять описание каждого ориентира в массив с помощью push. И наконец, когда придет время идти домой, будем методом pop изымать каждый пройденный ориентир из массива. var landmarks = []; landmarks.push("Мой дом"); landmarks.push("Дорожка к дому"); landmarks.push("Мигающий фонарь"); landmarks.push("Протекающий гидрант"); landmarks.push("Пожарная станция"); landmarks.push("Приют для кошек"); landmarks.push("Моя бывшая школа"); landmarks.push("Дом подруги"); Здесь мы создали пустой массив landmarks и методом push сохранили в нем все ориентиры, замеченные по дороге к дому подруги. Движемся в обратном порядке с помощью pop Вы добрались до дома подруги, и можно изучить массив ориентиров. Разумеется, первым стоит "Мой дом", потом "Дорожка к дому", и т. д. до конца массива, где находится элемент "Дом подруги". Теперь, когда наступит время идти домой, вам останется лишь изымать из массива по одному элементу, и всегда будет понятно, куда идти дальше. Landmarks — заметные объекты 64 Часть I. Основы landmarks.pop(); "Дом подруги" landmarks.pop(); "Моя бывшая школа" landmarks.pop(); "Приют для кошек" landmarks.pop(); "Пожарная станция" landmarks.pop(); "Протекающий гидрант" landmarks.pop(); "Мигающий фонарь" landmarks.pop(); "Дорожка к дому" landmarks.pop(); "Мой дом" Вот вы и дома! Заметили, что первый ориентир, который вы поместили в массив методом push, оказался также последним, который вы извлекли методом pop? А последний добавленный ориентир оказался первым извлеченным? Может показаться, что лучше бы первый добавленный элемент и извлекался всегда первым, однако извлекать элементы в обратном порядке в некоторых случаях удобно. Такой подход нередко используется в больших программах — именно поэтому push и pop в JavaScript всегда под рукой. ! Среди программистов такой способ работы с элементами называется «стек». Представьте, что стек — это стопка блинов. Всякий раз, когда готов новый блин, его кладут сверху стопки (как метод push), и, когда вы берете блин, чтобы его съесть, вы тоже берете его сверху (как метод pop). Снятие элементов со стека похоже на путешествие назад во времени: последним изымается элемент, который был в стеке первым. То же происходит с блинами: последний блин, который вы съедите, — это первый, который был приготовлен. На сленге программистов этот способ называется «последним вошел, первым вышел», английская аббревиатура LIFO (last in, fi rst out). Есть и альтернативный подход — «первым вошел, первым вышел», аббревиатура FIFO (fi rst in, fi rst out). Его также называют очередью, поскольку таким же образом устроены очереди — первый человек, вставший в очередь, будет первым, которого обслужат. 3. Массивы 65 Случайный выбор Используя массивы, можно написать программу, которая выдает случайные варианты из заданного списка (наподобие «шара судьбы»). Однако сначала нужно разобраться, откуда нам брать случайные числа. Использование Math.random() Случайные числа можно генерировать с помощью специального метода Math.random(), который при каждом вызове возвращает случайное число от 0 до 1: Math.random(); 0.8945409457664937 Math.random(); 0.3697543195448816 Math.random(); 0.48314980138093233 Важно помнить, что Math.random() всегда возвращает число меньше 1, то есть никогда не возвращает собственно 1. Если вам нужно число побольше, просто умножьте полученное из метода Math.random() значение на подходящий коэффициент. Например, если нужно случайное число от 0 до 10, умножьте Math.random() на 10: Math.random() \* 10; 7.648027329705656 Math.random() \* 10; 9.7565904534421861 Math.random() \* 10; 0.21483442978933454 Округление с помощью Math.fl oor() И все же эти случайные значения нельзя использовать как индексы в массиве, поскольку индексы должны быть целыми числами, а не десятичными дробями. Чтобы исправить этот недостаток, нужен метод Math.floor(), округляющий число до ближайшего снизу целого значения (по сути, он просто отбрасывает все знаки после запятой). Math.floor(3.7463463); 3 Math.floor(9.9999); 9 Math.floor(0.793423451963426); 0 Math random — случайное число Math floor — целая часть числа 66 Часть I. Основы Давайте используем оба метода, чтобы получить случайный индекс. Нужно лишь умножить Math.random() на длину массива и затем округлить полученное число методом Math.floor(). Например, если в массиве четыре элемента, это можно сделать так: Math.floor(Math.random() \* 4); 2 // может выпасть 0, 1, 2 или 3 При каждом запуске этот код будет возвращать случайное число от 0 до 3 (включая 0 и 3). Поскольку Math.random() всегда возвращает значение меньше 1, Math.random() \* 4 никогда не вернет 4 или большее число. Используя это случайное число как индекс, можно получить случайный элемент массива: var randomWords = ["Взрыв", "Пещера", "Принцесса", "Карандаш"]; var randomIndex = Math.floor(Math.random() \* 4); randomWords[randomIndex]; "Пещера" С помощью Math.floor(Math.random() \* 4); мы получили случайное число от 0 до 3. Сохранив это число в переменной randomIndex, мы использовали его как индекс для получения строки из массива randomWords. В сущности, можно сделать этот код короче, избавившись от переменной randomIndex: randomWords[Math.floor(Math.random() \* 4)]; "Принцесса" Программа случайного выбора вариантов Теперь давайте создадим массив с фразами, чтобы случайным образом выбирать их с помощью написанного ранее кода. Это и будет наш компьютерный «шар судьбы»! В комментариях указаны примеры вопросов, которые можно задать нашей программе. var phrases = [ "Звучит неплохо", "Да, это определенно надо сделать", "Не думаю, что это хорошая идея", "Может, не сегодня?", "Компьютер говорит нет" ]; // Мне выпить еще молочного коктейля? Random words — случайные слова Random index — случайный индекс Phrases — фразы 3. Массивы 67 phrases[Math.floor(Math.random() \* 5)]; "Не думаю, что это хорошая идея" // Мне пора делать домашнюю работу? phrases[Math.floor(Math.random() \* 5)]; "Может, не сегодня?” Мы создали массив phrases, в котором хранятся различные советы. Теперь, придумав вопрос, можно запросить случайный элемент из массива phrases, и полученный совет поможет принять решение! Обратите внимание: поскольку в массиве с советами пять элементов, мы умножаем Math.random() на 5. Таким образом, мы всегда получим одно из пяти значений индекса: 0, 1, 2, 3 или 4. Генератор случайных дразнилок Можно усовершенствовать код выбора вариантов, создав программу, которая при каждом запуске генерирует случайную дразнилку! var randomBodyParts = ["глаз", "нос", "череп"]; var randomAdjectives = ["вонючая", "унылая", "дурацкая"]; var randomWords = ["муха", "выдра", "дубина", "мартышка", "крыса"]; // Выбор случайной части тела из массива randomBodyParts: υ var randomBodyPart = randomBodyParts[Math.floor(Math.random() \* 3)]; // Выбор случайного прилагательного из массива randomAdjectives: ϖ var randomAdjective = randomAdjectives[Math.floor(Math.random() \* 3)]; // Выбор случайного слова из массива randomWords: ω var randomWord = randomWords[Math.floor(Math.random() \* 5)]; // Соединяем случайные строки в предложение: var randomInsult = "У тебя " + randomBodyPart + " словно " + 8 randomAdjective + " " + randomWord + "!!!"; randomInsult; "У тебя нос словно дурацкая выдра!!!" У нас есть три массива со словами, и в строках υ, ϖ и ω мы с помощью трех индексов берем из каждого массива по случайному слову. Затем мы склеиваем их, помещая результат в переменную randomInsult, — это и есть готовая дразнилка. В строках υ и ϖ мы используем множитель 3, поскольку и в randomAdjective, и в randomBodyPart по три элемента. Аналогично в строке ω мы умножаем на 5, ведь в randomWords пять элементов. Обратите внимание, что мы добавили между randomAdjective и randomWord Random body part — случайная часть тела Random adjective — случайное прилагательное Random insult — случайная дразнилка 68 Часть I. Основы строку с единственным пробелом. Запустите этот код несколько раз — при каждом запуске получится новая случайная дразнилка! ПОПРОБУЙТЕ! Если хотите сделать все действительно по-умному, замените строку ω на вот такую: var randomWord = randomWords[Math.floor(Math.random() \* 8 randomWords.length)]; В этой программе всегда нужно умножать Math.random() на длину массива; использование randomWords.length как множителя означает, что нам не понадобится менять этот код, если длина массива изменится. Вот еще один способ составления случайной дразнилки: var randomInsult = ["У тебя", randomBodyPart, "словно", 8 randomAdjective, randomWord + "!!!"].join(" "); "У тебя череп словно дурацкая дубина!!!" В этом варианте каждое слово дразнилки — это отдельный элемент массива, и мы соединяем все эти элементы методом join с разделителем-пробелом. Лишь в одном случае пробел не нужен — между randomWord и «!!!». В этом случае мы используем оператор +, чтобы соединить строки без пробела. Что мы узнали Как мы теперь знаем, массивы JavaScript предназначены для хранения списка значений. Мы научились создавать массивы и работать с ними и освоили много способов доступа к их элементам. Массивы JavaScript — один из способов хранения множества значений в одном месте. В следующей главе мы познакомимся с объектами — другим способом объединения значений в единую сущность. Для доступа к элементам объектов используются строковые ключи, а не индексы.

Первоначальный код

var randomBodyParts = ["глаз", "нос", "череп"]; var randomAdjectives = ["вонючая", "унылая", "дурацкая"]; var randomWords = ["муха", "выдра", "дубина", "мартышка", "крыса"]; // Выбор случайной части тела из массива randomBodyParts: υ var randomBodyPart = randomBodyParts[Math.floor(Math.random() \* 3)]; // Выбор случайного прилагательного из массива randomAdjectives: ϖ var randomAdjective = randomAdjectives[Math.floor(Math.random() \* 3)]; // Выбор случайного слова из массива randomWords: ω var randomWord = randomWords[Math.floor(Math.random() \* 5)]; // Соединяем случайные строки в предложение: var randomInsult = "У тебя " + randomBodyPart + " словно " + 8 randomAdjective + " " + randomWord + "!!!"; randomInsult; "У тебя нос словно дурацкая выдра!!!"

var randomBodyParts = ["глаз", "нос", "череп"];

var randomAdjectives = ["вонючая", "унылая", "дурацкая"];

var randomWords = ["муха", "выдра", "дубина", "мартышка", "крыса"];

/\*var randomWord = randomWords[Math.floor(Math.random() \*

randomWords.length)]; \*/ можно так написать, чтоб получить результат

// Выбор случайной части тела из массива randomBodyParts:

var randomBodyPart = randomBodyParts[Math.floor(Math.random() \* 3)];

// Выбор случайного прилагательного из массива randomAdjectives:

var randomAdjective = randomAdjectives[Math.floor(Math.random() \* 3)];

// Выбор случайного слова из массива randomWords:

var randomWord = randomWords[Math.floor(Math.random() \* 5)];

// Соединяем случайные строки в предложение:

//var randomInsult = "У тебя " + randomBodyPart + " словно " +

//randomAdjective + " " + randomWord + "!!!";

//randomInsult;

//'У тебя глаз словно дурацкая мартышка!!!'

var randomInsult = ["У тебя", randomBodyPart, "словно",

randomAdjective, randomWord + "!!!"].join(" "); а можно так, но тогда код /\*…\*/ не нужен

randomInsult;

'У тебя глаз словно дурацкая дубина!!!'

Массивы JavaScript — один из способов хранения множества значений в одном месте. В следующей главе мы познакомимся с объектами — другим способом объединения значений в единую сущность. Для доступа к элементам объектов используются строковые ключи, а не индексы.

УПРАЖНЕНИЯ

Чтобы укрепить знания, полученные в этой главе, выполните эти упражнения. #1. Новые дразнилки Сделайте генератор случайных дразнилок со своим набором слов

var randomPartsFace = ["глаза", "губы", "брови"];

var randomColors = ["зелёные", "синие", "красные"];

var randomSea = ["чёрном", "синем"];

var seaShals = ["берюза", "отблески"];

// Выбор случайной части лица из массива randomPartsFace:

var randomPartsFace = randomPartsFace[Math.floor(Math.random() \* 3)];

// Выбор случайного цвета из массива randomColors:

var randomColors = randomColors [Math.floor(Math.random() \* 3)];

// Выбор случайного цвета моря из массива randomSea:

var randomSea =randomSea[Math.floor(Math.random() \* 2)];

// Выбор случайного свойства из массива seaShals:

var seaShals = seaShals [Math.floor(Math.random() \* 2)];

// Объединение выбранных случайных элементов массивов в переменной randomResult:

var randomResult = "Твои " + randomColors + " " + randomPartsFace + " как " + "в " + randomSea + " " + "море " + seaShals + " !!! ";

randomResult; //пустая строка + " " + служит пробелом между randomSea и "море ", можно было бы пустую строку не писать, сделать пробел " пробел море ", между первой кавычкой и море “

'Твои красные глаза как в синем море отблески !!! '

#2. Изощренные дразнилки Усовершенствуйте генератор дразнилок, чтобы он создавал дразнилки такого типа: «У тебя [часть тела] еще более [прилагательное], чем [часть тела животного] у [животное]». Подсказка: нужно будет создать еще один массив.

var randomPartsBody = ["спина", "ж\*а", "рука"];

var randomAdjectives = ["шире", "мощнее", "здоровее"];

var randomMasleAndNeck = ["морда", "рожа", "шея"];

var randomAnimals = ["слона", "обезьяны", "жирафа"];

var randomPartsBody = randomPartsBody [Math.floor(Math.random() \* 3)];

var randomAdjectives = randomAdjectives[Math.floor(Math.random() \* 3)];

var randomMasleAndNeck = randomMasleAndNeck[Math.floor(Math.random() \* 3)];

var randomAnimals = randomAnimals[Math.floor(Math.random() \*

randomAnimals.length)]; //<= это выражение вместо var randomAnimals =[Math.floor(Math.random() \* 3)];

var randomAllResult = "У тебя " + randomPartsBody + " ещё более " +randomAdjectives + " чем " + randomMasleAndNeck + " у " + randomAnimals + " !"

randomAllResult;

'У тебя ж\*а ещё более шире чем шея у обезьяны ! '

#3. Оператор + или join? Сделайте две версии своего генератора дразнилок: одна пусть использует для составления дразнилки оператор +, а другая создает массив со словами и соединяет их через пробел с помощью join. Какой вариант вам больше нравится и почему?

var randomPartsBody = ["спина", "ж\*а", "рука"];

var randomAdjectives = ["шире", "мощнее", "здоровее"];

var randomMasleAndNeck = ["морда", "рожа", "шея"];

var randomAnimals = ["слона", "обезьяны", "жирафа"];

var randomPartsBody = randomPartsBody [Math.floor(Math.random() \* 3)];

var randomAdjectives = randomAdjectives[Math.floor(Math.random() \* 3)];

var randomMasleAndNeck = randomMasleAndNeck[Math.floor(Math.random() \* 3)];

var randomAnimals = randomAnimals[Math.floor(Math.random() \*

randomAnimals.length)];

// Соединяем случайные строки в предложение в массиве randomAllResult:

var randomAllResult = ["У тебя", randomPartsBody, "ещё более", randomAdjectives, "чем", randomMasleAndNeck, "у", randomAnimals + " !"].join(" ");

randomAllResult;

console.log(randomResult); // чтобы показывало в кодэпене нужно записать данные в функцию console.log(randomResult);

'У тебя ж\*а ещё более здоровее чем рожа у обезьяны !'

#4. Соединение чисел Как с помощью метода join превратить массив [3, 2, 1] в строку "3 больше, чем 2 больше, чем 1"?

var numbers = [3, 2, 1];

numbers.join(" > " ) ;

'3 > 2 > 1';

var numbers = [3, 2, 1];

numbers.join(" больше, чем " ); // +"?" можно поставить за скобками numbers.join(" больше, чем ") +"?"; и он тоже добавится в строке

'3 больше, чем 2 больше, чем 1'

ОБЪЕКТЫ

Объекты JavaScript очень похожи на массивы, но для доступа к элементам объектов используются строки, а не числа. Эти строки называют ключами, или свойствами, а элементы, которые им соответствуют, — значениями. Вместе эти фрагменты информации образуют пары «ключзначение». Причем если массивы используются главным образом как списки, хранящие множество элементов, то объекты часто применяют как одиночные сущности с множеством характеристик, или атрибутов. Например, в третьей главе мы создали несколько массивов, хранящих названия разных животных. Но что если нужно хранить набор различных сведений об одном конкретном животном? Создание объектов Для хранения всевозможной информации об одном животном подойдет JavaScript-объект. Вот пример объекта, где хранятся сведения о трехногой кошке по имени Гармония. var cat = { "legs": 3, "name": "Гармония", "color": "Черепаховый" }; Мы создали переменную под названием cat и присвоили ей объект с тремя парами «ключ-значение» (лапы, имя, окрас). При создании объекта используются фигурные скобки {} вместо квадратных, к которым Cat — кошка Legs — ноги Color — цвет, окрас 71 мы привыкли, создавая массивы. Внутри фигурных скобок можно вводить пары «ключ-значение», а вместе скобки и пары значений называются литералом объекта. Литерал объекта — это быстрый способ создания объекта вместе с его содержимым. ! Мы уже встречались с литералами массивов (например, ["a", "б", "в"]), числовыми литералами (например, 37), строковыми литералами (например, "лось") и булевыми литералами (true и false). Литерал — это когда значение записывается сразу и целиком, а не составляется постепенно, шаг за шагом. Например, чтобы создать массив с числами от 1 до 3, можно ввести литерал массива [1, 2, 3]. Альтернативный способ — создать пустой массив и методом push добавить в него значения 1, 2 и 3. Не всегда заранее известно, что за данные будут храниться в массиве или объекте, поэтому создавать массивы и объекты с помощью одних лишь литералов не получится. На рис. 4.1 показан базовый синтаксис создания нового объекта. При создании объекта ключ записывается перед двоеточием (:), а значение — после. Это двоеточие напоминает знак «равно», поскольку значения, стоящие слева, присваиваются именам (ключам), стоящим справа, что похоже на создание переменных со значениями. Все пары «ключ-значение» должны быть разделены запятыми — в нашем примере эти запятые стоят в конце строк. И обратите внимание, что после завершающей пары «ключ-значение» ("color": "Черепаховый") запятую ставить не нужно — следом за этой парой ставится закрывающая фигурная скобка. Ключи без кавычек Создавая первый объект, мы писали имена ключей в кавычках, однако это не обязательно. Следующая запись тоже является допустимым литералом объекта: Key — ключ { "key1": 99 } Ключ, обязательно строковый Значение любого типа Рис. 4.1. Общий синтаксис создания объекта 72 Часть I. Основы var cat = { legs: 3, name: "Гармония", color: "Черепаховый" }; JavaScript знает, что ключи всегда строковые, поэтому можно обходиться без кавычек. В этом случае имена ключей должны соответствовать тем же правилам, что и имена переменных: например, в них не должно быть пробелов. Но если ключ указан в кавычках, пробелы в его имени допустимы: var cat = { legs: 3, "full name": "Гармония Филомена Уси-Пусечка Морган", color: "Черепаховый" }; Помните, что, хотя ключ всегда является строковым (в кавычках он записан или без), значение, соответствующее этому ключу, может быть любого типа — даже переменной, в которой хранятся данные. Кроме того, весь объект можно записать одной строкой, хотя читать такую программу будет, пожалуй, не слишком удобно: var cat = { legs: 3, name: "Гармония", color: "Черепаховый" }; Доступ к значениям внутри объектов Хранящиеся в объектах значения можно получить с помощью квадратных скобок — так же, как элементы массива. Единственное различие в том, что вместо индекса (число) используется ключ (строка). cat["name"]; "Гармония" Точно так же, как необязательны кавычки при записи литерала объекта, их можно опускать и при доступе к значениям по ключу. Однако в этом случае код будет немного другим: Full name — полное имя 4. Объекты 73 cat.name; "Гармония" Такую запись называют точечной нотацией. Вместо того чтобы писать имя ключа в кавычках внутри квадратных скобок, мы просто ставим точку, после которой пишем имя ключа, без кавычек. И, аналогично ключам без кавычек при записи литерала, такой прием сработает, только если ключ не содержит специальных символов — например, пробелов. Теперь предположим, что вы хотите узнать, какие вообще ключи есть у данного объекта. Для этого в JavaScript есть удобное средство — команда Object.keys(): var dog = { name: "Оладушек", age: 6, color: "белый", 8 bark: "Гав тяф тяф!" }; var cat = { name: "Гармония", age: 8, color: "черепаховый" }; Object.keys(dog); ["name", "age", "color", "bark"] Object.keys(cat); ["name", "age", "color"] Object.keys(anyObject) возвращает массив, содержащий все ключи объекта anyObject. Добавление элементов объекта Пустой объект похож на пустой массив, только вместо квадратных скобок при его создании используются фигурные: var object = {}; Добавлять элементы объекта можно так же, как элементы массива, — но используя строки вместо чисел: var cat = {}; cat["legs"] = 3; cat["name"] = "Гармония"; cat["color"] = "Черепаховый"; cat; { color: "Черепаховый", legs: 3, name: "Гармония" } Мы начали с пустого объекта под названием cat, а затем поочередно добавили к нему три пары «ключ-значение». Потом мы ввели cat;, Object key — ключ объекта Dog — пес Bark — лай 74 Часть I. Основы и браузер отобразил содержимое объекта. Тут надо отметить, что разные браузеры могут показывать объекты по-разному. Например, Chrome (на момент написания этих строк) выводит объект cat в консоли следующим образом: Object {legs: 3, name: "Гармония", color: "Черепаховый"} Chrome перечисляет ключи в таком порядке — (legs, name, color), но другие браузеры могут выводить их в другой очередности. Дело в том, что JavaScript хранит ключи объектов, не упорядочивая их. В массивах элементы расположены строго один за другим: индекс 0 перед индексом 1, индекс 3 после индекса 2; однако в случае объектов неясно, как расположить элементы друг относительно друга. Должен ли ключ color стоять перед legs или после? «Правильного» ответа на этот вопрос нет, поэтому объекты хранят свои ключи без конкретной очередности, в результате чего разные браузеры показывают ключи в разном порядке. Так что никогда не полагайтесь в своих программах на тот или иной порядок ключей. Добавление ключей через точку Новые ключи также можно добавлять через точечную нотацию. Давайте перепишем этим способом предыдущий пример, то есть создадим пустой объект и заполним его данными: var cat = {}; cat.legs = 3; cat.name = "Гармония"; cat.color = "Черепаховый"; Если обратиться к несуществующему свойству объекта, JavaScript вернет специальное значение undefined, сообщая таким образом: «здесь ничего нет». Например: var dog = { name: "Оладушек", legs: 4, isAwesome: true }; dog.isBrown; undefined Is awesome — классный Is brown — коричневый 4. Объекты 75 Здесь мы определили три свойства объекта dog: name, legs и isAwesome. Свойства isBrown среди них нет, поэтому dog.isBrown возвращает undefined. Массивы объектов До этого момента мы рассматривали только массивы и объекты, в которых содержатся данные простых типов, такие как числа и строки. Однако ничто не мешает сделать элементом массива или объекта другой массив или объект. Например, так может выглядеть массив с объектами, описывающими динозавров: var dinosaurs = [ { name: "Тираннозавр рекс", period: "Верхнемеловой" }, { name: "Стегозавр", period: "Верхнеюрский" }, { name: "Платеозавр", period: "Триасовый" } ]; Получить сведения о первом динозавре можно уже известным нам способом — указав индекс в квадратных скобках: dinosaurs[0]; { name: "Тираннозавр рекс", period: "Верхнемеловой" } А если нужно только название первого динозавра, достаточно указать ключ объекта в еще одних квадратных скобках, следом за индексом: dinosaurs[0]["name"]; "Тираннозавр рекс" Другой вариант — воспользоваться точечной нотацией: dinosaurs[1].period; "Верхнеюрский" ! Точечную нотацию можно использовать только с объектами, для массивов она не подходит. Period — период 76 Часть I. Основы Массив друзей Давайте рассмотрим более сложный пример — массив объектов со сведениями о друзьях, где в каждый из объектов вложено по еще одному массиву. Сначала создадим объекты, а затем поместим их в массив. var anna = { name: "Анна", age: 11, luckyNumbers: [2, 4, 8, 16] }; var dave = { name: "Дэйв", age: 5, luckyNumbers: [3, 9, 40] }; var kate = { name: "Кейт", age: 9, luckyNumbers: [1, 2, 3] }; Мы создали три объекта, сохранив их в переменных anna, dave и kate. У каждого из этих объектов есть по три свойства: name, age и luckyNumbers. Каждому ключу name соответствует строковое значение, ключу age — числовое, а ключу luckyNumbers — массив, содержащий несколько чисел. Теперь создадим массив друзей: var friends = [anna, dave, kate]; Итак, в переменной friends находится массив с тремя элементами: anna, dave и kate (каждый из них является объектом). Мы можем получить любой из объектов по его индексу в массиве: friends[1]; { name: "Дэйв", age: 5, luckyNumbers: Array[3] } Здесь мы извлекли из массива второй объект, dave (по индексу 1). Вместо массива luckyNumbers Chrome напечатал Array[3], что означает «это массив с тремя элементами» (можно изучить содержимое этого массива с помощью Chrome, см. раздел «Исследование объектов в консоли» на с. 77.) Также мы можем получить значение, хранящееся в объекте, указав индекс объекта в квадратных скобках, поставив точку и написав соответствующий ключ: friends[2].name "Кейт" Этот код запрашивает элемент по индексу 2 (что соответствует переменной kate), а затем — свойство этого объекта, хранящееся по ключу "name" (это "Кейт"). Можно даже получить значение из массива, находящегося в объекте, который, в свою очередь, находится в массиве friends: Lucky numbers — счастливые числа Friends — друзья Array — массив 4. Объекты 77 friends[0].luckyNumbers[1]; 4 Использованные в этом примере индексы показаны на рис. 4.2. friends[0] — это элемент по индексу 0 из массива friends, то есть объект anna.friends[0].luckyNumbers — это массив [2, 4, 8, 16] из объекта anna. И наконец, friends[0].luckyNumbers[1] — это значение по индексу 1 из массива luckyNumbers — то есть число 4. Исследование объектов в консоли Chrome позволяет изучать содержимое объектов, показанных в консоли. Например, если вы введете friends[1]; Chrome отобразит то, что показано на рис. 4.3. var friends = [anna, dave, kate]; friends[0] { name: "Аннa", age: 11, luckyNumbers: [2, 4, 8, 16] }; friends[0].luckyNumbers friends[0].luckyNumbers[1] Рис. 4.2. Доступ ко вложенным значениям Рис. 4.3. Отображение объекта в интерпретаторе Chrome 78 Часть I. Основы Треугольник слева обозначает, что объект можно раскрыть. Для этого кликните мышкой по объекту, и увидите то, что показано на рис. 4.4. Рис. 4.4. Раскрытие объекта Массив luckyNumbers также можно раскрыть, кликнув по нему (рис. 4.5). Рис. 4.5. Раскрытие вложенного в объект массива Не беспокойтесь о свойствах с названием \_\_proto \_\_, они относятся к прототипу объекта. Мы поговорим о прототипах позже, в главе 12. Обратите внимание, что помимо элементов массива интерпретатор показывает его свойство length. Также вы можете просмотреть массив friends целиком и раскрыть каждый его элемент, как показано на рис. 4.6. Рис. 4.6. Все три объекта из массива friends, как отображает их интерпретатор Chrome Length — длина 4. Объекты 79 Что полезного можно сделать с объектами Теперь, когда вам известны разные способы создания объектов и добавления к ним свойств, давайте применим эти знания на практике, введя несколько простых программ. Учет долгов Предположим, вы решили открыть банк. Вы одолжили друзьям денег и теперь думаете, как вести учет того, кто и сколько вам должен. Можно использовать объект как способ связать строку с числом. Строкой в нашем случае будет имя друга, а числом — сумма, которую вам должны:

1 var owedMoney = {};

2 owedMoney["Джимми"] = 5;

3 owedMoney["Анна"] = 7;

4 owedMoney["Джимми"];

5

5 owedMoney["Элис"];

Undefined

В строке 1 мы создали пустой массив owedMoney. В строке 2 мы присвоили ключу "Джимми" значение 5, а в строке 3 присвоили значение 7 ключу "Анна". В строке 4, запросив значение, связанное с ключом "Джимми", мы получили 5. Затем в строке 5, пытаясь узнать значение, связанное с ключом "Элис", мы получили ответ undefined, поскольку такой ключ не задан. Теперь представим, что Джимми занял у вас еще немного денег (скажем, 3 доллара). Пора обновить данные в нашем объекте, добавив 3 к долгу Джимми — используем для этого оператор «плюс равно» (+=), речь о котором шла во второй главе.

owedMoney["Джимми"] += 3;

owedMoney["Джимми"];

8

Это примерно то же самое, что и owedMoney["Джимми"] = owedMoney["Джимми"] + 3. Также можно посмотреть на объект целиком, чтобы выяснить, сколько денег задолжал каждый из друзей:

Owed money — одолженная сумма денег 80 Часть I. Основы

owedMoney; { Джимми: 8, Анна: 7 }

Хранение информации о фильмах Предположим, у вас большая коллекция кино на DVD и Bluray. Правда было бы здорово хранить информацию об этих фильмах на компьютере, чтобы в случае чего быстро найти сведения о том или ином фильме? Для этого можно создать объект, каждый ключ в котором — это название фильма, а каждое значение — другой объект, в котором содержится информация об этом фильме. Да, хранящиеся в объекте значения тоже могут быть объектами!

var movies = {

"В поисках Немо": { releaseDate: 2003,

duration: 100,

actors: ["Альберт Брукс", "Эллен Дедженерес", 8 "Александр Гоулд"],

format: "DVD"

},

"Звездные войны: Эпизод VI — Возвращение джедая": {

releaseDate: 1983,

duration: 134,

actors: ["Марк Хэмилл", "Харрисон Форд", "Кэрри Фишер"],

format: "DVD"

},

"Гарри Поттер и Кубок огня": {

releaseDate: 2005,

duration: 157,

actors: ["Дэниел Рэдклифф", "Эмма Уотсон", "Руперт Гринт"],

format: "Blu-ray"

}

};

Наверное, вы заметили, что названия фильмов (ключи внешнего объекта) я поставил в кавычки, но ключи внутренних объектов записал без кавычек. Дело в том, что в названиях нужны пробелы — иначе пришлось бы писать нечто вроде ЗвездныеВойныЭпизодVIВозвращениеДжедая, а это уж совсем нелепо. Для ключей вложенных объектов кавычки необязательны, поэтому я их и не ставил. Код выглядит аккуратнее, когда в нем нет излишних знаков пунктуации. Теперь, если вы захотите что-то узнать о фильме, это легко сделать:

Movies — фильмы Release date — дата выхода Duration — продолжительность Actors — в ролях Format — формат 4.

Объекты 81

var findingNemo = movies["В поисках Немо"];

findingNemo.duration;

100

findingNemo.format;

"DVD"

Мы сохранили сведения о фильме «В поисках Немо» в переменной findingNemo. Теперь достаточно обратиться к свойствам этого объекта (таким как duration и format), чтобы получить интересующую нас информацию. Кроме того, в коллекцию легко добавить новые фильмы:

var cars = {

releaseDate: 2006,

duration: 117,

actors: ["Оуэн Уилсон", "Бонни Хант", "Пол Ньюман"],

format: "Blu-ray" };

movies["Тачки"] = cars;

Здесь мы создали новый объект со сведениями о мультфильме «Тачки» (Cars), а затем добавили его в объект movies с ключом "Тачки".

Коллекция растет, и вам может понадобиться простой способ просмотреть названия всех своих фильмов. Для этого подойдет Object.keys:

Object.keys(movies); ["В поисках Немо", "Звездные войны: Эпизод VI — Возвращение джедая", "Гарри Поттер и Кубок огня", "Тачки"]

Что мы узнали Теперь мы знаем, как устроены объекты JavaScript. Они во многом похожи на массивы и тоже нужны для хранения множества элементов данных в одном месте. Но есть важное отличие — для доступа к элементам объекта используются строки, тогда как элементы массива расположены по числовым индексам. Поэтому массивы отсортированы по порядку, а объекты нет. В дальнейших главах, когда мы больше узнаем о возможностях JavaScript, мы научимся использовать объекты для многих других задач. В следующей главе речь пойдет об HTML — языке разметки веб-страниц.

УПРАЖНЕНИЯ

Попрактикуйтесь в использовании объектов, выполнив эти упражнения.

#1. Подсчет очков Представьте, что вы играете в какую-нибудь игру со своими друзьями и вам нужно вести счет. Создайте для этого объект и назовите его scores. Пусть ключами будут имена ваших друзей, а значениями — набранные ими очки (0 или больше). Счет игроков надо будет увеличивать по мере того, как они зарабатывают новые очки. Как вы будете менять счет игрока, хранящийся в объекте scores?

//Моё решение

let scores = {};

scores["Джимми"] = 5;

scores["Анна"] = 7;

scores["Элис"] = 8;

console.log(scores["Джимми"]);//5

console.log(scores["Анна"]);

console.log(scores["Элис"]);

scores["Джимми"] += 3;

scores["Анна"] += 4;

scores["Элис"] += 5; // тоже самое, что scores["Элис"] = scores["Элис"] +5

console.log(scores);

#2. Вглубь объектов и массивов Пускай у вас есть такой объект: var myCrazyObject = { "name": "Нелепый объект", "some array": [7, 9, { purpose: "путаница", number: 123 }, 3.3], "random animal": "Банановая акула" }; Как одной строкой JavaScript-кода извлечь из этого объекта число 123? Проверьте свое решение, запустив его в консоли.

var myCrazyObject = {

name: "Нелепый объект",

"some array": [7, 9, { purpose: "путаница", number: 123 }, 3.3],

"random animal": "Банановая акула"

};

console.log(myCrazyObject["some array"] [2].number); // вызываем в console.log объект по его названию далее обращаемся к одному из его ключей, пишем имя ключа в скобках ["some array"], потому как имя состоит из двух слов и между ними ставится пробел, по этой же причине имя пишется в кавычках "some array", затем обращаемся по индексу [2] к объекту { purpose: "путаница", number: 123 }, т.к он находится в массиве, а счёт содержимого массива начинается с 0-ля, далее обращаемся к одному, интересующему нас свойству через точку .number и получаем искомый элемент 123.

ОСНОВЫ HTML

Встроенная в браузер JavaScript-консоль, которой мы до сих пор пользовались, хороша, когда нужно протестировать небольшой фрагмент кода, но для создания более масштабных программ понадобится чуть более гибкое и универсальное средство — вроде веб-страницы со встроенным JavaScript-кодом. В этой главе мы как раз и научимся создавать несложные странички на языке HTML. Гипертекстовый язык разметки HTML предназначен специально для создания веб-страниц. Слово гипертекстовый означает, что фрагменты текста связаны между собой гиперссылками — то есть ссылками в документе на другие объекты. А язык разметки — это способ встраивать в текст дополнительную информацию. Разметка указывает программам (таким как браузер), как отображать текст и что с ним делать. В этой главе я покажу, как создавать HTML-документы в текстовом редакторе — программе, предназначенной для работы с простым текстом без форматирования, в отличие от текстовых процессоров вроде Microsoft Word. Документы текстовых процессоров содержат форматированный текст (с различными типами и размерами шрифтов, цветами и т. п.), и устроены эти программы так, чтобы форматирование было легко менять. Кроме того, многие текстовые процессоры позволяют вставлять в текст картинки и другие графические элементы. Простой же текст является только текстом — без цветов, стилей, размеров и т. д. Вставить в такой текст картинку не выйдет, разве только составить ее из символов — скажем, как этого котика справа.

= /\\_/\ =

( °w° )

( // (\_\_ \_\_)//)

84 Часть I. Основы Текстовые редакторы Мы будем создавать HTML-документы в кросс-платформенном (совместимом с Windows, Mac OS и Linux) редакторе Sublime Text. Скачать Sublime Text можно бесплатно, однако спустя некоторое время вас попросят приобрести лицензию. На случай, если вам такой вариант не по нраву, я отобрал несколько полностью бесплатных альтернатив. Хотя в этой главе я буду ориентироваться на Sublime Text, работа с другими редакторами будет не сильно отличаться — благодаря относительной простоте текстовых редакторов как таковых. • Gedit — кросс-платформенный текстовый редактор, часть проекта GNOME (https://wiki.gnome.org/Apps/Gedit/). • Для Microsoft Windows хорошей альтернативой будет Notepad++ (http://notepad-plus-plus.org/). • В Mac OS вы можете воспользоваться TextWrangler (http://www.barebones.com/products/textwrangler/). Чтобы установить Sublime Text, зайдите на сайт http://www. sublimetext.com/. Инструкции по установке редактора отличаются для каждой из операционных систем, но написаны просто и понятно. В случае каких-либо проблем загляните в раздел Support («Поддержка») на сайте приложения. ПОДСВЕТКА СИНТАКСИСА Sublime Text будет отображать ваши программы в цвете — это называется подсветкой синтаксиса. Смысл в том, что программы легче читать, когда разные конструкции языка выделены разными цветами. Например, строки могут отображаться зеленым цветом, а ключевые слова вроде var — оранжевым. Sublime Text позволяет выбрать одну из множества схем подсветки. В этой книге используется схема IDLE — вы можете включить ее, войдя в меню Preferences Color Scheme и выбрав там IDLE, чтобы у вас в редакторе программы выглядели так же, как примеры кода в этой главе и далее. Наш первый HTML-документ Установив Sublime Text, запустите его и создайте новый файл, выбрав File New File. Затем выберите File Save, чтобы сохранить новый, пустой файл; назовите его page.html и сохраните на рабочий стол. Настало время писать HTML-код. Введите в файл page.html следующий текст: Page — страница 5. Основы HTML 85

**Привет, мир!**

Моя первая веб-страничка.

Сохраните обновленный файл page.html, выбрав File Save. Теперь посмотрим, на что это будет похоже в веб-браузере. Откройте Chrome и, удерживая CTRL, нажмите O (в Mac OS вместо CTRL используйте клавишу COMMAND). В появившемся окне выберите файл page.html, находящийся на рабочем столе. То, что вы должны после этого увидеть, изображено на рис. 5.1. Рис. 5.1. Ваша первая HTML-страница в Chrome Вы только что создали свой первый HTML-документ! Вы просматриваете его через браузер, однако находится он не в интернете — Chrome открыл его с вашего компьютера и, считав разметку, определил, как нужно отображать текст. Теги и элементы HTML-документы состоят из элементов. Каждый элемент начинается с открывающего тега и оканчивается закрывающим тегом. Например, в нашем первом документе пока всего два элемента: h1 и p (а также элемент meta, но его мы отдельно здесь рассматривать не будем. Он нужен, чтобы в браузере отображался русский текст). Элемент h1 начинается с открывающего тега

**и заканчивается закрывающим тегом**

, а элемент p начинается с открывающего тега

и заканчивается закрывающим тегом

. Все, что находится между открывающим и закрывающим тегами, называют содержимым элемента. Открывающие теги представляют собой название элемента в угловых скобках: < и >. Закрывающие теги выглядят так же, но перед именем элемента в них ставится наклонная черта (/). 86 Часть I. Основы Элементы заголовков У каждого элемента есть особое назначение и способ применения. Например, элемент h1 означает «это заголовок верхнего уровня». Содержимое, которое вы введете между открывающим и закрывающим тегами

**, браузер отобразит на отдельной строке крупным жирным шрифтом. Всего в HTML шесть уровней заголовков: h1, h2, h3, h4, h5 и h6. Выглядят они так:**

**Заголовок первого уровня**

**Заголовок второго уровня**

**Заголовок третьего уровня**

**Заголовок четвертого уровня**

**Заголовок пятого уровня**

**Заголовок шестого уровня**

**На рис. 5.2 показано, как эти заголовки выглядят в браузере. Рис. 5.2. Элементы заголовков разного уровня Элемент p Элемент p нужен для разделения текста на параграфы. Любой фрагмент текста, который вы поместите между тегами**

**, будет отображен как отдельный параграф, с отступами сверху и снизу. Давайте посмотрим, что происходит, если элементов**

**несколько. Для этого добавьте новую строку в документ page.html (прежние строки показаны серым цветом). 5. Основы HTML 87**

**Привет, мир!**

**Моя первая веб-страничка.**

**Добавим-ка еще параграф.**

**На рис. 5.3 показана страничка с нашим новым параграфом. Рис. 5.3. Та же страничка с еще одним параграфом Обратите внимание, что каждый параграф отображен с новой строки, а между параграфами сделан отступ. Все это благодаря тегу**

**. Пробелы в HTML и блочные элементы А как наша страничка будет выглядеть без тегов? Давайте посмотрим: Привет, мир! Моя первая веб-страничка. Добавим-ка еще параграф. На рис. 5.4 показана страничка без тегов. Рис. 5.4. Та же страничка без HTML-тегов Мало того что пропало форматирование, теперь весь текст отображается в одну строку! Дело в том, что в HTML все пробельные символы преобразуются в единственный пробел. Пробельные символы — это любые символы, которые отображаются в браузере как пробелы или 88 Часть I. Основы отступы, — например, это пробел, символ табуляции и символ перевода строки (тот самый, который вы вводите, нажимая ENTER или RETURN). Поэтому все пустые строки, которые вы вставите между фрагментами текста в HTML-документе, сожмутся до одного пробела. Элементы p и h1 — блочные; это значит, что их содержимое отображается отдельными блоками текста с новой строки и любое содержимое, идущее после такого блока, тоже начнется с новой строки. Строчные элементы А теперь добавим к нашему документу еще два элемента, em и strong: На рис. 5.5 показано, как выглядит страница с новыми тегами.**

**Привет, мир!**

**Моя *первая* веб-страничка.**

**Добавим-ка еще *параграф*.**

**Рис. 5.5. Элементы em и strong Элемент em отображает свое содержимое курсивом, а элемент strong — жирным шрифтом. И em, и strong относятся к строчным элементам, поскольку они, в отличие от блочных элементов, не выводят свое содержимое отдельной строкой. Чтобы отобразить текст одновременно жирным шрифтом и курсивом, поместите его внутрь обоих тегов. Обратите внимание, что в последнем примере теги стояли в такой последовательности: *параграф*. Очень важно правильным образом вкладывать элементы друг в друга: если один элемент находится внутри другого элемента, то его открывающий тег и его 5. Основы HTML 89 закрывающий тег также должны находиться внутри этого элемента. Например, такой вариант недопустим: *параграф* Закрывающий тег расположен здесь перед закрывающим тегом . Как правило, браузеры никак не сообщают о подобных ошибках, однако неправильно вложенные теги приведут к неверному отображению страниц. Полноценный HTML-документ До сих пор мы имели дело лишь с фрагментами HTML, тогда как полноценный HTML-документ должен включать некоторые дополнительные элементы. Давайте посмотрим на законченный HTML-документ и разберемся, зачем нужна каждая его часть. Добавьте в файл page.html следующие элементы:**

**Привет, мир!**

**Моя *первая* веб-страничка.**

***параграф*.**

**! Sublime Text автоматически ставит отступы при вводе некоторых строк кода, как показано в этом примере. По тегам (таким как ,**

**и т. д.) он определяет, внутри каких элементов находится каждая строка, и делает отступы в соответствии с этим. Перед тегами и**

**Sublime Text, в отличие от некоторых других редакторов, отступов не ставит. На рис. 5.6 показан законченный HTML-документ. Давайте по очереди рассмотрим элементы из файла page.html. Тег — всего лишь объявление, он сообщает: «это HTML-документ». Далее следует открывающий тег (закрывающий тег**

**находится в самом конце кода). Каждый Head — здесь «шапка документа» Title — название Body — тело документа 90 Часть I. Основы HTML-документ должен содержать элемент html верхнего уровня вложенности. Рис. 5.6. Законченный HTML-документ Внутри элемента html находятся элементы head и body. Элемент head содержит определенную информацию об HTML-документе, например элемент title, устанавливающий название документа, — обратите внимание, что текст на закладке браузера на рис. 5.6 («Моя первая настоящая HTML-страничка») соответствует содержимому title. Элемент title находится внутри элемента head, который, в свою очередь, находится внутри элемента html. Внутри элемента body находится содержимое, которое отображается в браузере. В данном случае мы просто скопировали эти данные из предыдущего примера. Иерархия HTML HTML-элементы подчинены строгой иерархии, которую можно себе представить в виде перевернутого дерева. На рис. 5.7 в виде дерева показан наш документ.**

**Привет, мир!**

**92 Часть I. Основы**

**Моя первая веб-страничка.**

[**Нажмите сюда**](http://comicsia.ru/collections/xkcd)**, чтобы почитать отличные комиксы.**

Сохраните файл и откройте страничку в браузере — она должна выглядеть как на рис. 5.9. Рис. 5.9. Веб-страница со ссылкой на http://comicsia.ru/collections/xkcd Если кликнуть по этой ссылке, браузер должен пе рейти по адресу http://comicsia.ru/collections/xkcd. Насладившись комиксами (выберите там тег «программисты» и почитайте смешные истории из жизни разработчиков), кликните на кнопку «назад», чтобы вернуться к нашей страничке. Атрибуты ссылок Давайте разберемся, как мы создали эту HTML-ссылку. Чтобы браузер знал, куда перейти по клику, мы добавили элементу a так называемый атрибут. Атрибуты HTMLдокументов напоминают пары «ключ-значение» в объектах JavaScript: у каждого атрибута есть имя и значение. Посмотрите еще раз на созданную нами ссылку: [Нажмите сюда](http://comicsia.ru/collections/xkcd) В данном случае у атрибута есть имя href и значение "http:// comicsia.ru/collections/xkcd" — то есть веб-адрес. На рис. 5.10 показаны все составные части этой ссылки. Ссылка отправит вас по любому адресу, который указан в качестве значения атрибута href. Href — от hypertext reference — гипертекстовая ссылка 5. Основы HTML 93 [Нажмите сюда](http://comicsia.ru/collections/xkcd) Открывающий тег Веб-адрес в кавычках Этот текст отобразится как ссылка Закрывающий тег Рис. 5.10. Базовый синтаксис для создания гиперссылки Атрибут title Также к ссылкам можно добавлять атрибут title — он задает текст, который появляется при наведении курсора на ссылку. Например, давайте изменим открывающий тег , чтобы он выглядел так: [Нажмите сюда](http://comicsia.ru/collections/xkcd) Теперь перезагрузите страничку. При наведении мышки на ссылку должна появиться надпись: «xkcd: Комиксы для гиков!», как на рис. 5.11. Рис. 5.11. Веб-страничка, содержащая ссылку на адрес http://comicsia.ru/collections/xkcd/ с атрибутом title ПОПРОБУЙТЕ! Создайте новый файл под названием links.html. Пусть его HTML-структура будет такой же, как у странички page.html, однако название и заголовок поменяйте на другие, а также добавьте три элемента p («параграф»). В каждом параграфе создайте ссылку на один из своих любимых сайтов. Убедитесь, что для всех элементов a заданы атрибуты href и title. Links — ссылки Что мы узнали В этой главе мы познакомились с основами HTML — языка для создания веб-страниц. Также мы создали простой HTML-документ со ссылкой на другую страницу. В следующей главе мы разберемся, как встраивать в нашу страничку JavaScript-код. Это облегчит создание более объемных программ по мере изучения новых возможностей JavaScript. Эта книга посвящена JavaScript, а не HTML, поэтому я рассмотрел лишь самые азы создания HTML-документов. Вот некоторые ресурсы, где можно узнать о HTML больше: На английском языке: • Курс HTML и CSS от Codeacademy: http://www.codecademy.com/ tracks/web/ • Mozilla Webmaker: https://webmaker.org/ На русском языке: • Введение в HTML от Mozilla Developer Network: https://developer. mozilla.org/ru/docs/Web/Guide/HTML/Introduction • <https://htmlacademy.ru/>

УСЛОВИЯ И ЦИКЛЫ

Условные конструкции и циклы — одни из самых важных понятий в JavaScript. Условная конструкция представляет собой команду: «если что-то истинно (true), сделай это, иначе сделай то». Пример: выполнив домашнее задание, вы можете съесть мороженое, но, если домашнее задание не готово, мороженое вам не светит. А цикл — это инструкция: «до тех пор, пока что-то истинно (true), продолжай делать это». Пример: пока вы испытываете жажду, продолжайте пить воду. Условные конструкции и циклы — понятия, лежащие в основе любой мало-мальски серьезной программы. Их называют управляющими конструкциями, поскольку они позволяют решать, какие части кода и когда выполнять, а также насколько часто это нужно делать, исходя из заданных вами условий. Для начала давайте разберемся, как встраивать JavaScript в HTMLфайл. Это позволит писать программы более сложные, чем те, с которыми мы имели дело до сих пор. Внедрение JavaScript-кода в HTML Вот HTML-файл, который мы создали в пятой главе, с некоторыми дополнениями — они показаны в цвете, тогда как прежний текст набран серым (чтобы упростить этот пример, я убрал из него ссылку на http://comicsia.ru/collections/xkcd/). 96 Часть I. Основы

**Привет, мир!**

Моя первая веб-страничка.

Как видите, мы добавили новый элемент под названием script. Этот элемент особенный: содержимое большинства элементов HTML отображается на страничке, однако то, что находится внутри тегов script, считается JavaScript-кодом и передается для выполнения интерпретатору JavaScript. Теперь рассмотрим код внутри элемента script: var message = "Привет, мир!"; υ console.log(message); Выполнение JavaScript, встроенного в HTML, заметно отличается от запуска кода в консоли. Введенный в консоли код выполнится при первом же нажатии ENTER, после чего вы увидите результат сработавшей команды. Однако код, встроенный в веб-страницу, выполняется сразу и целиком, от верхних строк к нижним, причем в консоль ничего автоматически не выводится — если мы не попросим браузер об этом отдельно. Для вывода в консоль можно воспользоваться командой console.log — это поможет следить за ходом выполнения программы. Метод console.log принимает любое значение и печатает (логирует) это значение в консоли. Например, загрузив в браузер наш последний пример, вы увидите в консоли вот что (разумеется, если она открыта): Привет, мир! Вызов console.log(message) в строке υ привел к появлению в консоли строки "Привет, мир!". Теперь, когда вы знаете, как с удобством писать длинные JavaScriptпрограммы, можно перейти к изучению условных конструкций. Script — скрипт, сценарий Message — сообщение 6. Условия и циклы 97 Условные конструкции В JavaScript есть два вида условных конструкций — это if и if... else. Оператор if выполняет фрагмент кода, если какое-то условие истинно (true). Например: если вы хорошо себя вели, то получите конфетку. А оператор if... else выполняет один фрагмент кода, если условие дает true, и другой фрагмент в противном случае. Например: если вы хорошо себя вели, получите конфетку, иначе вас не отпустят гулять. Конструкция if Cамая простая из управляющих конструкций JavaScript — это if. Она используется, чтобы запускать код, если некое условие истинно (true). Вернитесь к нашему HTML-файлу и замените содержимое элемента script следующими строками: υ var name = "Николай"; ϖ console.log("Привет, " + name); ω if (name.length > 6) { ξ console.log("Ну и длиннющее же у вас имя!"); } Сначала в строке υ мы создали переменную name и присвоили ей значение — строку "Николай". Затем, в строке ϖ, мы с помощью console.log напечатали строку "Привет, Николай". В строке ω мы использовали конструкцию if, чтобы проверить: длина name больше, чем шесть символов? Если это так, в строке ξ мы посредством console.log выводим: "Ну и длиннющее же у вас имя!". Как показано на рис. 6.1, конструкция if состоит из двух частей: условия и тела. Условие должно давать булево значение. А тело — одна или несколько строк JavaScript-кода, которые будут выполнены, если условие истинно (true). if (condition) { console.log("Делаем что-то"); } Код, который выполняется, если условие дает true, называется телом if Конструкция if проверяет, истинно ли условие Рис. 6.1. Общая структура конструкции if If — если If… else — если… иначе 98 Часть I. Основы После загрузки нашей HTML-странички со встроенным JavaScriptкодом в консоли должно появиться: Привет, Николай Ну и длиннющее же у вас имя! В имени Николай 7 букв, поэтому name.length вернет значение 7, и условие name.length > 6 даст true. В результате будет выполнено тело оператора if, и в консоли появится несколько фамильярное сообщение. Чтобы избежать выполнения if, поменяйте имя Николай на Ник (оставив остальной код без изменений): var name = "Ник"; Теперь сохраните файл и перегрузите страничку. На этот раз условие name.length > 6 даст false, поскольку name.length равно 3. В итоге тело оператора if выполнено не будет, а в консоли появится лишь: Привет, Ник Тело оператора if выполняется, только когда условие дает true. Если же условие дает false, интерпретатор игнорирует конструкцию if и переходит к следующей за ней строке. Конструкция if... else Как я уже говорил, оператор if запускает код своего тела, только если условие дает true. Но если вы хотите, чтобы по условию false тоже что-то происходило, вам нужна конструкция if... else. Давайте дополним предыдущий пример: var name = "Николай"; console.log("Привет, " + name); if (name.length > 6) { console.log("Ну и длиннющее же у вас имя!"); } else { console.log("Имя у вас не из длинных."); } Этот код делает практически то же, что и раньше, однако, если имя (name) не длиннее 6 символов, он выводит другое, альтернативное сообщение. 6. Условия и циклы 99 Как видно по рис. 6.2, конструкция if... else похожа на конструкцию if, однако у нее целых два тела, между которыми расположено ключевое слово else. Первое тело будет выполнено, если условие дает true, иначе выполняется код второго тела. if (condition) { console.log("Делаем что-то"); } else { console.log("Делаем что-то другое!"); } Код, который выполняется, если условие = true Условие (true или false) Код, который выполняется, если условие = false Рис. 6.2. Общая структура конструкции if... else Цепочка из конструкций if... else Зачастую нужно проверить несколько условий и сделать что-то, если одно из них дает true. Пример: вы пришли в китайский ресторан и выбираете, что бы такое съесть. Больше всего вы любите курицу с лимоном (lemon chicken), и, если она есть в меню, вы ее закажете. Если же ее нет, вы закажете говядину в соусе из черных бобов (beef with black bean). Однако если и это блюдо отсутствует, вы остановитесь на свинине в кислосладком соусе (sweet and sour pork). Наконец, в маловероятном случае, когда нет ни одного из этих блюд, вы закажете рис с яйцом, поскольку знаете, что его подают во всех китайских ресторанах. var lemonChicken = false; var beefWithBlackBean = true; var sweetAndSourPork = true; if (lemonChicken) { console.log("Отлично! Я буду курицу с лимоном!"); } else if (beefWithBlackBean) { console.log("Заказываю говядину."); } else if (sweetAndSourPork) { console.log("Ладно, закажу свинину."); } else { console.log("Что ж, остается рис с яйцом."); } 100 Часть I. Основы Чтобы создать цепочку if... else, начните с обычного оператора if и после закрывающей фигурной скобки его тела введите ключевые слова else if, а следом — еще одно условие и еще одно тело. После можно добавить еще else if, и так до тех пор, пока у вас не закончатся условия (которых может быть сколько угодно). Завершающая секция else будет выполнена, если ни одно из условий не дает true. На рис. 6.3 показана классическая цепочка конструкций if... else. if (condition1) { console.log("Сделай это, если условие 1 истинно"); } else if (condition2) { console.log("Сделай это, если условие 2 истинно"); } else if (condition3) { console.log("Сделай это, если условие 3 истинно"); } else { console.log("Иначе сделай это"); } Для каждого условия указан код, который выполняется, если условие дает true Код, который нужно выполнить, когда все условия дают false Рис. 6.3. Цепочка конструкций if... else Можно прочитать этот код так: • Если первое условие дает true, выполнить код из первого тела. • Иначе, если второе условие дает true, выполнить второе тело. • Иначе, если третье условие дает true, выполнить третье тело. • Иначе выполнить тело else. Имея цепочку if... else с завершающей секцией else, можно не сомневаться, что одно (и только одно) из тел будет выполнено. Как только выяснится, что одно из условий дает true, будет запущен код из соответствующего тела, а последующие условия проверяться уже не будут. Если запустить код из предыдущего примера, мы увидим в консоли «Заказываю говядину», поскольку beefWithBlackBean — первое из условий в цепочке if... else, которое равно true. Если же ни одно из условий не даст true, будет выполнено тело else. 6. Условия и циклы 101 Также обратите внимание: указывать завершающее else необязательно. Однако если вы этого не сделаете, то в случае, когда ни одно из условий не дает true, ничего из цепочки if... else выполнено не будет. var lemonChicken = false; var beefWithBlackBean = false; var sweetAndSourPork = false; if (lemonChicken) { console.log("Отлично! Я буду курицу с лимоном!"); } else if (beefWithBlackBean) { console.log("Заказываю говядину."); } else if (sweetAndSourPork) { console.log("Ладно, закажу свинину."); } В этом примере мы не стали указывать завершающую секцию else. Поскольку ни одного из ваших любимых блюд нет, в консоли не появится никаких сообщений (и, по всей видимости, вы останетесь без обеда).

ПОПРОБУЙТЕ!

Напишите программу с переменной name. Если в этой переменной находится ваше имя, напечатайте: «Привет мне!» — иначе напечатайте: «Привет, незнакомец!» (Подсказка: используйте === для сравнения переменной name с вашим именем.)

/\*let name = "Sergey";

if (name === "Sergey") {

console.log("Hi me");

}else { console.log("Hi, stranger");}\*/

Теперь дополните программу, чтобы она здоровалась с вашим папой, если в name его имя, и с вашей мамой, если в name ее имя. Если же там что-то иное, по-прежнему печатайте «Привет, незнакомец!».

/\*let name = "Sergey";

name = "Mam";

if (name === "Sergey") {

console.log("Hi, " + name);

}else if (name === "Mam") {

console.log("Hi, Mam");

}else { console.log("Hi, stranger");}\*/

Циклы

Как мы теперь знаем, условные конструкции позволяют запускать фрагмент кода, если условие дает true. Циклы, с другой стороны, позволяют выполнять фрагмент кода многократно — до тех пор, пока некое условие дает true. Примеры: до тех пор, пока в тарелке есть пища, следует продолжать есть; до тех пор, пока на лице грязь, следует продолжать умываться. Цикл while Самый простой из циклов — цикл while. Этот цикл снова и снова выполняет код своего тела, до тех пор, пока заданное условие не перестанет давать true. Используя цикл while, мы имеем в виду следующее: «Продолжай делать это, пока условие дает true. Но если оно даст false, остановись». While — до тех пор, пока 102 Часть I. Основы Как видно на рис. 6.4, цикл while начинается с ключевого слова while, после которого в скобках стоит условие, а за ним идет тело, заключенное в фигурные скобки. while (condition) { console.log("Делаем что-то"); i++; } Код, который будет выполняться снова и снова, до тех пор, пока условие дает true (что-то в этом коде должно влиять на условие, чтобы в какой-то момент оно дало false) Это условие проверяется при каждом повторе цикла Рис. 6.4. Общая структура цикла while Аналогично конструкции if, тело цикла while выполняется, если заданное условие дает true. Но, в отличие от if, после того как тело цикла выполнено, условие будет проверено снова, и, если оно все еще дает true, тело цикла начнет выполняться опять. И так будет продолжаться, пока условие не даст false. Считаем овец с помощью цикла while Предположим, у вас проблемы со сном и вы решили посчитать овец. Но раз уж вы программист, почему бы не написать программу, которая будет считать овец за вас? var sheepCounted = 0; υ while (sheepCounted < 10) { ϖ console.log("Посчитано овец: " + sheepCounted + "!"); sheepCounted++; } console.log("Хрррррррррр-псссс"); Мы создали переменную sheepCounted и задали ей значение 0. Дойдя до цикла while в строке υ, мы проверяем, правда ли, что sheepCounted меньше 10. Поскольку 0 меньше 10, выполняется код в фигурных скобках (тело цикла, которое начинается со строки ϖ) и выражение "Посчитано овец: " + sheepCounted + "!" выводится в консоль как «Посчитано овец: 0!». Далее команда sheepCounted++ увеличивает значение Sheep counted — посчитанные овцы 6. Условия и циклы 103 sheepCounted на 1, мы возвращаемся к началу цикла, и все повторяется снова: Посчитано овец: 0! Посчитано овец: 1! Посчитано овец: 2! Посчитано овец: 3! Посчитано овец: 4! Посчитано овец: 5! Посчитано овец: 6! Посчитано овец: 7! Посчитано овец: 8! Посчитано овец: 9! Хрррррррррр-псссс Тело цикла повторяется, пока sheepCounted не примет значение 10, после чего условие становится ложным (false), ведь 10 не меньше 10. И тогда программа переходит к строке, идущей после цикла, — в данном случае на консоль выводится "Хрррррррррр-псссс". Бесконечный цикл Имея дело с циклами, помните: если условие никогда не даст false, цикл будет повторяться бесконечно (по крайней мере до тех пор, пока вы не закроете страницу в браузере). Например, не будь в теле цикла строчки sheepCounted++;, в sheepCounted всегда был бы 0 и программа печатала бы: Посчитано овец: 0! Посчитано овец: 0! Посчитано овец: 0! Посчитано овец: 0! ... Поскольку повторения цикла ничем не ограничены, программа будет печатать эту строку снова и снова, без конца. Это называется бесконечным циклом. Цикл for Оператор for упрощает создание циклов, устроенных следующим образом: сначала создается переменная, а затем тело цикла выполняется снова и снова до тех пор, пока условие дает true, причем в конце каждого повтора значение переменной обновляется. Программируя цикл for, мы создаем переменную, задаем условие, указываем, как For — для 104 Часть I. Основы должна меняться переменная после каждого повтора, — и лишь затем переходим к написанию тела цикла. Например, вот как можно считать овец с помощью for: for (var sheepCounted = 0; sheepCounted < 10; sheepCounted++) { console.log("Посчитано овец: " + sheepCounted + "!"); } console.log("Хрррррррррр-псссс"); Согласно рис. 6.5, в составе цикла for есть три выражения, разделенные точками с запятой: это настройка, проверка условия и при - ращение. for (настройка; условие; приращение) { console.log("Делаем что-то"); } Код, который будет выполняться до тех пор, пока условие дает true Условие: true или false Этот код будет выполнен до начала цикла Код, который будет выполняться после каждого повтора тела цикла Рис. 6.5. Общая структура цикла for Настройка (var sheepCounted = 0) выполняется до запуска цикла. Как правило, здесь создают переменную для отслеживания количества повторов. В нашем случае это переменная sheepCounted с начальным значением 0. Условие (sheepCounted < 10) проверяется перед каждым повтором тела цикла. Если условие дает true, тело выполняется, иначе цикл заканчивает работу. В нашем случае цикл остановится, когда значение sheepCounted достигнет 10. Приращение (sheepCounted++) выполняется после каждого повтора тела цикла. Как правило, здесь изменяют значение переменной цикла. В этом примере мы после каждого повтора увеличиваем sheepCounted на 1. Циклы for удобны, когда нужно сделать что-то определенное количество раз. Например, эта программа три раза выведет слово «Привет!». var timesToSayHello = 3; for (var i = 0; i < timesToSayHello; i++) { console.log("Привет!"); } Times to say hello — сколько раз сказать «привет» 6. Условия и циклы 105 Вот что появится в консоли: Привет! Привет! Привет! Вообразите, что вы интерпретатор JavaScript, который выполняет этот код. Сначала вы создадите переменную timesToSayHello, задав ей значение 3. Дойдя до цикла for, вы выполните настройку, то есть создадите переменную i и присвоите ей значение 0. Далее вы проверите условие. Поскольку в переменной i сейчас 0, а в timesToSayHello — 3, условие даст true и вы запустите тело цикла, где печатается строка "Привет!". А затем выполните приращение, то есть увеличите i на 1. Теперь снова проверьте условие. Оно по-прежнему даст true, и вы опять перейдете к телу цикла, а затем к приращению. И так будет происходить до тех пор, пока i не примет значение 3. После этого условие даст false (3 не меньше, чем 3) — таким образом, вы завершите цикл. Цикл for, массивы и строки Очень часто цикл for используют для перебора всех элементов массива или всех символов строки. Например, вот цикл, который печатает названия всех животных, которые есть в зоопарке: var animals = ["лев", "фламинго", "белый медведь", "удав"]; for (var i = 0; i < animals.length; i++) { console.log("В этом зоопарке есть " + animals[i] + "."); } В этом цикле i сначала равняется 0, а затем возрастает до значения animals.length - 1, то есть 3. Числа 0, 1, 2 и 3 — индексы элементов в массиве animals. Это значит, что при каждом повторе цикла i принимает значение очередного индекса, а animals[i] соответствует очередному животному из массива animals. Когда в i число 0, animals[i] даст нам строку "лев". Когда в i число 1, animals[i] даст "фламинго" и т. д. Запустив эту программу, мы увидим: В этом зоопарке есть лев. В этом зоопарке есть фламинго. В этом зоопарке есть белый медведь. В этом зоопарке есть удав. 106 Часть I. Основы Как мы уже знаем из второй главы, к отдельным символам строки можно обращаться тем же способом, что и к элементам массива, — с помощью квадратных скобок. В следующем примере цикл for используется для вывода символов имени: var name = "Ник"; for (var i = 0; i < name.length; i++) { console.log("В моем имени есть буква " + name[i] + "."); } Вот что выдаст эта программа: В моем имени есть буква Н. В моем имени есть буква и. В моем имени есть буква к. Другие варианты применения for Как вы, может быть, догадываетесь, не обязательно сначала задавать переменной цикла значение 0, а затем каждый раз увеличивать ее на 1. Например, вот как можно напечатать все степени двойки, не превышающие числа 10 000: for (var x = 2; x < 10000; x = x \* 2) { console.log(x); } Здесь мы присваиваем x значение 2 и увеличиваем его командой x = x \* 2, то есть, удваиваем значение x при каждом повторе цикла. В результате x очень быстро возрастает: 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 6. Условия и циклы 107 Вуаля! Этот несложный цикл печатает все степени двойки меньше 10 000.

ПОПРОБУЙТЕ!

Напишите цикл for, который печатает степени тройки, не превышающие 10 000 (программа должна выводить 3, 9, 27 и т. д.)

for (var x = 3; x < 10000; x = x \* 3) {

console.log(x);

}

Перепишите это задание, вместо for использовав цикл while. (Подсказка: установите начальное значение перед входом в цикл.)

let x = 3;

while (x < 10000) {

x = x \* 3;

console.log(x);

}

Что мы узнали Мы разобрались с условными конструкциями и циклами. Условные конструкции нужны, чтобы выполнять некие действия, если условие дает true. А с помощью циклов можно многократно выполнять фрагмент кода — до тех пор, пока условие дает true. Условными конструкциями можно пользоваться, чтобы запускать код в подходящие для этого моменты, а циклами — чтобы программа выполнялась так долго, как требуется. И это открывает перед нами целый мир новых возможностей в программировании. В следующей главе мы воспользуемся мощью условных конструкций и циклов при создании нашей первой игры!

УПРАЖНЕНИЯ

Выполните эти упражнения, чтобы попрактиковаться в работе с условными конструкциями и циклами. #1. Прекрасные животные Напишите цикл for, который изменяет массив животных, делая их прекрасными! Например, если есть следующий массив: var animals = ["Кот", "Рыба", "Лемур", "Комодский варан"]; то ваш цикл должен сделать его таким:

["Кот - прекрасное животное", "Рыба - прекрасное животное", "Лемур - прекрасное животное", "Комодский варан - прекрасное животное"]

Подсказка: вам понадобится переприсвоить значения для каждого индекса, то есть присвоить новые значения уже существующим элементам. Например, сделать первое животное прекрасным можно так: animals[0] = animals[0] + " - прекрасное животное";

так правильно, подсказала Даша, тут я добавил в массив сторку + ' - прекрасное животное'

const animals = ['Кот', 'Рыба', 'Лемур', 'Комодский варан'];

let beautifulAnimals = []; //создали новый массив

for (let i = 0; i < animals.length; i++) {

beautifulAnimals.push(animals[i] + ' - прекрасное животное'); //используем метод .push и добавляем старый массив animals в новый

}

console.log(beautifulAnimals);

var animals = ["Кот", "Рыба", "Лемур", "Комодский варан"];

for (let i = 0; i < animals.length; i++) {

console.log(animals [i] + " - прекрасное животное");

} //Но ты их просто показал в логе, а надо добавить в массив.

#2. Генератор случайных строк Напишите генератор случайных строк. Для этого вам понадобится строка со всеми буквами алфавита:

var alphabet = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";

Чтобы выбирать из этой строки случайную букву, можно использовать примерно такой же код, как для генератора случайных дразнилок из третьей главы: Math.floor(Math.random() \* alphabet.length). Так вы получите случайный индекс в строке. Затем, воспользовавшись квадратными скобками, можно получить символ по этому индексу. Начните создавать случайную строку с пустой строки (var randomString = ""). Затем добавьте цикл while и при каждом его повторе добавляйте в строку новый случайный символ — до тех пор, пока длина строки randomString не превысит шесть символов (или любой другой длины на ваш выбор). Alphabet — алфавит Random string — случайная строка Добавлять символ в конец строки можно с помощью оператора +=. После того как цикл закончит работу, выведите получившуюся строку в консоль, чтобы полюбоваться на свое творение!

/\*const alphabet = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";

let randomString = "";

while (randomString.length <= 6) {

const randomNumber = Math.floor(Math.random() \* alphabet.length);

// прибавить к строке рандомную букву

randomString += alphabet[randomNumber];

}

console.log(randomString);

/\* вам понадобится строка со всеми буквами алфавита

const alphabet = "abcdefghijklmnopqrstuvwxyz";

// Начните создавать случайную строку с пустой строки

let randomString = "";

// пока длина строки randomString не превысит шесть символов

while (randomString.length <= 6) {

// Math.floor(Math.random()alphabet.length). Так вы получите случайный индекс в строке.

const randomNumber = Math.floor(Math.random() \* alphabet.length)

// Добавлять символ в конец строки можно с помощью оператор +=

// Затем, воспользовавшись квадратными скобками, можно получить символ по этому индексу.

randomString += alphabet[randomNumber]

}

// выведите получившуюся строку в консоль

console.log(randomString)\*/

// вариант по короче

/\*const alphabet = "abcdefghijklmnopqrstuvwxyz";

let randomString = "";

while (randomString.length <= 6) {

randomString += alphabet[Math.floor(Math.random() \* alphabet.length)]

}

console.log(randomString)\*/

3 Переведите англоязычный текст на «хакерский язык» (h4ck3r

sp34k)! Многим в интернете нравится заменять некоторые

буквы похожими на них числами — например, число «4» похоже

на букву «A», «3» похоже на «E», «1» — на «I», а «0» — на «О». Хотя

цифры напоминают скорее заглавные буквы, мы будем заменять

ими буквы строчные. Чтобы перевести обычный текст на «хакерский язык», понадобится строка с исходным текстом var input = "javascript is awesome"; и новая пустая строка для результата:

var output = "";

Теперь воспользуйтесь циклом for, чтобы перебрать все символы исходной строки. Встретив букву «a», добавьте к результирующей строке «4». Встретив «e», добавьте «3», встретив «i»,

добавьте «1», а встретив «o» — «0». В противном случае просто

добавляйте к результату исходный символ. И снова оператор +=

отлично подойдет для добавления символа в конец строки.

После завершения цикла выведите результирующую строку

в консоль. Если программа работает верно, вы должны увидеть

следующее: "j4v4scr1pt 1s 4w3s0m3".\*/

let input = "javascript is awesome";

let output = "";

for (let i = 0; i < input.length; i++) {

if (input[i] == 'a') {

output += 4;

} else if (input[i] == 'i') {

output += 1;

} else if (input[i] == 'e') {

output += 3;

} else if (input[i] == 'o') {

output += 0;

} else {

//добавить символ который был изначально в строке

output += input[i];

}

}

console.log(output);

ПИШЕМ ИГРУ «ВИСЕЛИЦА »

В этой главе мы разработаем игру «Виселица» и разберемся, как с помощью диалоговых окон сделать ее интерактивной, запрашивая у игрока

данные.

«Виселица» — игра на угадывание слов. Один игрок выбирает слово,

а второй пытается его отгадать. Например, если первый игрок загадал

слово КАПУСТА, он изобразит семь «пустых мест», по одному на каждую

букву слова:

\_ \_ \_ \_ \_ \_ \_

Второй игрок старается отгадать это слово, называя буквы. Каждый

раз, когда он угадывает букву, первый игрок заполняет пустоты, вписывая ее везде, где она встречается. Например, если второй игрок назвал

букву «А», первый должен вписать все «А» для слова КАПУСТА, вот так:

\_ А \_ \_ \_ \_ А

Если второй игрок назовет букву, которой нет в слове, у него отнимается очко, а первый игрок рисует руку, ногу или другую часть тела

человечка. Если первый игрок закончит рисовать человечка раньше, чем

второй угадает все буквы, второй игрок проиграл.

В нашем варианте «Виселицы» JavaScript будет выбирать слово,

а игрок-человек — отгадывать буквы. И рисовать человечка наша программа не будет, поскольку мы пока не знаем, как это делается (рисованием на JavaScript мы займемся в главе 13).

111

Взаимодействие с игроком

Для этой игры нам нужно, чтобы игрок (человек) мог каким-то образом вводить в программу свои ответы. Один из способов это сделать —

открывать диалоговое окно (в JavaScript оно называется prompt), в котором игрок может что-нибудь напечатать.

Создаем диалоговое окно

Сначала создадим новый HTML-документ. Выбрав в меню File Save As,

сохраните файл page.html из пятой главы под новым именем —

prompt.html. Чтобы создать диалоговое окно, введите следующий код

между тегов <script>, а затем откройте файл prompt.html в браузере:

var name = prompt("Как вас зовут?");

console.log("Привет, " + name);

Здесь мы создали новую переменную name и присвоили ей значение,

которое вернул вызов prompt("Как вас зовут?"). При вызове prompt

открывается маленькое диалоговое окно (часто его называют просто

диалог), показанное на рис. 7.1.

Рис. 7.1. Диалоговое окно prompt

Вызов prompt("Как вас зовут?") создает окно с запросом «Как вас

зовут?» и строкой для ввода текста. В нижней части этого диалога есть

две кнопки — «ОК» и «Отмена».

Если вы введете какой-нибудь текст и нажмете «ОК», этот текст станет значением, которое вернет в программу prompt. Например, если

я введу свое имя и нажму «ОК», JavaScript напечатает в консоли:

Привет, Ник

Поскольку я ввел Ник и нажал на «ОК», строка "Ник" попала в переменную name, а вызов console.log напечатал: "Привет, " + "Ник",

то есть "Привет, Ник".

Prompt —

здесь «запрос»

112 Часть I. Основы

! Когда вы во второй раз создадите какое-либо диалоговое окно Chrome,

в окне появится кнопка-флажок с подписью «Предотвратить создание дополнительных диалоговых окон на этой странице». Таким

образом Chrome защищает пользователей от веб-страниц, открывающих множество назойливых рекламных окошек. Выполняя примеры из этой главы, просто отключите этот флажок.

А ЧТО ЕСЛИ ВЫ НАЖМЕТЕ «ОТМЕНА»?

Если вы нажмете кнопку «Отмена», prompt вернет значение null. Как

нам известно из второй главы, null используется для обозначения чеголибо, что намеренно оставлено пустым.

После нажатия «Отмена» в консоли должно появиться:

Привет, null

В данном случае console.log печатает null как строку. Вообще-то

null строкой не является, но, поскольку в консоль можно выводить

только строки и вы попросили JavaScript напечатать "Привет, " + null,

JavaScript преобразовал null в строку "null", чтобы напечатать это

значение. Ситуация, когда JavaScript автоматически преобразует значение

к другому типу, называется неявным приведением типа.

Неявное приведение типа — пример того, как JavaScript старается быть

умным. Способа объединить строку и null не существует, и JavaScript делает

лучшее, на что он способен. В данном случае он знает, что для успешного

выполнения операции нужны две строки. Строковая версия значения null —

это "null", и в результате мы видим в консоли "Привет, null".

Используем confirm, чтобы получить

ответ «да» или «нет»

Функция confirm позволяет задать пользователю вопрос, на который он может ответить

«да» или «нет» (что соответствует булеву значению). В следующем примере мы используем confirm, чтобы спросить у пользователя, нравятся ли ему кошки (см. рис. 7.2).

Если получен утвердительный ответ, переменная likesCats принимает

значение true и мы печатаем: «Ты классная кошка!» Если же кошки

пользователю не нравятся, likesCats принимает значение false, и мы

отвечаем: «Что ж, не проблема. Все равно ты молодец!»

Confirm —

подтвердить

Likes cats —

нравятся

кошки

7. Пишем игру «Виселица» 113

var likesCats = confirm("Тебе нравятся кошки?");

if (likesCats) {

console.log("Ты классная кошка!");

} else {

console.log("Что ж, не проблема. Все равно ты молодец!");

}

Рис. 7.2. Диалог confirm

Ответ на вопрос, заданный с помощью confirm, возвращается в программу как булево значение. Если в окне, показанном на рис. 7.2, пользователь нажмет «ОК», confirm вернет true. Если же пользователь нажмет

«Отмена», confirm вернет false.

Используем alert для выдачи информации

Если требуется просто показать что-то пользователю, можно воспользоваться диалогом alert, который отображает сообщение с кнопкой «ОК».

Например, если вы считаете, что JavaScript — это здорово, вы можете

использовать alert так:

alert("JavaScript это здорово!");

На рис. 7.3 показано, как выглядит этот диалог.

Рис. 7.3. Диалог alert

Диалог alert просто отображает сообщение до тех пор, пока пользователь не нажмет «ОК».

Alert —

предупреждение

114 Часть I. Основы

Чем alert лучше console.log?

Зачем нужен диалог alert, когда есть console.log?

Во-первых, если необходимо просто сообщить о чем-то

игроку, alert позволяет сделать именно это — не требуя, чтобы игрок открыл консоль и прочитал сообщение.

Во-вторых, вызов alert (а также prompt и confirm)

приостанавливает работу интерпретатора JavaScript

до нажатия кнопки «ОК» (или «Отмена», в случае

prompt) и — значит, у пользователя будет достаточно

времени, чтобы прочитать сообщение. А при использовании console.log текст отображается в консоли,

а интерпретатор тут же переходит к следующей строке

программы.

Проектирование игры

Прежде чем перейти к созданию игры «Виселица», давайте подумаем

о ее структуре. Нам нужно, чтобы программа умела выполнять следующие действия:

1. Случайным образом выбирать слово.

2. Запрашивать у игрока вариант ответа (букву).

3. Завершать игру по желанию игрока.

4. Проверять, является ли введенный ответ буквой.

5. Вести учет угаданных букв.

6. Показывать игроку, сколько букв он угадал и сколько еще предстоит угадать.

7. Завершать игру, если слово отгадано.

Все эти действия, кроме первого и последнего (выбор слова и завершение игры), нужно выполнять многократно, причем заранее неизвестно, сколько раз (это зависит от ответов игрока). И, как мы теперь

знаем, если требуется повторять какие-то действия, значит, в программе

нужен цикл.

Однако в нашем списке действий ничего не говорится о том, что

и когда должно происходить. Чтобы выяснить этот вопрос и лучше представить себе структуру будущей программы, мы можем воспользоваться

псевдокодом.

7. Пишем игру «Виселица» 115

Используем псевдокод для проектирования игры

Псевдокод — удобный инструмент, который программисты часто

используют при проектировании программ. Слово «псевдокод» означает

«ненастоящий код». Хотя в псевдокоде есть циклы и условия, в целом

программа описывается обычным человеческим языком. Чтобы разобраться, что это значит, давайте посмотрим на описание нашей игры

в псевдокоде:

Выбрать случайное слово

Пока слово не угадано {

Показать игроку текущее состояние игры

Запросить у игрока вариант ответа

Если игрок хочет выйти из игры {

Выйти из игры

}

Иначе Если вариант ответа — не одиночная буква {

Сообщить игроку, что он должен ввести букву

}

Иначе {

Если такая буква есть в слове {

Обновить состояние игры, подставив новую букву

}

}

}

Поздравить игрока с победой — слово угадано

Как видите, это не программный код, который может выполнить

компьютер. Однако такая запись дает нам представление о структуре

программы прежде, чем мы перейдем к написанию кода и выяснению

мелких деталей, например, как именно выбирать случайное слово.

Отображение состояния игры

Одна из строк нашего псевдокода гласит: «Показать игроку текущее

состояние игры». Для игры «Виселица» это означает подставить в слово

угаданные игроком буквы, а также показать, какие буквы осталось угадать. Как мы будем это делать? В сущности, можно хранить состояние

игры тем же способом, что и в обычной «Виселице»: в виде последовательности «пустых мест», которые мы будем заполнять по мере того, как

игрок угадывает буквы.

Мы сделаем это с помощью массива «пустых мест» — по одному

элементу для каждой буквы в слове. Назовем этот массив «итоговым

массивом» и будем по ходу игры заполнять его угаданными буквами.

А каждое из «пустых мест» представим в виде строки со знаком подчеркивания: "\_".

116 Часть I. Основы

Сначала наш итоговый массив будет просто набором «пустых

мест», количество которых равно количеству букв в загаданном слове.

Например, если загадано слово «рыба», массив будет выглядеть так:

["\_", "\_", "\_", "\_"]

Если игрок угадает букву «ы», мы заменим второй элемент на «ы»:

["\_", "ы", "\_", "\_"]

А когда игрок угадает все буквы, массив примет вид:

["р", "ы", "б", "а"]

Также нам понадобится переменная для

хранения количества букв, которые осталось

угадать. Для каждого вхождения верно угаданной буквы эта переменная будет уменьшаться

на 1, и, когда она примет значение 0, мы поймем, что игрок победил.

Проектируем игровой цикл

Основная часть игры будет располагаться внутри цикла while (в нашем

псевдокоде этот цикл начинается со строки «Пока слово не угадано»).

В цикле мы будем отображать текущее состояние игры (то есть слово,

поначалу представленное одними знаками подчеркивания), запрашивать у игрока вариант ответа (и проверять, действительно ли тот ввел

одиночную букву), а также обновлять итоговый массив, подставляя введенную букву, если она действительно есть в слове.

Практически все компьютерные игры организованы в виде того или

иного цикла, нередко структурно похожего на цикл нашей «Виселицы».

В целом игровой цикл выполняет следующие задачи:

1. Принимает ввод от игрока.

2. Обновляет состояние игры.

3. Показывает игроку текущее состояние игры.

Такого рода цикл применяется даже в играх, где непрерывно что-то

меняется, — просто он выполняется очень быстро. В случае нашей

7. Пишем игру «Виселица» 117

«Виселицы» программа запрашивает у игрока вариант ответа, обновляет

итоговый массив (если ответ верный) и отображает новое состояние итогового массива.

Если игрок угадает все буквы в слове, мы должны показать ему законченное слово, а также вывести сообщение, поздравляющее с победой.

Программируем игру

Теперь, когда у нас есть представление о структуре игры, можно переходить к написанию кода. Сначала мы рассмотрим его по частям. После этого

вы увидите весь код целиком, чтобы с удобством ввести его и поиграть.

Выбираем случайное слово

Первым делом нам нужно выбрать случайное слово. Вот как это делается:

 var words = [

"программа",

"макака",

"прекрасный",

"оладушек"

];

 var word = words[Math.floor(Math.random() \* words.length)];

Наша игра начинается со строки , где мы создаем массив со словами (программа, макака, прекрасный и оладушек), из которого затем

будем выбирать слово для отгадывания (все слова должны быть записаны строчными буквами). Cохраним этот массив в переменной words.

В строке  мы используем Math.random и Math.floor, чтобы выбрать

из массива случайное слово — так же как в третьей главе выбирали слова

для генератора дразнилок.

Создаем итоговый массив

Далее создадим пустой массив под названием answerArray (итоговый

массив) и заполним его символами подчеркивания (\_), количество которых соответствует количеству букв в загаданном слове.

var answerArray = [];

 for (var i = 0; i < word.length; i++) {

answerArray[i] = "\_";

}

var remainingLetters = word.length;

Words — слова

Answer array —

массив

с ответом

Remaining

letters —

оставшиеся

буквы

118 Часть I. Основы

В строке  в начале цикла for создается переменная цикла i, которая сначала равна 0, а затем возрастает до word.length (не включая,

однако, само значение word.length). При каждом повторе цикла мы

добавляем в массив новый элемент — answerArray[i]. Когда цикл

завершится, длина answerArray будет такой же, как длина слова.

Например, если было выбрано слово «макака» (в котором шесть букв),

answerArray примет вид [" \_ ", " \_ ", " \_ ", " \_ ", " \_ ", " \_ "]

(шесть знаков подчеркивания).

Наконец, создадим переменную remainingLetters, приравняв ее

к длине загаданного слова. Эта переменная понадобится, чтобы отслеживать количество букв, которые осталось угадать. Каждый раз, когда

игрок угадает букву, мы будем декрементировать (то есть уменьшать)

значение этой переменной: на 1 для каждого вхождения буквы в слово.

Программируем игровой цикл

Основа игрового цикла выглядит так:

while (remainingLetters > 0) {

// Основной код

// Показываем состояние игры

// Запрашиваем вариант ответа

// Обновляем answerArray и remainingLetters для каждого

// вхождения угаданной буквы

}

Мы используем цикл while, который будет повторяться

до тех пор, пока условие remainingLetters > 0 дает true.

В теле цикла надо будет обновлять remainingLetters

для каждого правильного ответа игрока; когда игрок угадает все буквы, remainingLetters примет значение 0,

и цикл завершится.

Далее мы рассмотрим код, составляющий тело игрового цикла.

Отображение состояния игры

Первым делом в теле игрового цикла нужно показать игроку текущее

состояние игры:

alert(answerArray.join(" "));

Мы делаем это, объединяя элементы answerArray в строку с пробелом в качестве разделителя, а затем с помощью alert показываем

эту строку. Например, пусть загадано слово «макака» и игрок угадал

7. Пишем игру «Виселица» 119

буквы «м» и «а». Тогда итоговый массив примет вид: ["м", "а", " \_ ",

"а", " \_ ", "а"] и answerArray.join("") вернет строку "м а \_ а \_ а".

Диалог alert в этом случае будет выглядеть как на рис. 7.4.

Рис. 7.4. Отображение состояния игры с помощью alert

Обработка введенного ответа

Теперь нужно запросить у игрока ответ и убедиться, что он ввел одиночную букву.

 var guess = prompt("Угадайте букву или нажмите Отмена для выхода 

из игры.");

 if (guess === null) {

break;

 } else if (guess.length !== 1) {

alert("Пожалуйста, введите только одну букву.");

} else {

 // Обновляем состояние игры

}

В строке  prompt запрашивает у игрока ответ и сохраняет его

в переменной guess. Далее возможен один из четырех вариантов развития событий.

Первый вариант — если игрок нажмет кнопку «Отмена», guess примет значение null. Этот вариант мы проверяем в строке  командой

if (guess === null). Если это условие даст true, мы с помощью break

выйдем из цикла.

! Ключевое слово break можно использовать для немедленного выхода

из любого цикла, независимо от того, где именно внутри цикла это происходит, или от того, выполняется ли на этот момент условие while.

Второй и третий варианты — игрок не ввел ничего либо ввел

несколько букв. Если он просто нажал «ОК», ничего не вводя, в guess

окажется пустая строка (""), а guess.length вернет 0. Если же игрок

ввел больше одной буквы, guess.length вернет число больше 1.

В строке  мы с помощью else if (guess.length !== 1) обрабатываем эти варианты, то есть проверяем, что guess содержит в точности

Guess —

предположение

Break — здесь

«отмена»

120 Часть I. Основы

одну букву. В противном случае мы отображаем диалог alert, гласящий: «Пожалуйста, введите только одну букву».

Четвертый вариант — игрок, как и положено, ввел одну букву. Тогда

мы должны обновить состояние игры — это происходит в строке ,

в секции else. Об этом пойдет речь ниже.

Обновление состояния игры

Если игрок ввел корректный ответ, мы должны обновить answerArray

согласно этому ответу. Для этого добавим в тело else такой код:

 for (var j = 0; j < word.length; j++) {

 if (word[j] === guess) {

answerArray[j] = guess;

 remainingLetters--;

}

}

В строке  мы задали цикл for с новой переменной j, которая будет

менять значение от 0 до word.length, не включая само значение word.

length. (Мы назвали переменную j, поскольку имя i уже использовано в предыдущем цикле for.) В этом цикле мы проверяем каждую

букву переменной word. Например, пусть в word находится строка

"оладушек". Тогда при первом повторе цикла, когда j равно 0, word[j]

вернет "о". При следующем повторе word[j] вернет "л", затем "а", "д",

"у", "ш", "е" и, наконец, "к".

В строке  мы с помощью if (word[j] === guess) проверяем,

совпадает ли текущая буква (word[j]) с ответом игрока. Если это

так, мы обновляем итоговый массив, добавляя туда букву командой

answerArray[j] = guess. Для каждой буквы, совпадающей с ответом, мы обновляем соответствующую позицию итогового массива.

Этот код работает, поскольку переменную цикла j можно использовать

одновременно в качестве индекса в строке word и индекса в массиве

answerArray, как показано на рис. 7.5.

["\_", "\_", "\_", "\_", "\_", "\_", "\_", "\_"]

Индекс (j)

word

answerArray

0 1 2 3 4 5 6 7

" о л а д у ш e к "

Рис. 7.5. Один и тот же индекс можно использовать для word и answerArray

Например, представим, что мы только начали игру и дошли до цикла

for в строке . Пусть загадано слово «программа», в guess находится

буква "р", а answerArray имеет вид:

7. Пишем игру «Виселица» 121

["\_", "\_", "\_", "\_", "\_", "\_", "\_", "\_", "\_"]

При первом повторе for в строке  j равно 0, поэтому word[j] вернет "п". Наш ответ (guess) — это "р", поэтому мы пропускаем команду

if в строке  (ведь условие "п" === "р" дает false). При следующем

повторе j равно 1, и word[j] вернет "р". Это значение совпадает с guess,

и срабатывает оператор if. Команда answerArray[j] = guess присваивает элементу с индексом 1 (второй элемент) массива answerArray значение guess, и теперь answerArray имеет вид:

["\_", "р", "\_", "\_", "\_", "\_", "\_", "\_", "\_"]

При следующих двух повторах цикла word[j] вернет "о", а затем

"г", что не совпадает со значением guess. Однако когда j достигнет 4,

word[j] снова вернет "р". И снова мы обновим answerArray, на этот

раз присвоив значение guess элементу с индексом 4 (пятый элемент).

Теперь answerArray выглядит так:

["\_", "р", "\_", "\_", "р", "\_", "\_", "\_", "\_"]

Оставшиеся буквы не совпадают с "р", так что при дальнейших

повторах ничего не произойдет. Так или иначе, после завершения цикла

в answerArray будут внесены все совпадения guess с соответствующими позициями word.

Помимо обновления answerArray для каждого совпадения с guess

требуется уменьшать remainingLetters на 1. Мы делаем это в строке 

командой remainingLetters--;. Каждый раз, когда guess совпадает

с буквой из word, remainingLetters уменьшается на 1, и, когда игрок

угадает все буквы, remainingLetters примет значение 0.

Конец игры

Как мы знаем, игровой цикл while выполняется при условии

remainingLetters > 0, поэтому его тело будет повторяться до тех пор,

пока еще остаются неотгаданные буквы. Когда же remainingLetters

уменьшится до 0, цикл завершится. После цикла нам остается лишь

закончить игру — это позволяет сделать такой код:

alert(answerArray.join(" "));

alert("Отлично! Было загадано слово " + word);

122 Часть I. Основы

В первой строке мы последний раз отображаем итоговый массив. Во второй строке, опять же с помощью alert, мы поздравляем

игрока с победой.

Код игры

Итак, мы разобрали по частям весь код игры, осталось лишь

соединить все вместе. Ниже он приведен целиком, от начала

до конца. Я добавил в него комментарии, поясняющие, что происходит в том или ином месте программы. Обязательно вручную

введите код в компьютер — это поможет вам поскорее набить

руку в JavaScript. Создайте новый файл под названием hangman.

html и введите в него следующее:

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Виселица!</title>

</head>

<body>

<h1>Виселица!</h1>

<script>

// Создаем массив со словами

var words = [

"программа",

"макака",

"прекрасный",

"оладушек"

];

// Выбираем случайное слово

var word = words[Math.floor(Math.random() \* words.length)];

// Создаем итоговый массив

var answerArray = [];

for (var i = 0; i < word.length; i++) {

answerArray[i] = "\_";

}

var remainingLetters = word.length;

// Игровой цикл

while (remainingLetters > 0) {

// Показываем состояние игры

alert(answerArray.join(" "));

Hangman —

здесь «Виселица»

7. Пишем игру «Виселица» 123

// Запрашиваем вариант ответа

var guess = prompt("Угадайте букву, или нажмите Отмена для

выхода из игры.");

if (guess === null) {

// Выходим из игрового цикла

break;

} else if (guess.length !== 1) {

alert("Пожалуйста, введите одиночную букву.");

} else {

// Обновляем состояние игры

for (var j = 0; j < word.length; j++) {

if (word[j] === guess) {

answerArray[j] = guess;

remainingLetters--;

}

}

}

// Конец игрового цикла

}

// Отображаем ответ и поздравляем игрока

alert(answerArray.join(" "));

alert("Отлично! Было загадано слово " + word);

</script>

</body>

</html>

Если игра не запускается, проверьте, все ли вы ввели правильно.

Обнаружить ошибки вам поможет JavaScript-консоль. Например, если

вы сделали опечатку в имени переменной, в консоли появится сообщение с указанием, в какой строке ошибка, — примерно такое, как

на рис. 7.6.

Рис. 7.6. Ошибка JavaScript в консоли Chrome

Кликнув по ссылке hangman.html:30, вы увидите строку программы,

где произошла ошибка. В данном случае, изучив код, можно понять, что

мы напечатали remainingLetter вместо remainingLetters в условии цикла while.

Поиграйте в свою «Виселицу» некоторое время. Работает ли программа так, как вы этого ожидаете? Можете ли вы представить, как

выполняется во время игры JavaScript-код?

124 Часть I. Основы

Что мы узнали

Не так-то много страниц, а вы уже создали свою первую

игру на JavaScript! Как видите, циклы и условные конструкции совершенно необходимы при написании игр

и других интерактивных программ; без этих управляющих конструкций программе остается лишь завершиться,

едва начав работу.

В восьмой главе мы познакомимся с функциями, что

позволит нам запускать один и тот же фрагмент кода

из разных мест программы.

УПРАЖНЕНИЯ

Вот несколько упражнений по усовершенствованию игры «виселица», которую мы написали в этой главе. #1. Больше слов Добавьте новые слова в массив words. Не забывайте, что все буквы должны быть строчными. #2. Заглавные буквы Если игрок введет заглавную букву, она не совпадет с такой же строчной буквой в загаданном слове. Эту вероятность можно учесть, преобразовав введенную букву в строчный регистр. (Подсказка: для этого подойдет метод toLowerCase.) #3. Ограничение по ходам Сейчас наша «Виселица» позволяет угадывать буквы сколько угодно раз, до победного конца. Добавьте в программу переменную для учета попыток и завершайте игру, если игрок израсходовал все попытки. (Подсказка: проверяйте значение этой переменной в том же цикле while, где проверяется условие remainingLetters > 0, а с помощью операции && можно убедиться, что сразу два логических условия дают true — см. вторую главу.) #4. Исправьте ошибку В игре есть ошибка: если вы будете снова и снова вводить одну и ту же букву, которая есть в загаданном слове, remainingLetters будет снова и снова уменьшаться. Постарайтесь это исправить! (Подсказка: можно добавить проверку еще одного условия — что в соответствующем элементе answerArray все еще стоит знак подчеркивания. Если там другой символ, значит, эта буква уже угадана.)

//Ига виселица

//Первым делом нам нужно выбрать случайное слово. Вот как это делается:

let words = ["программа","макака","прекрасный","оладушек"];

//#1. Больше слов

//Добавьте новые слова в массив words. Не забывайте, что все

//буквы должны быть строчными.

words.push("подушка");// добавить слово в конец массива

words[5] = "пуля";// добавить слово в конец массива

console.log(words);

//#2. Заглавные буквы

/\*Если игрок введет заглавную букву, она не совпадет с такой же

строчной буквой в загаданном слове. Эту вероятность можно

учесть, преобразовав введенную букву в строчный регистр.

(Подсказка: для этого подойдет метод toLowerCase.)\*/

//console.log(words[0][0].toUpperCase() + words.slice(1).toLowerCase());

/\*Наша игра начинается со строки , где мы создаем массив со словами (программа, макака, прекрасный и оладушек), из которого затем

будем выбирать слово для отгадывания (все слова должны быть записаны строчными буквами). Cохраним этот массив в переменной words.

В строке  мы используем Math.random и Math.floor, чтобы выбрать

из массива случайное слово — так же как в третьей главе выбирали слова

для генератора дразнилок.\*/

// Выбираем случайное слово

let word = words[Math.floor(Math.random() \* words.length)];

// Создаем итоговый массив

/\*Далее создадим пустой массив под названием answerArray (итоговый

массив) и заполним его символами подчеркивания (\_), количество которых соответствует количеству букв в загаданном слове. В строке  в начале цикла for создается переменная цикла i, которая сначала равна 0, а затем возрастает до word.length (не включая,

однако, само значение word.length). При каждом повторе цикла мы

добавляем в массив новый элемент — answerArray[i]. Когда цикл

завершится, длина answerArray будет такой же, как длина слова.

Например, если было выбрано слово «макака» (в котором шесть букв),

answerArray примет вид [" \_ ", " \_ ", " \_ ", " \_ ", " \_ ", " \_ "]

(шесть знаков подчеркивания).

Наконец, создадим переменную remainingLetters, приравняв ее

к длине загаданного слова. Эта переменная понадобится, чтобы отслеживать количество букв, которые осталось угадать. Каждый раз, когда

игрок угадает букву, мы будем декрементировать (то есть уменьшать)

значение этой переменной: на 1 для каждого вхождения буквы в слово.\*/

let remainingLetters = word.length;

let answerArray = [];

for (let i = 0; i < word.length; i++) {

answerArray[i] = "\_";

/\*#4. Исправьте ошибку

В игре есть ошибка: если вы будете снова и снова вводить одну и ту

же букву, которая есть в загаданном слове, remainingLetters

будет снова и снова уменьшаться. Постарайтесь это исправить!

(Подсказка: можно добавить проверку еще одного условия — что

в соответствующем элементе answerArray все еще стоит знак

подчеркивания. Если там другой символ, значит, эта буква уже

угадана.)\*/

if (answerArray[i] == "\_") {

remainingLetters = word.length;

} else {answerArray.join(" ");}

}

//let remainingLetters = word.length;

/\*3 Ограничение по ходам

Сейчас наша «Виселица» позволяет угадывать буквы сколько

угодно раз, до победного конца. Добавьте в программу переменную для учета попыток и завершайте игру, если игрок

израсходовал все попытки. (Подсказка: проверяйте значение

этой переменной в том же цикле while, где проверяется условие remainingLetters > 0, а с помощью операции && можно

убедиться, что сразу два логических условия дают true —

см. вторую главу.) \*/

let attempt = 0;

// Игровой цикл

/\*Мы используем цикл while, который будет повторяться

до тех пор, пока условие remainingLetters > 0 дает true.

В теле цикла надо будет обновлять remainingLetters

для каждого правильного ответа игрока; когда игрок угадает все буквы, remainingLetters примет значение 0,

и цикл завершится.

Далее мы рассмотрим код, составляющий тело игрового цикла.

Отображение состояния игры

Первым делом в теле игрового цикла нужно показать игроку текущее

состояние игры:\*/

while (remainingLetters > 0 && attempt < 10) {

// Основной код

// Показываем состояние игры

// Запрашиваем вариант ответа

// Обновляем answerArray и remainingLetters для каждого

// вхождения угаданной буквы

// Показываем состояние игры

attempt++;

attempt === null;

alert(answerArray.join(" "));

/\*Мы делаем это, объединяя элементы answerArray в строку с пробелом в качестве разделителя, а затем с помощью alert показываем

эту строку. Например, пусть загадано слово «макака» и игрок угадал

буквы «м» и «а». Тогда итоговый массив примет вид: ["м", "а", " \_ ",

"а", " \_ ", "а"] и answerArray.join("") вернет строку "м а \_ а \_ а".

\*/

// Запрашиваем вариант ответа

/\*Теперь нужно запросить у игрока ответ и убедиться, что он ввел одиночную букву.\*/

/\*2 Если игрок введет заглавную букву, она не совпадет с такой же

строчной буквой в загаданном слове. Эту вероятность можно

учесть, преобразовав введенную букву в строчный регистр.

(Подсказка: для этого подойдет метод toLowerCase.)\*/

//let guess = prompt("Угадайте букву, или нажмите Отмена для выхода из игры."/\*"guess the letter, 0 for exit"\*/).toLowerCase();

let guess = prompt("Угадайте букву, или нажмите Отмена для выхода из игры.");

if (guess === null) {

// Выходим из игрового цикла

break;

} else if (guess.length !== 1) {

alert("Пожалуйста, введите одиночную букву.");

} else {

// Обновляем состояние игры

for (let j = 0; j < word.length; j++) {

if (word[j] === guess) {

answerArray[j] = guess;

remainingLetters--;

}

}

}

// Конец игрового цикла

}

// Отображаем ответ и поздравляем игрока

alert(answerArray.join(" "));

alert("Отлично! Было загадано слово " + word);

Вариант Даши

let words = ['программа', 'макака', 'прекрасный', 'оладушек']

words.push('подушка')

const word = words[Math.floor(Math.random() \* words.length)]

let answerArray = []

for (i = 0; i < word.length; i++) {

answerArray.push('\_')

}

let remainingLetters = word.length

console.log(remainingLetters)

while (remainingLetters > 0) {

const guess = prompt('Угадайте букву, или нажмите Отмена для выхода из игры.').toLowerCase()

if (guess === null) {

break

} else if (guess.length !== 1) {

alert('Пожалуйста, введите одиночную букву.')

} else {

for (let j = 0; j < word.length; j++) {

if (word[j] === guess) {

answerArray[j] = guess

remainingLetters--

}

}

}

alert(answerArray.join(' '))

}

alert(`Отлично! Было загадано слово '${word}'`)

8 ФУНКЦИИ

Функции — это механизм для многократного использования частей кода. Они позволяют запускать один и тот же код из разных мест программы без необходимости его копировать. Кроме того, если вы «спрячете» сложные фрагменты кода внутри функций, вам будет легче сосредоточиться на проектировании программы — так вы будете налаживать взаимодействие между функциями, а не барахтаться в мелких деталях, из которых состоит код этих фрагментов. Организация кода в виде небольших, легко контролируемых частей позволяет видеть общую картину и думать о строении программы на более высоком уровне. Функции очень удобны, когда нужно многократно выполнять в программе некие расчеты или другие действия. Мы уже пользовались готовыми функциями, такими как Math.random, Math.floor, alert, prompt и confirm. А в этой главе мы научимся создавать свои функции. Базовое устройство функции На рис. 8.1 показано строение функции. Код внутри фигурных скобок называется телом функции — аналогично циклам, где код в фигурных скобках зовется телом цикла. function () { console.log("Делаем что-то"); } Тело функции записывается в фигурных скобках Рис. 8.1. Синтаксис создания функции 127 Создаем простую функцию Давайте создадим функцию, которая печатает фразу «Привет, мир!». Введите в консоли браузера следующий код. Чтобы перейти к новой строке без выполнения уже введенных команд, используйте SHIFT-ENTER. var ourFirstFunction = function () { console.log("Привет, мир!"); }; Этот код создает новую функцию, сохраняя ее в переменной ourFirstFunction. Вызов функции Чтобы запустить код функции (то есть ее тело), нужно эту функцию вызвать. Для этого укажите ее имя, а следом — открывающую и закрывающую скобки, вот так: ourFirstFunction(); Привет, мир! При вызове ourFirstFunction выполняется ее тело, то есть команда console.log("Привет, мир!");, и текст, который мы таким образом выводим, появляется в консоли на следующей строке: "Привет, мир!" Однако, вызвав эту функцию из браузера, можно заметить в консоли еще одну строчку — с маленькой, указывающей влево стрелкой, как на рис. 8.2. Это значение, которое возвращает функция. Рис. 8.2. Вызов функции, возвращаемое значение которой не определено Возвращаемое значение — это значение, которое функция выдает наружу, чтобы потом его можно было использовать где угодно в программе. В данном случае это undefined, поскольку мы не указывали возвращаемое значение в теле функции, мы лишь дали команду вывести Our first function — наша первая функция 128 Часть I. Основы текст в консоль. Функция всегда будет возвращать undefined, если в теле функции нет указания вернуть что-нибудь другое. (В разделе «Возврат значения из функции» на с. 131 мы выясним, как это можно сделать.) ! В консоли Chrome и примерах кода в этой книге возвращаемые значения помечены разными цветами в зависимости от их типа данных, тогда как текст, напечатанный через console.log, всегда показан черным цветом. Передача аргументов в функцию Наша функция ourFirstFunction выводит одну и ту же строку при каждом вызове, однако хотелось бы, чтобы поведением функции можно было управлять. Чтобы функция могла изменять поведение в зависимости от значений, нам понадобятся аргументы. Список аргументов указывается в скобках после имени функции — как при ее создании, так и при вызове. Функция sayHelloTo использует аргумент (name), чтобы поздороваться с человеком, имя которого передано в аргументе: var sayHelloTo = function (name) { console.log("Привет, " + name + "!"); }; Здесь мы создали функцию и сохранили ее в переменной sayHelloTo. При вызове функция печатает строку "Привет, " + name + "!", заменяя name на значение, переданное в качестве аргумента. На рис. 8.3 показан синтаксис создания функции с одним аргументом. function ( argument ) { console.log("Передан аргумент: " + argument); } Аргумент можно использовать в теле функции Имя аргумента Рис. 8.3. Синтаксис создания функции с одним аргументом Вызывая функцию, которая принимает аргумент, введите значение, которое вы хотите использовать в качестве этого аргумента в скобках после имени функции. Например, чтобы поздороваться с Ником, можно ввести: Say hello to — скажи привет [кому-то] 8. Функции 129 sayHelloTo("Ник"); Привет, Ник! А с Анной поздороваться можно так: sayHelloTo("Анна"); Привет, Анна! Каждый раз при вызове функции переданный нами аргумент name подставляется в строку, которую печатает функция. Поэтому, когда мы передаем значение "Ник", в консоли появляется "Привет, Ник!", а когда пишем "Анна", мы видим в консоли "Привет, Анна!". Печатаем котиков! Кроме того, переданный в функцию аргумент может указывать, сколько раз требуется что-то сделать. Например, функция drawCats выводит в консоль смайлы — кошачьи мордочки (вот такие: =^.^=). Задавая аргумент howManyTimes, мы сообщаем ей, сколько таких смайлов нужно напечатать: var drawCats = function (howManyTimes) { for (var i = 0; i < howManyTimes; i++) { console.log(i + " =^.^="); } }; Тело функции представляет собой цикл for, который повторяется столько раз, сколько указано в аргументе howManyTimes (поскольку переменная i сначала равна 0, а затем возрастает до значения howManyTimes - 1). На каждом повторе цикла функция выводит в консоль строку i + « =^.^=». Вот что мы увидим, вызвав эту функцию со значением 5 в качестве аргумента howManyTimes: drawCats(5); 0 =^.^= 1 =^.^= 2 =^.^= 3 =^.^= 4 =^.^= Draw cats — рисовать котов How many times — сколько раз 130 Часть I. Основы Попробуйте задать howManyTimes значение 100, чтобы напечатать 100 кошачьих мордочек! Передача в функцию нескольких аргументов В функцию можно передать больше одного значения, задав несколько аргументов. Для этого перечислите аргументы в скобках после имени функции, разделив их запятыми. На рис. 8.4 показан синтаксис создания функции, принимающей два аргумента. function (argument1, argument2) { console.log("Первый аргумент: " + argument1); console.log("Второй аргумент: " + argument2); } В теле функции можно использовать оба аргумента Имена аргументов пишутся через запятую Рис. 8.4. Синтаксис создания функции с двумя аргументами Фунция printMultipleTimes похожа на drawCats, однако она принимает еще один аргумент с именем whatToDraw. var printMultipleTimes = function (howManyTimes, whatToDraw) { for (var i = 0; i < howManyTimes; i++) { console.log(i + " " + whatToDraw); } }; Функция printMultipleTimes печатает строку, переданную в аргументе whatToDraw столько раз, сколько указано в аргументе howManyTimes. Второй аргумент сообщает функции, что печатать, а первый — сколько раз это нужно печатать. Вызывая функцию с несколькими аргументами, перечислите нужные вам значения через запятую в скобках после имени функции. Например, чтобы напечатать кошачьи мордочки с помощью функции printMultipleTimes, вызывайте ее так: printMultipleTimes(5, "=^.^="); 0 =^.^= 1 =^.^= Print multiple times — напечатать многократно What to draw — что рисовать 8. Функции 131 2 =^.^= 3 =^.^= 4 =^.^= А чтобы четыре раза напечатать смайлик «^\_^», вызывайте printMultipleTimes так: printMultipleTimes(4, "^\_^"); 0 ^\_^ 1 ^\_^ 2 ^\_^ 3 ^\_^ Здесь при вызове printMultipleTimes мы указали значение 4 для аргумента howManyTimes и строку «^\_^» для аргумента whatToDraw. В результате цикл выполнил четыре повтора (переменная i менялась от 0 до 3), каждый раз печатая i + " " + "^ \_ ^". Чтобы дважды напечатать (>\_\_\_\_< 5) { ϖ return; } return "Пятая буква вашего имени: " + name[4] + "."; }; В строке υ мы проверяем длину переданного имени — уж не короче ли оно пяти символов? Если это так, в строке ϖ мы выполняем return, чтобы незамедлительно выйти из функции. Давайте попробуем эту функцию в деле. fifthLetter("Николай"); "Пятая буква вашего имени: л." В имени Николай больше пяти букв, так что функция fifthLetter благополучно завершается, вернув пятую букву имени Николай, то есть л. Попробуем вызвать ее еще раз с именем покороче: fifthLetter("Ник"); undefined Когда мы вызвали fifthLetter для имени Ник, функция распознала, что имя недостаточно длинное, и сразу завершилась, выполнив оператор return в строке ϖ. Поскольку никакого значения после этого return не указано, функция вернула undefined. Многократное использование return вместо конструкции if... else Можно многократно использовать return внутри разных конструкций if, чтобы возвращать из функции разные значения в зависимости от входных данных. Предположим, вы пишете игру, в которой игроки награждаются медалями согласно набранным очкам. Счету меньше трех очков соответствует бронзовая медаль, счету от трех до шести — серебряная, а счету от семи и выше — золотая. Fifth letter — пятая буква 138 Часть I. Основы var medalForScore = function (score) { if (score < 3) { υ return "Бронзовая"; } ϖ if (score < 7) { return "Серебряная"; } ω return "Золотая"; }; В строке υ мы возвращаем значение "Бронзовая" и выходим из функции, если счет меньше трех очков. Если мы достигли строки ϖ, значит, счет как минимум равен трем очкам, поскольку, будь он меньше трех, мы бы уже вышли из функции (выполнив return в первом операторе if). И наконец, если мы достигли строки ω, значит, на счету как минимум семь очков, проверять больше нечего и можно спокойно вернуть значение "Золотая". Хотя мы проверяем здесь несколько условий, необходимости использовать цепочку конструкций if... else нет. Мы используем if... else, когда хотим убедиться, что будет выбран лишь один из вариантов. Однако если в каждом варианте выполняется return, это также гарантирует однозначный выбор (поскольку выйти из функции можно лишь один раз). СОКРАЩЕННАЯ ЗАПИСЬ ПРИ СОЗДАНИИ ФУНКЦИЙ Есть длинный и короткий способы записи функций. Я использую длинную запись, поскольку она наглядно демонстрирует, что функция хранится в переменной. Тем не менее вам стоит знать и о короткой записи, поскольку ее используют многие JavaScript-разработчики. Возможно, и вы сами, достаточно поработав с функциями, предпочтете короткую запись. Вот пример длинной записи: var double = function (number) { return number \* 2; }; Короткая запись той же функции выглядит так: function double(number) { return number \* 2; } Medal for score — медаль за очки Score — счет, очко (очки) 8. Функции 139 Как видите, при длинной записи мы явно создаем переменную и сохраняем в ней функцию, так что имя double записывается прежде ключевого слова function. Напротив, при короткой записи сначала идет ключевое слово function, а затем название функции. В этом случае JavaScript создает переменную double неявным образом. На техническом сленге длинная запись называется функциональным выражением, а короткая — объявлением функции. Что мы узнали Функции позволяют повторно использовать фрагменты кода. Они могут работать по-разному в зависимости от переданных аргументов и могут возвращать значение в то место кода, откуда они были вызваны. Также функции дают возможность называть фрагменты кода понятными именами, чтобы, глядя на название, мы могли сразу понять, что функция делает. В следующей главе мы узнаем, как писать JavaScript-код для работы с HTML-элементами.

// УПРАЖНЕНИЯ

/\*#1. Математические расчеты и функции

Создайте две функции, add и multiply; пусть каждая принимает по два аргумента. Функция add должна складывать аргументы и возвращать результат, а функция multiply — перемножать аргументы. С помощью только этих двух функций вычислите следующее несложное выражение:

36325 \* 9824 + 777\*/

/\*function multiply(number1, number2) {

return number1 \* number2;

};

function add(number3, multiply) {

return number3 + multiply;

};

console.log(add(777, multiply(36325,9824)));\*/

/\*#2. Совпадают ли массивы?

Напишите функцию areArraysSame, которая принимает два

массива с числами в качестве аргументов. Она должна возвращать true, если эти массивы одинаковые (то есть содержат одни

и те же числа в одном и том же порядке), или false, если массивы различаются. Убедитесь, что ваша функция работает правильно, запустив такой код:\*/

function areArraysSame (arr1, arr2) {

if (arr1.length !== arr2.length) return false;

else {

for (let i = 0; i < arr1.length; i++) {

if (arr1[i] !== arr2[i]) { // [i] это число в массиве

return false;

}

}

return true;

}

}

console.log(areArraysSame([1, 2, 3], [4, 5, 6]));

console.log(areArraysSame([1, 2, 3], [1, 2, 3]));

console.log(areArraysSame([1, 2, 3], [1, 2, 3, 4]));

/\*Подсказка 1: вам понадобится перебрать все значения из первого массива в цикле for и убедиться, что они совпадают со значениями из второго массива. Вы можете вернуть false прямо

из тела for, если обнаружите несовпадающие значения.

Подсказка 2: вы можете сразу выйти из функции, пропустив

цикл for, если у массивов разная длина.\*/