ВЕДЬ ЭТО ТАК ПРОСТО!

Программирование на JavaScript®

для чайников®

Издательство ДИАЛЕКТИКА

Анимация робота

Создание развивающих игр

Программирование веб-приложений

Крис Минник
Ева Холланд
Авторы книги *JavaScript для чайников*

Программирование на JavaScript®

для чайников®

JavaScript® For Kids

by Chris Minnick
and Eva Holland

for dummies®
A Wiley Brand

Программирование на JavaScript®

Крис Минник
Ева Холланд

для чайников®

Диалектика

Москва ◆ Санкт-Петербург
2019

ББК 32.973.26-018.2.75
М62
УДК 681.3.07

Компьютерное издательство "Диалектика"
Главный редактор *С.Н. Тригуб*
Зав. редакцией *В.Р. Гинзбург*
Перевод с английского и редакция *И.В. Василенко*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com

Минник, Крис, **Холланд**, Ева.
М62 Программирование на Javascript для чайников. : Пер. с англ. — СПб. : ООО "Диалектика", 2019. — 320 с. : ил. — Парал. тит. англ.

ISBN 978-5-907144-39-2 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Крис Минник, Ева Холланд

Программирование на Javascript для чайников

В издании использованы карикатуры Банка Изображений Cartoonbank.ru

Подписано в печать 21.03.2019. Формат 70x100/16
Усл. печ. л. 25,8. Уч.-изд. л. 15,6
Доп. тираж 300 экз. Заказ № 2435

Отпечатано в АО "Первая Образцовая типография"
Филиал "Чеховский Печатный Двор"
142300, Московская область, г. Чехов, ул. Полиграфистов, д. 1
Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8 (499) 270-73-59

ООО "Диалектика", 195027, Санкт-Петербург, Магнитогорская ул., д. 30, лит. А, пом. 848

ISBN 978-5-907144-39-2 (рус.)

ISBN 978-1-119-11986-9 (англ.)

Оглавление

Содержание

Об авторе

Крис Минник — настоящий знаток JavaScript, способный решить любую задачу. В свободное время увлекается плаванием, написанием книг и игрой на гитаре.

Ева Холланд — эксперт по вопросам программирования, учредитель компании WhatzThis?, выпускающей обучающие курсы. В сферу ее интересов входят танцы, вечеринки и, конечно же, написание книг.

Введение

Эта книга призвана познакомить вас с основами программирования на языке JavaScript. Здесь вы найдете пошаговые инструкции по написанию программ, аналогичных используемым на современных веб-сайтах. Рассчитанная на начинающих пользователей любого возраста, данная книга станет незаменимым пособием для тех, кто только начинает изучать программирование и не имеет опыта в написании кода.

Сегодня JavaScript является наиболее распространенным языком программирования в мире. Поэтому начинать обучение программированию лучше всего именно с него. Данная книга полностью посвящена написанию программ на JavaScript.

Изучать язык JavaScript и писать на нем первые программы очень просто и интересно. Обладая достаточным воображением и базовыми навыками разработки JavaScript-кода, вы быстро научитесь создавать собственные программы, имеющие широкое практическое применение.

Наилучший учитель в программировании — это каждодневная практика. Только практическое закрепление любых полученных знаний позволит вам в кратчайшие сроки овладеть всеми необходимыми приемами. Регулярное написание кода и его отладка — вот в чем секрет успеха умелого программиста.

Об этой книге

В книге сделана попытка описать язык JavaScript и способы программирования на нем максимально просто и понятно. Начав с азов, вы вскоре перейдете к изучению сложных концепций. Забавные проекты и игры, на примерах которых проводится обучение, упростят освоение материала и заинтересуют каждого читателя. По завершении обучения вы получите навыки, достаточные для разработки игр, код которых не стыдно опубликовать на сайте или показать друзьям.

Независимо от уровня вашего знания языка JavaScript, сведения, приведенные в книге, помогут вам овладеть многими

приемами программирования, которые применяются при написании веб-приложений самого разного уровня сложности.

В книге рассмотрены следующие фундаментальные принципы и концепции программирования:

- ✓ базовая структура программ JavaScript;
- ✓ операторы и выражения JavaScript;
- ✓ структурирование программ с помощью функций;
- ✓ циклическое выполнение операций;
- ✓ совместное использование JavaScript, HTML и CSS;
- ✓ ветвление кода и условная конструкция `if...else`.

Программирование на JavaScript сводится не только к изучению операторов и структур языка. Написание программ невозможно представить без использования вспомогательных инструментов, предлагаемых сторонними программистами. Результатом многолетней истории развития языка и совершенствования приемов программирования стала разработка богатого наследия, состоящего из идеально отлаженных решений, предлагаемых для всеобщего использования. В процессе освоения материала книги вы будете знакомиться со многими популярными инструментами, упрощающими написание и тестирование кода.

Для лучшего понимания приведенной в книге информации примите к сведению несколько важных замечаний. Во-первых, весь код JavaScript, HTML и CSS отображается моноширинным шрифтом, например:

```
document.write("Hi!");
```

Во-вторых, страницы книги имеют вполне определенную ширину и далеко не всегда позволяют разместить каждую строку кода JavaScript, HTML и CSS целиком. В итоге приходится разбивать такие строки на несколько. Для обозначения строки кода, которая в книге продлевается на следующую строку, используется символ ↳.

```
document.getElementById("thisIsAnElementInTheDocument")
  ↳.addEventListener("click",doSomething,false);
```

И в-третьих, код HTML и CSS не чувствителен к регистру символов. Его можно вводить как строчными, так и прописными символами. В то же время язык JavaScript чувствителен к регистру символов. Поэтому, чтобы получить ожидаемый результат, всегда следуйте приведенному в книге написанию ключевых слов, в точности повторяя регистр символов.

Простые предположения

Вам не нужно быть хакером или разработчиком, чтобы начать писать код и научиться основам программирования. Вам также не нужно в совершенстве знать устройство компьютера. Более того, вам не придется изучать двоичную систему счисления.

Тем не менее при написании книги мы рассчитывали, что вы знаете, как включается и выключается компьютер, в совершенстве владеете клавиатурой и мышью, а еще умеете подключаться к Интернету и использовать браузер по назначению. Если вы уже знаете, как создаются веб-страницы (а в этом нет ничего сложного!), то начать изучение нового материала вам будет несколько проще, чем остальным читателям.

Для полного понимания кода и правильного написания собственных программ на языке JavaScript вам понадобится быть внимательным и аккуратным. Только уделяя пристальное внимание деталям, вы станете успешным программистом!

Пиктограммы, используемые в книге

Ниже показаны пиктограммы, предназначенные для того, чтобы выделить определенную информацию, изучение которой представляет особую важность.

Обращает ваше внимание на то, что может оказаться весьма полезным в работе. Помеченная этой пиктограммой информация не обязательна для изучения, но если вы любите технические подробности, то будете рады приведенным здесь сведениям.

Этой пиктограммой помечены определенные действия или процедуры, которые заметно облегчат вашу жизнь и ускорят выполнение поставленных задач.

Увидев такую пиктограмму, уделите помеченным ею сведениям самое пристальное внимание. Она указывает на информацию, о которой нужно помнить при решении текущей задачи, или на уже пройденный материал, который вы могли забыть.

Такой пиктограммой обозначаются рекомендации, помогающие избежать неудач при выполнении сложных операций.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: info@dialektika.com
WWW: http://www.dialektika.com

Наши почтовые адреса:

в России: 195027, Санкт-Петербург, Магнитогорская ул., д. 30, ящик 116
в Украине: 03150, Киев, а/я 152

Часть I
Основы JavaScript

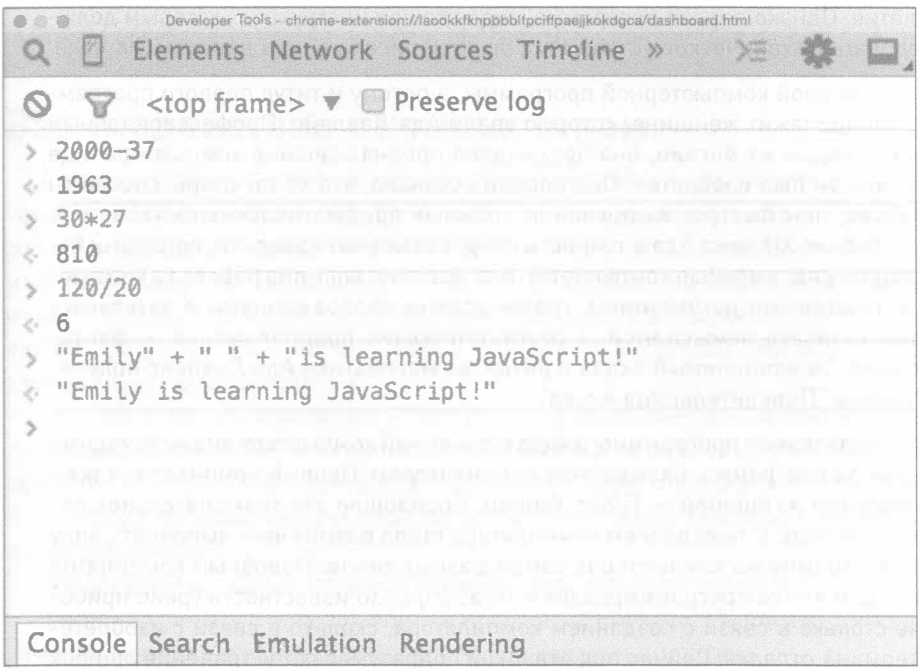
В этой части...

- ✓ Основы веб-программирования
- ✓ Синтаксис JavaScript
- ✓ Отправка и получение данных
- ✓ Разработка веб-приложений

ГЛАВА 1

Основы веб-программирования

Языку программирования JavaScript свойственны, казалось бы, совершенно несовместимые характеристики: простота изучения и широкие функциональные возможности. В этой главе вы познакомитесь с основами написания программ на JavaScript, узнаете о возможностях этого языка и введете первые строки кода на нем. Но перед тем, как начать знакомство с удивительным миром JavaScript, нужно снабдить компьютер всеми необходимыми для написания кода инструментами. Вы узнаете обо всех программах, которые понадобятся вам при создании работоспособных приложений JavaScript.



Что такое программирование

Программа представляет собой набор понятных для компьютера инструкций, которым он следует при выполнении своих действий. Процесс создания программы известен как разработка или написание программного кода. Компьютеры не умеют принимать самостоятельные решения. Все выполняемые ими операции определяются в программе. Написанием кода программ, в которых содержатся инструкции для компьютера, занимаются программисты.

Обобщающее название для компьютерных программ — *программное обеспечение* (software).

Язык программирования — женское изобретение

Компьютеры как электронные устройства были изобретены в 30-х годах прошлого столетия. Однако первая программа, содержащая инструкции, которым должно следовать механическое устройство, была написала еще в далеком XIX веке.

Авторство первой компьютерной программы, а потому и титул первого программиста, принадлежит женщине, которую звали Ада Лавлейс. Профессиональный математик родом из Англии, она предвидела предназначение компьютера еще до того, как он был изобретен. Она первой осознала, что компьютеры способны на большее, чем быстрое выполнение сложных арифметических вычислений. Еще в середине XIX века Ада в точности предсказала возможности, которыми будут обладать современные компьютеры. В те далекие времена работа за компьютером с текстовыми документами, графическими изображениями и звуковыми данными казалась немыслимой, а соответствующие предположения — фантастическими. За неоценимый вклад в развитие математики Ада Лавлейс получили прозвище "Повелительница чисел".

Для преобразования программного кода в понятный компьютеру вид необходима специальная программа, называемая *компилятором*. Первый компилятор также был изобретен женщиной — Грейс Хоппер. Произошло это знаменательное событие в 1944 году. С появлением компилятора стало возможным выполнять одну и ту же программу на компьютерах самых разных типов. Подобный компилятор создан и для языка программирования JavaScript. Но известность Грейс приобрела не столько в связи с созданием компилятора, сколько в связи с изобретением термина *отладка*. Сейчас под отладкой подразумевают устранение ошибок и неполадок в работе программы. Термин впервые появился в английском языке — de-bugging — и буквально означал "удаление насекомых". Происхождение его не случайно, поскольку один из первых в мире компьютеров Грейс починила, в буквальном смысле слова вытащив из него мотылька. За свой вклад в развитие компьютерной индустрии Грейс получила титулы "Королева программирования" и "Удивительная Грейс".

Компьютерные программы находят применение при решении самых разных задач. Без них сегодня невозможно представить многие сферы деятельности.

- ✓ Проигрывание музыкальных композиций и видеороликов.
- ✓ Выполнение математических, финансовых и научных расчетов.
- ✓ Конструирование автомобилей.
- ✓ Производство лекарств.
- ✓ Разработка игр.
- ✓ Управление робототехникой.
- ✓ Навигация искусственных спутников и космических аппаратов.
- ✓ Печать журналов и книг.
- ✓ Обучение.

Этот список можно продолжать до бесконечности. Куда ни глянь — кругом компьютеры, а где компьютеры — там и программы, запускаемые на них.

Взаимодействие с компьютером

Сердцем каждого компьютера по праву считается центральное процессорное устройство (Central Processing Unit — CPU), или просто *процессор*. Состоит процессор из огромного количества миниатюрных электронных переключателей, называемых транзисторами. Каждый из них может одновременно находиться только в одном из двух положений: включенном или выключенном. На машинном уровне назначение любой программы состоит в указании положения, в которое переводится каждый из электронных переключателей.

При выполнении инструкций программы управление положением транзисторов осуществляется с помощью двоичного, или бинарного (binary), кода. Таким образом, операции над данными в компьютере сводятся к обработке двоичных значений (нулей и единиц), в которых кодируются привычные нам числа, текстовые символы и другие виды информации.

Любая операция, выполняемая компьютером, представляется ему как череда нулей и единиц, поданных в определенном в программе порядке. Например, английская буква a в двоичном исчислении представляется следующим кодом.

```
0110 0001
```

Каждая единица или ноль в двоичном представлении называется *битом*, а комбинация из восьми битов известна как *байт*. Это очень маленькая величина. Обычно размер компьютерного файла измеряется в *килобайтах*, *мегабайтах* или даже *гигабайтах*. Несмотря на столь запутанное представление, выраженное таким образом значение, конечно же, указывает на количество 8-битовых единиц информации, хранящихся в файле.

Наиболее распространенные единицы измерения размера файла показаны в табл. 1.1.

Таблица 1.1. Единицы измерения размера файла

Название	Количество байтов	Объем хранимой информации
Килобайт (Кбайт)	1024	Два-три абзаца обычного текста
Мегабайт (Мбайт)	1 048 576	800 страниц текста
Гигабайт (Гбайт)	1 073 741 824	250 музыкальных композиций (в формате MP3)
Терабайт (Тбайт)	1 099 511 627 776	350 000 фотографий
Петабайт (Пбайт)	1 125 899 906 842 624	41 943 дисков Blue-ray

Стандартная программа начального уровня, включающая инструкции, графические файлы и данные других типов, имеет размер от нескольких килобайтов до нескольких мегабайтов. Разумеется, никому не понравится писать программы в двоичной системе, отображая инструкции в виде длинных последовательностей единиц и нолей. Это сложно и отнимает слишком много времени и сил. Намного проще создавать программы на простом языке, подобном тому, на котором мы все с вами разговариваем, а затем транслировать их в понятный компьютеру двоичный (машинный) код. В качестве такого универсального языка и выступает язык программирования.

Любая программа пишется на одном из языков программирования и состоит из набора текстовых инструкций, составленных согласно принятым в этом языке правилам. Перед тем как выполнить программу, ее необходимо преобразовать (или скомпилировать) в машинный код. В процессе компиляции все текстовые инструкции трансформируются в битовый код, понятный компьютеру.

Выбор языка программирования

Существует несколько сотен языков программирования. Возникает справедливый вопрос: зачем нужно такое большое количество языков, если все они служат единой цели — преобразованию текстовых инструкций в машинный код? Разумеется, никому не интересно создавать новые языки программирования просто так, и все они имеют свое назначение.

Существует несколько причин широкого разнообразия языков программирования. Каждый новый язык разрабатывается, чтобы:

- ✓ упростить написание программ;
- ✓ получить возможность писать программы для новых типов компьютеров;
- ✓ создавать новые типы программного обеспечения.

К наиболее распространенным языкам программирования относятся следующие:

- ✓ C;
- ✓ C++;
- ✓ C#;
- ✓ Objective C;
- ✓ Java;
- ✓ JavaScript;
- ✓ Perl;
- ✓ Python;
- ✓ Ruby;
- ✓ Scratch;
- ✓ Swift;
- ✓ Visual Basic.

Приведенный список далеко не полный и не отражает всей картины. Детальный список современных языков программирования вы найдете по следующему адресу:

`https://ru.wikipedia.org/wiki/Проект:Информационные_технологии/Списки/Список_языков_программирования`

Имея в своем распоряжении так много инструментов написания программ, на каком языке программирования остановить свой выбор? Как и в любом другом случае, ответ зависит от преследуемой цели. Например, если вы планируете создавать программы для iPhone, то используйте Objective C, JavaScript или Swift. Чтобы написать игру для компьютеров Mac или Windows, лучше всего воспользоваться языком программирования C, Java или JavaScript. А вот для создания интерактивного веб-сайта вам достаточно одного только JavaScript.

Обратили внимание на закономерность? JavaScript упоминается в каждом из случаев!

Зачем нужен JavaScript

На заре развития Интернета веб-страницы состояли преимущественно из текста, имеющего простое форматирование, и гиперссылок на другие страницы (чаще всего в пределах одного и того же сайта). На них отсутствовало все то, к чему все мы давно привыкли: формы, анимация, графические объекты и даже художественное форматирование текста.

Самое интересное, что никто не жаловался. Веб-сайты только начинали появляться, а каждый новый сайт становился значимым событием мирового масштаба — информация на нем изучалась до последнего слова. Сама возможность открытой публикации информации для всего мирового сообщества вызывала нескрываемый восторг, и мало кто беспокоился о ее художественной ценности. Чтение, а не созерцание, — вот с чего начинался Интернет в первые годы своего существования!

Прошло несколько лет, и пользователи перестали довольствоваться скупой текстовой информацией на экране. Запросы росли очень быстро, и вскоре веб-страницы запестрели яркой графикой, цветным текстом, вычурными формами и другими объектами, призванными радовать глаз своей красочностью.

Из всех старых инструментов веб-разработки наибольшее распространение получил язык программирования JavaScript. Он и сейчас успешно применяется программистами при создании веб-страниц, а все благодаря своей возможности наполнять веб-страницы самыми разными художественными и интерактивными элементами.

Исходно JavaScript использовался для добавления на веб-страницы элементов управления, с которыми могли взаимодействовать пользователи. Его функциональные возможности необычайно широки. С его помощью создаются как простые формы, обеспечивающие проверку введенных данных, так и сложные 3D-игры, запускаемые в браузере. Если, посетив веб-страницу, вы видите на ней анимацию, динамически изменяемое содержимое, интерактивную карту или запускаемую в браузере игру, то знайте: скорее всего, без JavaScript тут не обошлось.

Чтобы в полной мере представить возможности языка программирования JavaScript, запустите браузер и откройте в нем следующие сайты.

- **ShinyText (`http://cabbi.bo/ShinyText`).** Это экспериментальный сайт, на котором JavaScript применяется для отображения на экране объемного художественного текста. К тексту применяются всевозможные эффекты, обладающие собственными настройками, например Reflection Power (Степень отражения) или Repulsion Power (Степень расталкивания). Изменив их значения, можно в режиме реального времени наблюдать, как это отразится на анимации символов. Для лучшего понимания эффекта поводите указателем мыши поверх текста. Пример того, чего позволяет добиться этот сайт, показан на рис. 1.1.

Даже не понимая (пока!), как это работает, поэкспериментируйте с настройками текста на сайте ShinyText, чтобы познакомиться с возможностями JavaScript по управлению трехмерными объектами.

- **Interactive Sock Puppet (`http://www.mediosyproyectos.com/puppetic`).** Еще один прекрасный пример трехмерной анимации с помощью JavaScript. На этот раз вам предстоит управлять забавным кукольным персонажем, отдаленно напоминающим дракона. Как нетрудно заметить (рис. 1.2), он весьма эмоционален.

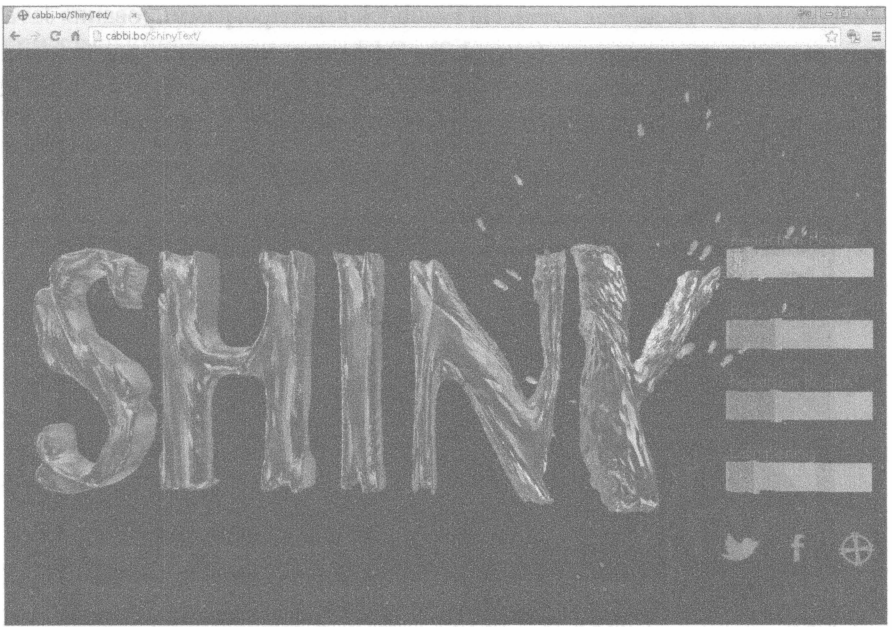


Рис. 1.1. На сайте ShinyText язык программирования JavaScript применяется для имитации трехмерного текста

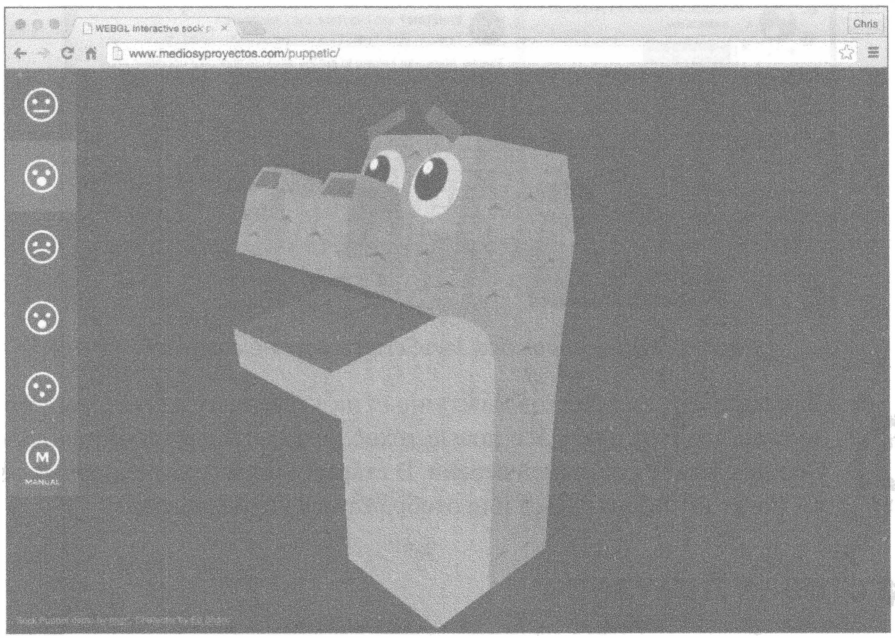


Рис. 1.2. На сайте Interactive Sock Puppet язык программирования JavaScript применяется для управления эмоциями кукольного дракона

✓ **Facebook (`www.facebook.com`).** На Facebook язык программирования JavaScript используется повсеместно (рис. 1.3). Эффекты анимации, воспроизведение видеороликов и автоматическое обновление списка постов — эти и многие другие действия выполняются исключительно благодаря JavaScript.

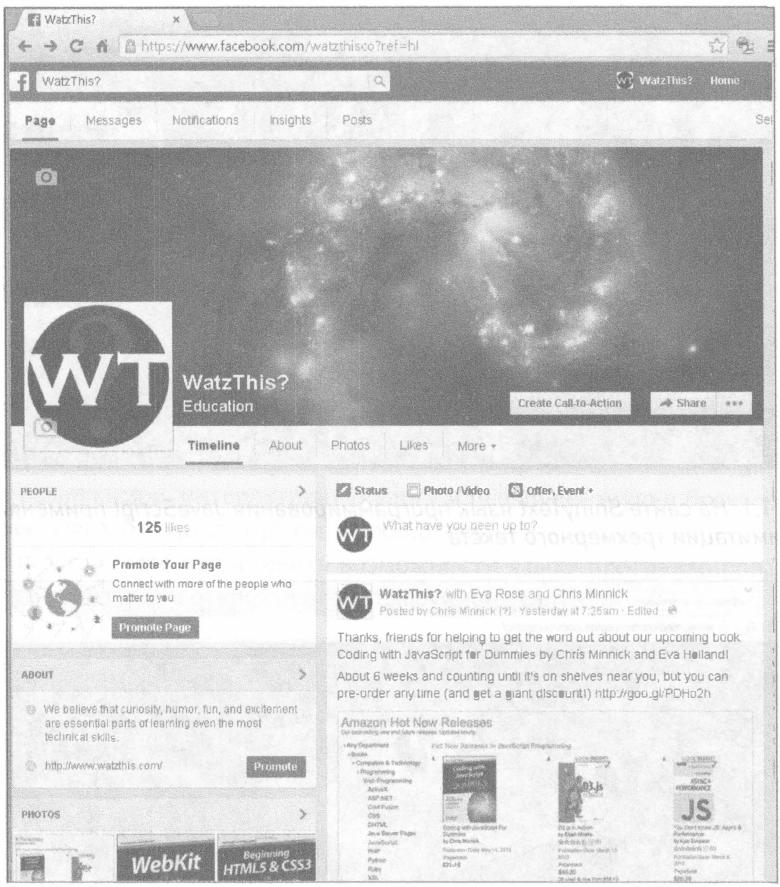


Рис. 1.3. На Facebook без JavaScript никак не обойтись

Для просмотра эффектов, доступных на указанных сайтах, ваш браузер должен поддерживать все новые технологии. Лучше всего использовать Google Chrome последней версии. В старых браузерах трехмерные эффекты могут не отображаться или отображаться не полностью.

Подготовка браузера

Для успешной работы с кодом JavaScript вам, кроме всего прочего, понадобится поддерживающий его браузер. Большинство популярных браузеров прекрасно выполняет программы JavaScript, поэтому специально устанавливать один из них вам вряд

ли придется. Скорее всего, вы будете пользоваться тем браузером, который уже есть на вашем компьютере.

К наиболее распространенным браузерам относятся Firefox, Safari, Chrome, Internet Explorer и Opera. Я рекомендовал бы остановиться на Google Chrome. Он самый распространенный среди пользователей и снабжен поддержкой всех без исключения функциональных возможностей JavaScript.

Если в вашем компьютере Chrome отсутствует, то загрузите его с сайта Google (`http://www.google.com/chrome/browser/desktop`), воспользовавшись текущим браузером, и установите в папку по умолчанию. Просто следуйте инструкциям, отображаемым на экране (в установке Chrome нет ничего сложного). Завершив процедуру, запустите браузер, чтобы удостовериться в его работоспособности.

В следующем разделе вы познакомитесь с инструментами разработчика Google Chrome, которые используются программистами и веб-дизайнерами для изучения и отладки кода JavaScript, отвечающего за представление содержимого веб-страницы в окне браузера.

Инструменты веб-разработчика

После установки и запуска браузера Chrome внимательно изучите верхнюю часть его окна. В правом верхнем углу вы видите значок с изображением трех горизонтальных линий. Это кнопка меню приложения, или просто кнопка Chrome. Щелкнув на ней, вы отобразите на экране главное меню браузера, подобное показанному на рис. 1.4.

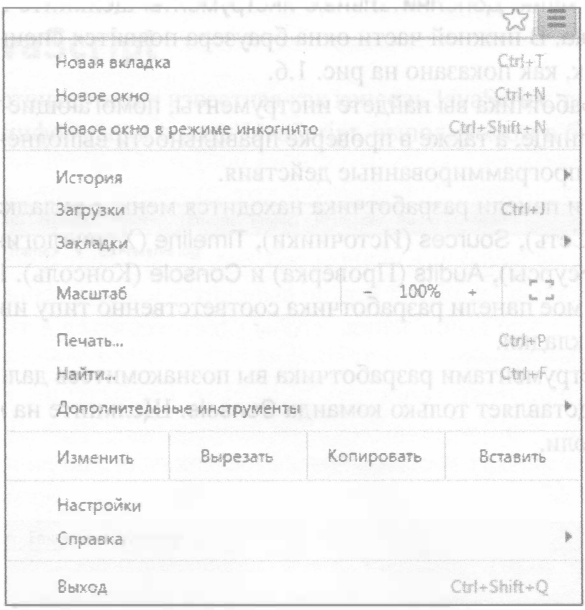


Рис. 1.4. Меню Chrome

В нижней части меню найдите команду **Дополнительные инструменты**. При наведении на нее указателя мыши на экране появляется вложенное меню (рис. 1.5). Именно в нем содержатся инструменты браузера, помогающие веб-разработчикам в их нелегком деле.

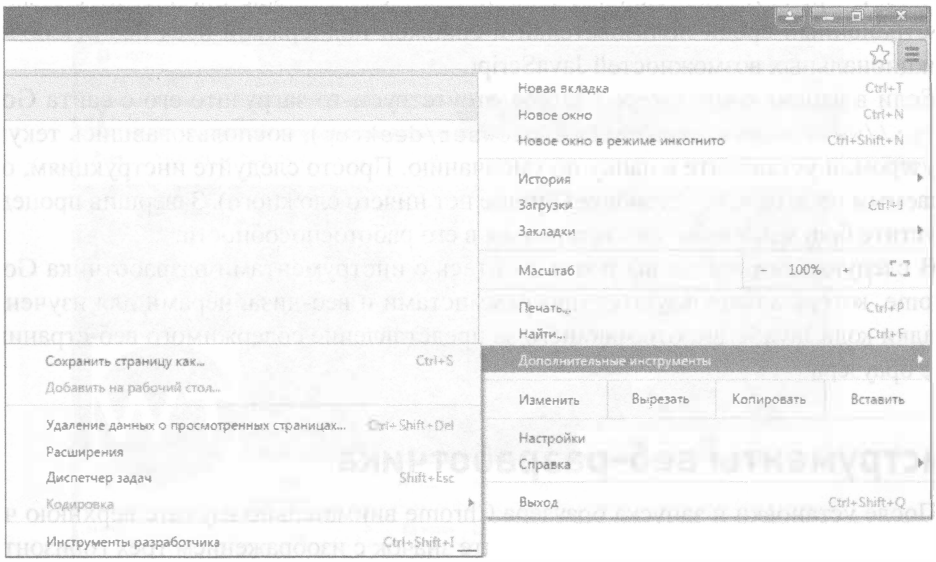


Рис. 1.5. Вложенное меню Дополнительные инструменты

Во вложенном меню **Дополнительные инструменты** щелкните на команде **Инструменты разработчика**. В нижней части окна браузера появится специальная панель, которая выглядит так, как показано на рис. 1.6.

На панели разработчика вы найдете инструменты, помогающие в поиске неисправностей на веб-странице, а также в проверке правильности выполнения кода JavaScript, отвечающего за запрограммированные действия.

В верхней части панели разработчика находится меню с вкладками **Elements** (Элементы), **Network** (Сеть), **Sources** (Источники), **Timeline** (Хронология), **Profiles** (Профили), **Resources** (Ресурсы), **Audits** (Проверка) и **Console** (Консоль). Щелкая на них, вы измените содержимое панели разработчика соответственно типу информации, указанному в названии вкладки.

Детально с инструментами разработчика вы познакомитесь далее, а на данный момент интерес представляет только команда **Console**. Щелкните на ней, чтобы отобразить вкладку консоли.

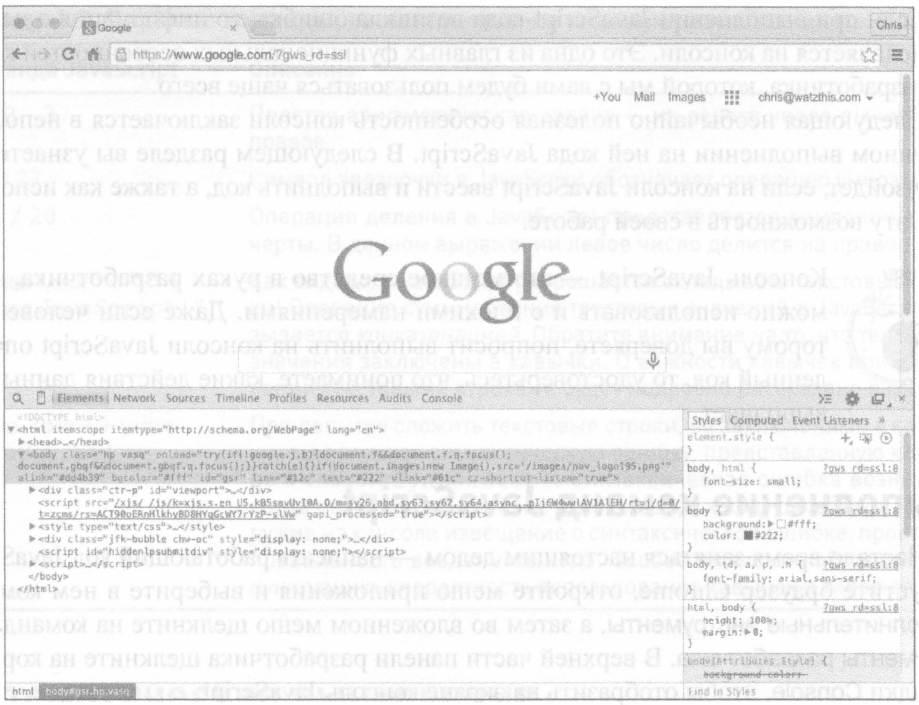


Рис. 1.6. Панель разработчика в окне браузера

Консоль JavaScript

Консоль разработчика, также известная как консоль JavaScript, показана на рис. 1.7. На нее выводится информация о коде JavaScript, выполняемом в браузере в текущий момент.

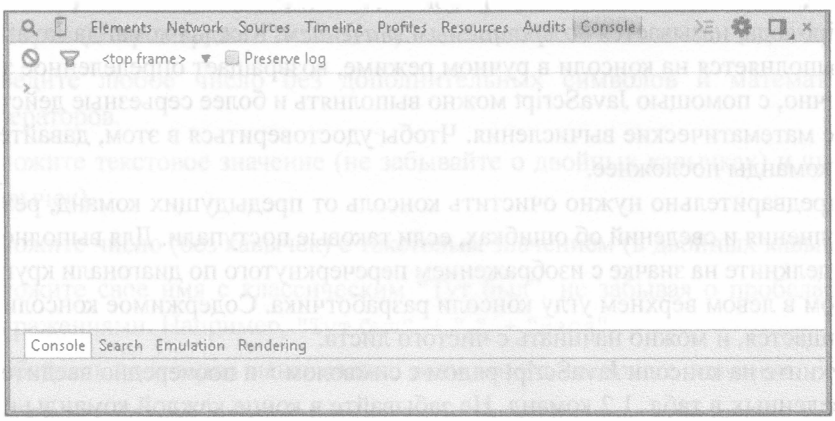


Рис. 1.7. Консоль JavaScript

Если при выполнении JavaScript-кода возникла ошибка, то информация о ней тут же появляется на консоли. Это одна из главных функциональных особенностей консоли разработчика, которой мы с вами будем пользоваться чаще всего.

Следующая необычайно полезная особенность консоли заключается в непосредственном выполнении на ней кода JavaScript. В следующем разделе вы узнаете, что произойдет, если на консоли JavaScript ввести и выполнить код, а также как использовать эту возможность в своей работе.

Консоль JavaScript — это мощное средство в руках разработчика, но ее можно использовать и с плохими намерениями. Даже если человек, которому вы доверяете, попросит выполнить на консоли JavaScript определенный код, то удостоверьтесь, что понимаете, какие действия данный код выполняет.

Выполнение команд JavaScript

Настало время заняться настоящим делом — написать работающий код JavaScript. Запустите браузер Chrome, откройте меню приложения и выберите в нем команду **Дополнительные инструменты**, а затем во вложенном меню щелкните на команде **Инструменты разработчика**. В верхней части панели разработчика щелкните на корешке вкладки **Console**, чтобы отобразить на экране консоль JavaScript.

Чтобы выполнить отдельные команды JavaScript, следуйте таким инструкциям.

1. Щелкните на консоли JavaScript рядом с символом >, чтобы установить курсор.
2. Введите `1+1` и нажмите клавишу <Return> в Mac или <Enter> в Windows.

Результат выполнения введенного вами выражения выводится в следующей строке, а перед ним отображается символ <. Угловая скобка, направленная в противоположную сторону, указывает на то, что строка получена в результате выполнения кода JavaScript, а не введена вами. Любой результат, выводимый на консоли при выполнении JavaScript-кода, называется *возвращаемым значением*. Каждая команда JavaScript, которая выполняется на консоли в ручном режиме, возвращает определенное значение.

Конечно, с помощью JavaScript можно выполнять и более серьезные действия, чем простые математические вычисления. Чтобы удостовериться в этом, давайте протестируем команды посложнее.

Но предварительно нужно очистить консоль от предыдущих команд, результатов их выполнения и сведений об ошибках, если таковые поступали. Для выполнения этой задачи щелкните на значке с изображением перечеркнутого по диагонали круга, расположенном в левом верхнем углу консоли разработчика. Содержимое консоли полностью очищается, и можно начинать с чистого листа.

Щелкните на консоли JavaScript рядом с символом > и поочередно введите каждую из приведенных в табл. 1.2 команд. Не забывайте в конце каждой команды нажимать <Return> (в Mac) или <Enter> (в Windows).

Таблица 1.2. Пробные команды, выполняемые на консоли разработчика

Команда JavaScript	Описание
`2000 - 37`	Простая арифметическая задача — из левого числа вычитается правое
`30 * 27`	Символ звездочки в JavaScript обозначает операцию умножения
`120 / 20`	Операция деления в JavaScript представляется символом косой черты. В данном выражении левое число делится на правое
`"`*Ваше имя*`" + " " + "учит JavaScript!"`	Как видите, в JavaScript разрешается складывать текстовые строки! Операция суммирования текстовых значений в JavaScript называется *конкатенацией*. Обратите внимание на то, что текстовые значения заключены в кавычки. О важности кавычек при управлении текстовыми строками будет подробно рассказано в главе 2
Ваше имя `+ + учит JavaScript!`	При попытке сложить текстовые строки, не заключенные в кавычки, вы получите синтаксическую ошибку, представленную ключевым словом `SyntaxError`. Синтаксическая ошибка возникает, когда вы вводите нечто, не относящееся к коду JavaScript. Обнаружив на консоли извещение о синтаксической ошибке, проверьте правильность введенного кода. Особое внимание обращайте на пунктуацию, корректность использования кавычек и типов данных

Математические вычисления

Теперь попробуйте выполнить простые математические вычисления самостоятельно. Только сначала очистите консоль JavaScript от записей, появившихся в результате выполнения команд, описанных в предыдущем разделе.

Смело приступайте к решению следующих задач!

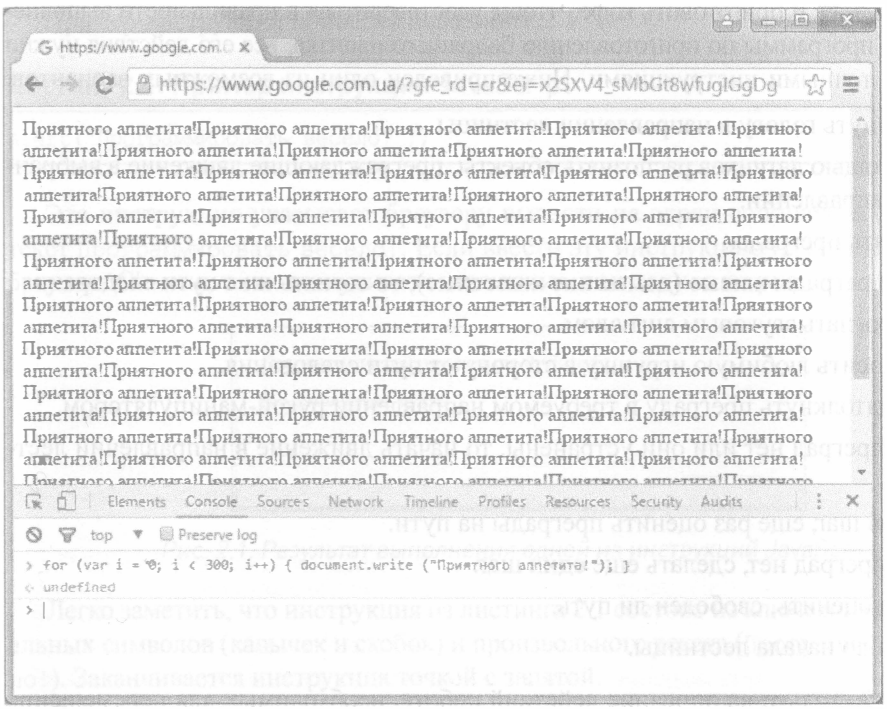
- ✓ Умножьте два десятичных числа.
- ✓ Введите в одну строку выражение, содержащее несколько математических операторов (например, `1 + 1 * 4 / 8`).
- ✓ Введите любое число без дополнительных символов и математических операторов.
- ✓ Сложите текстовое значение (не забывайте о двойных кавычках) и число (без кавычек).
- ✓ Сложите число (без кавычек) с текстовым значением (в двойных кавычках).
- ✓ Сложите свое имя с классическим “Тут был”, не забывая о пробелах между выражениями. Например, `"Тут был" + " " + "Вася"`.
- ✓ Добейтесь получения максимально возможного числового значения.
- ✓ Добейтесь получения минимально возможного числового значения.

- ✓ Попробуйте выполнить невозможную математическую операцию, как, например, деление на ноль.
- ✓ Узнайте, что произойдет, если число умножить на текстовое значение (заключенное в двойные кавычки). Например, введите `343 + "Привет!"`. В качестве результата будет возвращено значение `NaN`, что означает “Not A Number” (Не числовое значение).

ГЛАВА 2

Синтаксис JavaScript

Как и языки, на которых мы разговариваем, языки программирования имеют свой синтаксис, определяющий правила написания на них программ. Как только вы освоите принципы написания программ на JavaScript, он перестанет казаться необычным, поскольку синтаксически подобен английскому языку.



Если вам кажется, что учитель английского языка придирается к вам из-за каждой погрешности в написании слов, то не спешите с выводами. Это вы еще не знакомы с требованиями к синтаксису JavaScript. Синтаксические ошибки в коде JavaScript непростительны, так как вызывают сбои в работе программы.

В этой главе вы познакомитесь с синтаксисом языка программирования JavaScript и научитесь избегать опечаток при написании кода программ.

Точность — вежливость королей

Поскольку компьютеры не приемлют двусмысленности, инструкции для них должны быть предельно четкими и однозначными.

В главе 1 было введено понятие программы и описано, как набор инструкций преобразуется в машинный код, понятный компьютеру. Как вы уже знаете, эта операция называется компиляцией.

В задачи программиста, кроме всего прочего, входит разбиение кода на отдельные фрагменты, каждый из которых предельно просто проверить на наличие ошибок. Например, являясь счастливым обладателем домашнего робота, можно дать ему указание спуститься на первый этаж и приготовить кофе. Чтобы удостовериться в правильности выполнения роботом программы по приготовлению бодрящего напитка, все его действия нужно представить точными инструкциями. Ниже приведен один из возможных вариантов.

1. Повернуть голову в направлении лестницы.
2. С помощью датчиков распознать объекты, преграждающие движение в выбранном направлении.
3. Оценить преграды.
4. Если преграды живые (домашние животные), то согнать их с пути следования.
   - Прогнать звуковым сигналом.
   - Бросить любимую игрушку в сторону от пути следования.
   - Подтолкнуть преграду в требуемом направлении рукой-манипулятором.
5. Если преград нет или они устранены, то начать движение в направлении лестницы.
6. Сделав шаг, еще раз оценить преграды на пути.
7. Если преград нет, сделать еще один шаг.
8. Снова оценить, свободен ли путь.
9. Дойти до начала лестницы.

Как видите, только на описание действий робота, необходимых для перемещения к началу лестницы, потребовалось 9 инструкций. На самостоятельное приготовление кофе уходит намного меньше времени!

Настоящая компьютерная программа, отвечающая за перемещение робота к лестнице, содержит намного больше инструкций, чем представлено в приведенном выше списке. На каждом этапе необходимо указывать продолжительность работы каждого из сервоприводов, обрабатывать данные, поступающие с датчиков, а также принимать решения о том, как поступить с найденными преградами. И это далеко не все операции! Детализации действий нет предела.

Все указания для робота в программе JavaScript записываются в виде специальных команд, именуемых *инструкциями*.

Детально примеры управления робототехническими устройствами с помощью JavaScript рассмотрены на сайте `http://nodebots.io`.

Инструкции

В нашем языке речь записывается в виде предложений. В языке программирования JavaScript предложениям обычного языка сопоставляются *инструкции*. Как и предложения, инструкции состоят из отдельных элементов, расположение которых регулируется специальными правилами. Если их не придерживаться, то компьютер попросту не поймет программу.

Пример инструкции приведен в листинге 2.1.

Листинг 2.1. Пример инструкции JavaScript

```
alert("Программировать весело!");
```

Эта инструкция указывает браузеру вывести на экран сообщение, содержащее текст `Программировать весело!`. Если ввести эту инструкцию на консоли JavaScript браузера Chrome, то можно увидеть извещение, показанное на рис. 2.1.

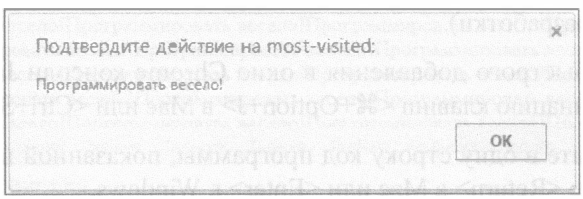


Рис. 2.1. Результат выполнения одной из инструкций JavaScript

Легко заметить, что инструкция из листинга 2.1 состоит из ключевого слова, специальных символов (кавычек и скобок) и произвольного текста (`Программировать весело!`). Заканчивается инструкция точкой с запятой.

В JavaScript вас никто не ограничивает количеством инструкций в программе. Как и в художественном произведении, написанном на обычном языке, в программе JavaScript может содержаться произвольное количество строк кода.

Приведенная выше инструкция начинается с ключевого слова `alert`, известного JavaScript. На самом деле многие инструкции начинаются с ключевых слов, но это не обязательное правило.

Инструкции разделяются точкой с запятой. В обычных текстах роль точки с запятой играет точка, которой заканчивается большинство предложений. В языке программирования JavaScript точкой с запятой заканчиваются все без исключения инструкции.

Синтаксические правила

Чтобы компьютер понимал введенный вами код JavaScript, при его написании необходимо соблюдать определенные правила. Ниже указаны наиболее важные из них.

- ✓ Ключевые слова JavaScript строго заданы.
- ✓ Пробелы не учитываются.

Давайте детально рассмотрим каждое из них. Проще всего это сделать на примере программы, выводящей на экран 300 уже известных вам извещений `Программировать весело!`. Ее код показан в листинге 2.2.

Листинг 2.2. Программа вывода на экран 300 сообщений

```
for (var i = 0; i < 300; i++) { document.write ("Программировать
   ↳весело!"); }
```

Чтобы запустить эту простую программу, выполните следующие действия.

1. Запустите браузер Google Chrome.

2. Отобразите в окне браузера консоль JavaScript (воспользовавшись панелью разработки).

   Для быстрого добавления в окно Chrome консоли JavaScript нажмите комбинацию клавиш <⌘+Option+J> в Mac или <Ctrl+Shift+J> в Windows.

3. Введите в одну строку код программы, показанной в листинге 2.2. Нажмите <Return> в Mac или <Enter> в Windows.

   Если все введено правильно, то в окне браузера появится множество (точнее — 300) одинаковых сообщений, как показано на рис. 2.2.

В данном коде основная часть работы выполняется в специальной программной структуре, известной под названием *цикл* `for`. Она позволяет выполнить одни и те же действия строго заданное количество раз. Подробно о возможностях циклических структур в JavaScript рассказывается в главах 17 и 18.

Еще раз внимательно изучите код листинга 2.2. Легко заметить, что текст сообщения `Программировать весело!` заключен в кавычки. Они указывают на то, что текст в них стоит рассматривать именно как текст, а не как ключевые слова JavaScript.

Текстовые строки

В программировании текст, заключенный в кавычки, называется строкой или строковым значением. Справедливое название, если учесть, что все символы, заключенные в кавычки, выводятся на экран последовательно и в том порядке, в котором они введены.

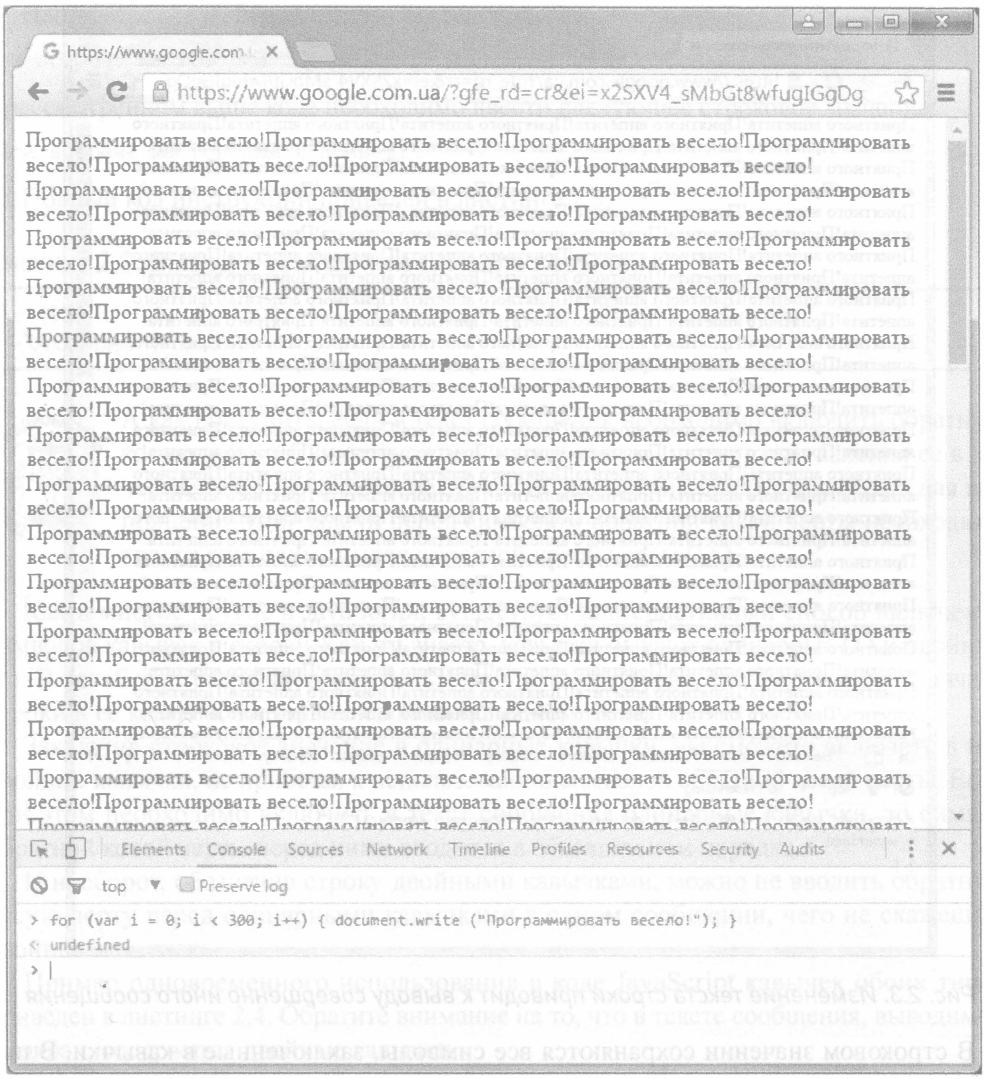


Рис. 2.2. Результат выполнения кода, приведенного в листинге 2.2

Изменить строковое значение в JavaScript очень просто. Попробуйте в примере, приведенном в листинге 2.2, заменить строковое значение `Программировать весело!` любым другим текстом, например пожеланием приятного аппетита. Выполнив измененный указанным образом код, вы получите на экране новые сообщения.

Как видно на рис. 2.3, теперь программа из листинга 2.2 выводит в окне браузера извещения `Приятного аппетита!`.

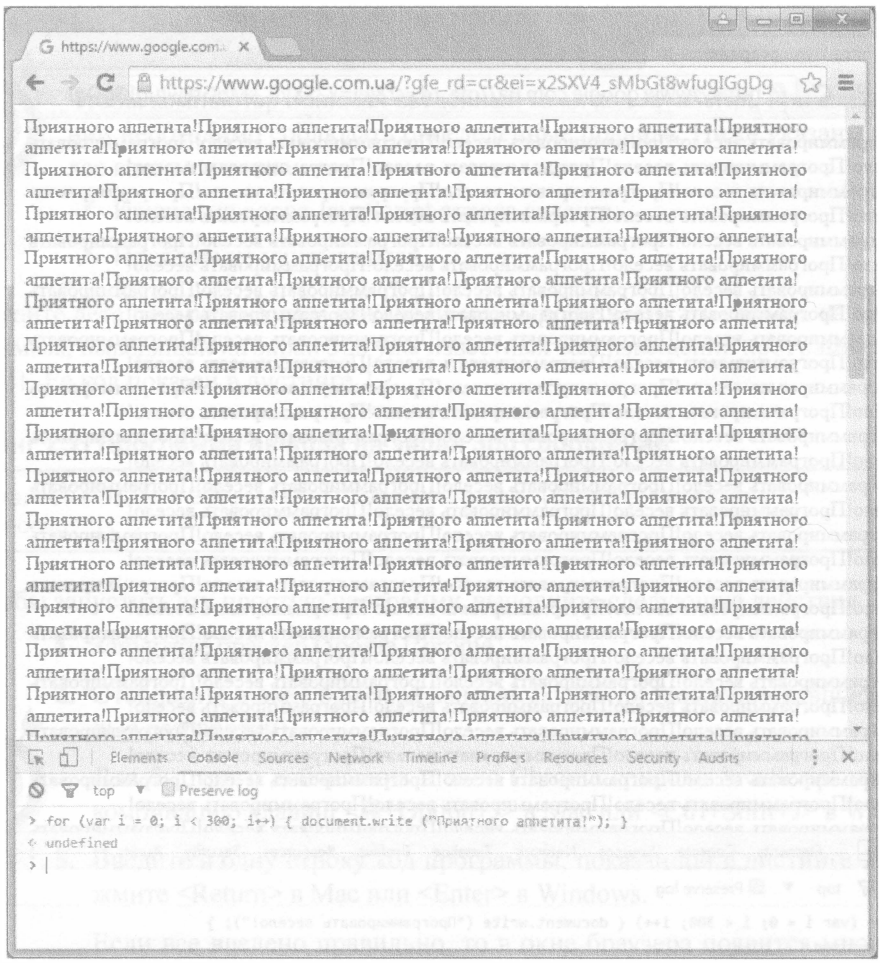


Рис. 2.3. Изменение текста строки приводит к выводу совершенно иного сообщения

В строковом значении сохраняются все символы, заключенные в кавычки. В подобном синтаксисе команды существует определенное противоречие: как поступить, если кавычки нужно включить в выводимое на экран сообщение? Должен же быть способ разграничения кавычек, включаемых в сообщение, и символов, определяющих его границы!

Для обозначения кавычек, выступающих в качестве части строкового значения, используется символ обратной косой черты (\). Он ставится перед кавычками и указывает на то, что следующий символ относится к тексту сообщения, а не к ключевому слову JavaScript. Существует целый ряд специальных знаков, включение которых в строковое значение требует добавления перед ними символа обратной косой черты. Кавычки — всего лишь один из них.

Например, для вывода на экран сообщения

```
При встрече мы говорим "Привет!"
```

в рассмотренном выше коде необходимо ввести следующее строковое значение.

```
"При встрече мы говорим \"Привет!\""
```

Полный код инструкции приведен в листинге 2.3.

Листинг 2.3. Вывод в сообщении символа кавычек

```
for (var i = 0; i < 300; i++) { document.write ("При встрече мы
   ↳говорим \"Привет!\""); }
```

А как поступить, если в текст сообщения необходимо включить обратную косую черту? Существует ли вообще возможность представления ее в качестве текстового символа? Конечно же, да! Выглядит странно, но для выполнения этой задачи перед символом обратной косой черты необходимо ввести еще один символ обратной косой черты (\\).

Как и многое другое в JavaScript, существует альтернативный способ включения символов кавычек в строку. Как вы знаете, кавычки бывают одинарные (') и двойные ("). В JavaScript разрешается использовать и те, и другие, главное, чтобы в начале строки и ее конце использовались кавычки одного типа.

Заключив строковое значение в одинарные кавычки, вы сможете включать в нее двойные кавычки, не прибегая к использованию символов обратной косой черты. Если при этом необходимо включить в текст сообщения одинарные кавычки, то символ обратной косой черты перед ними вводится в обязательном порядке.

И наоборот, обозначив строку двойными кавычками, можно не вводить обратную косую черту перед одинарными кавычками в самом сообщении, чего не скажешь о двойных кавычках.

Пример одновременного использования в коде JavaScript кавычек обоих типов приведен в листинге 2.4. Обратите внимание на то, что в тексте сообщения, выводимом на экран, содержатся двойные кавычки.

Листинг 2.4. Совместное применение кавычек обоих типов

```
for (var i = 0; i < 300; i++) { document.write ('При встрече мы
   ↳говорим "Привет!"'); }
```

Ввод ключевых слов

В отличие от строк, весь остальной код записывается по строгим правилам. Код, расположенный вне одинарных или двойных кавычек, относится к ключевым словам JavaScript и определяет операции, выполняемые программой.

При вводе ключевых слов JavaScript очень важно не допускать опечаток и сохранять исходный регистр символов. Как несложно заметить, следующие выражения распознаются в JavaScript по-разному.

```
FOR
for
For
```

Только второе слово относится к ключевым словам JavaScript. Если вы попытаетесь ввести в коде программы остальные два варианта написания, то получите сообщение об ошибке, подобное показанному на рис. 2.4.

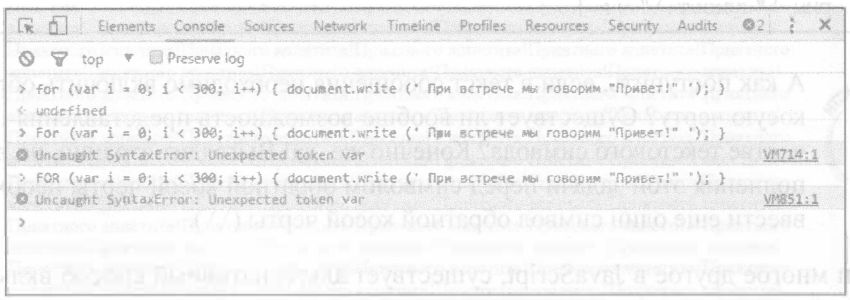


Рис. 2.4. Использование прописных символов во многих ключевых словах JavaScript приводит к возникновению ошибок

Подробно ключевое слово `for` и его назначение в коде JavaScript будут рассмотрены в главе 17.

При вводе кода JavaScript будьте предельно аккуратны. Во многих случаях синтаксические ошибки в программе вызваны опечатками в ключевых словах (чаще всего пропущена буква или буквы следуют в неправильном порядке).

Как и ошибки в школьных сочинениях, опечатки в коде JavaScript очень сложно отследить на этапе создания программы. Возьмите себе за правило никогда не торопиться при вводе ключевых слов. По окончании набора всегда проверяйте каждую инструкцию на наличие синтаксических ошибок. Тем самым вы сэкономите огромное количество времени и сил, которые еще понадобятся вам при поиске причин более серьезных неполадок.

Пробелы в коде

Давайте договоримся понимать под пробелами любые интервалы между словами и символами, в том числе отступы и разрывы строк. Как уже упоминалось, в коде JavaScript пробелы полностью игнорируются. В рассмотренном выше примере вывода сообщений в окне браузера использование отступов и пробелов в сочетании с разбивкой кода на отдельные строки позволяет лучше понять общую структуру программы (листинг 2.5).

Листинг 2.5. Пробелы и структурирование программы

```
for (var i = 0; i < 300; i++) {
   document.write ("Программировать весело!");
}
```

В листинге 2.5 показан только один из способов структурирования программы. Это предложенный нами вариант, не претендующий на абсолютную правильность.

Обратите внимание на то, что после открывающей фигурной скобки ({) добавлен разрыв строки. Еще один разрыв строки находится перед закрывающей фигурной скобкой (}). Обычно фигурные скобки используются для обособления некой части кода (группы инструкций) в отдельный *блок*. В нашем случае в таком блоке содержится только одна инструкция, отвечающая за вывод на экран повторяющегося 300 раз сообщения.

Фигурные скобки всегда указывают на место в коде, где не лишне добавить пробел, отступ или разрыв строки. Еще один символ, после которого гарантированно ставится разрыв строки, — это точка с запятой, обозначающая конец инструкции.

Если начать вводить показанный в листинге 2.5 код на консоли JavaScript и в конце первой строки (после открывающей фигурной кавычки) нажать клавишу <Return> (в Mac) или <Enter> (в Windows), то будет возвращена ошибка. А все потому, что нажатие <Return> (<Enter>) указывает выполнить введенный выше код. Разумеется, выполнение только первой строки программы не позволяет получить значимый результат. Чтобы вместо выполнения строки добавить в ее конец разрыв, нужно вместе с <Enter> нажимать клавишу <Shift>. Этой же комбинацией клавиш воспользуйтесь и в конце второй строки.

Строки внутри блока кода, заключенного в фигурные кавычки, вводятся с отступом. Отступами в коде обычно выделяются инструкции, выполняемые внутри других инструкций. Поскольку инструкция `for` образует циклическую структуру, отступами в нем "отбиваются" все вложенные (повторно выполняемые) инструкции.

Для добавления в начале строки отступа лучше всего использовать клавишу пробела. Стандартный отступ образуется при нажатии клавиши пробела два или четыре раза. Некоторые программисты предпочитают добавлять отступы с помощью клавиши <Tab>. Здесь каждый решает сам. Главное, чтобы отступы были одинаковыми во всей программе. Если вы с первых строк устанавливаете отступы двумя символами пробела, то не стоит дальше в коде переходить на использование отступов шириной в четыре пробела или один символ табуляции.

Комментарии

Комментарии в программе не относятся ни к инструкциям, ни к выводимому на экран тексту. Это может показаться странным, но зачастую комментарии играют в программе более значимую роль, чем весь остальной код. Комментарии не содержат инструкций, что не мешает им представлять особую важность.

Обычно программисты снабжают свой код комментариями в следующих случаях.

- ✓ Программа подлежит дальнейшей модификации или исправлению. Тот, кому предстоит заниматься этим в дальнейшем, будет бесконечно благодарен тому, кто снабдил все выполняемые программой действия подробным описанием.
- ✓ В коде программы задействованы специальные или нестандартные методики выполнения операций.
- ✓ Программу все еще можно улучшить, несмотря на то что она выполняет возложенные на нее задачи. В комментарии добавляются сведения о том, как лучше всего совершенствовать имеющийся код.
- ✓ Отдельные инструкции нужно временно заблокировать для выполнения.

Комментарии бывают двух типов: вводимые в одну или в несколько строк.

- ✓ **Однострочный комментарий.** Комментарием в коде объявляется сразу вся строка. Для создания однострочного комментария в начало строки помещаются два подряд символа косой черты (//). Например, в листинге 2.6 однострочными комментариями объявлены первые три строки программы. Выполняющая действие инструкция содержится в единственной, четвертой по счету, строке.

Листинг 2.6. Однострочные комментарии

```
// Следующая инструкция не выполняется
// alert("Поберегись!");
// Приведенный далее код выполняется
alert("Хорошего настроения!"); // приятное пожелание
```

- ✓ **Многострочный комментарий.** Многострочным такой комментарий называется потому, что располагается сразу в нескольких (иногда во многих) строках кода. На начало многострочного комментария указывает оператор /*. Завершается такой комментарий оператором */. Пример многострочного комментария показан в листинге 2.7.

Листинг 2.7. Многострочный комментарий

```
/*
   Программа AlertMe разработана Крисом Минником и Евой Холланд.

   Программа извещения пользователей об использовании
   ими языка программирования JavaScript называется AlertMe.
   Она разработана Крисом Минником и Евой Холланд.
*/
```

ГЛАВА 3

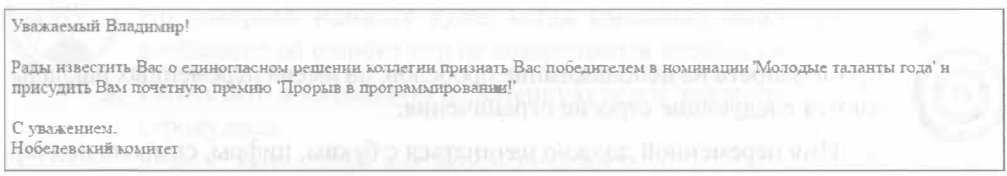
Отправка и получение данных

Программы бывают самые разные, как по размеру, так и по назначению. Но все они сходны в том, что обрабатывают информацию. Как правило, в программах JavaScript от вас потребуется обеспечить следующие три возможности:

- ✓ получение данных от пользователей;
- ✓ вывод информации пользователям;
- ✓ хранение и обработка данных в промежутке времени между получением и возвратом информации.

Данные, которые программа получает от пользователей, называются *входными*. Возвращаемые пользователю данные именуются *выходными*. Запросив у пользователей всю необходимую информацию, нужно продумать, как ее сохранить в программе. Только после этого данные обрабатываются и выводятся обратно пользователю. При этом нужно обязательно учитывать типы данных, с которыми приходится работать программе.

Вопрос о том, что проще, получать данные от пользователей или выводить их на экран, перед нами даже не стоит. Обе операции требуют приблизительно одинаковых усилий. В этой главе мы поговорим о том, как правильно в JavaScript обмениваться данными с пользователями и выполнять над данными необходимые действия.



Переменные

В повседневной жизни вещи хранятся на полках в кладовых аккуратно упакованными в коробки и корзины. В подобном виде их удобно перевозить в случае переезда на новое место.

Язык программирования не имеет никакого отношения к физическим вещам и не оперирует такими понятиями, как упаковка. Он обрабатывает только те данные, которые помещаются в специальные программные контейнеры. Для хранения и перемещения данных в программе используются элементы, называемые *переменными*. Переменной назначается имя, которое представляет в коде данные, содержащиеся в ней.

Используя переменные, в программе можно легко обработать разные наборы входных данных и получить отличные результаты при одном и том же коде JavaScript.

Создание переменной

Создать переменную в JavaScript очень просто: необходимо ввести ключевое слово `var` и указать имя переменной. Не забудьте в конце строки ввести точку с запятой.

```
var book;
```

Вам придется проявить смекалку, придумывая переменным названия. Это далеко не такое простое занятие, как кажется на первый взгляд. Включите фантазию, но не увлекайтесь слишком вычурными названиями. В первую очередь, имя должно отражать назначение переменной, и только потом свидетельствовать о творческой натуре ее создателя.

Приведенные ниже примеры объявления переменных в полной мере соответствуют требованиям, выдвигаемым к ним в большинстве программ. Одного взгляда на их названия достаточно, чтобы понять, для каких целей они применяются.

```
var myFirstName;
var favoriteFood;
var birthday;
var timeOfDay;
```

Обратите внимание на то, что пробелы в названиях переменных использовать недопустимо, поэтому для выделения в длинных именах отдельных слов применяются прописные буквы. Да, программистам приходится постоянно идти на ухищрения, чтобы обойти ограничения, принятые в выбранных ими языках программирования. Подобный способ именования переменных получил название *горбатый регистр* (несложно догадаться, почему).

Кроме запрета на использование пробелов, на имена переменных накладываются следующие строгие ограничения.

- ✓ Имя переменной должно начинаться с буквы, цифры, символа подчеркивания (_) или знака доллара ($).
- ✓ Название переменной может состоять только из букв, цифр, символов подчеркивания и знака доллара.

- ✓ Имена переменных чувствительны к регистру символов.
- ✓ В качестве переменных нельзя применять *зарезервированные* в JavaScript ключевые слова. Их не так много, но при создании в программе переменных о них нужно помнить (табл. 3.1).

Таблица 3.1. Зарезервированные слова языка программирования JavaScript

break	case	class	catch
const	continue	debugger	default
delete	do	else	export
extends	finally	for	function
if	import	in	instanceof
let	new	return	super
switch	this	throw	try
typeof	var	void	while
with	yield		

Сохранение данных в переменных

Сразу после создания переменную можно использовать по назначению — для хранения в ней информации. Поместив в переменную определенные данные, вы сможете обработать их в коде, обратившись к соответствующей переменной по имени. Вот как это делается.

1. Запустите браузер Chrome и отобразите в его окне консоль JavaScript.
2. Создайте новую переменную с именем `book`, введя на консоли следующую строку и нажав клавишу <Return> (Mac) или <Enter> (Windows).

   ```
   var book;
   ```

   Вы только что создали в коде контейнер или переменную с названием `book`.

   После нажатия клавиши <Return> или <Enter> возвращается неопределенное значение `undefined`. Так и должно быть. Код JavaScript выполняется, но в переменной не содержится никаких данных, поэтому на экран выводится несколько необычный результат.

   Может показаться странным, что результат выполнения кода не определен. Но поверьте, намного хуже, когда консолью JavaScript возвращается сообщение об ошибке или не возвращается вообще ничего.

3. Поместите в созданную переменную некое значение, введя следующую строку кода.

   ```
   book = "JavaScript для любознательных";
   ```

   Вы только что сохранили в переменной ее первое значение.

   После нажатия клавиши <Return> или <Enter> на консоль возвращается содержимое переменной.

При назначении переменной значения вводить ключевое слово `var` перед ее именем не нужно. Оно используется только единожды — при создании переменной. При последующем изменении значения переменной оно опускается.

4. Постарайтесь забыть о том, какие данные хранятся в переменной. Теперь представьте, что вам позарез нужно их вспомнить, поскольку вы обещали поделиться названием книги со своим другом. Чтобы извлечь данные из переменной или возвратить ее значение, достаточно ввести на консоли ее название.

   ```
   book
   ```

   На экране немедленно появится содержимое переменной, имя которой вы только что ввели, как показано на рис. 3.1.

Рис. 3.1. Вывод значения переменной на консоли JavaScript

Как вы успели заметить, после названия переменной не нужно вводить точку с запятой (`;`). Имя переменной не рассматривается как отдельная инструкция JavaScript, поэтому и не требует завершения специальным символом. Запрашивая имя переменной, вы выполняете действие, равнозначное обычной математической операции.

5. Для изменения значения переменной введите на консоли JavaScript следующую инструкцию.

   ```
   book = "JavaScript для профессионалов";
   ```

6. Введите на консоли слово **book** и нажмите <Return> или <Enter>.

   Возвратится новое значение переменной — `JavaScript для профессионалов`.

Кроме текста, в переменной допускается хранить данные некоторых других типов. В оставшейся части главы мы как раз и поговорим об основных (или простых) типах данных, используемых в JavaScript.

Данные, хранящиеся в переменной, чаще всего называют *значением* переменной.

Типы данных

Назначение переменных в JavaScript очень простое — хранить информацию и предоставлять ее по первому требованию программы. В создании переменных нет ничего сложного, поэтому с хранением в них данных у вас не должно возникать затруднений. В программировании выделяют большое количество типов данных: числовой, текстовый, дата-время и др. В JavaScript существуют определенные ограничения на использование данных разных типов, и вам, конечно же, нужно знать о них.

Тип данных указывает, как правильно интерпретировать каждое из обрабатываемых значений. Например, значение 03-02-2017 можно представить как дату (3 февраля 2017 года) или же как математическое выражение (из трех вычитается два, а затем из полученного результата вычитается 2017). Как видите, результат полностью зависит от применяемого к текущему значению типа данных.

В JavaScript распознаются три основных типа данных: строковый, числовой и булев.

Строковый тип данных

В данных строкового типа хранится текст. Об управлении текстовыми данными мы уже говорили в главе 2. Далее вы узнаете, какие еще операции применимы к тексту, кроме ввода и вывода на экран.

Весьма полезной оказывается возможность подсчета количества символов в текстовой строке. Эта операция выполняется с помощью ключевого слова `.length`, вводимого после строки или имени переменной, в которой хранится текстовая строка.

Например, чтобы определить длину строки, хранящейся в переменной `book`, введите на консоли JavaScript команду `book.length`. В качестве результата возвращается числовое значение, в точности соответствующее количеству символов в значении переменной (рис. 3.2).

Рис. 3.2. Определение длины строки

Каждая строка, даже пустая, имеет свою длину. Как легко догадаться, длина пустой строки равняется нулю. Поскольку длина является характеристикой строки, ее еще называют *свойством*.

В JavaScript термин *свойство* используется повсеместно. К свойствам относят любую характеристику, которая тем или иным образом описывает рассматриваемый элемент программы. В реальной жизни к свойствам относят, например, цвет (автомобиля) или имя (человека). В программировании у строки есть несколько свойств, среди которых и ее длина (`length`). Для определения длины строки используется следующее выражение.

```
"Текст строки".length
```

Оно подсчитывает количество символов в строке. В нашем случае их 11, но если ввести эту команду на консоли JavaScript, то будет возвращен несколько иной результат: `12`. Как же так?

Все дело в том, что в JavaScript пробел тоже считается символом, равно как и знаки пунктуации, а также цифры. Свойством `length` учитывается каждый введенный с клавиатуры символ.

Наряду со свойствами над строками можно выполнять определенные действия. В программировании операции, которые выполняются по отношению к чему-то или над чем-то, называются *методами*.

Чаще всего в коде JavaScript используется строковый метод `indexOf`. В его задачу входит просмотр содержимого строки и поиск в нем необходимого символа или группы символов. В качестве результата этот метод возвращает позицию, в которой располагается искомый символ (или символы). В следующей инструкции проводится поиск расположения в строке слова `текст`.

```
"Полный текст строки.".indexOf("текст");
```

После выполнения данной команды на консоли JavaScript возвращается результат `7`. Давайте несколько модифицируем инструкцию, изменив в ней искомый текст на `П`.

```
"Полный текст строки.".indexOf("П");
```

Будет возвращено значение `0`. Немного странно, не правда ли?

Мы плавно подошли к рассмотрению одной из главных концепций JavaScript — отсчета от нуля. В отличие от нас с вами, у JavaScript нет пальцев, поэтому счет в нем ведется, начиная с нуля, а не единицы. Вот почему в предыдущем примере, когда определяется положение в строке первого символа (`П`), возвращается значение `0`.

Если бы у JavaScript была своя спортивная команда, то первый ее игрок неизменно носил бы футболку с цифрой `0`.

Числовой тип данных

Следующий распространенный тип данных JavaScript, требующий подробного рассмотрения, — числовой. Числа бывают положительными и отрицательными, а также целыми и дробными. В отличие от текста (строк) они сохраняются в переменных без использования кавычек.

Диапазон положительных чисел, которые разрешается использовать в JavaScript, необычайно широкий — от запредельно маленьких до невероятно больших. Не хочется утомлять вас количеством нулей в значениях, определяющих границы возможного диапазона. Вам достаточно знать, что максимально допустимое числовое значение превышает количество звезд во Вселенной. Да что там! Оно гарантированно больше количества атомов во Вселенной. Таким образом, с помощью JavaScript можно решить абсолютно *любую* математическую задачу, которую только можно представить.

Остерегаться стоит не больших чисел, а комбинирования в одном выражении сразу нескольких типов данных, например текста и чисел.

Иногда JavaScript проявляет чудеса смекалки. Если отобразить в окне браузера консоль JavaScript и ввести на ней `"10" + 10`, то оба числа будут восприниматься как текст. Как результат, выполняется операция конкатенации строк, приводящая к выводу на экран текстового значения `1010`.

С другой стороны, если ввести на консоли код `10 * "10"`, то он будет распознано как математическое выражение, в котором запись `"10"` представляет число 10. Результатом выполнения такого кода будет число 100. Предположение оказывается вполне разумным, поскольку в JavaScript операция умножения текстовых строк попросту отсутствует.

Булев тип данных

Данные булева типа представляются одним из двух возможных значений: `true` (истина) и `false` (ложь).

Булевы значения вступают в силу при выполнении в JavaScript операций сравнения, о чем подробно рассказывается в части V. Например, если в коде проверяется равенство чисел 3 и 30, то программой возвращается булево значение `false`.

Название булева тип данных заимствовано от фамилии известного математика Джорджа Буля.

Давайте немного поэкспериментируем с булевыми значениями. Запустите в браузере Chrome консоль JavaScript и поочередно введите приведенные ниже выражения, не забыв в конце каждой строки нажимать клавишу <Return> или <Enter>. Как видите, мы снабдили каждое выражение однострочными комментариями, чтобы помочь вам разобраться в их действиях. Вам не нужно вводить эти комментарии на консоли JavaScript — на конечный результат они никак не влияют.

```
1 < 10 // 1 меньше, чем 10?
100 > 2000 // 100 больше, чем 2000?
2 === 2 // 2 равно 2?
false === false // false равно false?
40 >= 40 // 40 больше или равно 40?
Boolean (0) // Каково булево значение для 0?
```

```
Boolean (false) // Каково булево значение для false?
"apples" === "oranges" // "apples" равно "oranges"?
"apples" === "apples" // "apples" равно "apples"?
```

Кроме выражений, возвращающих значение `false`, в JavaScript в качестве `false` допустимо использовать следующие ключевые слова.

`0`
`null`
`undefined`
`""` (пустая строка)
`false`

Запрос данных у пользователя

Теперь, когда вы знаете, что для хранения значений самых разных типов в JavaScript применяются переменные, нужно определиться со способами получения данных от пользователей.

Самый простой способ запроса данных у пользователя заключается в использовании команды `prompt`. Чтобы изучить, как она работает, запустите в браузере Chrome консоль JavaScript и введите следующую инструкцию.

```
prompt("Как тебя зовут, мой друг?");
```

После нажатия клавиши <Return> или <Enter> на экране появится всплывающее сообщение, содержащее всего одно текстовое поле, как показано на рис. 3.3.

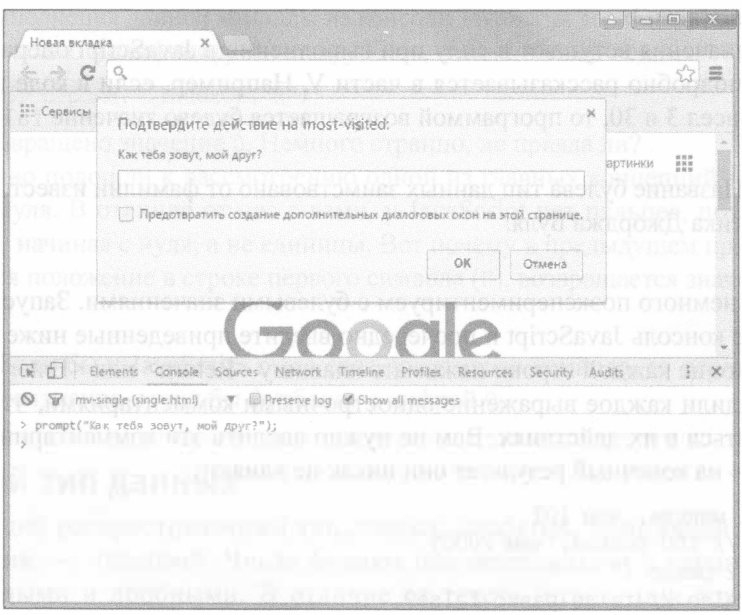


Рис. 3.3. Запрос данных у пользователя

Введите в нем свое имя и щелкните на кнопе OK. Всплывающее сообщение исчезнет, а введенный в нем текст немедленно отобразится на консоли JavaScript (рис. 3.4).

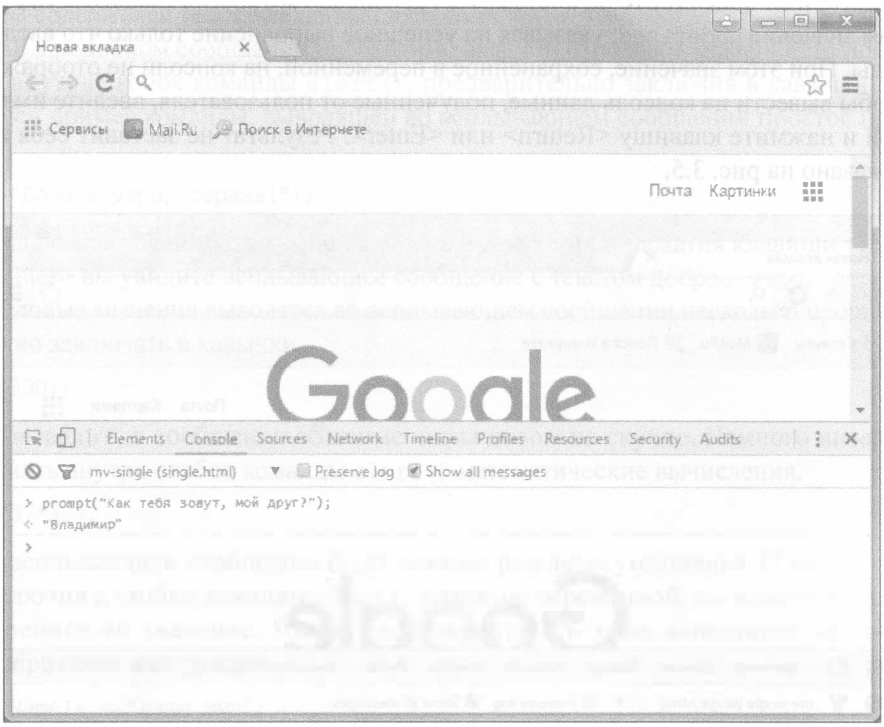


Рис. 3.4. Вывод на консоли введенного ранее имени

Такой подход полностью оправдан, когда требуется сразу же вывести на экран полученные от пользователя сведения. Как же поступить, если запрошенные данные необходимо в дальнейшем обработать специальным образом? Конечно же, сохранить их в переменной!

Сохранение данных, полученных от пользователя

Чтобы сохранить данные в переменной, ее необходимо сначала создать с помощью ключевого слова `var`. Для одновременного назначения переменной значения, полученного от пользователя, введите после имени переменной знак равенства (=), а после него — команду `prompt`.

```
var username = prompt("Как тебя зовут, мой друг?");
```

В JavaScript знак равенства (=) известен как оператор присвоения значения. С его помощью переменной, указанной в левой части выражения, присваивается значение, введенное справа от знака равенства. Подробно этот и другие важные операторы JavaScript будут рассмотрены в главе 9.

После нажатия клавиши <Return> или <Enter> вы снова увидите на экране всплывающее сообщение, в котором запрашивается ваше имя.

Введите имя и щелкните на кнопке OK. На консоли JavaScript появится уже знакомое вам значение `undefined`, указывая на успешное выполнение только что введенной команды. При этом значение, сохраненное в переменной, на консоли не отображается.

Чтобы вывести на консоль данные, полученные от пользователя, введите имя переменной и нажмите клавишу <Return> или <Enter>. Результат не заставит себя ждать, как показано на рис. 3.5.

Рис. 3.5. Вывод значения переменной, в которой хранятся запрошенные у пользователя сведения

Ответ на запрос

Вы уже знаете, как получать данные от пользователя и сохранять их в переменной. Далее давайте посмотрим, как с помощью JavaScript решается не менее актуальная задача: взаимодействие с пользователем.

Команда alert()

С помощью команды alert() в браузере выводится всплывающее сообщение, в котором содержится текст, введенный в скобках.

Если в подобном сообщении необходимо вывести строковое значение, то поместите его внутри скобок команды alert(), предварительно заключив в кавычки. Ниже приведен пример команды, выводящей во всплывающем сообщении простое приветствие.

```
alert("Доброе утро, страна!");
```

После ввода такой инструкции на консоли JavaScript и нажатия клавиши <Return> или <Enter> вы увидите всплывающее сообщение с текстом Доброе утро, страна!.

Числовые значения выводятся во всплывающем сообщении несколько проще — их не нужно заключать в кавычки.

```
alert(300);
```

Но выводить в сообщении обычные числа довольно скучно. Намного интереснее выполнять внутри скобок команды alert() математические вычисления.

```
alert(37*37);
```

Во всплывающем сообщении будет показан результат умножения 37 на 37.

Заключив в скобки команды alert() название переменной, вы выведете на экран присвоенное ей значение. Чтобы удостовериться в этом, выполните на консоли JavaScript такие инструкции.

```
var myNameIs = "ваше имя";
alert(myNameIs);
```

Во всплывающем сообщении будет указано имя, введенное вами в первой инструкции.

В команде alert допустимо комбинировать несколько типов данных. Это открывает перед вами новые, весьма интересные возможности. Например, введите и последовательно выполните на консоли JavaScript каждую из следующих инструкций.

```
var firstName = "ваше имя";
var yourScore = 30;
alert("Привет, " + firstName + ". Ваш заработок: " + yourScore);
```

Вы легко удостоверитесь, что с помощью команды alert() можно выводить на экран самые необычные извещения. Их содержимое зависит только от вашей фантазии и чувства юмора (рис. 3.6).

Метод document.write()

С точки зрения JavaScript *документом* считается вся веб-страница. Поэтому для изменения чего-либо на веб-странице с помощью кода JavaScript вам необходимо обратиться к объекту document.

Один из способов изменения текущей веб-страницы заключается в использовании метода write().

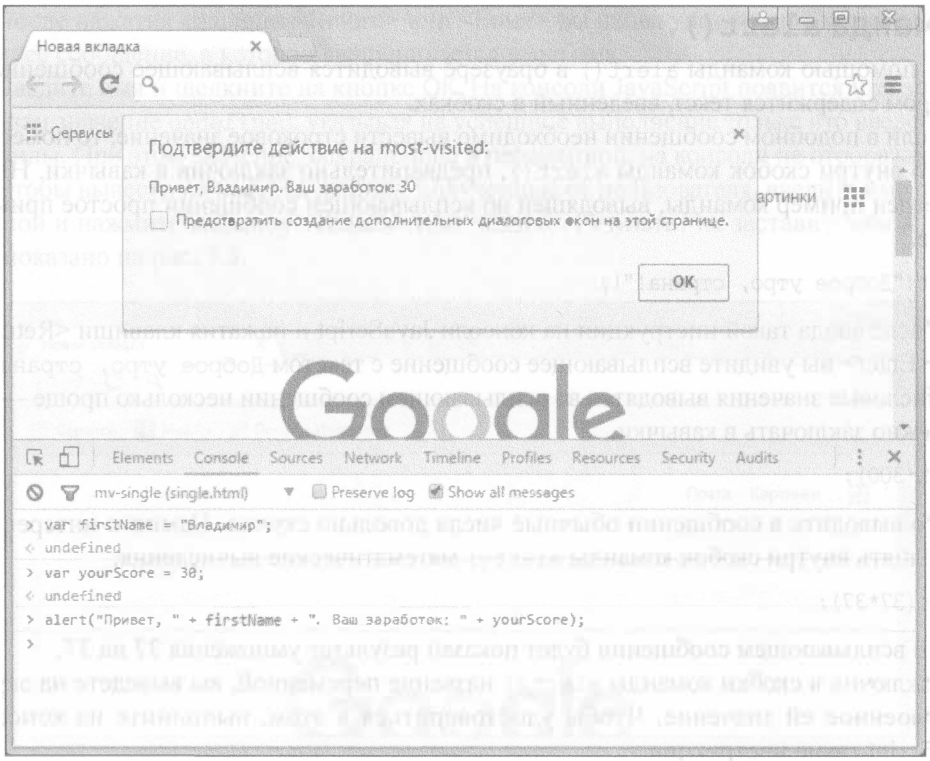


Рис. 3.6. Сообщения должны быть понятными пользователю

Тайная жизнь объектов

Наряду со строками и числами в JavaScript поддерживается специальный тип данных, известный как объекты. Он настолько функциональный, что позволяет хранить любые данные, которые представляются отдельными свойствами, а обрабатываются специально заданными методами.

Во многом объекты JavaScript подобны объектам, окружающим нас в повседневном мире. Например, у вас в гараже может стоять желтый трейлер. В JavaScript ему можно сопоставить объект `truck` со свойством `color`, установленным равным `yellow`.

```
truck.color="yellow";
```

Данный объект может обладать методом `drive`, который вызывается следующим образом.

```
truck.drive();
```

Под методом подразумевается команда, изменяющая некий объект или вносящая изменения в определенные данные.

Каждый документ (веб-страница) поддерживает использование метода `write()`, позволяющего добавлять в него все, что вводится в круглых скобках после имени метода. Технически метод `write()` выполняется так же, как и команда `alert()`. Чтобы познакомиться с результатом его воздействия на веб-страницу, откройте новое окно браузера и последовательно введите на консоли JavaScript следующие инструкции.

```
document.write("Привет, страна!");
document.write(333 + 100);
```

При такой записи команд между выводимыми на веб-странице сообщениями не добавляется ни разрыв строки, ни пробел. Это затрудняет их восприятие, поэтому в конец метода `document.write()` нужно добавить ключевое слово `<br>`, устраняющее этот недостаток.

```
document.write("Как жизнь?<br>");
document.write("Замечательно! Не перестает радовать!<br>");
document.write("Рад за тебя!");
```

Чтобы очистить содержимое текущего окна браузера, введите в его строке адреса **`chrome://newtab`** или просто откройте новую вкладку.

Результат выполнения последних трех строк кода на консоли JavaScript показан на рис. 3.7.

Ключевое слово `<br>` относится к языку HTML и называется тегом. Изучать HTML мы будем в главе 5.

Обработка входных и выходных данных

Самое время, используя полученные навыки, совместить в одном коде обработку входных данных и вывод на веб-страницу полезных сведений, основанных на полученной информации. В этом JavaScript нет равных!

Следуя приведенным ниже инструкциям, напишите себе письмо в окне браузера, воспользовавшись консолью JavaScript. Не забывайте в конце каждой команды (после ввода точки с запятой) нажимать клавишу <Return> или <Enter>.

1. Создайте с помощью следующей инструкции переменную, хранящую ваше имя.

   ```
   var toName = "ваше имя";
   ```

2. Создайте переменную, указывающую отправителя письма, воспользовавшись такой инструкцией.

   ```
   var fromName = "Нобелевский комитет";
   ```

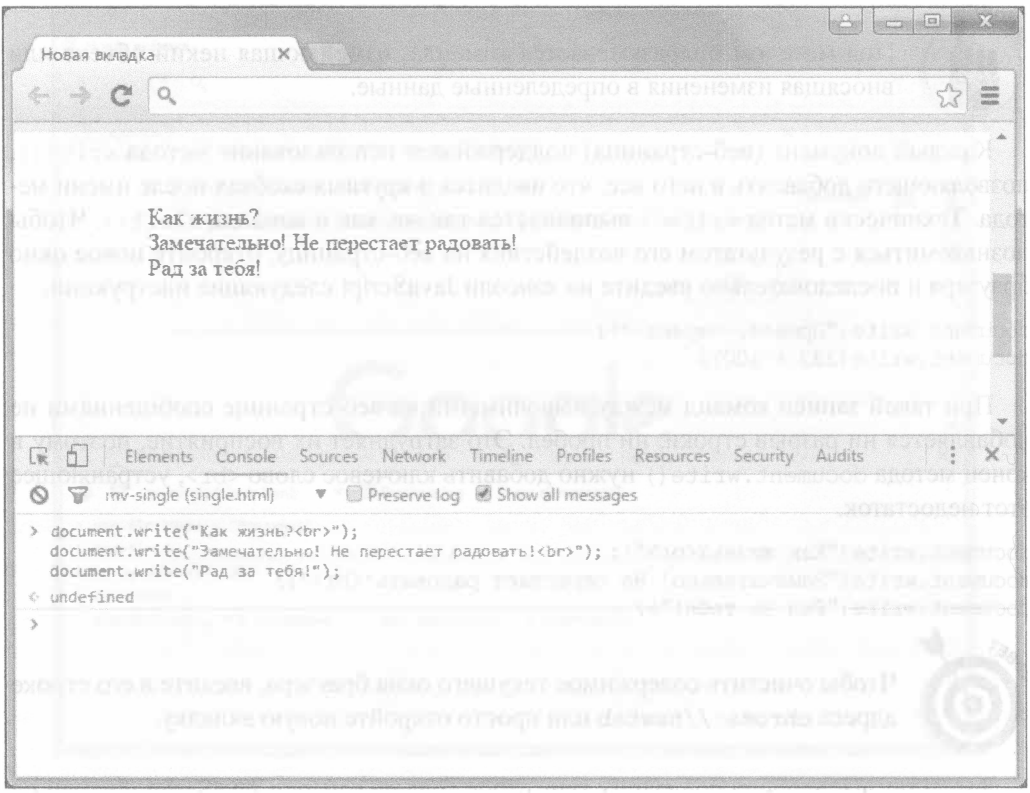


Рис. 3.7. Три сообщения в окне браузера

Вместо `Нобелевский комитет` можно ввести название любого другого отправителя на свое усмотрение.

3. Введите основной текст (содержимое) письма и сохраните его в отдельной переменной.

Не забывайте применять теги `<br>` для добавления разрывов в конце каждой строки создаваемого сообщения. Клавишу <Return> или <Enter> нажимайте только после ввода точки с запятой.

Не стоит сильно увлекаться текстом письма, ведь оно создается исключительно с обучающей целью.

```
var letterBody = "Рады известить Вас о единогласном решении
коллегии признать Вас победителем в номинации 'Молодые таланты
года' и присудить Вам почетную премию 'Прорыв
в программировании!'"
```

4. Воспользовавшись методом `document.write()`, выведите все части письма в окне браузера.

```
document.write("Уважаемый " + toName + "!<br><br>");
document.write(letterBody + "<br><br>");
document.write("С уважением,<br>");
document.write(fromName);
```

Конечный вид вашего письма должен быть таким, как на рис. 3.8.

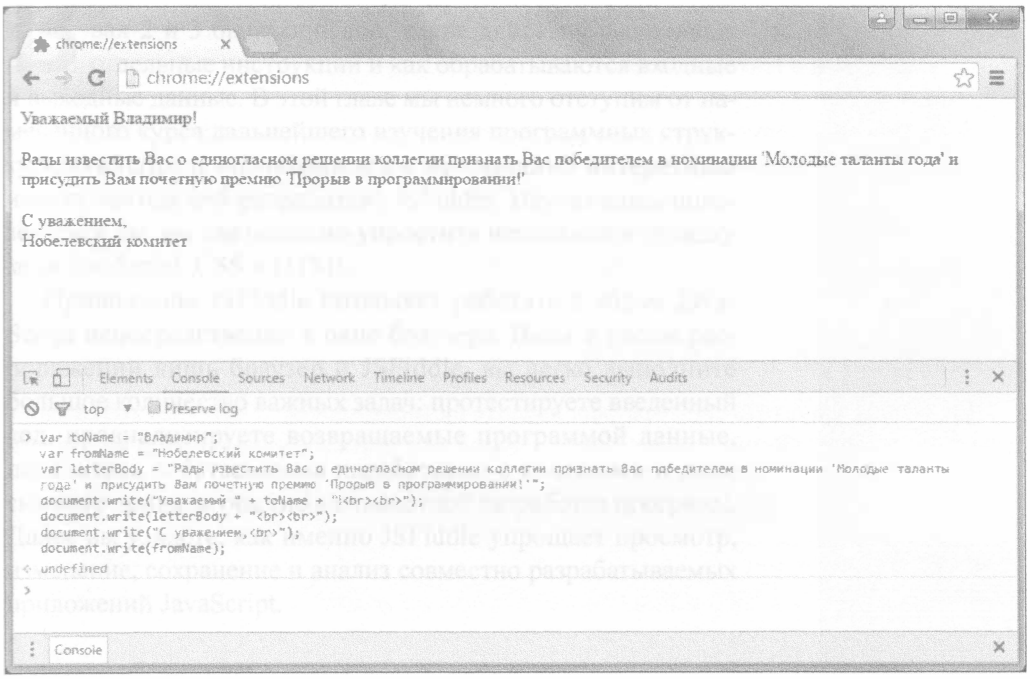


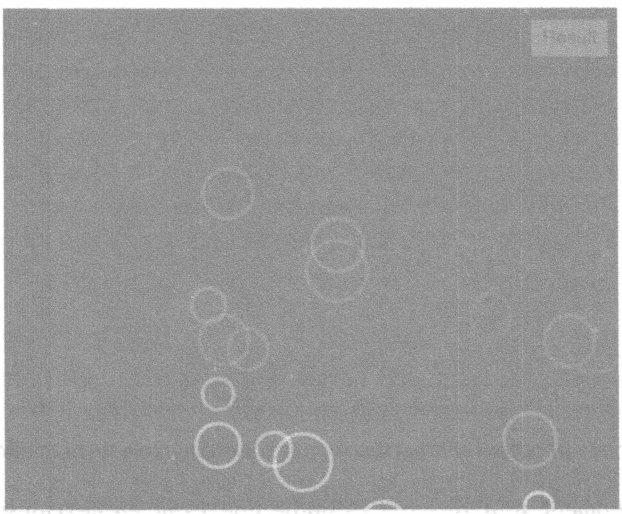
Рис. 3.8. Хвалебное письмо самому себе в окне браузера

ГЛАВА 4

Разработка веб-приложений

В главе 1 вы познакомились с консолью JavaScript. В главах 2 и 3 было описано, какие действия выполняют отдельные инструкции и как обрабатываются входные и выходные данные. В этой главе мы немного отступим от намеченного курса дальнейшего изучения программных структур JavaScript и познакомимся с необычайно интересным инструментом веб-разработки: JSFiddle. Научившись пользоваться им, вы значительно упростите написание и отладку кода JavaScript, CSS и HTML.

Приложение JSFiddle позволяет работать с кодом JavaScript непосредственно в окне браузера. Имея в своем распоряжении лишь браузер и JSFiddle, вы легко выполните большое количество важных задач: протестируете введенный код, проанализируете возвращаемые программой данные, поделитесь собственными наработками с коллегами и даже сможете принять участие в совместной разработке программ. Далее вы узнаете, как именно JSFiddle упрощает просмотр, изменение, сохранение и анализ совместно разрабатываемых приложений JavaScript.



До настоящего момента мы оперировали понятием *веб-приложение*. Под этим незамысловатым термином скрывается программа, выполняемая в браузере и, как правило, содержащая JavaScript-код. К веб-приложениям относится, например Google Earth, которое одновременно является и веб-сайтом. В его задачу как приложения входит демонстрация фотографий выбранных пользователем регионов земного шара, снятых в высоком разрешении. Но поскольку вы получаете доступ к этому приложению, вводя в браузере URL-адрес, то можете смело утверждать, что Google Earth одновременно играет роль сайта в Интернете. Заметьте, что далеко не все веб-приложения являются сайтами.

Итак, ближе к делу! Давайте не медля приступим к использованию приложения JSFiddle для создания занимательных анимационных эффектов. К концу изучения его основных возможностей вы научитесь настраивать анимацию пузырьков самым детальным образом. Название приложения — это комбинация аббревиатуры "JS" (JavaScript) и английского слова "fiddle" (играть).

Знакомство с JSFiddle

Для начала запустите свой браузер, введите в строке адреса `http://jsfiddle.net` и нажмите клавишу <Return> или <Enter>. Общий вид сайта JSFiddle показан на рис. 4.1.

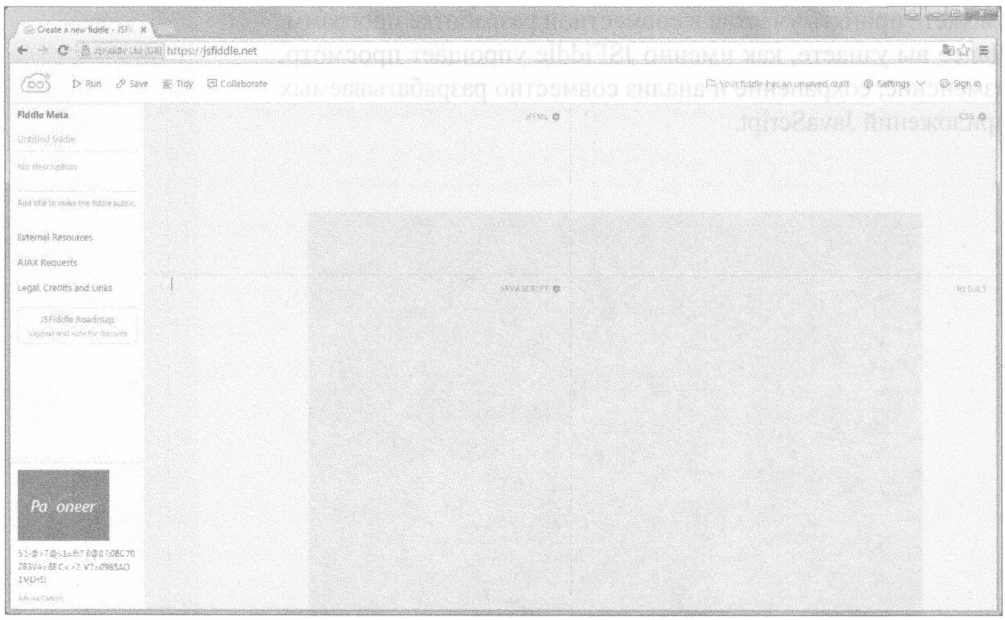


Рис. 4.1. JSFiddle обладает простым и понятным пользовательским интерфейсом

Пользовательский интерфейс приложения JSFiddle представлен четырьмя панелями. Три из них предназначены для ввода анализируемого кода JavaScript, CSS и

HTML. Четвертая панель отводится для вывода результатов выполнения введенного на остальных трех панелях кода. В левой части окна находится панель инструментов, позволяющая выполнять вспомогательные команды. Панель, расположенная вдоль верхнего края окна, содержит кнопки выполнения, сохранения и очистки введенного в центральной области кода.

Для изменения размеров любой из областей окна JSFiddle перетащите ее границу с помощью мыши.

На данный момент нас больше всего интересует панель JavaScript. Во многом она подобна уже знакомой вам консоли JavaScript, встроенной в браузер Google Chrome. Но между ними есть важное отличие: код, введенный на панели JavaScript, не выполняется до тех пор, пока вы не укажете сделать это специально.

Для выполнения программы JavaScript в приложении JSFiddle следуйте таким инструкциям.

1. Щелкните на панели JavaScript.
2. Введите следующую команду.
   ```
   alert("Привет, страна!");
   ```
3. Щелкните на кнопке Run (Выполнить), расположенной в верхней части окна.
4. На экране появится всплывающее сообщение приложения JSFiddle, содержащее текст `Привет, страна!`.
5. Закройте окно сообщения, щелкнув в нем на кнопке OK.

В поведении введенной выше однострочной программы JavaScript нет ничего необычного. Если вы знакомы с материалом первых трех глав, то знаете, чего ожидать от команды `alert()`.

К преимуществам приложения JSFiddle относится простота тестирования не только JavaScript-кода. С его помощью легко выполняется код CSS и HTML, вводимый на соседних панелях, а также анализируется взаимодействие всех трех технологий в рамках одной программы. В следующих нескольких разделах вы детально познакомитесь с функциональными возможностями всех панелей JSFiddle, выполнив на них несколько простых примеров. Но сначала давайте рассмотрим общие задачи, решаемые с ее использованием.

Просмотр кода

Настало время открыть все карты! Все программы, рассматриваемые в книге, доступны для просмотра, выполнения, копирования и анализа по такому адресу:

```
http://jsfiddle.net/user/forkids/fiddles
```

Немного похоже на домашнее задание, если вы еще помните, что это такое! Все приведенные на указанном сайте примеры полностью отлажены и готовы к использованию.

По указанному адресу содержится ваш рабочий кабинет. Он общедоступный, поэтому пользоваться программами, приведенными в нем, разрешено всем, кто только захочет.

Несмотря на то что коды всех рассматриваемых в книге проектов уже введены нами и сохранены на сайте `http://jsfiddle.net`, нам бы очень хотелось, чтобы вы самостоятельно воспроизвели этапы их разработки. Научиться писать программы можно, только вводя их код вручную. Конечно, при изучении материала книги вы вольны свободно использовать код всех приведенных в ней примеров: копировать, изменять, переписывать и даже использовать в собственных разработках. Включите воображение!

Готовые примеры

Перед тем как приступить к детальному изучению сложных проектов, рассматриваемых в книге, давайте познакомимся с примерами, которые не вошли в их число. Откуда они взялись? На сайте JSFiddle любой желающий может завести учетную запись и выложить на всеобщее обозрение проекты собственного написания. Это путь истинных программистов!

Выкладывая свои проекты на сайте JSFiddle, вы соглашаетесь с тем, что каждый его посетитель может использовать их по своему усмотрению, в том числе создавать на их основе собственные программы. Тем не менее считается неприличным пользоваться чужими наработками, не поставив в известность их автора. Упоминаемые далее проекты полностью скопированы и приведены в неизменном виде, чтобы вы могли познакомиться с их кодом, представленным в авторском исполнении. Чтобы узнать имя автора каждой из программ, щелкните на левой панели на команде Fiddle Options (Параметры примера) для отображения общих сведений.

Для знакомства с программами, включенными в список наших любимых проектов JSFiddle, выполните следующие действия.

1. Отобразите в браузере страницу `http://jsfiddle.net/user/forkids/fiddles`.

На ней приведен список всех проектов, рассматриваемых в последующих главах книги.

Полный список состоит из нескольких страниц, навигация по которым осуществляется с помощью соответствующих ссылок в нижней части страницы.

2. Найдите в списке интересующий вас пример.

После открытия программа запускается автоматически.

Ознакомившись с действиями, выполняемыми интересующей вас программой, попробуйте изменить в ней отдельные значения и операторы, а затем запустите заново. Отследите изменения в результатах.

Любые изменения, вносимые в примеры, которые взяты с сайта JSFiddle, не заменяют код исходного проекта. Поэтому чувствуйте себя свободно, редактируя чужие решения. Самое худшее, чего вы можете добиться своими действиями, — это изменить программу так, что она перестанет выполняться в вашем браузере.

Код CSS

Панель ввода CSS-кода находится в правом верхнем углу окна приложения JSFiddle. Наряду с программами JavaScript, приложение JSFiddle обеспечивает обработку кода CSS (Cascading Style Sheets — каскадные таблицы стилей). По большей части технология CSS применяется для настройки внешнего вида текстовых и графических элементов веб-страницы. К ней вы обращаетесь, например, когда необходимо задать цвет текста, отличный от черного.

Детально возможности языка CSS рассматриваются в главе 6. На данный момент просто попробуйте изменить одну из программ, написанных с его использованием.

1. Перейдите в браузере на страницу `http://jsfiddle.net/forkids/vaj023L5`.

   Загрузится демонстрационное приложение, воспроизводящее анимацию всплывающих пузырьков (рис. 4.2).

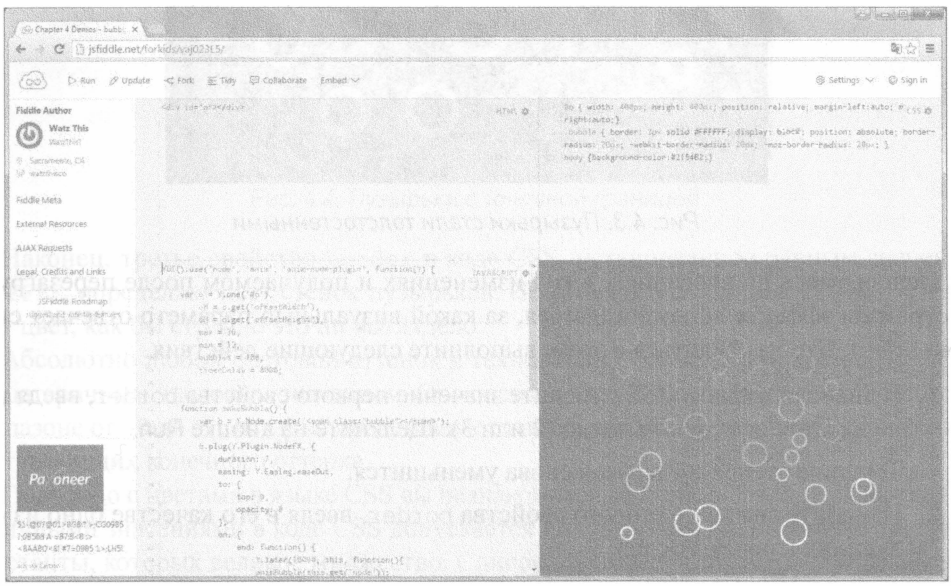


Рис. 4.2. Демонстрация забавной анимации

2. Внимательно изучите все четыре панели окна.

   В трех из них содержится код приложения, а в четвертом отображается анимация. Сможете догадаться, какой код за что отвечает?

3. Взгляните на правую верхнюю панель (панель кода CSS).

   На ней содержится всего три строки кода.

4. Отыщите код, содержащий ключевые слова `border: 3px solid #FFFFFF;`, и замените его кодом `border: 8px solid #FFFFFF;`.

5. Щелкните на кнопке Run, чтобы перезагрузить анимацию в окне браузера.

   Стенки пузырьков стали заметно толще, как показано на рис. 4.3.

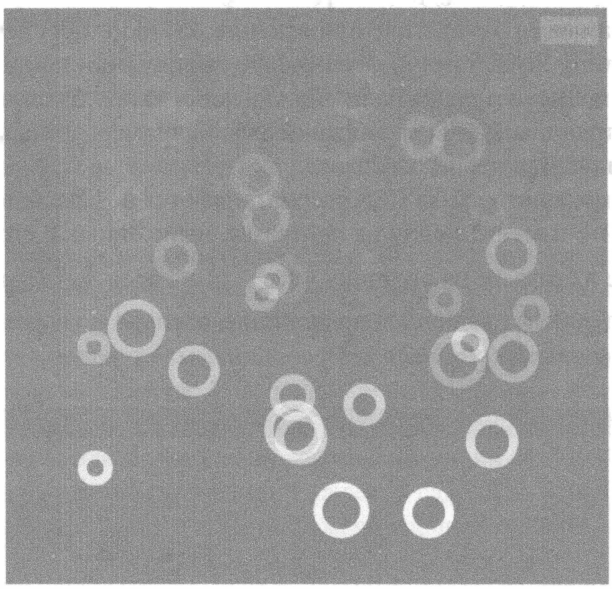


Рис. 4.3. Пузырьки стали толстостенными

Основываясь на внесенных в код изменениях и получаемом после перезагрузки программы эффекте легко догадаться, за какой визуальный параметр отвечает свойство `solid`. Чтобы убедиться в этом, выполните следующие действия.

1. На панели с кодом CSS измените значение первого свойства `border`, введя после двоеточия меньшее число (2 или 3). Щелкните на кнопке Run.

   Толщина стенок пузырьков снова уменьшится.

2. Измените значение второго свойства `border`, введя в его качестве одно из следующих ключевых слов.

   - `dotted`
   - `dashed`

- `double`
- `groove`
- `ridge`
- `inset`
- `outset`

3. Щелкните на кнопке Run, чтобы познакомиться с результатом внесенных изменений.

   Воспользовавшись одной из именованных границ, вы измените стиль стенок пузырьков. Пример пузырьков с типом стенок `dotted` показан на рис. 4.4.

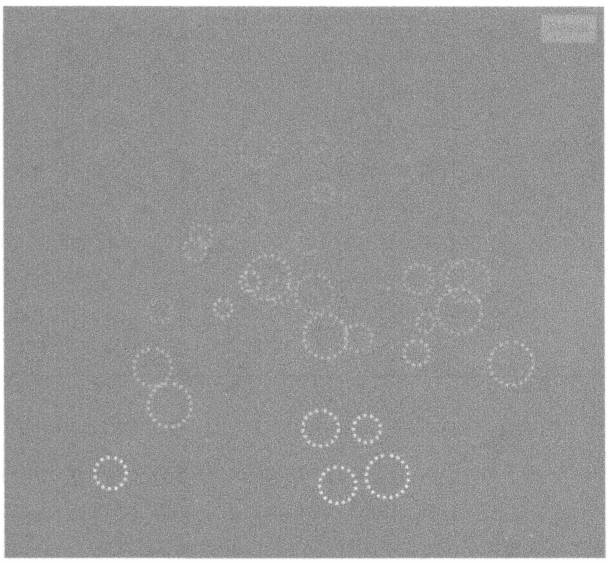


Рис. 4.4. Пузырьки с точечной границей

Наконец, третье свойство `border` в коде CSS, установленное равным значению `#FFFFFF`, определяет цвет стенок пузырьков. Не удивляйтесь, запись `#FFFFFF` указывает цвет, как бы странно это ни выглядело.

Абсолютно любой цветовой оттенок в технологии CSS можно представить шестнадцатеричным числовым значением. Оно состоит из трех пар чисел, изменяющихся в диапазоне от `00` до `FF`, которые представляют насыщенность красной, зеленой и синей составляющих конечного оттенка.

Подробно с цветами в языке CSS вы познакомитесь в главе 6. Наряду с шестнадцатеричными значениями в коде CSS допускается использовать именованные цветовые константы, которых великое множество. Список наиболее распространенных именованных цветов и их шестнадцатеричные аналоги приведены в табл. 4.1.

Таблица 4.1. Стандартные цвета HTML

Английское название	Русское название	Шестнадцатеричное числовое значение
Aqua	Циан	#00FFFF
Black	Черный	#000000
Blue	Синий	#0000FF
Fuchsia	Фуксия	#FF00FF
Gray	Серый	#808080
Green	Зеленый	#008000
Lime	Лайм	#00FF00
Maroon	Коричнево-малиновый	#800000
Navy	Темно-синий	#000080
Olive	Оливковый	#808000
Orange	Оранжевый	#FFA500
Purple	Пурпурный	#800080
Red	Красный	#FF0000
Silver	Серебристый	#C0C0C0
Teal	Сине-зеленый	#008080
White	Белый	#FFFFFF
Yellow	Желтый	#FFFF00

Чтобы изменить цвет пузырьков, выполните следующие действия.

1. Подберите цвет по имени или шестнадцатеричному значению согласно табл. 4.1.
2. На панели кода CSS замените новым цветом текущее цветовое значение (`#FFFFFF`).
3. Щелкните на кнопке Run.

   Удостоверьтесь, что цвет пузырьков изменился.

Код HTML

Теперь рассмотрим код HTML. Он приводится в левом верхнем углу окна браузера. Он намного проще кодов JavaScript и CSS. Да что там говорить, в нашем примере его почти нет!

Язык HTML, подробно о котором мы расскажем в главе 5, отвечает за создание макета веб-страницы и контейнеров для программ JavaScript, из которых они запускаются. В рассматриваемом выше примере анимации пузырьков задача HTML сводится к выделению места, на котором отображается наблюдаемый пользователем эффект.

Не стоит думать, что сам по себе код HTML малофункционален. Чтобы разубедиться в этом, давайте внесем в него некоторые изменения и понаблюдаем за получаемым результатом.

1. Расположите курсор после ключевого слова `</div>` и введите следующее выражение.

   ```
   <h1>I love bubbles!</h1>
   ```

   Код на панели HTML при этом должен выглядеть следующим образом.

   ```
   <div id="o"></div><h1>I love bubbles!</h1>
   ```

   По правде говоря, HTML относят не к языкам программирования, а языкам разметки документа. Причина рассмотрена в главе 5.

2. Щелкните на кнопке Run, чтобы ознакомиться с изменениями в получаемом эффекте.

   Как видно из рис. 4.5, теперь область анимации снабжается подписью.

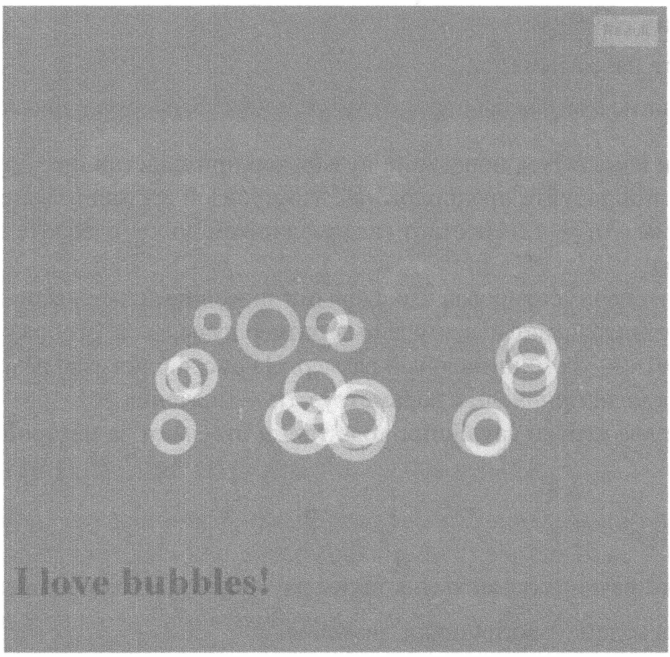


Рис. 4.5. Цветные пузырьки теперь не одиноки — под ними находится подпись

В языке HTML ключевые слова `<h1>` и `</h1>` называются тегами. В теги заключаются текст и другие элементы, которым нужно придать должное форматирование. Браузер распознает теги и обрабатывает необходимым образом помеченное ими содержимое. В нашем случае тег `<h1>` представляет заголовок первого уровня, в котором размер символов наибольший из всех возможных.

Еще один часто используемый в HTML тег — `<p>`. С его помощью в документе размечаются абзацы. Чтобы добавить этот тег на веб-страницу, выполните следующие действия.

1. Установите курсор после тега </h1> и нажмите клавишу <Return> или <Enter>, чтобы образовать новую строку кода.
2. Введите ключевое слово , а затем произвольный текст абзаца. В конце текста абзаца не забудьте добавить закрывающий тег .
3. Щелкните на кнопке Run, чтобы увидеть эффект от внесенных в код изменений.

Код JavaScript

Панель JavaScript находится в левой нижней части окна приложения JSFiddle. На ней вводится самый интересный код, отвечающий за анимацию пузырьков. Давайте попробуем его протестировать.

1. Перейдите на панель JavaScript, найдите строку, содержащую запись `max = 36`, и измените ее на `max = 80`.
2. Щелкните на кнопке Run.

   Очень многие пузырьки, но далеко не все, существенно увеличиваются в размерах.

Заметив, как изменяется поведение пузырьков при подстановке другого значения параметра `max`, попробуйте предположить, что произойдет при изменении числового значения в строке `min = 12`. Не стоит гадать слишком долго, сделайте это и протестируйте новый код!

Немного поэкспериментировав с числовыми значениями, несложно догадаться, что параметр `max` устанавливает максимальный размер пузырьков, отображаемых на экране, а параметр `min` — их минимальный размер. На рис. 4.6 показан результат присвоения параметру `max` значения `80`, а параметру `min` — значения `20`.

Следующие две строки JavaScript-кода также отвечают за настройку параметров анимации.

```
bubbles = 100
timerDelay = 8000
```

Их числовые значения изменяются таким же образом, как и раньше.

1. Введите в коде JavaScript новое значение.
2. Щелкните на кнопке Run для ознакомления с изменениями в анимации.

Если изменять эти параметры не одновременно, а по очереди (или попробовать угадать, от каких слов происходят их названия), то легко выяснить, что параметр `bubbles` указывает количество создаваемых пузырьков, а настройка `timerDelay` определяет скорость их появления на экране.

Попробуйте экспериментальным путем определить оптимальные значения для настроек `bubbles` и `timerDelay`. ***Подсказка***: значение `timerDelay` указывается в миллисекундах (тысячных долях секунды). Таким образом, 8 000 миллисекунд равняются 8 секундам. Измените значение, заданное по умолчанию, на число 10 000; щелкните на кнопке Run и понаблюдайте за полученным эффектом. Теперь замените текущее значение числом 1000, снова щелкните на кнопке Run и оцените разницу в полученной анимации.

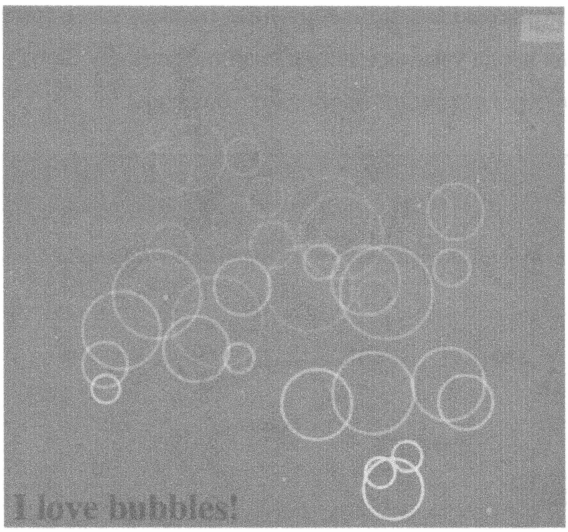


Рис. 4.6. Изменение параметров `max` и `min` приводит к перераспределению количества больших и маленьких пузырьков на экране

Наше предположение о том, что параметр `timerDelay` контролирует скорость анимации, полностью оправдалось. Поэкспериментировав описанным выше способом с параметром `bubbles`, вы убедитесь, что эта настройка определяет общее количество создаваемых пузырьков.

Регистрация в JSFiddle

Для успешного изучения всех примеров, рассматриваемых в книге, регистрироваться в приложении JSFiddle совсем необязательно. Но если вы планируете в дальнейшем использовать его для просмотра чужих проектов и предоставления на всеобщее рассмотрение собственных программ, то без учетной записи вам точно не обойтись.

Чтобы создать в JSFiddle собственную учетную запись, выполните следующие действия.

1. При открытом в приложении проекте щелкните на кнопке **Fork** (Заимствовать), расположенной в верхней части окна.

   Использование в собственных проектах кода, написанного другими программистами, называется *заимствованием*.

2. Скопируйте URL-адрес текущего проекта, отображаемый в строке адреса браузера, или выпишите его на отдельном листе, чтобы после регистрации в JSFiddle получить возможность его заимствовать.

3. В правом верхнем углу окна щелкните на ссылке **Login/Sign** (Войти/Зарегистрироваться).

Появится страница ввода учетных данных пользователя (рис. 4.7).

4. Под полями для ввода учетных данных найдите ссылку Sign in и щелкните на ней.

   Появится форма регистрации нового пользователя.

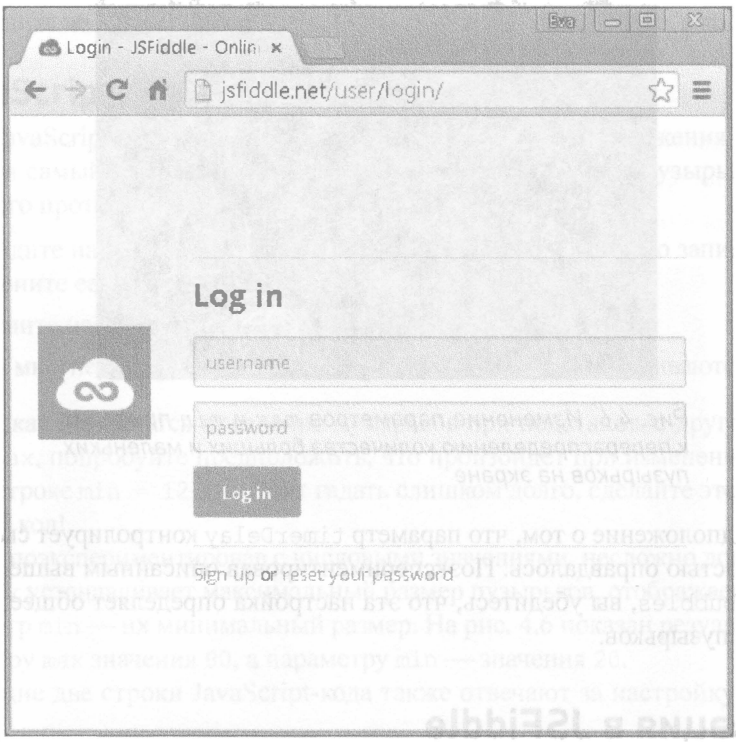


Рис. 4.7. Страница регистрации в JSFiddle

5. Заполните все обязательные поля формы и щелкните на кнопке Create an account (Создать учетную запись).

   Отобразится страница настройки профиля новой учетной записи, показанная на рис. 4.8. Просмотрите ее и при необходимости введите дополнительные сведения о себе. Это не обязательное требование; им можно пренебречь. Сохраните профиль пользователя.

6. На верхней панели щелкните на ссылке Editor (Редактор).

   Появится основное окно приложения JSFiddle с именем вашей учетной записи в правом верхнем углу.

7. Вставьте или введите скопированный в п. 2 адрес URL в строку адреса браузера, после чего нажмите клавишу <Return> или <Enter>.

   Мы вернулись к проекту анимации пузырьков.

8. Щелкните еще раз на кнопке Fork, чтобы связать текущую версию программы со своей учетной записью.

   Обратите внимание на то, что теперь в адресе URL текущего проекта содержится имя вашей учетной записи.

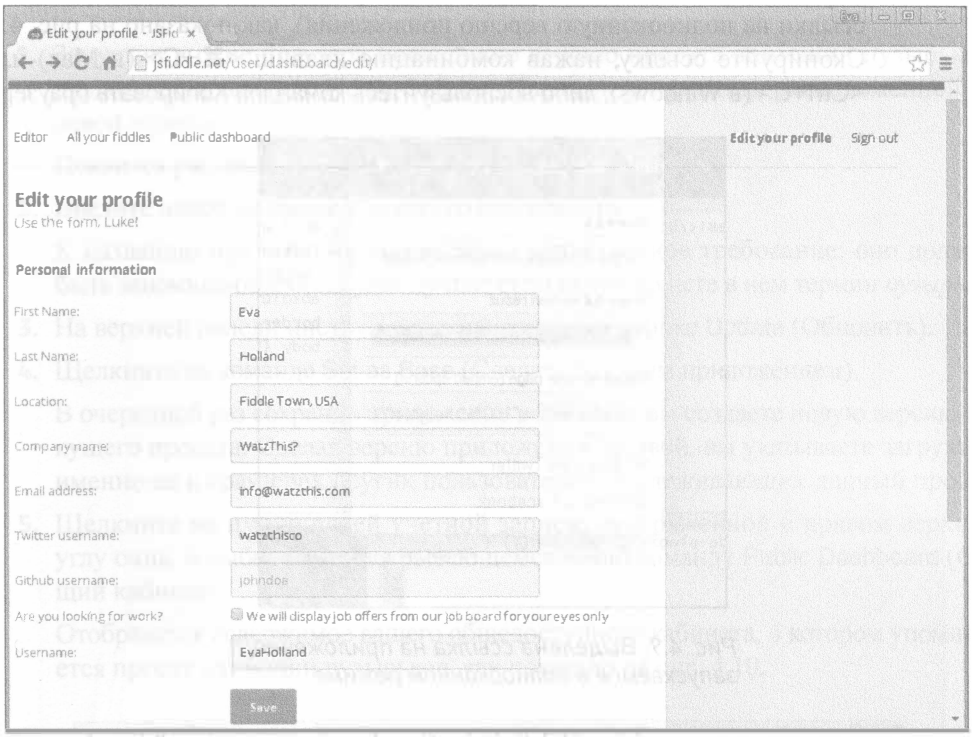


Рис. 4.8. Страница изменения профиля пользователя

Общий доступ к проектам

После создания собственной версии проекта анимации пузырьков можно смело хвастаться ею перед друзьями. Чтобы выполнить эту задачу, следуйте таким инструкциям.

1. Щелкните в верхней части окна на кнопке Share (Поделиться).

   Появится раскрывающееся меню, в котором содержатся команды создания ссылок на текущее приложение, в том числе запускаемое в полнооконном виде. С его помощью также можно опубликовать проект на Facebook или в Твиттере.

При публикации своего творения на Facebook или в Твиттере не забывайте упоминать о нас (`@watzthisco` в Твиттере или `www.facebook.com/watzthisco`). Мы обязательно оценим ваши старания!

2. Выделите в меню **Share** содержимое второго текстового поля (создание ссылки на полнооконную версию приложения), как показано на рис. 4.9. Скопируйте ссылку, нажав комбинацию клавиш <⌘+C> (в Mac) или <Ctrl+C> (в Windows), либо воспользуйтесь командой **Копировать** браузера.

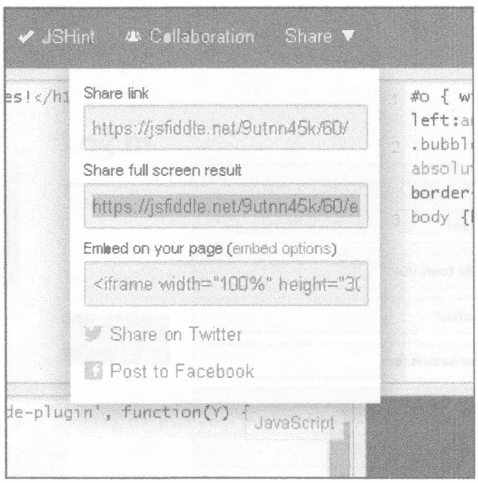


Рис. 4.9. Выделена ссылка на приложение, запускаемое в полнооконном режиме

3. Откройте в окне браузера новую вкладку, нажав комбинацию клавиш <⌘+T> (в Mac) или <Ctrl+T> (в Windows). Вставьте скопированную на предыдущем этапе ссылку в строку адреса и нажмите клавишу <Return> или <Enter>.

   Приложение запустится на всю ширину окна браузера. Панели ввода кода JavaScript, CSS и HTML при этом не отображаются.

Если приложение отказывается запускаться в вашем браузере в полнооконном режиме, то попробуйте заменить `https` на `http` в строке адреса, после чего нажмите клавишу <Return> или <Enter>.

Чтобы вернуться к исходному варианту приложения анимации пузырьков, найдите ссылку на него на следующей странице:

```
https://jsfiddle.net/user/forkids/fiddles
```

Вам больше по душе работать в личном кабинете над собственными проектами? Далее вы узнаете, как сохранить собственный вариант приложения под новым именем.

Сохранение собственного приложения

Получив в свое распоряжение отдельную учетную запись, можно приступать к созданию и сохранению новых версий приложения.

Чтобы создать в JSFiddle собственную версию проекта анимации пузырьков, выполните следующие действия.

1. Запустив в окне браузера последнюю версию проекта анимации пузырьков, щелкните на команде **Fiddle Options** (Параметры проекта), расположенной на левой панели.

   Появится раскрывающееся меню.

2. Введите новое название для своего приложения.

   К названию приложения выдвигается единственное требование: оно должно быть запоминающимся. Будет лучше, если вы упомянете в нем термин *пузырьки*.

3. На верхней панели инструментов щелкните на кнопке **Update** (Обновить).

4. Щелкните на команде **Set as Base** (Сделать базовым приложением).

   В очередной раз сохранив приложение в JSFiddle, вы создаете новую версию текущего проекта. Сделав версию приложения базовой, вы указываете загружать именно ее в браузерах других пользователей, отслеживающих данный проект.

5. Щелкните на имени своей учетной записи, отображенной в правом верхнем углу окна, и выберите в раскрывающемся меню команду **Public Dashboard** (Общий кабинет).

   Отобразится содержимое вашего общедоступного кабинета, в котором упоминается проект анимации пузырьков, как показано на рис. 4.10.

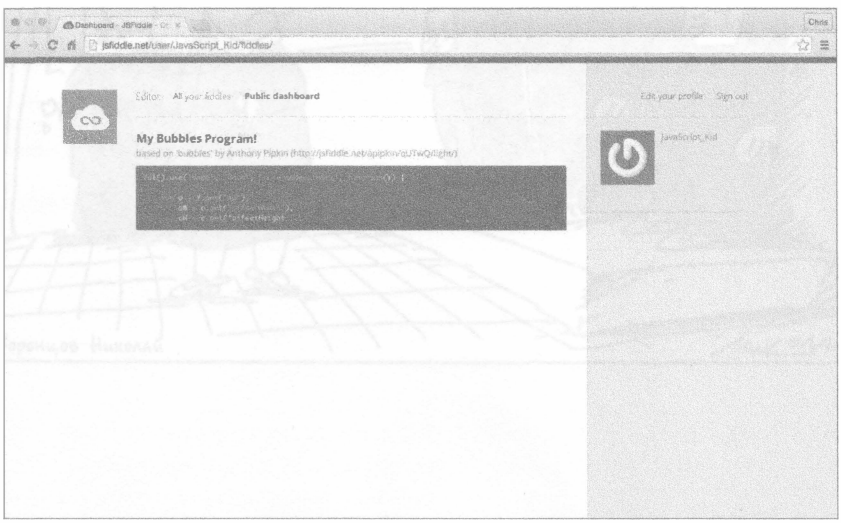


Рис. 4.10. Кабинет с общедоступными проектами на JSFiddle

Часть II
Анимация в Интернете

В этой части...

- ✓ JavaScript и HTML
- ✓ JavaScript и CSS
- ✓ Создание двигающегося робота

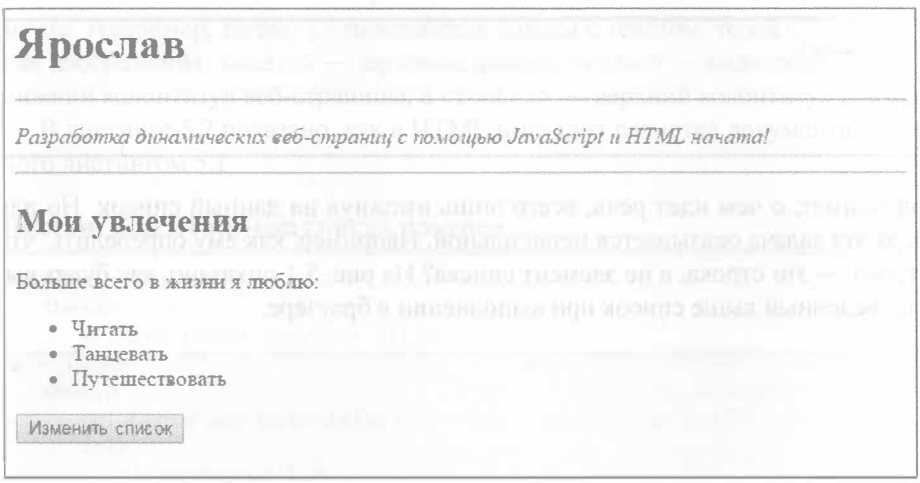
ГЛАВА 5

JavaScript и HTML

В веб-разработке язык программирования JavaScript редко используется сам по себе. Но, как и пицца без сыра, редко какой веб-сайт обходится без JavaScript-кода.

Возможность динамического изменения содержимого веб-страниц и обеспечение их интерактивными элементами, отвечающими за взаимодействие с пользователями, делает JavaScript невероятно популярным инструментом веб-программирования. Но для того, чтобы научиться эффективно использовать его при создании собственных страниц, необходимо детально познакомиться с базовыми технологиями создания веб-страниц.

В этой главе вы познакомитесь с основным языком веб-разработки — HTML. Вы также узнаете, как правильно использовать JavaScript для изменения и создания HTML-кода.



Написание кода HTML

Аббревиатура "HTML" появилась как сокращение от "Hypertext Markup Language" (Язык разметки гипертекста). Столь замысловатое название указывает на способность HTML добавлять в созданные с его помощью документы гиперссылки на другие страницы и ресурсы. При этом возможности HTML намного шире, чем может показаться при первом знакомстве. Они выходят за рамки одного только перекрестного связывания документов ссылками.

Средствами HTML создается основная структура веб-страницы, включая текст, рисунки и другие элементы, управление которыми осуществляется с помощью языка программирования JavaScript.

Текст без HTML

Языки разметки, к которым относится и HTML, разрабатывались, чтобы представить документы (письма, книги, заметки и т.п.) в виде законченной структуры, понятной и доступной для управления компьютером.

В листинге 5.1 приведен простой список покупок, который будет понятен каждому, кто хоть раз посещал супермаркет.

Листинг 5.1. Список покупок

```
Овощи для салата:
морковь;
сельдерей;
шпинат.
```

Любой поймет, о чем идет речь, всего лишь взглянув на данный список. Но для компьютера эта задача оказывается непосильной. Например, как ему определить, что первая строка — это строка, а не элемент списка? На рис. 5.1 показано, как будет выглядеть приведенный выше список при выполнении в браузере.

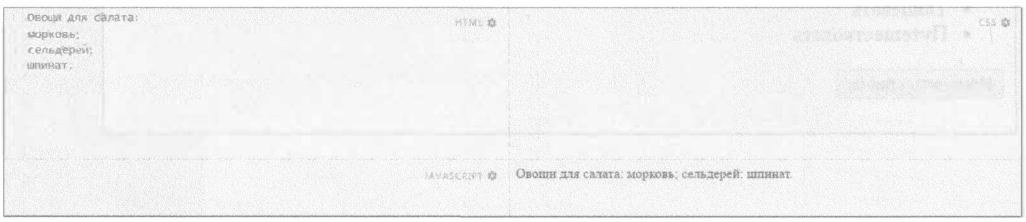


Рис. 5.1. Выполнение текста из листинга 5.1 как HTML-кода в приложении JSFiddle

Становится очевидным: чтобы документ стал понятным для браузера, его необходимо разметить, для чего язык HTML подходит как нельзя лучше.

Теги в HTML

Структура HTML-документа создается с помощью *тегов*. Теги представляют собой контейнеры, в которые помещаются отдельные элементы документа. Ориентируясь по тегам, компьютер распознает содержимое документа и представляет веб-страницу в виде, задуманном разработчиком.

Каждый тег состоит из ключевых слов, заключенных в угловые скобки (< и >). Также он имеет две вариации: открывающий и закрывающий теги. Отличие закрывающего тега от открывающего проявляется в добавлении перед ключевым словом косой черты (/). Например, закрывающим для тега будет .

Разметка документа с помощью тегов достаточно проста: поместите между открывающим и закрывающим тегами элемент документа (текст, рисунок или что-то еще), который необходимо пометить. В частности, абзац с текстом выделяется в HTML следующим образом.

```
<p>Это абзац с текстом. Он отделяется от других элементов
документа дополнительными интервалами, а его первая строка
вводится с отступом</p>
```

Открывающий тег, расположенное после него содержимое и закрывающий тег все вместе называются *HTML-элементом*.

Язык HTML содержит набор тегов для разметки всех возможных элементов документа. Например, тегом помечаются абзацы с текстом, тегом <img> — графические изображения, <audio> — звуковые файлы, <video> — видеоролики, <header> — нижний колонтитул веб-страницы, а <footer> — верхний колонтитул документа.

В листинге 5.2 показано, как в HTML выглядит разметка документа, представленного листингом 5.1.

Листинг 5.2. HTML-код списка покупок

```
<html>
   <head>
      <title>Список покупок</title>
   </head>
   <body>
      <h1>Овощи для салата</h1>
      <ol>
         <li>морковь</li>
         <li>сельдерей</li>
         <li>шпинат</li>
      </ol>
   </body>
</html>
```

Результат выполнения листинга 5.2 в браузере показан на рис. 5.2. Как видите, на веб-странице список покупок представляется вполне разборчиво.

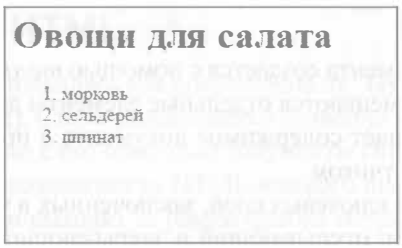


Рис. 5.2. Список покупок, отображенный в окне браузера после выполнения HTML-кода из листинга 5.2

Обратите внимание на то, что сами теги HTML на веб-странице не отображаются. И это вполне понятно, ведь их задача сводится лишь к тому, чтобы указать браузеру, как должны выглядеть те или иные элементы документа.

Структура веб-страницы

Создавать веб-страницы очень просто, особенно если заранее представить, как они должны выглядеть в окне браузера. Кроме того, вам нужно смириться с тем, что большинство веб-страниц имеет одинаковую структуру, а их создание подчиняется нескольким общим правилам.

Первое правило HTML-разметки документа гласит, что каждый элемент начинается открывающим тегом и заканчивается закрывающим тегом. Не допускается изменять порядок следования тегов в паре или пропускать один из них (за редким исключением).

Например, в листинге 5.2 закрывающий тег `</html>` располагается после открывающего тега `<html>`, но не наоборот. Эта пара тегов содержит в себе всю веб-страницу. Остальные открывающие теги представленного HTML-документа также сопровождаются закрывающими тегами. Все они вложены в элемент `html`.

В частности, внутри элемента `html` находится элемент `head`. Таким образом, закрывающий тег `</head>` должен располагаться перед закрывающим тегом `</html>`. А вот элемент `ol` (представляющий нумерованный список) вкладывается в элемент `body`, поэтому тег `</ol>` предшествует тегу `</body>`.

Правило создания HTML-документов гласит о том, что любая веб-страница должна содержать элементы `head` и `body`.

- ✓ **Элемент head.** Этот элемент представляет заголовок веб-документа. Сложный код JavaScript чаще всего включается в элемент `head`, хотя он и не отображается в окне браузера.

  В коде, представленном в листинге 5.2, элемент `head` содержит всего один элемент, `title`. Последний включает текст, отображаемый в заголовке окна браузера или в названии вкладки. Кроме того, содержимое элемента `title` представляет ссылку на веб-страницу в списке результатов поиска, проводимого в Интернете.

✓ **Элемент body.** В элемент `body` включается все, что должно непосредственно отображаться на веб-странице при ее открытии в браузере.

Из листинга 5.2 видно, что в нашем примере элемент `body` содержит несколько вложенных элементов.

- **Элемент h1.** Самый крупный текстовый заголовок веб-страницы размечается элементом `h1`. Обычно заголовками документ разбивают на несколько логических или структурных частей. В частности, если бы настоящая глава была веб-страницей, то элементами `h1` в ней выделялись бы заголовки самого высокого уровня.
- **Элемент ol.** Сразу после элемента `h1` располагается элемент `ol`. Его название образовано от "ordered list" (упорядоченный список). Элементы таких списков нумеруются как числами, так и буквами (согласно порядку их следования в алфавите). В HTML также поддерживается возможность создания неупорядоченных (маркированных) списков, для чего применяется элемент `ul`.
- **Элемент li.** За элементом `ol` следует элемент `li`. В HTML элементы `ol` и `ul` всегда наполняются элементами `li`. Название элемента `li` происходит от "list item" (элемент списка); каждый из элементов `li` представляет отдельный элемент нумерованного или маркированного списка.

Создание первой веб-страницы

Чтобы создать в приложении JSFiddle код своей первой веб-страницы, выполните следующие действия.

1. Запустите браузер и загрузите в нем сайт `http://jsfiddle.net`.
2. Перетащите границы панелей в центральной части окна так, чтобы панель HTML занимала наибольшее пространство.

   В текущем примере мы будем работать только с кодом HTML, поэтому не беспокойтесь о том, что для других панелей отводится слишком мало места.
3. Введите на панель HTML следующий код, представляющий шаблон веб-страницы.

```
<html>
   <head>
      <title>Шаблон веб-страницы</title>
   </head>
   <body>
      <h1>Базовый шаблон веб-страницы</h1>
   </body>
</html>
```

Как только вы введете `<html>`, на экране появится предупреждающее сообщение, подобное показанному на рис. 5.3. В нем указывается, что во вводе этого тега нет необходимости. Он настолько шаблонный, что приложение JSFiddle вводит теги `<html>` и `</html>` автоматически, выполняя вместо вас всю рутинную работу. Мелочь, а приятно!

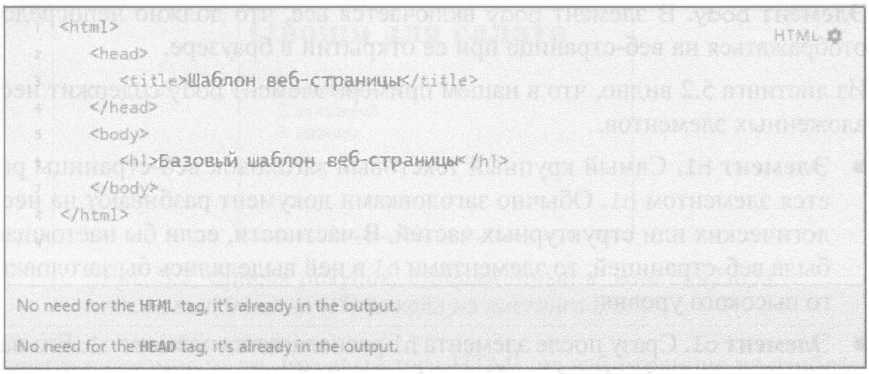


Рис. 5.3. Предупреждение от JSFiddle

4. Поскольку базовый шаблон страницы создается автоматически, удалите на панели HTML все, кроме содержимого элемента `body` (текста, добавленного между тегами `<h1>` и `</h1>`).

Не забывайте о том, что элементы `html`, `head` и `body` содержатся во всех без исключения веб-страницах. Приложение JSFiddle добавляет их автоматически, поэтому вручную их вводить не придется. При использовании других инструментов веб-разработки вам может понадобиться добавлять обязательные теги вручную. Будьте бдительны!

5. Щелкните на кнопке Run.

   Текст, заключенный в теги `<h1>` и `</h1>`, выводится на панели просмотра результатов (правая нижняя панель в окне JSFiddle), принимая форматирование заголовка первого уровня, как показано на рис. 5.4.

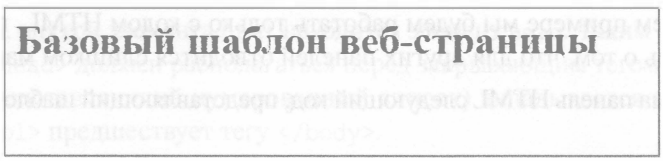


Рис. 5.4. Простой HTML-заголовок

Базовые HTML-элементы

В набор языка HTML входит не так уж и много элементов. Поскольку в нашей книге основное внимание уделяется JavaScript, мы не будем здесь рассматривать все HTML-элементы. Но мы познакомим вас с основными тегами языка HTML, которые понадобятся вам при разработке веб-страниц, используемых в книге в качестве примеров. Языку HTML в Интернете посвящено множество сайтов. С нерассмотренными в книге HTML-элементами вы легко познакомитесь на одном из них.

Существует огромное количество интересных книг, в которых рассматривается язык HTML и его применение в веб-разработке. Среди них можно порекомендовать *HTML5 и CSS3 для чайников*.

Наиболее часто используемые при написании веб-страниц элементы HTML описаны в табл. 5.1.

Таблица 5.1. Базовые элементы языка HTML

Элемент	Название	Описание
h1-h6	Заголовок (от 1-го по 6-й уровень)	Заголовки разделов
p	Абзац	Текстовый абзац
em	Выделение (курсив)	Определяет форматирование выделения в тексте. В большинстве браузеров соответствует курсивному начертанию
strong	Выделение (полужирный)	Определяет форматирование очень важного текстового фрагмента. В большинстве браузеров соответствует полужирному начертанию
a	Ссылка	Гиперссылка на другие страницы и ресурсы
ul	Неупорядоченный список	Маркированный список
ol	Упорядоченный список	Нумерованный список
li	Элемент списка	Отдельный элемент маркированного или нумерованного списка
img	Рисунок	Графическое изображение
hr	Разделитель	Сплошная горизонтальная линия через всю страницу
div	Контейнер	Контейнер для отдельных частей документа

Давайте попробуем воспользоваться приведенными в табл. 5.1 элементами для создания своей первой веб-страницы.

1. В приложении JSFiddle удалите все, что находится на панели HTML, и щелкните на кнопке Run.

   Панель результатов (справа внизу) должна очиститься от любых данных.

2. Воспользовавшись элементом `h1`, создайте в документе заголовок первого уровня, в котором содержится ваше имя.

3. Разместите после только что добавленного заголовка первого уровня горизонтальную линию.

4. Добавьте в документ элемент абзаца, включив между тегами `<p>` и `</p>` любимое высказывание, несколько слов о себе или следующий текст.

   ```
   Разработка динамических веб-страниц с помощью JavaScript и HTML начата!
   ```

5. Поместите в элемент em весь текст абзаца (включенный в элемент p), добавленный в предыдущем пункте.
6. Щелкните на кнопке Run.

   Если вы в точности следовали приведенным выше инструкциям, то ваш код должен выглядеть так, как показано ниже.

```
<h1>Ярослав</h1>
<hr>
<p><em>Разработка динамических веб-страниц с помощью JavaScript
      и HTML начата!</em></p>
```

   Результат выводится на отдельную панель, расположенную в правой нижней части окна, как показано на рис. 5.5.

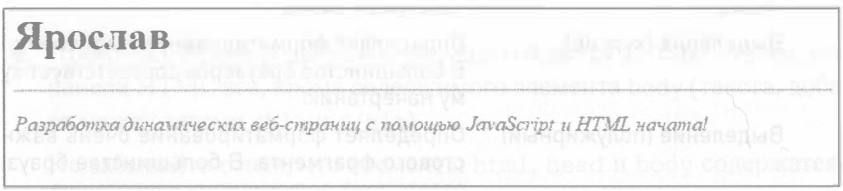


Рис. 5.5. Начало положено

7. Создайте еще одну горизонтальную линию под абзацем.
8. Добавьте заголовок второго уровня (используя элемент h2), заключив в него текст Мои увлечения.
9. Создайте второй элемент абзаца, содержащий следующий текст.

   Больше всего в жизни я люблю:

10. Сформируйте неупорядоченный (маркированный) список, состоящий из трех пустых элементов.

    Код такого списка очень простой.

```
<ul>
   <li>элемент1</li>
   <li>элемент2</li>
   <li>элемент3</li>
</ul>
```

11. Заполните элементы списка названиями своих увлечений.

    В листинге 5.3 показан примерный вид кода HTML, который вы должны получить, выполнив все описанные выше действия.

Листинг 5.3. Заполненная данными первая страница

```
<h1>Ярослав</h1>
   <hr>
   <p><em>Разработка динамических веб-страниц с помощью JavaScript
         и HTML начата!</em></p>
   <hr>
   <h2>Мои увлечения</h2>
   <p>Больше всего в жизни я люблю:</p>
   <ul>
      <li>Читать</li>
      <li>Танцевать</li>
      <li>Путешествовать</li>
   </ul>
```

12. Щелкните на кнопке Run, чтобы просмотреть результат выполнения полученного HTML-кода в браузере. Конечный вид страницы должен быть подобен показанному на рис. 5.6.

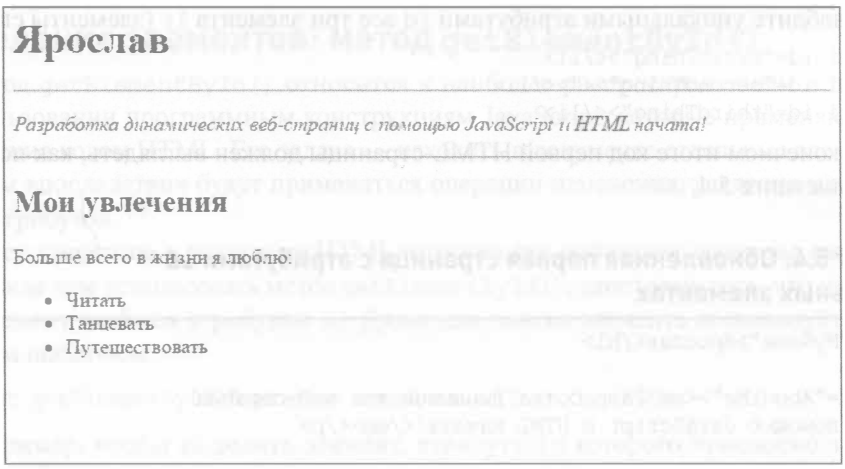


Рис. 5.6. Первая страница вашего резюме

Атрибуты HTML-элементов

Атрибуты позволяют браузерам получать более детальные сведения об элементах HTML. Атрибуты элементов представляются в программе JavaScript парой *имя/значение*. Например, в элементе `img` допускается использовать атрибут, определяющий расположение рисунка, а также дополнительный атрибут, указывающий содержимое, которое выводится вместо графического изображения, если таковое не найдено по указанному пути.

```
<img src="Изображение.jpg" alt="Мое фото">
```

В приведенном примере у тега `img` сразу два атрибута: `src` и `alt`.

Каждый элемент HTML обладает своим перечнем атрибутов. Одни из них, такие как `src` и `alt`, описывают поведение элемента или определяют его действие. Другие, подобно атрибуту `id`, всего лишь снабжают браузер вспомогательными сведениями об элементе.

К слову, атрибут `id` уникальным образом идентифицирует каждый элемент веб-страницы. В частности, в вашем документе может содержаться большое количество элементов `li`. Если в их элементы добавить атрибуты `id`, то ими можно будет управлять по отдельности, в том числе в JavaScript-коде.

В последующих проектах атрибут `id` будет применяться настолько часто, что нужно остановиться на его описании более подробно.

Давайте воспользуемся атрибутом `id` в элементах HTML вашей первой страницы.

1. Добавьте атрибут `id` в элемент `h1`, который будет включать ваше имя.

   ```
   <h1 id="MyName">
   ```

2. Включите в элемент `p`, расположенный между двумя элементами `hr`, атрибут `id`, снабдив им открывающий тег. Пусть атрибут `id` содержит значение `AboutMe`.

   ```
   <p id="AboutMe">
   ```

3. Снабдите уникальными атрибутами `id` все три элемента `li` (элементы списка).

   ```
   <li id="firstThing"></li>
   <li id="secondThing"><"></li>
   <li id="thirdThing"></li>
   ```

   В конечном итоге код первой HTML-страницы должен выглядеть, как показано в листинге 5.4.

Листинг 5.4. Обновленная первая страница с атрибутами `id` в отдельных элементах

```
<h1 id="MyName">Ярослав</h1>
   <hr>
   <p id="AboutMe"><em>Разработка динамических веб-страниц
      с помощью JavaScript и HTML начата!</em></p>
   <hr>
   <h2>Мои увлечения</h2>
   <p>Больше всего в жизни я люблю:</p>
   <ul>
      <li id="firstThing">Читать</li>
      <li id="secondThing">Танцевать</li>
      <li id="thirdThing">Путешествовать</li>
   </ul>
```

4. Щелкните на кнопке Run, чтобы просмотреть результат выполнения представленного кода в приложении JSFiddle. Легко заметить, что ничего не изменилось. Совсем ничего! Веб-страница выглядит, как и прежде — до назначения элементам атрибутов `id`. Конечно, это не то, что вы ожидали. Ощутимый эффект от использования атрибутов `id` вы получите только при выполнении кода JavaScript.

Атрибут `id` должен обязательно начинаться с латинского символа. Использование русских букв в его значениях недопустимо.

Изменение HTML-кода с помощью JavaScript

Язык программирования JavaScript позволяет изменять код HTML любой части веб-страницы. Обычно это делается в ответ на действие пользователя, просматривающего HTML-документ. Далее вы узнаете, как это можно сделать.

Перед тем как начать, вам необходимо познакомиться с несколькими методиками программирования. Первая из них заключается в использовании метода `getElementById()`.

Как вы знаете из главы 3, метод в коде JavaScript выполняет некое действие по отношению к объекту, к которому он относится.

Выделение элементов: метод `getElementById()`

Метод `getElementById()` относится к наиболее распространенным и простым в использовании программным конструкциям JavaScript, которые применяются для управления кодом HTML. Его задача — находить в документе отдельные элементы, к которым впоследствии будут применяться операции изменения, удаления или добавления атрибутов.

Поиск элементов в документе HTML известен как *выделение элемента* в коде.

Прежде чем использовать метод `getElementById()`, удостоверьтесь, что необходимый элемент снабжен атрибутом `id`. Далее для поиска элемента воспользуйтесь следующим шаблоном.

```
document.getElementById("значение атрибута id")
```

Например, чтобы выделить элемент, атрибуту `id` которого присвоено значение `myName`, используйте следующую программную конструкцию.

```
document.getElementById("myName")
```

Содержимое элемента: свойство `innerHTML`

Выделение в коде отдельного элемента всегда сопровождается его изменением. (Иначе для чего еще его потребовалось выделять?)

Свойством `innerHTML` обладает каждый HTML-элемент веб-страницы. Оно указывает на то, что содержится между открывающим и закрывающим тегами элемента, и позволяет его изменить.

Свойство описывает одну из *характеристик* объекта, в противоположность методу, который определяет *действие*, выполняемое объектом или над объектом.

В качестве примера давайте рассмотрим следующий элемент HTML.

```
<p id="myParagraph">Введенный <em>мною</em> абзац!</p>
```

Для выделения в коде и изменения значения его свойства `innerHTML` используется такая команда.

```
document.getElementById("myParagraph").innerHTML = "Введенный
   ↳<em>мною</em> абзац!";
```

Изменение списка в HTML

Теперь, когда вы познакомились с методом `getElementById()` и свойством `innerHTML`, давайте попробует изменить с их помощью код HTML нашей первой страницы.

Чтобы сделать задачу интереснее, добавим на страницу кнопку, щелчок на которой приводит к выполнению JavaScript-кода, отвечающего за модификацию кода HTML.

1. Кнопка создается с помощью следующего кода HTML, размещаемого на веб-странице сразу под списком.

   ```
   <button id="changeList" type="button">
      Изменить список
   </button>
   ```

   Браузер распознает приведенный выше элемент как кнопку, на которой размещается текст, указанный между тегами `<button>` и `</button>`.

2. Создайте на панели JavaScript три следующие переменные.

   ```
   var item1;
   var item2;
   var item3;
   ```

   Эти переменные будут использоваться для хранения данных, получаемых от пользователя перед их выводом на веб-страницу.

3. Для распознавания щелчков на только что созданной кнопке используется следующий код JavaScript, вводимый в JSFiddle на соответствующей панели.

   ```
   document.getElementById("changeList").onclick = newList;
   ```

   В данном коде метод `getElementById()` используется для выделения элемента кнопки (его атрибут `id` имеет значение `changeList`). Именно для этих целей элементам веб-страницы и назначаются идентификаторы.

   После распознавания кнопки ее состояние отслеживается с помощью обработчика событий `onclick`. Название этого программного элемента говорит само

за себя. Он указывает, как браузеру необходимо реагировать на те или иные события, происходящие на странице (в данном случае щелчок пользователем на кнопке).

Как только происходит щелчок на кнопке, обработчик этого события запускает функцию, которая называется `newList`. Функция представляет собой небольшую программу внутри основной программы. Подробно о функциях рассказывается в главе 12, а сейчас вам достаточно знать, что код функции выполняется только по требованию, т.е. по вызову из основной программы.

4. Следующие строки кода необходимы, чтобы запросить у пользователя текст новых элементов списка.

```
function newList(){
   item1 = prompt("Введите первый элемент: ");
   item2 = prompt("Введите второй элемент: ");
   item3 = prompt("Введите третий элемент: ");
   updateList();
}
```

В первой строке кода объявляется сама функция. Между фигурными скобками (`{` и `}`) указываются команды, которые функция выполняет при вызове. Вызывается функция при упоминании в коде ее имени (`newList`).

В нашем случае функция запрашивает у пользователя три новых элемента списка, для чего используется уже известная вам команда `prompt`.

Наконец, в код JavaScript добавляется еще одна функция, `updateList()`, описанная далее.

5. Следующий код отвечает за обновление элементов списка.

```
function updateList() {
   document.getElementById("firstThing").innerHTML = item1;
   document.getElementById("secondThing").innerHTML = item2;
   document.getElementById("thirdThing").innerHTML = item3;
}
```

Функция `updateList()` распознает каждый из элементов списка по атрибуту `id`. Далее с помощью свойства `innerHTML` содержимое элемента (находящееся между открывающим и закрывающим тегами) заменяется значением переменной, хранящей запрошенные у пользователя данные.

В результате выполнения функции `updateList` элементы списка в окне браузера заменяются введенными пользователем значениями.

Окончательный вариант JavaScript-кода, отвечающий за выполнение всех описанных выше действий, имеет вид, показанный в листинге 5.5. Внимательно изучите его перед выполнением, стараясь не допускать синтаксических ошибок при вводе ключевых слов. В противном случае программа может не выполнить всех возложенных на нее обязанностей.

Листинг 5.5. JavaScript-код первой веб-страницы

```
var item1;
var item2;
var item3;

document.getElementById("changeList").onclick = newList;

function newList() {
   item1 = prompt("Введите первый элемент: ");
   item2 = prompt("Введите второй элемент: ");
   item3 = prompt("Введите третий элемент: ");
   updateList();
}

function updateList() {
   document.getElementById("firstThing").innerHTML = item1;
   document.getElementById("secondThing").innerHTML = item2;
   document.getElementById("thirdThing").innerHTML = item3;
}
```

6. Щелкните на кнопке Run, чтобы выполнить код.

Если вы не допустили ошибок при вводе кода JavaScript и HTML, то сможете, щелкнув на кнопке, ввести новые названия для всех трех элементов списка, содержащегося на веб-странице, и понаблюдать, как они заменяют уже имеющиеся (рис. 5.7).

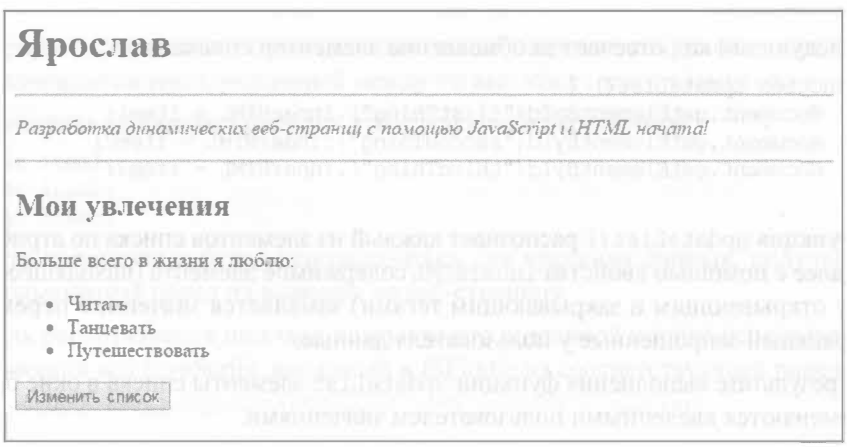


Рис. 5.7. Конечный вид веб-страницы, которая содержит программу управления кнопкой, отвечающей за изменение элементов списка

ГЛАВА 6

JavaScript и CSS

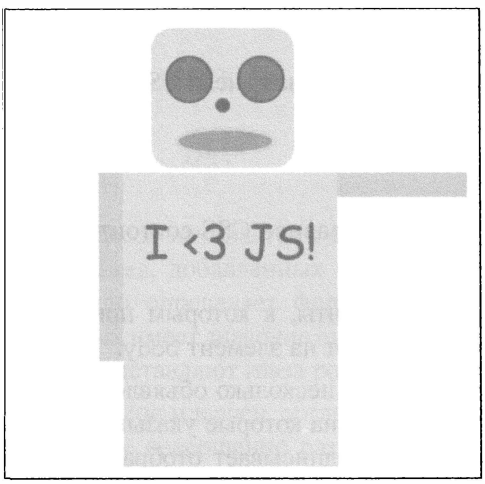
У каждого из нас есть свои таланты. Кто-то хорошо владеет искусством айкидо, кто-то прекрасно вышивает крестиком, а кто-то устанавливает рекорды по поеданию гамбургеров. Увлечения приходят и уходят, и то, что сегодня кажется чрезвычайно интересным, завтра будет выглядеть абсолютно скучным занятием.

Покупая новые наряды и примеряя их, мы изменяем свой внешний вид. Но это не делает нас умнее или нравственнее. Надев новое пальто, вы вряд ли измените свой внутренний мир.

Подобным образом можно изменять внешний вид веб-страницы отдельно от ее структуры и содержимого.

Язык, отвечающий за изменение одного только внешнего вида веб-страниц, получил название "CSS" (Cascading Style Sheet — каскадные таблицы стилей). В этой главе вы узнаете о возможностях этой технологии и научитесь применять ее для оформления собственных веб-приложений.

С помощью CSS удобно изменять цветовую гамму веб-страницы, добавлять границы к отдельным ее элементам, назначать фон и детально указывать размер каждого объекта, отображаемого в браузере. Кроме того, язык CSS применяется для точного позиционирования элементов и управления специальными эффектами, в том числе анимацией.



Дуглас — программный робот

В проекте этой главы мы займемся стилизацией и управлением программным роботом со странным именем "Дуглас", выводимым на целевой веб-странице. Будучи созданным умелыми разработчиками, он прекрасно владеет JavaScript, имеет нулевой пробег (только что сошел с конвейера) и не требует специального обслуживания (отсутствуют изнашивающиеся детали).

Но у столь искусного творения есть один серьезный недостаток: он выглядит грустным. Конечно, нельзя от робота требовать проявления человеческих эмоций, но активная жизненная позиция и немного веселого характера ему все же не повредят. Пусть наш робот обладает собственным задорным стилем.

Для начала запустите свой браузер и загрузите в нем страницу по такому адресу:

```
http://jsfiddle.net/user/forkids/fiddles
```

Теперь выполните следующие действия.

1. Найдите проект с названием Chapter 6: Robot Style — Start и щелкните на его заголовке.

   Чтобы перейти к проекту сразу, миновав предыдущие действия, загрузите страницу с таким адресом:

   ```
   http://jsfiddle.net/forkids/m9nfspxp
   ```

   На панелях HTML и CSS уже содержится некий код, в котором нам предстоит разобраться.

2. Щелкните в верхней части окна на кнопке Fork и скопируйте код проекта в свой личный кабинет.

Дуглас готов к нашим экспериментам — из него выйдет отменный стиляга!

Знакомство с CSS

Изучите первые три строки кода, приведенные на панели CSS приложения JSFiddle.

```
body {
   font-family: Arial;
}
```

Все вместе они образуют правило CSS. Правило CSS состоит из двух основных компонентов.

- ✓ **Селектор.** Селектор определяет элементы, к которым применяется правило CSS. В нашем случае селектор указывает на элемент `body`.
- ✓ **Блок объявлений.** Содержит одно или несколько объявлений, определяющих параметры форматирования элементов, на которые указывает селектор. В рассматриваемом примере объявление предписывает отображать текст элемента шрифтом Arial.

Селекторы CSS

Селектор является неотъемлемой частью правила CSS, в котором он вводится перед открывающей фигурной скобкой ({). Селектор указывает браузеру, к каким элементам применять форматирование, описанное в текущем правиле CSS.

Указав в правиле CSS элемент, к которому применяется форматирование, определенное в правиле, будьте готовы к тому, что оно будет автоматически применено и ко всем вложенным в него объектам.

Существует несколько способов указания элементов в селекторе правила CSS. Далее рассмотрены только три из них; все они применяются в коде CSS, отвечающем за оформление нашего программного робота.

- **Селектор элементов.** Он включен в первые два правила CSS-кода нашего робота.

```
body {
   font-family: Arial;
}
p {
   font-size: 1em;
}
```

  В селекторе элементов обращение к HTML-элементу, к которому будет применяться форматирование, взятое из правила, осуществляется по его имени. Таким образом, в селекторе элементов представлены одни лишь имена элементов. В нашем случае первое правило применяется к элементу `body` (всему, что содержится между тегами `<body>` и `</body>`), а второе — к элементу `p` (текстовым абзацам).

- **Селектор классов.** Обратите внимание, как представлен селектор в третьем правиле CSS.

```
.eye {
   background-color: blue;
   width: 20%;
   height: 20%;
   border-radius: 50%;
}
```

  Селектор классов начинается с точки (`.`), после которой указываются значения атрибутов `class`, добавленных в элементы HTML. В частности, приведенное выше правило определяет форматирование всех HTML-элементов, атрибут `class` которых имеет значение `eye` (`class="eye"`). Несложно догадаться, что эти элементы представляют глаза робота Дугласа.

  Селектор классов идеален, когда требуется применить одинаковое форматирование к группе совершенно разных элементов. В нашем примере одинаковое форматирование применяется к глазам робота (синий цвет, одинаковый размер), которые имеют и отличительные характеристики.

✓ **Селектор идентификаторов.** Селектор идентификаторов начинается с символа решетки (#), после которого указываются значения атрибутов `id` целевых элементов HTML. Например, глаза нашего робота имеют разные идентификаторы.

```
#righteye {
   position: absolute;
   left: 20%;
   top: 20%;
}
#lefteye {
   position: absolute;
   left: 60%;
   top: 20%;
}
```

Селекторы идентификаторов лучше всего использовать при создании правил для отдельных элементов веб-страницы.

Все атрибуты `id`, используемые в одном документе, должны быть уникальными.

Если взглянуть на панель кода HTML, то легко заметить, что глаза робота, хотя и принадлежат к одному классу, имеют совершенно разные атрибуты `id`. Это сделано для того, чтобы иметь возможность задавать расположение этих элементов по отдельности.

Объявления CSS

Все объявления в коде CSS вводятся в блоке объявлений соответствующих правил. Каждое объявление состоит из двух частей.

✓ **Свойство.** Определяет параметр, который изменяется правилом. Это может быть цвет, размер или расположение элемента. После названия свойства всегда вводится двоеточие (`:`).

✓ **Значение.** Указывает величину, на которую изменяется свойство.

Все объявления обязательно заканчиваются точкой с запятой (`;`). Строгого ограничения на количество объявлений в блоке одного правила не существует.

В частности, блок объявлений для правила CSS, определяющего форматирование элементов класса `eye`, состоит из четырех строк.

```
.eye {
   background-color:blue;
   width:20%;
   height:20%;
   border-radius: 50%;
}
```

Форматирование веб-страниц с помощью правил CSS

Свойства CSS отвечают за изменение отдельных характеристик элементов HTML-документа. Наш подопечный, робот Дуглас, в полной мере ощущает влияние свойств на свою внешность. С их помощью устанавливаются цвет его глаз, размер туловища и рук, радиус окружности головы, а еще — расположение всех основных частей тела.

Давайте попробуем изменить внешность Дугласа, модифицировав отдельные CSS-правила, применяемые к нему.

1. Найдите правило CSS, определяющее форматирование элемента `p`.

   На панели кода CSS оно введено вторым.

2. Измените значение свойства `font-size` на `2.5em`.

   После изменения правило должно выглядеть следующим образом.

```
p {
   font-size: 2.5em;
}
```

Для определения размера текста на веб-странице используются несколько единиц измерения. Наиболее распространенные — пиксели (`px`), проценты (`%`) и цицеро (`em`). Последние две указывают размер шрифта относительно базового значения. Например, при определении размера шрифта как `2.5em` текст элемента будет увеличен в 2,5 раза.

3. В верхней части окна щелкните на кнопке **Run**, чтобы ознакомиться с результатами изменений, внесенных на правой нижней панели.

   Робот должен принять вид, подобный показанному на рис. 6.1.

4. Найдите правило, устанавливающее форматирование элемента `body`.

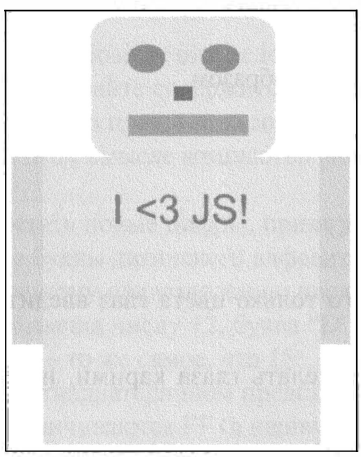


Рис. 6.1. Надпись на роботе крупным шрифтом

5. Установите внешний вид элемента, задав следующие параметры форматирования. Обратите внимание на использование кавычек.

   `"Comic Sans MS", cursive, sans-serif`

   Правило CSS должно выглядеть следующим образом.

```
body {
   font-family: "Comic Sans MS", cursive, sans-serif;
}
```

6. Щелкните на кнопке Run и просмотрите полученные результаты.

   Надпись на торсе робота выглядит неровной, как показано на рис. 6.2.

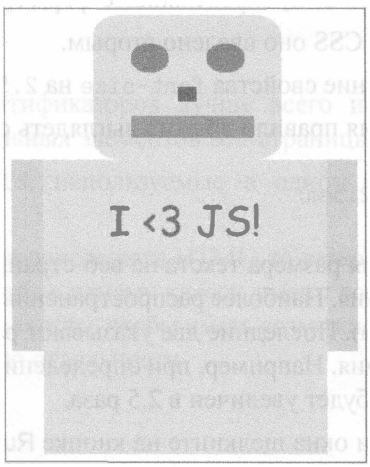


Рис. 6.2. Дуглас знает толк в чистописании

Теперь давайте изменим цвет глаз Дугласа. Какой вы предпочитаете?

7. Отыщите на панели CSS правило, устанавливающее форматирование глаз робота.

   Выглядит оно следующим образом.

```
.eye {
   background-color:blue;
   width: 20%;
   height: 20%;
   border-radius: 50%;
}
```

8. Для изменения одного только цвета глаз введите другое значение свойства `background-color`.

   В частности, чтобы сделать глаза карими, необходимо ввести следующее выражение.

   `background-color: brown;`

9. Для просмотра результатов изменения свойства щелкните на кнопке Run.

Легко заметить, что при установке цвета глаз `brown` глаза не становятся карими в привычном понимании этого слова. Подобным образом после присвоения свойству `background-color` значения `green` глаза робота станут зелеными, но несколько непривычного оттенка. Чтобы получить в точности карие глаза, воспользуйтесь одним из именованных HTML-цветов (см. главу 4) или представьте требуемый оттенок правильно подобранным шестнадцатеричным значением.

Цвета

В языке CSS можно указывать несколько миллионов цветовых оттенков, отображаемых современными браузерами, которые образованы как комбинация красного, зеленого и синего цветовых компонентов.

Ниже приведен пример представления цветового оттенка в виде шестнадцатеричного числового значения.

```
#9BE344
```

Именно в таком виде цветовые оттенки чаще всего представляются в коде CSS.

Хеш-символ (`#`) указывает на то, что следующее значение представляет собой шестнадцатеричный код. Следующие после него две цифры (`9B`) определяют оттенок красного цвета (красная компонента). Четвертый и пятый символы (`E3`) задают зеленый оттенок, а шестой и седьмой (`44`) — насыщенность синей составляющей. Итоговый цвет получается как смешение всех трех основных компонентов. В браузере данное значение представляется правдоподобным зеленым цветом.

При первом знакомстве шестнадцатеричное представление цветового кода вызывает недоумение. Зачем числа представлять и цифрами, и буквами? Ответ очень простой: для увеличения количества возможных комбинаций.

Представьте, что для счета вы используете не десять, а сразу шестнадцать пальцев. В привычной для нас системе счета используется всего десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Если цифр не 10, а 16, то для обозначения недостающих шести цифр приходится использовать буквы. Поскольку в байте содержится 8 бит информации, а в двух байтах — 16 бит, компьютеру намного проще пользоваться для счета 16 цифрами, а не десятью, привычными нам. В этом смысле компьютер напоминает диковинное существо с 16 пальцами.

Вместо того чтобы изобретать новые цифры, принято использовать вместо недостающих шести цифр первые буквы латинского алфавита. Таким образом, буква "A" соответствует шестнадцатеричному представлению числа 10, буква "B" сопоставима с числом 11, буква "C" равнозначна числу 12, буква "D" приравнивается к 13, буква "E" — к числу 14, а буква "F" — то же самое, что 15.

Двухразрядные числа в шестнадцатеричном представлении начинаются с 00 (в нашем представлении — 0) и заканчиваются FF (в нашем представлении — 255).

Теперь становится понятным, что когда мы говорим об оттенке красного цвета насыщенностью 00, то имеем в виду полное его отсутствие. Наименьшее возможное количество цвета (скорее всего, незаметное для глаза) представляется шестнадцатеричным числом 01. А вот оттенком со значением FF обозначается максимально насыщенный красный цвет.

Вместо угадывания цветовых оттенков по их шестнадцатеричным значениям при выборе цвета воспользуйтесь одним из онлайн-ресурсов, на которых цветовые образцы визуально сопоставляются с их числовыми значениями, например `www.colorpicker.com`.

Размер элемента

Каждый элемент HTML-документа имеет прямоугольную форму. Даже когда графический объект выглядит круглым (подобно глазам нашего робота), в программе он все равно представляется как круг, вписанный в квадрат. В подобном прямоугольном мире размеры всех объектов задаются предельно просто — достаточно определить их точную ширину и высоту. Именно поэтому CSS при установке размера элементов оперирует только этими двумя понятиями.

Измеряя ширину и высоту окружающих нас объектов, мы используем такие единицы измерения, как миллиметры, сантиметры и метры. Работая с CSS, приходится привыкать к совершенно другим пространственным размерностям. В веб-разработке в качестве единиц измерения привычнее пользоваться пикселями (`px`) и процентами (`%`).

- ✓ **Пиксели.** Точка наименьшего размера, которая только может отображаться браузером. При указании длины или ширины в пикселях вы определяете размер элемента максимально точно. Проблема заключается в том, что при таком подходе объекты в браузере — независимо от размера его окна — всегда будут оставаться одного и того же размера. Это не всегда удобно при просмотре веб-страниц на экранах очень маленького и слишком большого размеров.
- ✓ **Проценты.** Установив ширину и высоту элемента HTML в процентах, вы зададите его размер относительно размера окна браузера.

Размеры робота Дугласа в коде CSS устанавливаются в относительных единицах измерения (`%`). Это вполне оправданное решение, позволяющее комфортно знакомиться с ним как на экране смартфона, так и на огромном мониторе. Размер изображения на экране всегда будет подстраиваться под размер окна браузера. Вам совсем не обязательно приобретать большой монитор, чтобы познакомиться с эффектом автоматического изменения размера изображения. Достаточно перетащить границы области просмотра результатов в приложении JSFiddle. Сразу бросается в глаза, что даже малейшее изменение области приводит к пропорциональному расширению или сжатию изображения робота.

Вам может быть знакомо понятие *адаптивный дизайн*. Таким термином обозначают содержимое веб-страницы, подстраивающееся под текущий размер экрана, на котором оно просматривается. Изображение робота Дугласа — это прекрасный пример использования технологии адаптивного дизайна в практических целях.

Настало время заняться модификацией Дугласа с помощью CSS.

1. Найдите правило, которое устанавливает форматирование элемента, относящегося к классу `.eye`.
2. Установите как ширину, так и высоту элемента равными `30%`.

   Круги, представляющие глаза робота, заметно увеличатся в размере. К тому же они перестанут отображаться по середине "лица" (рис. 6.3). Последнее требует немедленного исправления.

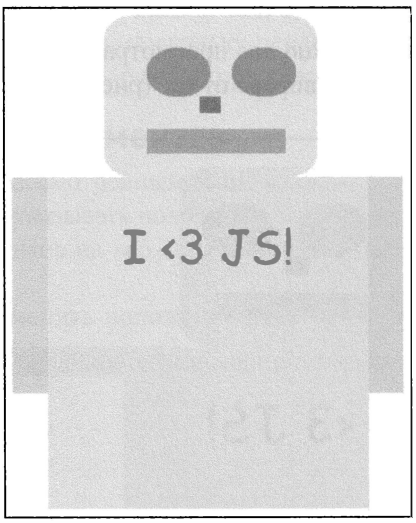


Рис. 6.3. Эти крупные глаза напротив сместились вправо

3. Найдите в коде CSS правило, устанавливающее форматирование элементов класса `.arm`.

```
.arm {
   background-color: #cacaca;
   position: absolute;
   top: 35%;
   width: 5%;
   height: 40%;
}
```

   Несложно догадаться, что этот класс применяется для управления цветом и размером обеих рук робота.

Кроме того, в правиле для класса `.arm` устанавливается расстояние, на котором располагаются руки робота от верхнего края окна. Детально об абсолютном и относительном позиционировании объектов в окне рассказывается далее.

4. Установите ширину рук робота равными `3%` и щелкните на кнопке Run, чтобы ознакомиться с изменениями во внешнем виде Дугласа.
5. Отыщите правило CSS, определяющее характеристики левой руки.
6. Добавьте в него два свойства, устанавливающие ширину (`width`) больше высоты (`height`) элемента. Например, так, как показано ниже.

```
#leftarm {
   position: absolute;
   left: 70%;
   width: 27%;
   height: 5%;
}
```

7. Щелкните на кнопке Run для просмотра полученного эффекта. Теперь робот указывает куда-то вправо от вас (рис. 6.4).

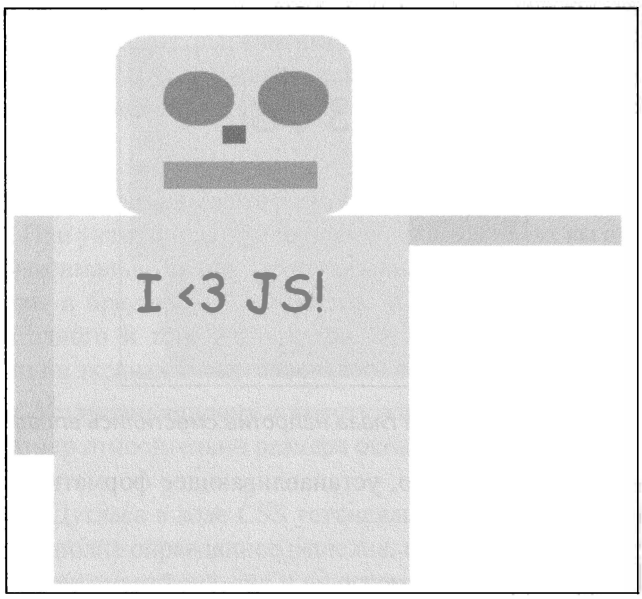


Рис. 6.4. Вам туда!

На последнем этапе приведенных выше инструкций в правило CSS с селектором идентификаторов добавляются два новых свойства, `wight` и `height`, даже несмотря на то что ширина и высота элемента, на который они указывают, уже заданы правилом CSS с селектором классов.

Как ни странно, ширина и высота руки изменяются согласно правилу с селектором идентификаторов. При этом уже установленные ранее правилом с селектором классов значения полностью игнорируются.

Приведенный выше пример демонстрирует каскадную методику определения форматирования элементов. (А вы думали, таблицы стилей названы каскадными просто так?)

Каскадное форматирование

Левая рука робота Дугласа характеризуется только одним свойством `width` и одним свойством `height`. Что же произойдет, если попытаться задать ее размер сразу двумя парами таких свойств, установленных в разных правилах? Какому из них браузер отдаст предпочтение и каким будет окончательный размер элемента?

Принимая решение, браузер учитывает несколько факторов: порядок следования правил в коде и их приоритетность.

В каскадном методе применения правил CSS атрибут `id` всегда имеет более высокий приоритет, чем атрибут `class`, поскольку первый идентифицирует элемент уникальным образом, чего не скажешь о селекторе класса.

Позиционирование элементов

В дополнение ко всем рассмотренным выше возможностям (изменение цвета и размера) язык CSS позволяет указывать положение элементов в HTML-документе. Определение положения элементов на веб-странице часто называют *позиционированием* объектов.

Давайте попробуем изменить положение некоторых частей тела нашего робота.

1. Найдите в коде CSS правило, определяющее форматирование правого глаза.

```
#righteye {
   position: absolute;
   left: 20%;
   top: 20%;
}
```

   Первое свойство этого правила, `position`, указывает браузеру, как отсчитывать следующие настройки позиционирования (в нашем случае — `top` и `left`). При абсолютном позиционировании элемент, к которому применяется свойство CSS (глаз робота), не будет смещаться в сторону соседним объектом, как это часто случается при неправильном относительном позиционировании нескольких элементов (например, глаз смещается с головы). Даже если два элемента с абсолютным типом позиционирования по недосмотру окажутся в одном и том же месте HTML-документа, то они попросту перекроют друг друга.

   Правый глаз смещен относительно верхнего края головы на 20% от ее высоты. При этом он также смещен влево относительно правого края головы на 20% от ее ширины.

2. Сместите правый глаз влево и вверх, уменьшив значения свойств `left` и `top` соответственно.

```
#righteye {
   position: absolute;
   left: 10%;
   top: 10%;
}
```

3. Щелкните на кнопке Run, чтобы ознакомиться с результатом выполненных действий (рис. 6.5).

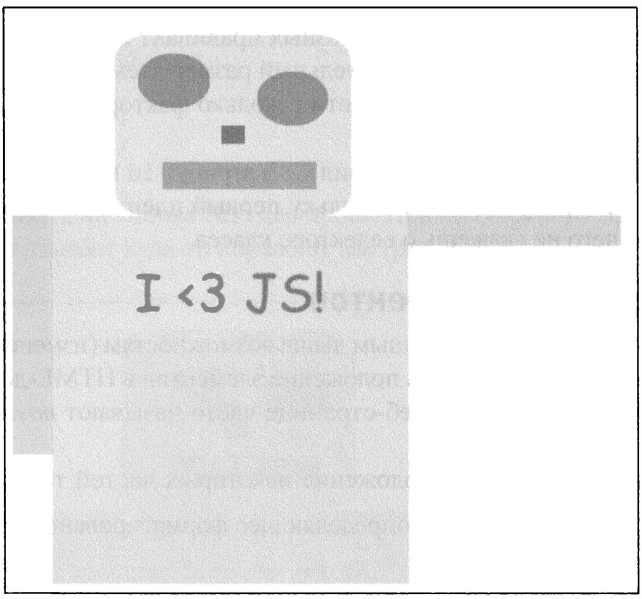


Рис. 6.5. Неужели?! Правый глаз поднялся вверх

Робот собственного изготовления

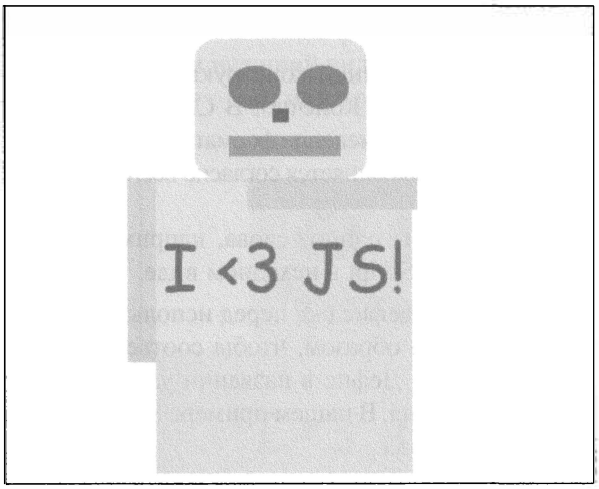
Теперь ваша очередь проявить творческую инициативу и, воспользовавшись полученными в этой главе навыками, поэкспериментировать с внешним видом программного робота. Смело изменяйте цвета, размер и положение отдельных его элементов. Добейтесь самого необычного внешнего вида, который только в состоянии выдать "на гора" ваша необузданная фантазия!

ГЛАВА 7

Анимированный робот

Все вместе — HTML, JavaScript и CSS — прекрасная команда, способная добиться потрясающих результатов. Каждый из языков дополняет остальных два, позволяя полностью контролировать все происходящее в окне браузера.

В этой главе мы воспользуемся всеми тремя языками программирования, чтобы заставить Дугласа исполнить незамысловатый танец робота.



Изменение кода CSS с помощью JavaScript

Как и HTML-код, язык JavaScript позволяет изменять программные конструкции CSS. Ничего принципиально нового для этого не нужно. Методика вам уже известна по предыдущим главам.

Для начала необходимо выделить элемент, форматирование которого требуется изменить. В главе 5 рассказывалось о том, что для этих целей применяется метод `getElementById`. Например, для выделения левого глаза Дугласа используется следующий код.

```
document.getElementById("lefteye")
```

После выделения целевого элемента его форматирование изменяется с помощью свойства `style` селектора, за которым указывается свойство, требующее изменения. Таким образом, код изменения цвета левого глаза принимает такой вид.

```
document.getElementById("lefteye").style.backgroundColor = "purple";
```

Вам ничего не кажется странным в приведенной конструкции, если сравнивать ее с используемым для тех же целей кодом CSS? Конечно! В CSS название свойства `backgroundColor` записывается несколько иначе. Изменение форматирования элемента, заданного свойствами CSS, с помощью JavaScript выполняется согласно всего двум правилам.

- ✓ Если название свойства CSS состоит из одного слова, например `margin` или `border`, то оно применяется в коде JavaScript в исходном виде.
- ✓ Свойства CSS, содержащие в названии дефис (-), перед использованием в коде JavaScript преобразуются специальным образом, чтобы соответствовать стандартам этого языка программирования. Дефис в названии удаляется, а второе слово отображается с прописного символа. В нашем примере `background-color` трансформировалось в `backgroundColor`.

В табл. 7.1 показано, как названия наиболее распространенных свойств CSS приводятся к стандартам JavaScript.

Таблица 7.1. Преобразование названий свойств CSS для использования в JavaScript

Свойство CSS	Свойство объекта *style*
background-color	backgroundColor
border-radius	borderRadius
font-family	fontFamily
margin	margin
font-size	fontSize
border-width	borderWidth
text-align	textAlign
color	color

Регистр символов, в котором вводятся ключевые слова JavaScript, очень важен. Нельзя вводить приведенные выше названия свойств с использованием одних лишь строчных символов.

Управление программным роботом в JavaScript

Изменение кода CSS с помощью языка JavaScript открывает перед нами новые возможности по созданию программных конструкций, позволяющих управлять внешним видом HTML-элементов в ответ на действия пользователей.

Чтобы попрактиковаться в управлении свойствами CSS с помощью JavaScript, давайте обучим робота Дугласа простому танцу.

1. Перейдите в общедоступный кабинет по такому адресу:

   `http://jsfiddle.net/user/forkids/fiddles`

   Найдите в нем программу с названием **Chapter 7: Start** и запустите ее в браузере.

   Вам уже знаком внешний вид робота по предыдущей главе; с момента последнего изменения с ним ничего принципиально нового не произошло (рис. 7.1).

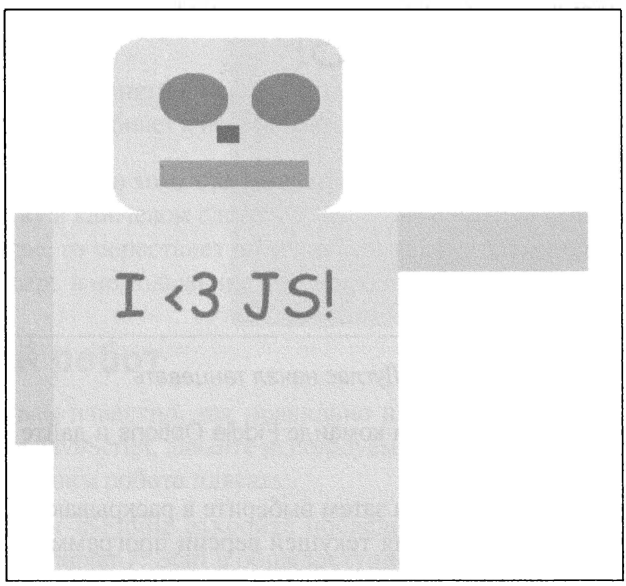


Рис. 7.1. Дуглас — робот, управляемый с помощью JavaScript

2. Щелкните на кнопке **Fork** для добавления копии программы в свой личный кабинет JSFiddle.
3. Чтобы изменить вид левого глаза Дугласа, введите на панели JavaScript следующий код.

   ```
   document.getElementById("lefteye").style.backgroundColor = "purple";
   ```

4. Щелкните на кнопке Run и просмотрите результат своих деяний.

   Левый глаз робота приобретает фиолетовый оттенок.

5. Нажмите <Return> (в Mac) или <Enter> (в Windows), чтобы создать новую строку кода на панели JavaScript. Введите следующий код.

```
document.getElementById("head").style.transform = "rotate(15deg)";
```

6. Щелкните на кнопке Run для просмотра результатов выполнения инструкции.

   Дуглас наклоняет голову влево. Учитывая вытянутую в том же направлении руку, можно представить, что он выполняет танцевальное па (рис. 7.2).

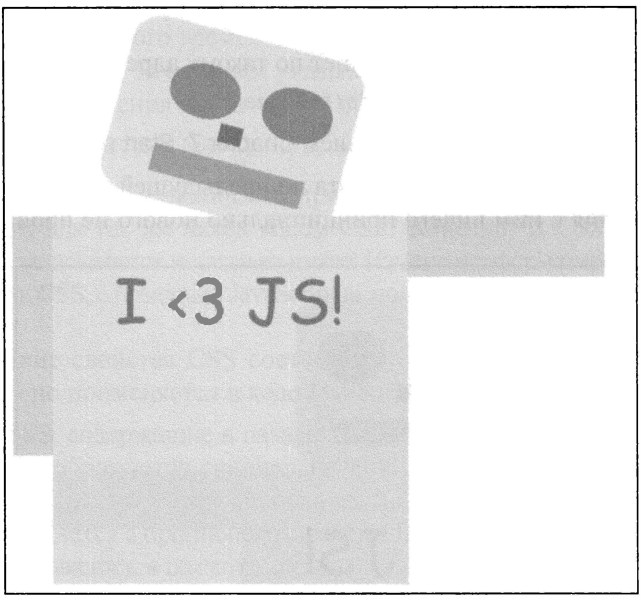


Рис. 7.2. Дуглас начал танцевать

7. На левой панели щелкните на команде Fiddle Options и дайте программе новое название.

8. Щелкните на кнопке Update, а затем выберите в раскрывающемся меню команду Set as Base для сохранения текущей версии программы в общедоступном кабинете.

Дальнейшие эксперименты

Изменяя с помощью JavaScript одни только свойства элементов, вы получаете неограниченные возможности по управлению роботом Дугласом. Ниже приведены только некоторые, самые очевидные действия, описание которых вы найдете в комментариях к соответствующим инструкциям. Не поленитесь протестировать каждую из них.

```
// Добавление к туловищу сплошной черной границы толщиной 2 px
document.getElementById("body").style.border = "2px black solid";
// Скругление уголков рта
document.getElementById("mouth").style.borderRadius = "4px";
// Обводка правого глаза контуром, состоящим из желтых точек
document.getElementById("righteye").style.border = "4px yellow dotted";
// Изменение цвета левой руки
document.getElementById("leftarm").style.backgroundColor = "#FF00FF";
// Окрашивание надписи в другой цвет
document.getElementById("body").style.color = "#FF0000";
// Создание прически
document.getElementById("head").style.borderTop = "5px black solid";
```

Теперь ваша очередь придумывать, как еще можно изменить внешний вид и поведение Дугласа. Наряду с собственными идеями попробуйте с помощью JavaScript заставить робота выполнить следующие действия:

- ✓ наклоните голову Дугласа в противоположном направлении;
- ✓ сделайте нос робота круглым;
- ✓ окрасьте правую руку Дугласа в зеленый цвет;
- ✓ накрасьте губы роботу розовой помадой.

Если вам не совсем понятно, как выполнить предложенные задачи, то посетите в JSFiddle общедоступный кабинет и просмотрите программу Chapter 7: Changing CSS with JS.

Не забывайте о том, что JavaScript не прощает ошибок. Если вы сделаете опечатку в ключевом слове, пропустите точку или введете символ не в том регистре, то перестанет выполняться вся программа, введенная на панели JavaScript, а не только инструкция, содержащая ошибку.

Танцующий робот

Теперь, когда вам известно, как правильно изменять код CSS с помощью языка программирования JavaScript, давайте используем полученные знания для чего-то впечатляющего — заставим робота плясать.

1. В приложении JSFiddle откройте последнюю версию программного робота (см. ссылку Chapter 7: Start в общедоступном кабинете) и щелкните на кнопке Fork, чтобы скопировать ее в личный кабинет.
2. На левой панели щелкните на команде Fiddle Options и измените имя проекта на Animated Robot (Движущийся робот).
3. Для сохранения проекта щелкните на кнопке Update.

   Вы только что создали копию робота, которого будете обучать танцам. Если что-то пойдет не так, всегда можно вернуться к исходному варианту Дугласа (в общедоступном кабинете) и попробовать еще раз.

Стоит признать, что танцор из Дугласа неважный, но отдельные движения выучить он все же сможет. Первое, чему мы его научим, — это "вздергиванию" глаз вверх. Далеко не самое достойное занятие для робота, но для закрепления полученных знаний вполне сгодится. Оно очень простое и заключается в быстром перемещении глаз вверх и в последующем медленном их опускании вниз. Поверьте, под музыку это движение действует просто гипнотически.

На рис. 7.3 показан робот у которого вверх приподнят только один глаз.

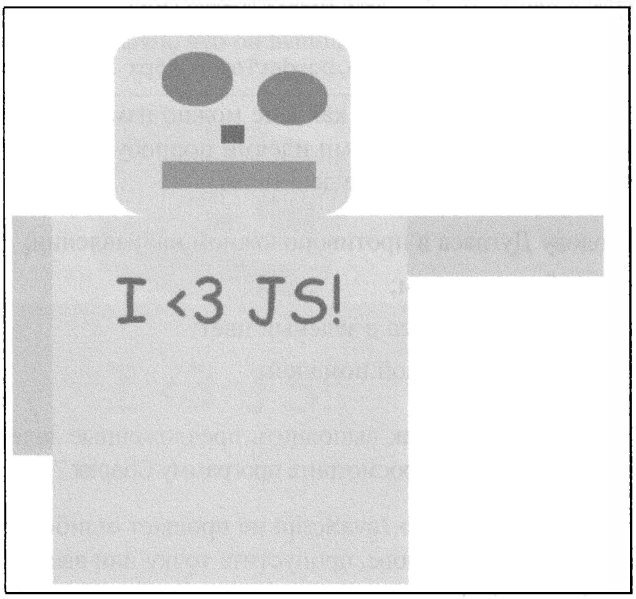


Рис. 7.3. Дуглас умеет вращать глазами

В создаваемой далее программе мы попробуем добиться того, что движения отдельных частей тела робота будут выполняться только после щелчка на них мышью. Начнем со все того же правого глаза — давайте сделаем так, чтобы после щелчка на нем мышью он поднимался вверх.

1. На панели JavaScript введите следующий код.

   ```
   var rightEye = document.getElementById("righteye");
   ```

   Мы только что объявили переменную, которая будет представлять в коде JavaScript правый глаз Дугласа. Отныне вам не нужно каждый раз использовать конструкцию `document.getElementById("righteye")` — вместо нее достаточно ввести переменную `rightEye`.

2. Отслеживание щелчков мышью на правом глазе робота осуществляется с помощью такого JavaScript-кода.

   ```
   rightEye.addEventListener("click", moveUpDown);
   ```

События

За отслеживание событий, которые возникают при взаимодействии пользователя с элементами веб-страницы, отвечает специальная программная конструкция, называемая прослушивателем событий. Воспользовавшись ею, можно определить действие, которое будет выполняться в ответ на возникновение того или иного события.

В JavaScript реакция программы на событие устанавливается с помощью метода `addEventListener()`. При распознавании и обработке событий в JavaScript мы будем оперировать следующими понятиями.

- ✓ **Событие.** Под *событием* подразумевается действие, происходящее в браузере. Самые известные события, обрабатываемые программами, — это щелчок мышью, нажатие клавиши, перетаскивание объектов, наведение указателя мыши на элемент и выделение текста.

  К событиям необязательно относят действия, выполняемые в браузере пользователями. Некоторые события происходят без участия человека: загрузка страницы, отображение определенного элемента, возникновение ошибки или завершение анимации. Ниже приведен краткий список наиболее значимых для браузера событий.

  - `click`. Щелчок пользователя мышью.
  - `submit`. Отправка данных формы.
  - `drag`. Перетаскивание элемента мышью.
  - `drop`. Отпускание элемента после перетаскивания.
  - `copy`. Копирование пользователем некого содержимого.
  - `paste`. Вставка скопированного ранее содержимого.
  - `mouseover`. Наведение пользователем мыши на элемент
  - `load`. Загрузка страницы.

  Чтобы прослушивать определенное событие, подставьте его название в метод `addEventListener()`, предварительно заключив в кавычки.

  ```
  rightEye.addEventListener("click", listener);
  ```

- ✓ **Приемник.** Определив событие, необходимо снабдить целевой объект методом `addEventListener()`, отвечающим за его отслеживание. В частности, для отслеживания правым глазом робота событий щелчка мышью применяется следующий код.

  ```
  rightEye.addEventListener("click", listener);
  ```

  Элемент или объект, который отслеживает событие, указанное в обработчике, называется приемником событий. В нашем случае в качестве приемника событий выступает элемент `rightEye` (как видите, в этой переменной хранится значение, возвращенное методом `document.getElementById("righteye")`).

- ✓ **Обработчик.** Третий компонент метода `addEventListener()` — это обработчик событий. Он представляется объектом, реагирующим на наступление события.

  В нашем случае в качестве обработчика выступает функция, которую нам еще предстоит написать, с названием `moveUpDown`. В задачи этой функции входит перемещение глаз робота вверх и вниз.

  ```
  rightEye.addEventListener("click", moveUpDown);
  ```

  Подытожив вышесказанное, можно представить обработки события в следующем виде.

  приемник`.addEventListener("`*событие*`", `*обработчик*`);`

Теперь, когда вы знаете, что такое событие и как оно обрабатывается, давайте создадим и применим обработчик событий, который запускается в ответ на щелчок на глазе робота Дугласа.

Код обработки события

На текущий момент на панели JavaScript должен содержаться следующий код.

```
var rightEye = document.getElementById("righteye");
rightEye.addEventListener("click", moveUpDown);
```

Если запустить программу в существующем виде, то легко обнаружить, что ничего не происходит. А все потому, что мы еще не создали обработчик события.

Обработчик — это, как правило, функция или небольшая программа, включенная в общую, бо́льшую программу JavaScript. Данная функция запускается только тогда, когда происходит событие, которое отслеживает приемник.

В нашем проекте обработчик (функция `moveUpDown()`) отвечает за анимацию правого глаза робота Дугласа. Для создания функции обработки события выполните следующие действия.

1. Перейдите на панель JavaScript и установите курсор в конце последней строки, после чего нажмите <Return> (в Mac) или <Enter> (в Windows).
2. Введите в новой строке такую инструкцию.

   ```
   function moveUpDown(e) {
   ```

   С нее начинается код новой функции, и ей дается название. Обратите внимание на параметр `e`, приведенный в скобках после названия функции. При вызове функции методом `addEventListener()` параметр `e` представляет информацию о происходящем событии, которая обрабатывается внутри функции. Подробно об этом — далее.
3. Нажмите в конце строки (после символа `{`) клавишу <Return> (в Mac) или <Enter> (в Windows). Введите следующий код.

   ```
   var robotPart = e.target;
   var top = 0;
   ```

```
    var id = setInterval(frame, 10) // прорисовка каждые 10 мс
    function frame() {
       robotPart.style.top = top + '%';
       top++;
       if (top === 20){
          clearInterval(id);
       }
    }

}
```

На первый взгляд, приведенный выше код кажется сложным, но на самом деле он предельно прост. Перед тем как приступить к его анализу, давайте выполним программу и посмотрим на возвращаемые ею результаты.

4. Щелкните на кнопке **Run**.

   Если все введено правильно, то, щелкнув на правом глазе робота, вы заметите, что он смещается вверх, как показано на рис. 7.4.

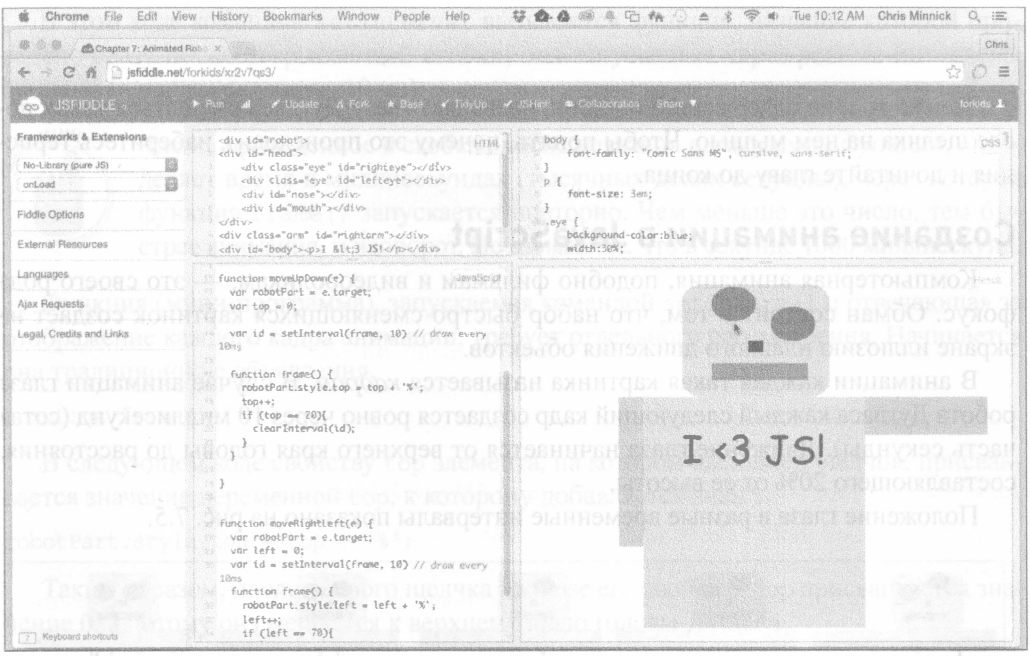


Рис. 7.4. То ли удивление, то ли заигрывание — робот хитро вздергивает правый глаз вверх

Если программа отказывается работать так, как описано выше, то проверьте ее код на наличие опечаток и ошибок. Он должен в точности повторять код, приведенный в листинге 7.1.

Листинг 7.1. Код JavaScript для вздергивания правого глаза робота вверх

```
var rightEye = document.getElementById("righteye");
rightEye.addEventListener("click", moveUpDown);

function moveUpDown(e) {
   var robotPart = e.target;
   var top = 0;

   var id = setInterval(frame, 10) // прорисовка каждые 10 мс

   function frame() {
      robotPart.style.top = top + '%';
      top++;
      if (top === 20){
         clearInterval(id);

      }
   }

}
```

Только код, не содержащий ошибок, позволяет глазу Дугласа смещаться вверх после щелчка на нем мышью. Чтобы понять, почему это происходит, наберитесь терпения и дочитайте главу до конца.

Создание анимации в JavaScript

Компьютерная анимация, подобно фильмам и видеороликам, — это своего рода фокус. Обман состоит в том, что набор быстро сменяющихся картинок создает на экране иллюзию плавного движения объектов.

В анимации каждая такая картинка называется *кадром*. В случае анимации глаза робота Дугласа каждый следующий кадр создается ровно через 10 миллисекунд (сотая часть секунды). Движение глаза начинается от верхнего края головы до расстояния, составляющего 20% от ее высоты.

Положение глаза в разные временные интервалы показано на рис. 7.5.

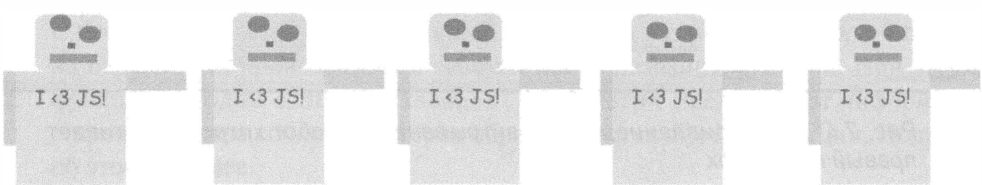


Рис. 7.5. Движение глаза в процессе анимации

Теперь давайте рассмотрим код JavaScript, отвечающий за анимацию глаза, и проанализируем, как достигается желаемый эффект. Сразу после строки объявления функции расположен следующий код.

```
var robotPart = e.target;
```

В этом коде с помощью объекта события (автоматически возвращаемого методом `addEventListener()`) определяется та часть тела робота (элемент HTML), на которой пользователь щелкает мышью. Сведения об этом элементе (в нашем случае — `rightEye`) сохраняются в новой переменной `robotPart`.

Следующий код создает новую переменную `top`, в которой хранится значение `0`.

```
var top = 0;
```

В переменной `top` указывается положение глаза в каждом кадре анимации.

Приведенная далее строка кода содержит команду `setInterval`, отвечающую за смену кадров на экране.

```
var id = setInterval(frame, 10); // прорисовка каждые 10 мс
```

В этом коде командой `setInterval` вызывается функция, название которой приведено сразу после открывающей скобки; она запускается через равные интервалы времени (в нашем случае — 10 мс).

Число, приведенное в скобках после названия вызываемой функции, определяет время в миллисекундах (тысячных долях секунды), через которое функция `frame()` запускается повторно. Чем меньше это число, тем быстрее анимация, и наоборот. Не забывайте, что в секунде 1000 миллисекунд.

Функция (мини-программа), запускаемая командой `setInterval` и отвечающая за отображение каждого кадра анимации, требует отдельного рассмотрения. Начинается она традиционно с объявления.

```
function frame() {
```

В следующем коде свойству `top` элемента, на котором выполнен щелчок, присваивается значение переменной `top`, к которому добавляется знак `%`.

```
robotPart.style.top = top + '%';
```

Таким образом, после первого щелчка на глазе его свойству `top` присваивается значение 0, а потому он смещается к верхнему краю головы Дугласа.

В следующем коде значение переменной `top` увеличивается на единицу. Для этого используется оператор *инкремента*.

```
top++;
```

Детально об этом операторе мы еще поговорим в главе 9.

Далее проверяется, достигнут ли последний, двадцатый кадр анимации. В операторе сравнения проверяется условие равенства переменной `top` числу `20`.

```
if (top === 20){
```

Только в случае истинности этого условия выполняется следующий оператор, определяющий конец анимации. Завершение анимации устанавливается командой `clearInterval`.

```
clearInterval(id);
```

В конце нам нужно завершить все начатые ранее блоки кода, для чего используются закрывающие фигурные кавычки.

```
        }
    }
}
```

Анимация других элементов

После написания кода функции `moveUpDown()`, использующейся для анимации правого глаза, настройка анимации для второго глаза будет сводиться к добавлению в программу еще одного обработчика события.

1. Чтобы сохранить написанный выше код, щелкните в верхней части окна на кнопке **Update**.
2. Под оператором объявления переменной `rightEye` добавьте еще одно объявление — переменной, представляющей левый глаз.

   ```
   var leftEye = document.getElementById("lefteye");
   ```

3. Теперь добавьте еще один обработчик событий, как раз под оператором обработки события для правого глаза.

   ```
   leftEye.addEventListener("click", moveUpDown);
   ```

4. Щелкните на кнопке **Run**.

   Снабдив программу всеми описанными выше операторами, вы заставите ее запускать анимацию как для правого, так и для левого глаз. Анимация для левого глаза выполняется с помощью все той же функции `moveUpDown()`.

Добавление еще одной функции анимации

Робот Дуглас умеет не только вздергивать глаза. В его репертуаре как минимум еще одно па. Конечно, оно не столь эффектное, как у профессиональных танцоров, и заключается в плавном повороте вытянутой левой руки справа налево (синхронно с движением глаза).

Для воссоздания такого движения роботом нам понадобится еще один обработчик событий, в котором задействуется измененная функция `moveUpDown()`, обеспечивающая движение руки справа налево, а не сверху вниз.

1. Создайте новую переменную под двумя уже имеющимися, введя следующий код. Она будет представлять левую руку робота.

   ```
   var leftArm = document.getElementById("leftarm");
   ```

2. Напишите новый обработчик событий, расположив его под имеющимися.
   ```
   leftArm.addEventListener("click", moveRightLeft);
   ```
3. Скопируйте код функции `moveUpDown()`, выделив его и нажав комбинацию клавиш <⌘+C> (в Mac) или <Ctrl+C> (в Windows).
4. Вставьте копию кода функции `moveUpDown()` под его оригиналом.

Теперь вам предстоит поработать над копией кода функции `moveUpDown()`, изменив ее так, чтобы обеспечить горизонтальное движение элемента, а не вертикальное.

1. Переименуйте копию функции `moveUpDown()` в `moveRightLeft()`.
   ```
   function moveRightLeft(e) {
   ```
2. В коде функции измените вторую строку следующим образом.
   ```
   var left = 0;
   ```
3. Внесите изменения в первую строку вложенной функции `frame()`, заменив ее такой инструкцией.
   ```
   robotPart.style.left = left + '%';
   ```
4. Во второй строке функции `frame()` введите следующий код.
   ```
   left++:
   ```
5. Измените третью строку кода функции `frame()` на такую.
   ```
   if (left === 70){
   ```

Завершив редактирование кода функции `moveRightLeft()`, вы должны получить программу, приведенную в листинге 7.2.

Листинг 7.2. Код функции `moveRightLeft`

```
function moveRightLeft(e) {
   var robotPart = e.target;
   var left = 0;
   var id = setInterval(frame, 10) // прорисовка каждые 10 мс
   function frame() {
      robotPart.style.left = left + '%';
      left++;
      if (left === 70){
         clearInterval(id);
      }
   }
}
```

Для сохранения измененной программы и ее тестирования щелкните на кнопке **Update**, расположенной в верхней части окна. Начиная с этого момента щелчок на левой руке робота Дугласа приводит к отображению анимации, показанной на рис. 7.6.

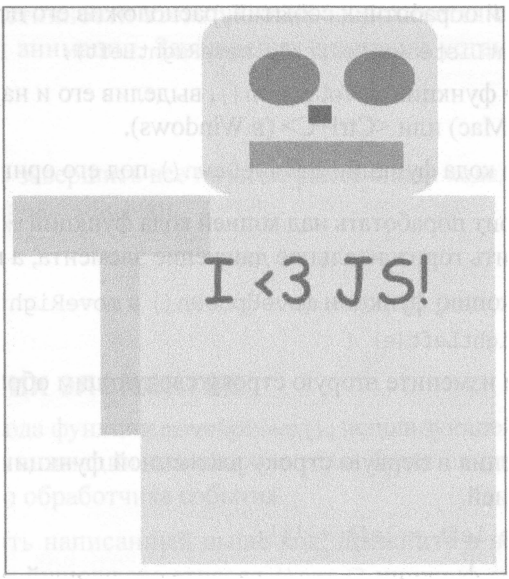


Рис. 7.6. В танце Дуглас двигает руками

По завершении модификации код всей программы анимации робота должен выглядеть так, как показано в листинге 7.3.

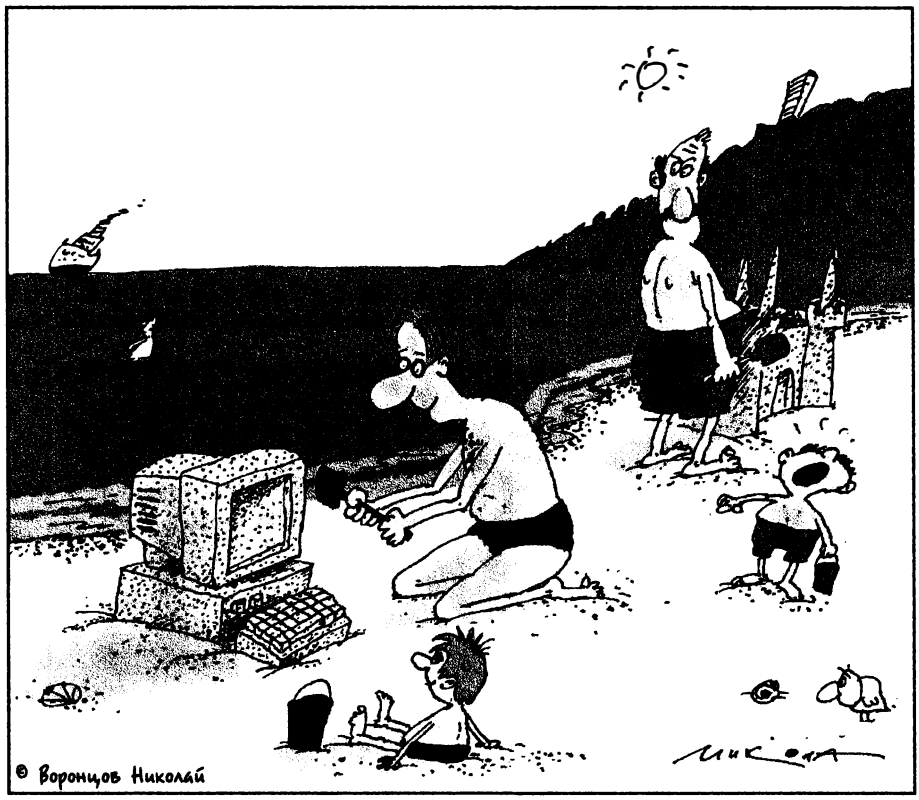
Листинг 7.3. Код JavaScript, обеспечивающий танцевальные движения робота рукой и глазами

```
var rightEye = document.getElementById("righteye");
var leftEye = document.getElementById("lefteye");
var leftArm = document.getElementById("leftarm");

rightEye.addEventListener("click", moveUpDown);
leftEye.addEventListener("click", moveUpDown);
leftArm.addEventListener("click", moveRightLeft);

function moveUpDown(e) {
   var robotPart = e.target;
   var top = 0;
   var id = setInterval(frame, 10) // прорисовка каждые 10 мс
   function frame() {
      robotPart.style.top = top + '%';
      top++;
      if (top === 20){
         clearInterval(id);
      }
   }
}
```

```
function moveRightLeft(e) {
   var robotPart = e.target;
   var left = 0;
   var id = setInterval(frame, 10) // прорисовка каждые 10 мс
   function frame() {
      robotPart.style.left = left + '%';
      left++;
      if (left === 70){
         clearInterval(id);
      }
   }
}
```

Настал ваш черед обучить Дугласа новым танцевальным движениям. Включите ритмичную музыку и щелкайте в такт ей на глазах и руке робота! В завершение добавьте в обработчик событий код управления другими частями тела Дугласа, например носа, рта и, наконец, правой руки.

Часть III
Выполнение действий



В этой части...

- ✓ Автомобиль мечты и операнды
- ✓ Выполнение действий с помощью операторов
- ✓ Игра в слова на языке JavaScript

ГЛАВА

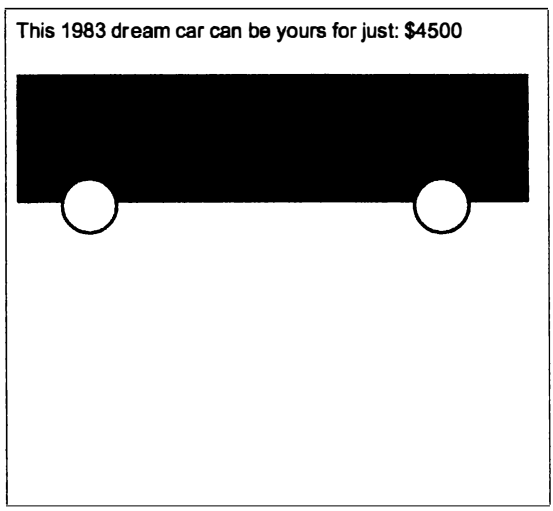
8

Автомобиль мечты и операнды

В повседневной жизни для описания выполняемых действий мы используем существительные и глаголы. Чтобы сделать речь приятнее для собеседника, мы в нее добавляем прилагательные, наречия, местоимения, союзы, предлоги и междометия. При этом основная смысловая нагрузка возлагается преимущественно на существительные и глаголы.

На языке JavaScript лингвистические конструкции, сходные с предложениями, называются инструкциями. Инструкции, в свою очередь, состоят из операндов (аналогичны существительным) и операторов (подобны глаголам).

В этой главе вы познакомитесь с разными типами операндов и научитесь использовать операторы для написания программ на языке программирования JavaScript.



Знакомство с операндами

Выражением в языке программирования JavaScript считается любой фрагмент кода, результат выполнения которого приводит к получению некоего результата.

Под обработкой данных мы понимаем некие действия, приводящие к возврату определенного значения. В качестве наглядного примера рассмотрим следующие операции.

- ✓ Выполнение инструкции `1+1` приводит к возвращению значения `2`. Иными словами, в результате сложения двух величин возвращается значение `2`.
- ✓ К инструкциям также относится код `x=2`. Его выполнение приводит к присвоению переменной `x` значения `2`.

Инструкции строятся на основе операндов (таких, как число `1` или переменная `x`) и операторов (например, `=` или `+`). О типах данных, которые разрешается использовать в операндах JavaScript, речь шла в главе 3. Кроме того, в качестве операндов допустимо применять массивы и объекты.

Как и с какой целью в коде JavaScript создаются объекты, вы узнаете далее. Подробно же о массивах и их назначении в JavaScript рассказывается в главе 11.

Использование разных типов данных в операндах лучше всего изучать на реальных примерах. Далее мы предлагаем вам самостоятельно отгадать, к какому типу данных относятся предложенные нами операнды JavaScript. Вам будет предоставлен ряд подсказок, которые помогут сориентироваться в остальных случаях. В дальнейшем вам придется принимать самостоятельные решения, хотя правильные ответы мы все же приведем ниже.

Итак, начнем! Определите, к какому типу данных — числовому, строковому или булеву — относятся приведенные ниже операнды.

- ✓ `100`. Вполне очевидно, что данный операнд относится к числовому типу данных. Он не заключен в кавычки и содержит одни лишь цифры.
- ✓ `"Привет, мир JavaScript!"`. Все заключенные в кавычки символы относятся к строкам. Таким образом, этот операнд представлен строковым типом данных.
- ✓ `false`. Определенно, булев тип данных. Кавычки отсутствуют, а ключевое слово `false` относится исключительно к булевым значениям.
- ✓ `"true"`. Несмотря на наличие ключевого слова `true`, это значение не относится к булевым, поскольку заключено в кавычки, указывающие на строковый тип данных.

Настал ваш черед определить тип данных следующих операндов.

- ✓ `186`
- ✓ `"007"`

- ✓ `"Число 9"`
- ✓ `true`
- ✓ `86`
- ✓ `"Полдень наступает в 12:00"`

Правильно ли вы определили типы данных операндов, можно узнать, сверившись с приведенными ниже ответами.

- ✓ `186` относится к числам.
- ✓ `"007"` представляет строку.
- ✓ `"Число 9"` — также строка.
- ✓ `true` соответствует булеву типу данных.
- ✓ `86` — однозначно число.
- ✓ `"Полдень наступает в 12:00"` относится к строкам.

Далеко не всегда операнды представляются литеральными значениями. На самом деле чаще всего программисты используют в качестве операндов переменные, которым впоследствии назначают необходимые значения. Ниже приведено несколько примеров таких операций. Постарайтесь правильно определить в каждом из случаев тип данных переменной (булев, числовой или строковый) и соответствующего операнда.

- ✓ `distance = 3000`
- ✓ `distance = 800 * 4`
- ✓ `doTheMath = 7 + 8 + 36 + 18 + 12`
- ✓ `countrySong = "мама" + "авто" + "дождь" + "дом"`

Если вы утверждаете, что три первые переменные имеют числовой тип данных, а последняя — строковый, то абсолютно правы!

Чтобы проверить правоту нашего утверждения, выполните следующее упражнение, позволяющее убедиться в этом самостоятельно.

1. Запустите браузер Chrome и откройте в нем консоль JavaScript (рис. 8.1), нажав <⌘+C> (в Mac) или <Ctrl+C> (в Windows).
2. Тип данных на консоли определяется с помощью команды `typeof`.

   В случае числа она имеет вид, подобный приведенному в следующем примере.

   ```
   typeof 8
   ```

   В качестве результата на консоли отображается значение `"number"`, как показано на рис. 8.2.

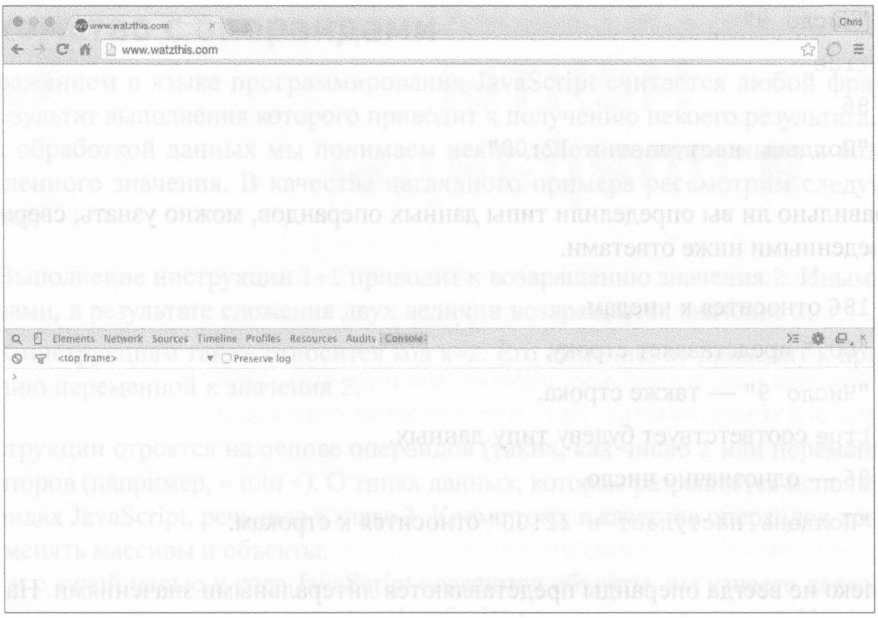


Рис. 8.1. Консоль JavaScript в Chrome

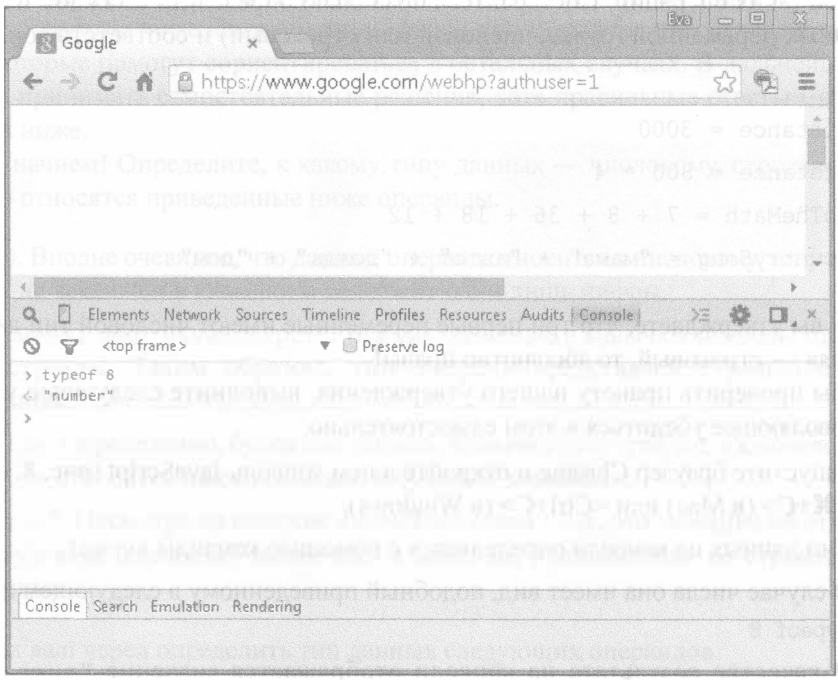


Рис. 8.2. Определение типа данных операнда

3. С помощью следующей инструкции создайте новую переменную и присвойте ей значение.

   ```
   var doTheMath = 7 + 8 + 36 + 18 + 12
   ```

   На консоль выводится запись `undefined`, сообщая об успешном выполнении кода.

4. Определите тип только что созданной переменной, используя уже известную вам команду `typeof`.

   ```
   typeof doTheMath
   ```

   В качестве результата на консоли вновь отображается значение `"number"`.

5. Для проверки правильности определения на консоли строкового типа данных введите следующий код.

   ```
   typeof "моя любимая программа"
   ```

   Конечно, в данном случае возвращается значение `"string"`.

6. Теперь попробуйте определить тип данных для текста, не заключенного в кавычки.

   ```
   typeof automobile
   ```

   Результат вполне ожидаемый — `"undefined"`. Значение, введенное без кавычек, по умолчанию определяется как имя переменной. Поскольку ранее в коде переменная `automobile` не объявлялась, то и данные, хранящиеся в ней, имеют неопределенный тип (`undefined`).

Управление объектами

В дополнение к числам, строкам и булевым значениям операнды JavaScript могут содержать объекты, представленные отдельным типом данных. Как уже рассказывалось в главе 5, объекты JavaScript обладают *свойствами* (задающими характеристики объекта) и методами (определяющими возможности объекта).

Самое время заняться созданием объектов в собственном коде и управлением ими! Чтобы создать объект, предварите его имя, как и в случае с переменной, ключевым словом `var`, после которого введите знак равенства (=).

```
var myObject =
```

Все, что указывается после знака равенства, в корне отличается от известного вам по команде объявления переменной. На то, что далее в коде создается объект, указывают обязательные фигурные скобки (`{ }`).

```
var myObject = {};
```

Внутри таких скобок указываются свойства и методы, присущие объекту. Каждый из них определяется как пара *имя*: *значение*. Если объект имеет несколько свойств и методов, то они отделяются в фигурных скобках символами запятых.

Для практического закрепления полученных знаний об объектах давайте воспользуемся приложением JSFiddle.

1. Запустите браузер и загрузите в нем сайт `http://jsfiddle.net`.

   Начните с чистого листа, убедившись, что все четыре панели в окне браузера пусты и не содержат кода (HTML, CSS и JavaScript), а также результатов его выполнения.

2. На панели JavaScript введите следующий код, отвечающий за создание объекта `dreamCar`.

```
var dreamCar = {
   make: "Oldsmobile",
   model: "98",
   color: "brown",
   year: 1983,
   bodyStyle: "Luxury Car",
   price: 4500
}
```

   Не бойтесь указывать в объекте собственные свойства и методы, если они выглядят полезнее созданных нами.

Как видите, объект идеально подходит для совмещения в одном элементе нескольких свойств с разными типами данных. В частности, объект `dreamCar` снабжается строковыми свойствами `make` (производитель), `model` (модель), `color` (цвет) и `bodyStyle` (тип кузова), а также числовыми свойствами `year` (год производства) и `price` (цена).

3. После кода определения объекта введите на панели JavaScript такое выражение. Оно позволит вам узнать тип данных объекта `dreamCar`.

```
alert("Тип данных dreamCar: " + typeof dreamCar);
```

4. Щелкните в верхней части окна на кнопке Run.

   В результате выполнения кода JavaScript на экране появится окно сообщения, в котором указывается, что `dreamCar` относится к объектам (рис. 8.3).

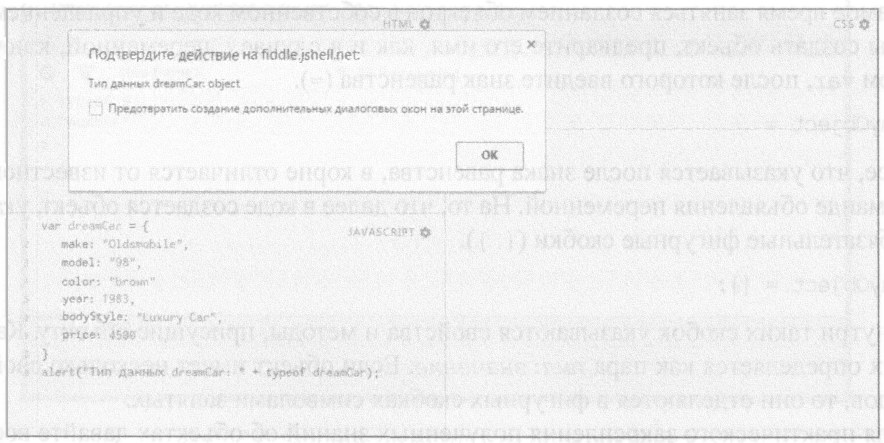


Рис. 8.3. Переменная `dreamCar` — объект!

Конфигурирование автомобиля мечты

Разговоры в сторону! Давайте комплектовать автомобиль мечты всем необходимым и выводить его на скоростные дороги.

Создание автомобиля в HTML отличается от его производства на заводском конвейере. Начинается оно с создания фрейма.

1. На панели HTML в приложении JSFiddle введите следующий код.

```
<div id="car">
   Твой автомобиль мечты <span id="modelyear"></span> стоит:
       $<span id="pricetag"></span>
   <div id="body">
      <div id="frontwheel"></div>
      <div id="backwheel"></div>
   </div>
</div>
```

Вы только что создали базовую структуру своего автомобиля мечты, снабдив ее ценником. На следующих этапах мы ее слегка усовершенствуем.

2. На панели CSS введите такой код.

```
#car {
   font-family: Arial;
}
#body {
   position: absolute;
   top: 50px;
   width: 80%;
   height: 100px;
   background-color: #000000;
   text-align: center;
}
#backwheel {
   position: absolute;
   left: 10%;
   top: 130px;
   background-color: #ffffff;
   border: 3px solid black;
   border-radius: 50%;
   width: 40px;
   height: 40px;
}
#frontwheel {
   position: absolute;
   left: 50%;
   top: 130px;
   background-color: #ffffff;
   border: 3px solid black;
   border-radius: 50%;
   width: 40px;
   height: 40px;
}
```

Замечательно! Автомобиль создан и стилизован: кузов, колеса и даже цена — все на своих местах.

3. Щелкните в верхней части окна на кнопке Run, чтобы увидеть результат своих трудов.

   На четвертой панели окна отображается ваш автомобиль в базовой комплектации, как показано на рис. 8.4.

Рис. 8.4. Самая базовая комплектация

Нам предстоит модифицировать автомобиль мечты, улучшив его с помощью JavaScript.

1. На панели JavaScript создайте объект `dreamCar`.

   Мы использовали следующий код.

```
var dreamCar = {
   make: "Oldsmobile",
   model: "98",
   color: "Коричневый",
   year: 1983,
   bodyStyle: "Luxury Car",
   price: 4500
};
```

2. Введите инструкцию, отвечающую за обновление цены при запуске программы.

```
document.getElementById("pricetag").innerHTML = dreamCar.price;
```

3. Следующая команда обеспечивает обновление года производства автомобиля.

```
document.getElementById("modelyear").innerHTML = dreamCar.year;
```

4. Наконец, обновление цвета автомобиля выполняется с помощью такой инструкции.

```
document.getElementById("body").style.backgroundColor = dreamCar.
   ↳color;
```

5. Осталось написать код вывода названия производителя и модели автомобиля на боковой поверхности кузова.

```
document.getElementById("body").innerHTML = dreamCar.make + " " +
   ↳dreamCar.model;
```

6. Для выполнения кода обновления автомобиля мечты щелкните на кнопке Run. Результат не замедлит появиться на четвертой панели окна (рис. 8.5).

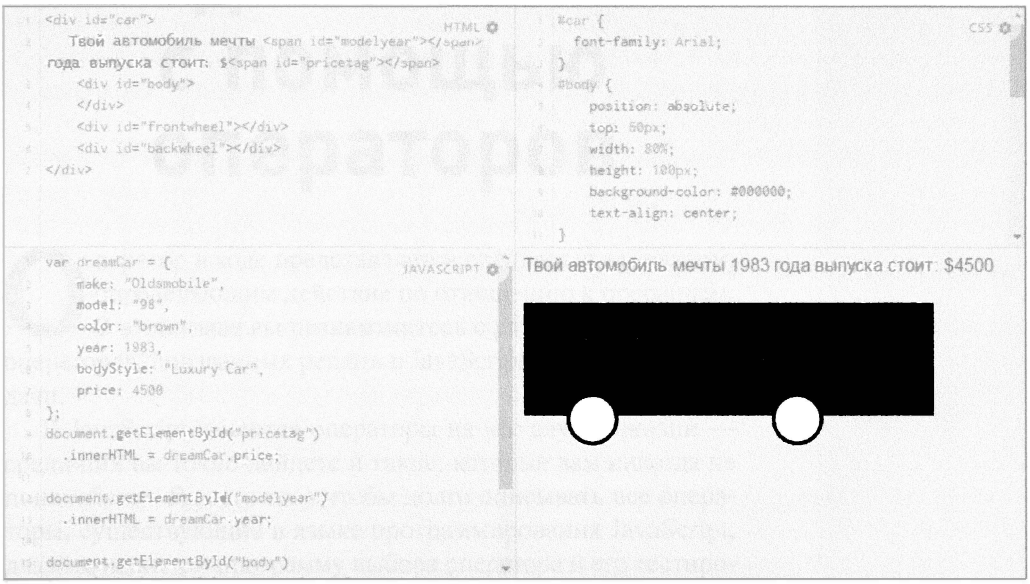


Рис. 8.5. Так и хочется прокатиться на новой модели!

Попробуйте самостоятельно изменить настройки автомобиля и обновить его вид в приложении JSFiddle, выполнив полученный код JavaScript. Обратите внимание на то, как каждое из свойств объекта и CSS влияет на внешний вид автомобиля, отображаемого на веб-странице.

ГЛАВА

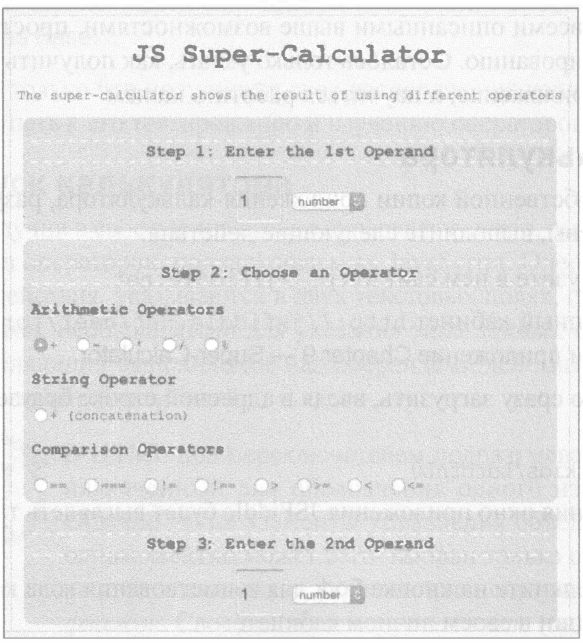
9

Выполнение действий с помощью операторов

Оператор в коде представляется специальным значком, определяющим действие по отношению к операндам. В этой главе вы познакомитесь с различными типами операторов, призванных решать в JavaScript самые разные задачи.

В JavaScript имеются операторы на все случаи жизни — среди них вы точно найдете и такие, которые вам никогда не понадобятся. Вместо того чтобы долго описывать все операторы, существующие в языке программирования JavaScript, давайте напишем программу выбора оператора и его тестирования на предложенных операндах.

Наша программа будет скромно называться “Super-Calculator”, а обрабатывать она будет как числа, так и строки.



Создаем суперкалькулятор

Скорее всего, вам не доводилось встречать калькулятор, подобный тому, разработкой которого мы сейчас займемся. Как и любой другой калькулятор, он умеет выполнять над числами базовые математические операции — складывать, вычитать, умножать и делить. Но в отличие от них, наш калькулятор способен на нечто необычное — возвращать остаток деления одного числа на другое.

Но и это еще не все! Наш суперкалькулятор умеет объединять слова. В частности, если взять слово "Java" и сложить его со словом "Script", то будет образовано хорошо известное вам название — "JavaScript". Обычные калькуляторы не выполняют действий над отдельными словами, но только не наше приложение. Невероятно, но факт!

Более того, с помощью суперкалькулятора можно объединять не только слова, но целые фразы и даже предложения. Эта операция в программировании называется *конкатенацией* строк. Не стоит воспринимать оператор конкатенации как нечто необычное — его действие необычайно простое и во многом напоминает сложение в математике. В коде JavaScript он даже представлен таким же оператором, как и сложение в арифметике. Поставив оператор конкатенации между двумя строковыми значениями, вы получите строку, содержащую текст обеих исходных строк.

Кроме описанных выше, в арсенал суперкалькулятора входят операторы сравнения. С их помощью, например, можно определить, равно ли число 3 числу 8. К тому же операторы сравнения позволяют узнать большее или меньшее из двух чисел. Заметьте, что в качестве результата оператор сравнения всегда возвращает булево значение — истина (`true`) или ложь (`false`). В такой точной дисциплине, как программирование, нет места приблизительным понятиям в стиле "похоже на" или "немного больше".

Калькулятор, обладающий всеми описанными выше возможностями, просто неоценим при обучении программированию. Осталось только узнать, как получить в свое распоряжение столь мощное приложение, и научиться работать на нем.

Получение копии калькулятора

Для запуска и получения собственной копии приложения-калькулятора, разрекламированного нами в начале главы, выполните следующие действия.

1. Запустите браузер и загрузите в нем сайт `http://jsfiddle.net`.
2. Перейдите в общедоступный кабинет `http://jsfiddle.net/user/forkids/fiddles` и отыщите в нем приложение **Chapter 9 — Super-Calculator**.

   Суперкалькулятор можно сразу загрузить, введя в адресной строке браузера такой URL:

   `http://jsfiddle.net/forkids/LdtbfnL0`

   После загрузки приложения окно приложения JSFiddle будет выглядеть так, как показано на рис. 9.1.
3. В верхней части окна щелкните на кнопке **Fork** для заимствования кода калькулятора и создания его копии в своем личном кабинете.

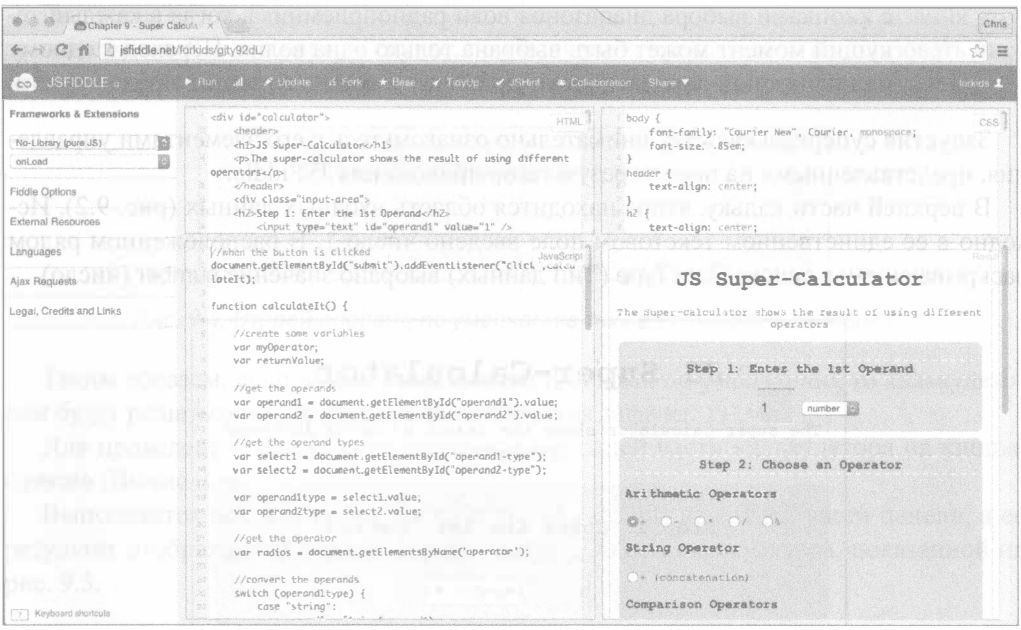


Рис. 9.1. Суперкалькулятор, созданный на языке программирования JavaScript

4. Щелкнув на команде **Fiddle Options**, расположенной на левой панели, введите свое название суперкалькулятора.
5. В верхней части окна щелкните на кнопке **Update**, а затем выберите команду **Set as Base**.

Вы только что получили в свое распоряжение копию суперкалькулятора. Можете приступать к его тестированию и изучению операторов JavaScript.

Запуск калькулятора

Получив в свое распоряжение суперкалькулятор, вы легко изучите назначение каждого из операторов, поддерживаемых JavaScript. Операнды, над которыми выполняются действия, указываются в двух текстовых полях, снабженных раскрывающимися списками, необходимыми для указания типа данных вводимых значений. Оператор, определяющий выполняемое над операндами действие, устанавливается с помощью одного из переключателей.

В HTML под переключателем подразумевают элемент управления, предназначенный для обозначения одного из вариантов входных данных. Обычно такие элементы управления объединяются в группы, и в группе одномоментно может быть выбран только один переключатель. Для выбора сразу нескольких вариантов используются элементы управления типа *флажок*. Свое название *переключатель* (radio button) получил по аналогии

с кнопками выбора диапазонов волн радиоприемника, когда в каждый текущий момент может быть выбрана только одна волна. Переход к одному из диапазонов автоматически приводит к отключению от предыдущего.

Запустив суперкалькулятор, внимательно ознакомьтесь с его элементами управления, представленными на панели результатов приложения JSFiddle.

В верхней части калькулятора находится область входных данных (рис. 9.2). Исходно в ее единственном текстовом поле введено число 1. В расположенном рядом раскрывающемся списке **Data Type** (Тип данных) выбрано значение **number** (число).

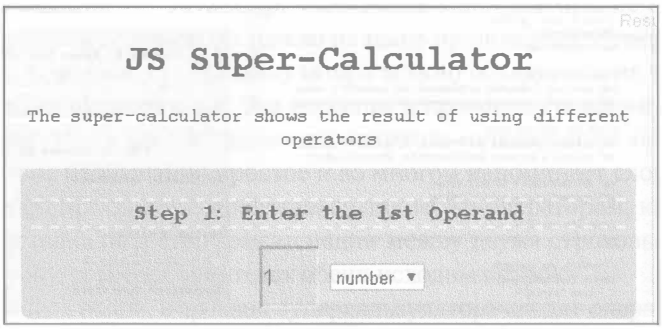


Рис. 9.2. По умолчанию в качестве первого операнда указано число 1

Оператор, выполняющий конечное действие, выбирается чуть ниже — под областью ввода первого операнда. Все операторы калькулятора разделены на три группы: арифметические (**Arithmetic Operators**), строковые (**String Operators**) и сравнения (**Comparison Operators**). В следующих разделах мы детально рассмотрим их все. Здесь же давайте ознакомимся с первым математическим оператором — оператором сложения, представленным в калькуляторе знаком "плюс" (+), как показано на рис. 9.3. Этот оператор выбран в калькуляторе по умолчанию.

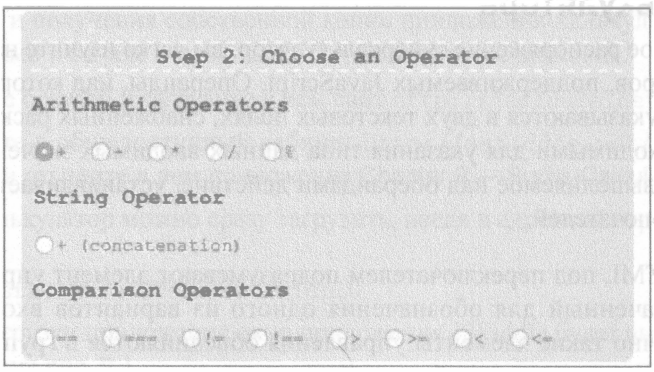


Рис. 9.3. Оператор сложения выбирается автоматически

Под областью выбора задействуемого оператора находится область указания второго операнда. Как и в первом случае, по умолчанию предлагается использовать числовое значение 1 (рис. 9.4).

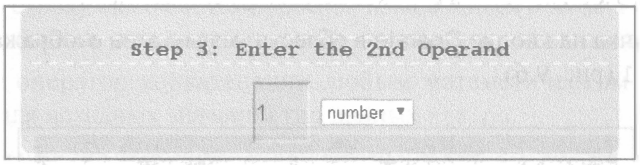


Рис. 9.4. Второй операнд по умолчанию также установлен равным 1

Таким образом, если воспользоваться настройками по умолчанию, то калькулятором будет решаться детсадовская математическая задачка: 1+1=?.

Для проведения вычислений щелкните в нижней части калькулятора на кнопке **Operate** (Вычислить).

Выполняется операция, настройки которой указаны в верхней части панели, а ее результат отображается в области результатов (**Output**) калькулятора, показанной на рис. 9.5.

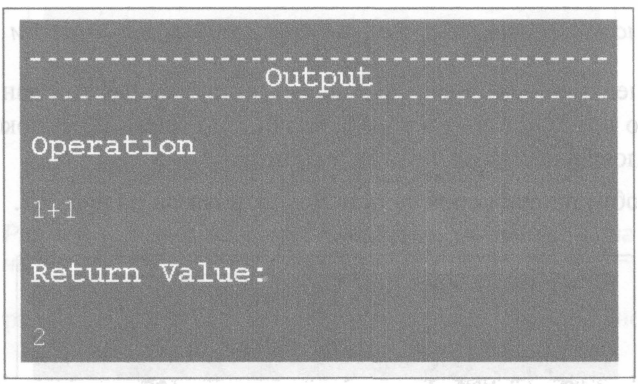


*Рис. 9.5. Результат операции, выполненной калькулятором, отображается в разделе **Output** калькулятора*

Конечно, вам заведомо известен результат этой простейшей арифметической задачи. Взглянув на такой пример использования калькулятора, у вас может сложиться впечатление, что приставка "супер" в названии не имеет никакого отношения к его возможностям. Не торопитесь с выводами! В следующих разделах вы узнаете об истинном назначении программы Super-Calculator.

Математические и строковые операции

Продолжим наше изучение суперкалькулятора, выполняя простые математические операции, в которых принимают участие всего два операнда. Вы узнаете много интересного о вычислениях в коде JavaScript.

1. Оставьте операнды равными значениям, заданным по умолчанию (`1` и `1`). Но вместо математического оператора сложения выберите оператор конкатенации строк, также представленный значком "плюс" (+). Разумеется, каждому из операндов необходимо задать строковый (string) тип данных.

   После щелчка на кнопке Operate в области результатов отображается строковое значение `11` (рис. 9.6).

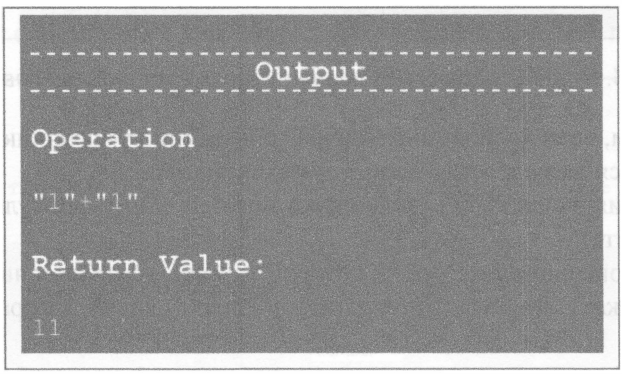


Рис. 9.6. Результат конкатенации строковых значений `1` и `1`

2. Оставьте неизменным оператор конкатенации, а также тип данных обоих операндов, но измените в текстовых полях калькулятора строковые значения. В первом поле введите текст `Java`, а во втором — `Script`.

   Результат объединения срок `Java` и `Script` показан на рис. 9.7.

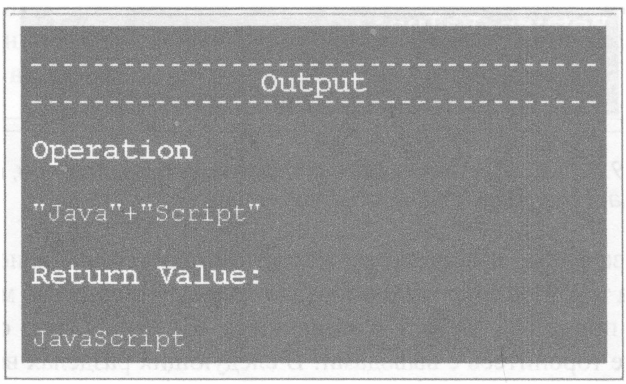


Рис. 9.7. Конкатенация строк `Java` и `Script` приводит к вполне ожидаемому результату

3. Снова оставьте неизменными оператор конкатенации и тип данных операндов, после чего в первом текстовом поле введите свое имя, сопроводив его пробелом в конце, а во втором — свою фамилию.

4. Щелкните на кнопке Operate.

   В качестве результата суперкалькулятор выведет строку, содержащую ваши имя и фамилию, разделенные пробелом. Обратите внимание, что в результате конкатенации в строку включаются пробелы, находящиеся в начале или конце исходных строк (они не игнорируются, но и не добавляются автоматически).

5. Замените оператор конкатенации любым математическим оператором, но оставьте для исходных значений тип данных `string`.

6. Щелкните на кнопке Operate.

   Результат выполнения последней операции показан на рис. 9.8. На панель Output выводится значение `NaN`, означающее Not a Number (не числовое значение). Определив, что суммируемые значения не относятся к числовым, калькулятор, управляемый JavaScript, любезно извещает вас об этом.

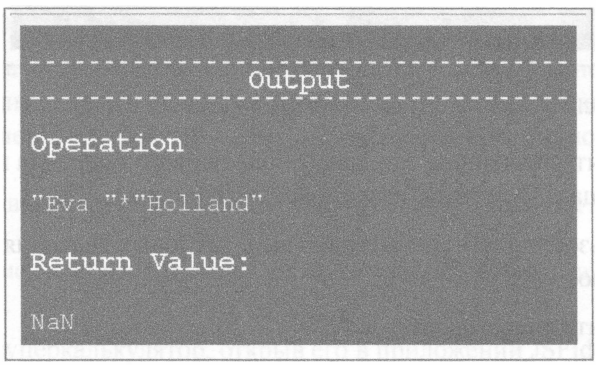


Рис. 9.8. Попытавшись просуммировать нечисловые значения, вы получите результат `NaN`

7. В поле первого операнда введите значение `9` и выберите для него числовой тип данных.

8. Измените в калькуляторе операцию на представленную оператором `%` (остаток от деления по модулю числа).

   В результате определения модуля числа отбрасывается его дробная часть. В нашем случае результат представлен не модулем числа, а остатком, полученным при делении одного из операндов на другой.

9. Измените значение второго операнда на `2`, а его тип — на `number`.

10. Щелкните на кнопке Operate.

    В качестве результата возвращается число `1`. Такой результат не вызывает сомнений, поскольку значение 9 не делится нацело на число 2 без остатка (тогда бы возвращалось `0`). Остаток в данном случае составляет `1`.

11. Измените второй операнд на `2.5`.

    Попробуйте самостоятельно определить результат этой операции.

12. Щелкните на кнопке Operate.

    Результат составит `1.5`, так как 9 делится нацело на 2,5 трижды, а неделимый остаток составляет 1,5, что и отражено на рис. 9.9.

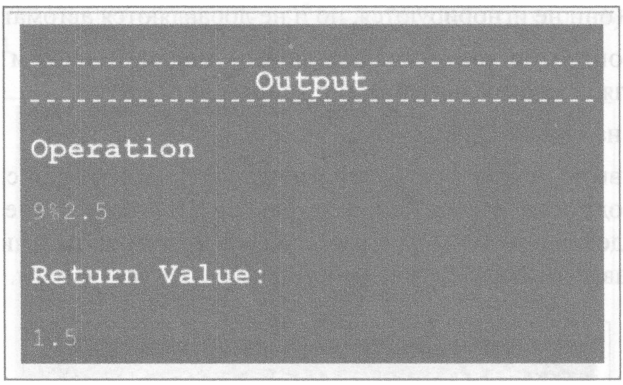


Рис. 9.9. Остаток от деления числа 9 на 2,5 составляет 1,5

13. Измените значение в текстовом поле первого операнда на 1000000000000 (единица с 12 нулями, известная как триллион).

    Использовать десятичные запятые в числовых значениях JavaScript недопустимо.

14. Измените оператор на `*`.

15. В текстовом поле второго операнда также введите значение 1000000000000.

16. Щелкните на кнопке Operate.

    Результат последней операции показан на рис. 9.10. На первый взгляд, он выглядит несколько необычно. А все потому, что значение представлено в *экспоненциальном формате*. Он позволяет записывать большие числовые значения в компактном виде. Чтобы преобразовать число из экспоненциальной формы в обычную, привычную нам со школьной скамьи, необходимо после целой части первого числа добавить его дробную часть столько раз, сколько указано после знака "плюс" (+).

Как видите, умножение триллиона на триллион в JavaScript дает результат `1e+24`. В обычном виде это число преобразуется в единицу с 24 нулями после нее. На приведенном примере прекрасно видно, что экспоненциальная форма представления числа намного удобнее, чем обычная (1000000000000000000000000).

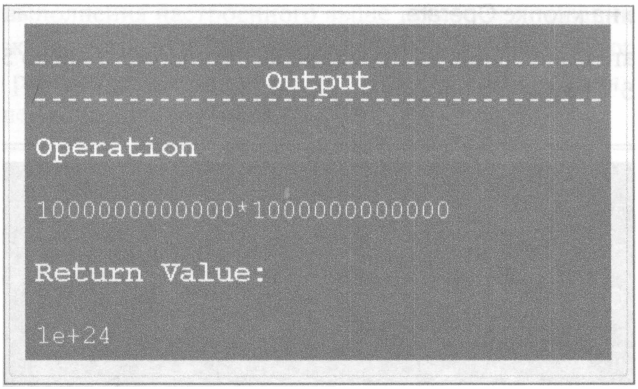


Рис. 9.10. Результат умножения триллиона на триллион

Операции сравнения

В JavaScript операторы сравнения применяются в условных конструкциях, обеспечивающих принятие решения на основе значений переменных. Условные конструкции обычно представлены операторами `if...else`, которые детально будут рассматриваться в главе 14.

Операторы сравнения в качестве результата всегда возвращают значение `true` или `false`.

Давайте вернемся к нашему суперкалькулятору и посмотрим, на что способны операторы сравнения.

1. Запустите суперкалькулятор, открыв его в приложении JSFiddle.
2. В текстовое поле первого операнда введите значение `5`, определив для него числовой (number) тип данных.
3. В списке доступных операторов выберите ==, отвечающий за сравнение двух операндов.
4. Введите в текстовом поле второго операнда такое же числовое значение (`5`) и укажите для него все тот же числовой тип данных.
5. Щелкните на кнопке Operate.

   Результат сравнения двух числовых операндов выводится в области Output калькулятора.

Чудесно! В математике, предельно точной дисциплине, 5 всегда равно 5.

Несколько усложним задачу, чтобы сделать операцию сравнения не столь прямолинейной, как в предыдущем случае.

1. Оставьте в текстовых полях обоих операндов значение `5`, но для одного из них измените тип данных на строковый (string).
2. Убедитесь в том, что выбран оператор равенства (==), а не какой-то другой.

3. Щелкните на кнопке Operate.

   В результате выполнения настроенной выше операции число 5 сравнивается со строкой "5". Результат показан на рис. 9.11.

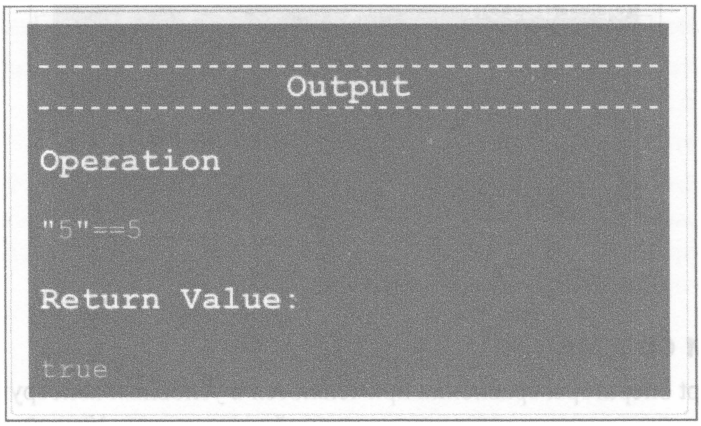


Рис. 9.11. Сравнение числа 5 со строковым значением "5"

Полученный выше результат показывает, что сравнение разнотипных значений, хотя и одинаковых по содержимому, приводит к возвращению значения `true`, указывающего на истинность такого утверждения. Подобное становится возможным благодаря *приведению* данных к общему типу, выполняемому автоматически перед проведением сравнения. В нашем случае строковое значение "5" автоматически преобразуется в число 5 и только после этого сравнивается с другим операндом.

Подобное автоматическое преобразование типов данных часто служит источником сбоев и неполадок в программах. Именно поэтому лучше избегать использования в коде JavaScript операторов равенства. Вместо этого попробуйте применить методику, описанную далее.

В JavaScript поддерживается еще один оператор сравнения, не выполняющий автоматическое приведение типов данных обоих операндов. Чтобы познакомиться с ним, выполните следующие действия.

1. В текстовых полях обоих операндов введите одно и то же число.

   Можно использовать произвольное число, но нам больше нравится 37.

2. Для первого операнда задайте тип данных number, а для второго — string.
3. В качестве оператора, выполняющего действие сравнения, выберите ===.

   Это оператор строгого равенства.

4. Щелкните на кнопке Operate.

Результат выполнения настроенного выше действия показан на рис. 9.12. В отличие от уже известного вам оператора равенства (==) оператор строгого равенства (===) рассматривает одинаковые записи, имеющие разные типы данных, как совершенно разные значения.

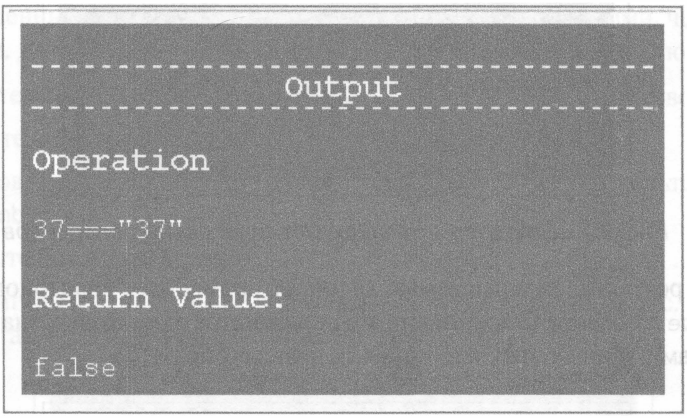


Рис. 9.12. При строгом рассмотрении строка и число не равны

Нередко возникают ситуации, когда требуется определить неравенство двух величин. Для этих целей в JavaScript имеется набор отдельных операторов. Они подобны операторам равенства, за исключением направленности действия. Можете рассматривать их как антиподы операторов равенства.

После того как вы отказались от оператора равенства (==), старайтесь избегать в JavaScript и его точного антипода — оператора неравенства (!=).

1. В текстовом поле настройки первого операнда введите произвольное число (например, `00`) и задайте для него тип данных **number**.
2. Выберите оператор неравенства (`!=`).
3. Введите значение `99` в текстовое поле второго операнда и выберите для него строковый (**string**) тип данных.
4. Щелкните на кнопке **Operate**.

   Результат текущей операции сравнения показан на рис. 9.13. Значение `false`, возвращаемое калькулятором, указывает на то, что оператор `!=` отрицает неравенство указанных значений.
5. Оставьте операнды прежними, но измените оператор сравнения на `!==`.
6. Щелкните на кнопке **Operate**.

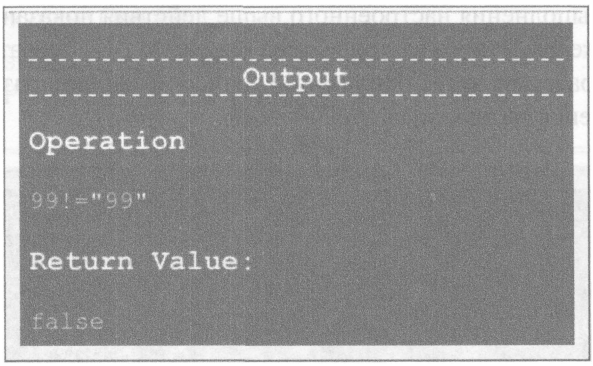


Рис. 9.13. Оператор != игнорирует типы данных операндов

7. Теперь результат, возвращаемый калькулятором, более правдоподобный. Значение `true` в области **Output** говорит о неравенстве одинаковых значений с разными типами данных, что, несомненно, правда (рис. 9.14).

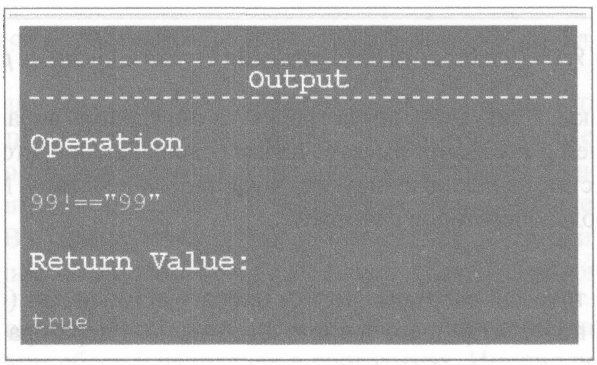


Рис. 9.14. Для распознавания различий в типах данных используется оператор !==

Операторы сравнения также умеют определять большее или меньшее из двух чисел. Давайте посмотрим, как выполняются эти действия в нашем суперкалькуляторе.

1. Измените первый операнд на `10` и укажите для него числовой (**number**) тип данных.
2. Выберите оператор "больше чем" (>).

Значки операторов "больше чем" и "меньше чем" представлены угловыми скобками. Различить их очень просто. Если стрелка указывает вправо, по направлению письма, то она представляет оператор "больше чем". Если же угловая стрелка направлена против направления письма, то она обозначает оператор "меньше чем".

3. Измените второй операнд на `5` и определите для него все тот же числовой тип данных.

4. Щелкните на кнопке Operate.
5. Результат проведенного указанным образом сравнения вполне ожидаем — десять всегда больше пяти.

В JavaScript также существуют операторы "больше или равно" и "меньше или равно". В определенных случаях они становятся более востребованными, чем оба рассмотренных выше. Для их проверки в суперкалькуляторе выполните следующие действия.

1. Измените первый операнд на `10`, не забыв указать для него числовой тип данных.
2. Выберите оператор "больше или равно" (>=).
3. В текстовом поле второго операнда также введите `10` и назначьте ему тип данных number.
4. Щелкните на кнопке Operate.
5. Результат настроенной выше операции сравнения показан на рис. 9.15. Он верен (`true`), так как 10 не больше, но все же равно 10.

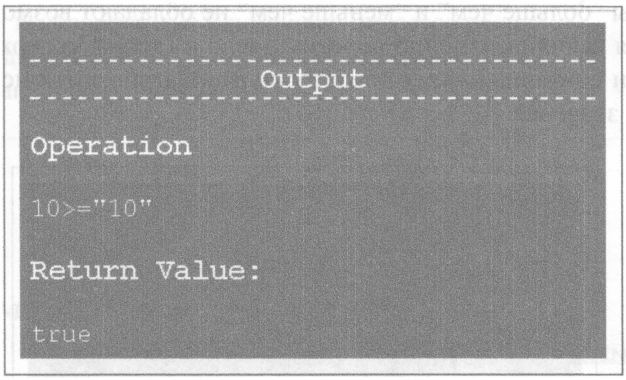


Рис. 9.15. По мнению JavaScript, 10 однозначно больше или равно 10

Операторы "меньше чем" и "меньше или равно" работают так же, как и их антиподы — операторы "больше чем" и "больше или равно". При первом же удобном случае попробуйте удостовериться в этом.

Другие возможности суперкалькулятора

Отдельные операторы языка программирования JavaScript могут показаться странными, но только до тех пор, пока вы не начнете применять их в собственных программах. Поняв логику действия, лежащую в их основе, вы найдете такие операторы невероятно удобными и эффективными. Далее вы узнаете о действии некоторых из них.

1. Измените первый операнд на `1`, а тип его данных — на number.
2. Выберите оператор сложения (+).
3. Измените второй операнд на `1`, а тип его данных — на string.

4. Щелкните на кнопке Operate.

   Вы получите результат 11. Подобное поведение JavaScript вполне объяснимо. Поскольку одно из значений имеет строковый тип данных, вторая единица также преобразуется в строковое значение, а вместо математического оператора сложения применяется оператор конкатенации строк. Как вы уже знаете, объединение строк "1" и "1" приводит к образованию строки "11".

5. Измените тип данных первого операнда на string (строковый), а само значение — на 10.

6. Воспользуйтесь оператором "больше чем" (>).

7. Измените тип данных первого операнда на number (числовой), а само значение — на 5.

8. Щелкните на кнопке Operate.

   Удивительно! Результат этой операции, приведенный на рис. 9.16, как ни странно, — true. В отличие от операторов "больше или равно" и "меньше или равно", операторы "больше чем" и "меньше чем" не обладают возможностью точного сравнения величин. Именно поэтому число и строка, содержащая число, при сравнении с помощью этих операторов рассматриваются исключительно как числовые значения.

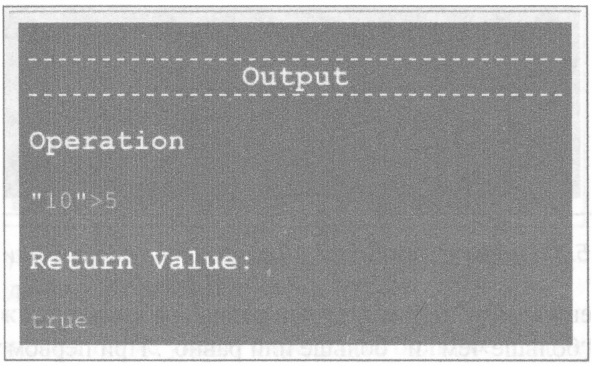


Рис. 9.16. Строковое значение "10" в JavaScript больше числа 5

Приложение Super-Calculator скрывает множество других неожиданных возможностей. Не поленитесь потратить немного вашего драгоценного времени на их изучение. Поэкспериментируйте со всеми представленными в нем операторами, подставляя в выражения самые разные типы данных. Если вам интересно, попробуйте разобраться в коде HTML, CSS и CSS, лежащем в основе приложения. Если вы знаете английский язык, то вам сильно помогут комментарии к коду.

Наконец, попробуйте изменить код приложения Super-Calculator. Возможно, вам удастся модифицировать его в нечто еще более функциональное?

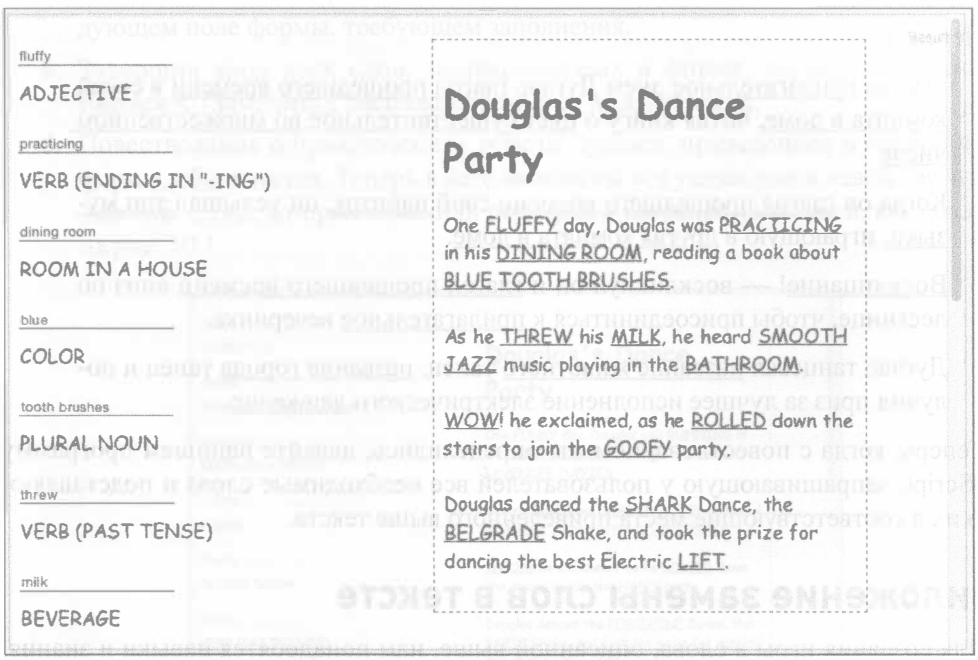
ГЛАВА

10

Игра в слова на JavaScript

Игра в слова, рассматриваемая далее, во многом подобна заполнению формы — от вас требуется дополнить лист с данными отдельными сведениями, представленными самыми разными частями речи: существительными, глаголами, прилагательными, наречиями и т.п. Если немного поразмыслить, то реализовать такую игру с помощью JavaScript не составит большого труда. Базовая структура приложения создается исключительно с помощью кода HTML, CSS и JavaScript, а замена слов в нем выполняется с помощью таких уже известных вам программных элементов, как переменные.

Из этой главы вы узнаете, как с помощью оператора конкатенации строк добиться включения введенных вами слов в общее повествование, созданное заранее.



История с переменными

Первая версия игры в слова посвящена похождениям робота Дугласа, созданного нами с помощью JavaScript в одном из предыдущих проектов. В приведенном тексте отдельные слова заменены названиями частей речи, которые они представляют. Так пользователю, запустившему данную игру, будет проще угадывать слова, которые необходимо подставить в общий сюжет.

Ниже приведена полная история приключений Дугласа в том виде, в котором она будет представлена игроку. Значащие слова подставляются в ней вместо подчеркнутых фраз, указывающих их назначение в предложении.

> One adjective day, Douglas was verb ending in "–ing" in his room in house, reading a book about color plural noun.
>
> As he past-tense verb his beverage, he heard type of music playing in the different room in house.
>
> exclamation! he exclaimed, as he past-tense verb down the stairs to join the adjective party.
>
> Douglas danced the name of animal Dance, the name of city Shake, and took the prize for dancing the best Electric verb.

Перевод:

> Одним прилагательное днем Дуглас глагол прошедшего времени в своей комната в доме, читая книгу о цвет существительное во множественном числе.
>
> Когда он глагол прошедшего времени свой напиток, он услышал тип музыки, играющую в другая комната в доме.
>
> Восклицание! — воскликнул он и глагол прошедшего времени вниз по лестнице, чтобы присоединиться к прилагательное вечеринке.
>
> Дуглас танцевал название животного танец, название города танец и получил приз за лучшее исполнение электрического движение.

Теперь, когда с повествованием мы определились, давайте напишем программу JavaScript, запрашивающую у пользователей все необходимые слова и подставляющую их в соответствующие места приведенного выше текста.

Приложение замены слов в тексте

Для создания игры в слова, описанной выше, нам понадобятся навыки и знания, полученные при изучении всех предыдущих проектов. В программе используются коды HTML и CSS, а также переменные, операторы, обработчики событий и некоторые другие уже известные вам программные конструкции.

Перед тем как приступить к самостоятельному написанию кода игры, давайте загрузим готовое приложение и протестируем его.

1. Запустите браузер и загрузите в нем приложение JSFiddle, отобразив в нем общедоступный кабинет по такому адресу:

   `http://jsfiddle.net/user/forkids/fiddles`

2. Найдите в списке приложение **Chapter 10 — Word Replacement Game** и щелкните на его названии.

   Запустится готовый проект игры в слова.

3. В зависимости от размера экрана вашего компьютера вам может понадобиться изменить размер панели с формой, чтобы отобразить на ней весь текст повествования.

   Следите за тем, чтобы подписи в левой части формы помещались в одну строку и не перекрывали следующую. В правой части формы вы должны наблюдать рамку с пунктирной границей, в которой в конечном счете отображается повествование с подставляемыми в него словами.

4. Щелкните в поле над линией, подчеркивающей первое поле в левой части формы, чтобы выделить находящийся в нем текст.

   Введя в поле необходимое слово, нажмите клавишу <Tab> или щелкните в следующем поле формы, требующем заполнения.

5. Завершив ввод всех слов, запрашиваемых в форме, щелкните на кнопке **Replace it!** (Заменить), расположенной в ее нижней части.

6. Повествование о приключениях робота Дугласа, приведенное в правой части формы, обновляется. Теперь в него включены все указанные в левой части приложения слова, которые заменили подчеркнутые ранее части речи, как показано на рис. 10.1.

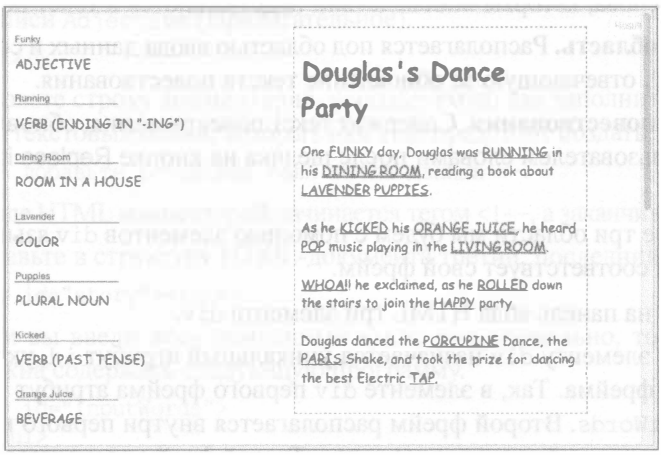


Рис. 10.1. Заполненная словами история о приключениях робота Дугласа

Теперь, когда вы познакомились с поведением конечного приложения, давайте попробуем воссоздать его "с нуля". Научившись разрабатывать игры, вы сможете легко модифицировать и улучшать их в дальнейшем. Вам даже не составит большого труда переписывать их согласно другим потребностям.

Создавать игру замены слов мы будем в приложении JSFiddle, поэтому начнем с его запуска и входа под собственной учетной записью.

1. Запустите браузер Google Chrome и откройте в нем новую вкладку, нажав комбинацию клавиш <⌘+T> (в Mac) или <Ctrl+T> (в Windows). Также в главном меню Chrome можно выбрать команду **Новая вкладка**. Перейдите к странице `http://jsfiddle.net`.
2. Убедитесь в том, что вошли в приложение под собственной учетной записью. Ее имя должно отображаться в правом верхнем углу страницы.

   Все дальнейшие действия выполняются из вашего личного кабинета.
3. На левой панели щелкните на команде **Fiddle Options**. Введите название создаваемого приложения (игры).

   Дайте волю фантазии, но не забывайте о том, что название должно отражать суть истории.
4. Щелкните на кнопке **Save** (Сохранить), расположенной в верхней части окна.

Все готово для написания кода игры. Начнем, пожалуй, с HTML.

Код HTML

Создаваемая нами игра в слова представлена веб-страницей, условно разделенной на три области.

- ✓ **Область ввода данных.** В этой части страницы программа запрашивает у пользователя слова, которые впоследствии подставляются в текст повествования.
- ✓ **Нижняя область.** Располагается под областью ввода данных и содержит кнопку **Replace it!**, отвечающую за обновление текста повествования.
- ✓ **Область повествования.** Содержит текст повествования, обновляемый введенными пользователем словами после щелчка на кнопке **Replace it!**.

Создавать все три области мы будем с помощью элементов `div` языка HTML. Каждой из областей соответствует свой фрейм.

1. Добавьте на панель кода HTML три элемента `div`.

   Каждому элементу `div` назначается уникальный атрибут `id`, указывающий назначение фрейма. Так, в элементе `div` первого фрейма атрибут `id` имеет значение `inputWords`. Второй фрейм располагается внутри первого и имеет атрибут `id`, установленный равным `buttonDiv`. Третий фрейм располагается отдельно от двух первых и снабжается идентификатором `story`.

```
<div id="inputWords">

<div id="buttonDiv"></div>

</div>
<div id="story"></div>
```

2. Создайте форму с текстовыми полями, в которые вводятся слова, подставляемые в повествование.

   В HTML решить эту задачу проще всего с использованием элемента неупорядоченного списка. Для образования формы ввода данных создайте внутри первого фрейма (элемента `div`) следующую структуру.

```
<ul>
   <li></li>
</ul>
```

3. Каждое текстовое поле представляется отдельным элементом `input`, находящимся внутри элемента `li`.

   Снабдите его всеми необходимыми атрибутами. Например, для первого текстового поля их всего два: `type="text"` и `id="adj1"`.

```
<input type="text" id="adj1" />
```

Обратите внимание на символ косой черты в конце тега `<input>`. В HTML5 вводить элемент `input` в таком формате совсем необязательно. Чтобы обеспечить запуск нашей игры в браузерах, поддерживающих старые редакции языка HTML, в конце таких тегов, как `<input>` и `<br>`, косую черту все же лучше добавлять.

4. Добавьте подписи под полями, воспользовавшись тегами `<br>`.

```
<br />Adjective
```

5. Наконец, добавьте закрывающий тег к элементу `li`, расположив его после подписи `Adjective` (Прилагательное).

```
</li>
```

6. Добавьте строку комментария, используемую как заполнитель для остальных текстовых полей, которые нам еще предстоит создать.


```
<!-- добавьте остальные поля ввода данных -->
```

   В коде HTML комментарий начинается тегом `<!--`, а заканчивается тегом `-->`.

7. Добавьте в структуру HTML-документа третий, последний элемент `div`.

```
<div id="story"></div>
```

   Если вы ввели весь описанный выше код правильно, то панель HTML должна содержать следующую программу.

```
<div id="inputWords">
   <ul>
      <li><input type="text" id="adj1" /><br />Adjective</li>
<!-- добавьте остальные поля ввода данных -->
```

```
    </ul>
<div id="buttonDiv"></div>
</div>
<div id="story"></div>
```

8. Щелкните на кнопке **Update** для просмотра нашего приложения в текущем виде.

   На панели результатов должно отображаться нечто подобное представленному на рис. 10.2.

Рис. 10.2. В начале разработки приложение игры в слова имеет предельно простой вид

На последнем этапе написания HTML-кода требуется создать кнопку.

9. Поместите курсор в элемент `div` с идентификатором (`id`), имеющим значение `buttonDiv`, и введите такой код.

   ```
   <button id="replaceButton">Replace it!</button>
   ```

10. Еще раз щелкните на кнопке **Update**.

    На данном этапе наша игра представлена единственным полем ввода данных с подписью и расположенной под ним кнопкой, как показано на рис. 10.3.

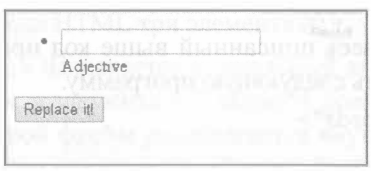


Рис. 10.3. Элементы управления на своих местах

Поскольку другие поля ввода данных являются всего лишь копией первого, мы не будем детально описывать код, с помощью которого они создаются. Можете самостоятельно скопировать HTML-код имеющегося поля и вставить его в программу необходимое количество раз. Не забудьте изменить атрибуты каждой из копий. Кроме того, можете скопировать готовый HTML-код работающего приложения в свою копию игры.

Далее мы займемся написанием кода CSS нашего приложения.

Стилизация игры

Перед тем как заняться настройкой внешнего вида игры, давайте обсудим общую структуру кода CSS.

Используя в приложении большое количество кода CSS, размещайте его таким образом, чтобы в дальнейшем быстро находить в нем нужные правила и вносить все необходимые правки.

Один из способов организации кода CSS предполагает его помещение в те HTML-элементы, внешний вид которых он изменяет. В подобных случаях правила CSS приводятся в порядке следования HTML-элементов в общем коде программы.

Если придерживаться предложенного выше метода оформления приложения, то первый CSS-стиль применяется к элементу `body`, затем следуют стили, определяющие форматирование контейнера элементов `input`, далее — стили оформления неупорядоченного списка и только после них — стили полей ввода данных и т.д. Общая структура понятна!

Выяснив порядок назначения стилей в приложении, давайте перейдем к непосредственному оформлению нашей игры с помощью правил CSS.

1. Чтобы изменить шрифт, которым отображается в приложении весь текст, измените свойство `font-family` элемента `body`.

   В качестве значения свойства используйте шрифт Comic Sans.

   ```
   body {
      font-family: "Comic Sans MS";
   }
   ```

   Список других стандартных шрифтов, допустимых для применения в свойствах CSS, приведен на следующем сайте:

   `https://webref.ru/course/css-text/font-family`

2. Контейнер (фрейм), содержащий все поля ввода данных, оформляется с помощью следующего правила CSS.

   ```
   #inputWords {
      float:left;
      width: 45%;
   }
   ```

   Свойство `float:left` приведенного выше правила указывает расположить элемент `div` у левой границы своего контейнера (в данном случае — элемента `body` или всей страницы). Остальные элементы будут располагаться вокруг него.

На практике такое форматирование элемента `div` приводит к расположению формы с полями ввода данных слева от контейнера (элемента `div`), в котором выводится текст повествования.

3. Форматирование неупорядоченного списка выполняется с помощью следующего CSS-кода.

```
ul {
   list-style-type: none;
   padding: 0px;
   margin: 0px;
}
li {
   line-height: 2em;
   text-transform: uppercase;
   margin-top: 8px;
}
```

В этих правилах определяются следующие свойства, устанавливающие внешний вид списка.

- Свойство `list-style-type` указывает скрывать маркеры (кружки) в начале каждого элемента списка.
- Установив свойства `padding` и `margin` равными `0px`, вы выровняете список строго по левому краю своего контейнера, а потому и веб-страницы.
- С помощью свойства `line-height` элемента `li` устанавливается дополнительный интервал между элементами списка. Если это свойство опустить, то по умолчанию элементы списка будут располагаться очень близко один от другого.
- В свойстве `text-transform` элемента `li` указывается выводить весь текст подписей к полям ввода данных в верхнем регистре.
- Свойство `margin-top` еще немного увеличивает интервал между отдельными элементами списка.

4. Снабдив приложение описанными выше правилами, щелкните на кнопке **Update**, чтобы ознакомиться с текущей версией приложения.

Приведенные далее правила устанавливают непосредственное форматирование полей ввода данных, кнопки и текста повествования.

5. Введите на панели CSS следующий код.

```
input[type=text] {
border-width: 0 0 1px 0;
border-color: #333;
}
#buttonDiv {
text-align: center;
}
#replaceButton {
margin-top: 30px;
```

```
width: 200px;
}
#story {
margin-top: 12px;
width: 45%;
border: 1px dashed blue;
padding: 8px;
float: left;
}
.replacement {
text-decoration: underline;
text-transform: uppercase;
}
```

6. Для сохранения результатов выполненной работы щелкните на кнопке Update.
7. Панель результатов при этом должна выглядеть так, как показано на рис. 10.4.

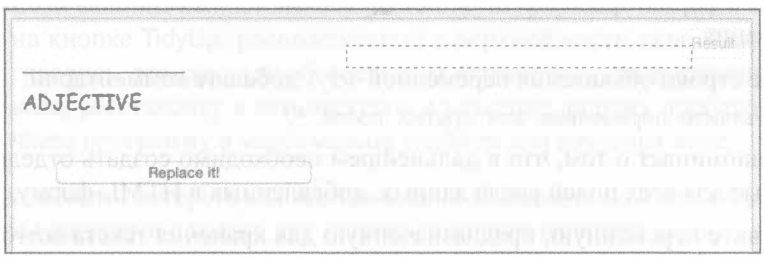


Рис. 10.4. Панель результатов после добавления правил CSS в код приложения

Самое интересное мы оставили напоследок. Сложно представить, что веб-приложение, выполняющее действия, будет разрабатываться без участия JavaScript. Давайте посмотрим, каким образом код JavaScript "вдыхает жизнь" в статичную веб-страницу.

Код JavaScript

Первое, что необходимо написать на панели JavaScript, — код обработчика события для элемента кнопки. В нем используется метод `addEventListener()`, известный вам из главы 7.

```
var replaceButton = document.getElementById("replaceButton");
replaceButton.addEventListener("click",replaceIt);
```

В первой строке создается переменная `replaceButton`, содержащая ссылку на элемент `button` (кнопка). Вторая строка добавляет в программу обработчик события, который указывает запускать функцию `replaceIt()` при возникновении события `click` (щелчок).

Следующий этап заключается в написании кода функции `replaceIt()`.

1. В первой строке кода объявляется функция, и ей дается название.

```
function replaceIt() {}
```

2. В фигурных скобках с новой строки введите инструкцию создания переменной, содержащей ссылку на элемент `div`, в котором отображается текст конечного повествования.

   ```
   var storyDiv = document.getElementById("story");
   ```

   Переменная будет `storyDiv` востребована в дальнейшем коде, а пока нам необходимо обеспечить извлечение данных, введенных пользователями в поля формы.

3. Создайте переменную, хранящую содержимое первого поля ввода данных.

   ```
   var adj1 = "<span class='replacement'>"+
      ↳document.getElementById("adj1").value + "</span>";
   ```

   Обратите внимание на то, как с помощью оператора конкатенации значение из поля (полученное методом `document.getElementById()`) объединяется с элементом `span`. Последний позволяет применять к введенным пользователями данным форматирование, отличное от задаваемого для остального текста повествования.

4. После строки объявления переменной `adj1` добавьте комментарий.

   ```
   /* Объявите переменные для других полей */
   ```

   Он напоминает о том, что в дальнейшем необходимо создать отдельные переменные для всех полей ввода данных, добавленных в HTML-форму.

5. Объявите переменную, предназначенную для хранения текста всего повествования.

   Пусть она называется `theStory`.

   ```
   var theStory;
   ```

6. Снабдите повествование заголовком, сохранив его в переменной и представив элементом `h1` (HTML).

   ```
   theStory = "<h1>Douglas's Dance Party</h1>";
   ```

7. Добавьте в переменную `theStory` первую часть повествования, воспользовавшись оператором конкатенации с присвоением (`+=`).

   Оператор `+=` добавляет к уже имеющемуся значению переменной новые данные. В нашем случае значение переменной дополняется новым текстом.

   ```
   theStory += "One " + adj1 + " day,";
   ```

8. Снова включите в код комментарий, указывающий на необходимость дополнения переменной `adj1` содержимым остальных полей ввода данных и частей повествования.

   ```
   /* Дополните повествование, воспользовавшись оператором += */
   ```

9. Для отображения повествования, сохраненного в переменной `theStory`, в соответствующем фрейме `div` используется свойство `innerHTML`.

   ```
   storyDiv.innerHTML = theStory;
   ```

На данный момент код JavaScript нашего приложения должен выглядеть следующим образом.

```
var replaceButton = document.getElementById("replaceButton");
replaceButton.addEventListener("click", replaceIt);

function replaceIt() {
   var storyDiv = document.getElementById("story");
   var adj1 = "<span class='replacement'>" + document.
      getElementById("adj1").value + "</span>";
   /* Объявите переменные для других полей */
   var theStory = "<h1>Douglas's Dance Party</h1>";
   theStory += "One " + adj1 + " day,";
   /* Дополните повествование, воспользовавшись оператором += */
   storyDiv.innerHTML = theStory;
}
```

Если ваш код не настолько хорошо структурирован, как наш, то щелкните на кнопке **TidyUp**, расположенной в верхней части окна. Как вы уже догадались, эта команда обеспечивает автоматическое выравнивание строк кода, расстановку в нем отступов и удаление лишних пробелов. Вы получаете программу в максимально удобном для изучения виде.

Наступил момент, которого мы все так ждали! Щелкните на кнопке **Update**, расположенной над панелями с кодом, для запуска тестового варианта игры в слова.

Вводите слова в единственное поле формы и щелкайте на кнопке **Replace it!**, чтобы добавлять их в невероятно короткое повествование. Результат этого действа показан на рис. 10.5.

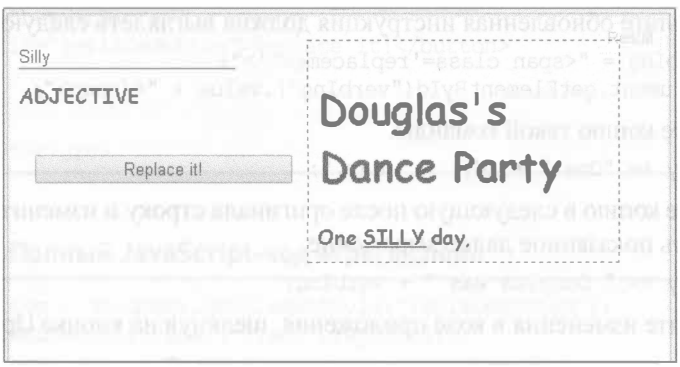


Рис. 10.5. Сгенерированная программой история

Завершение программы

Сейчас программа нашей игры содержит все необходимые для успешного выполнения компоненты. На завершающем этапе необходимо выполнить всего три простые операции, обеспечивающие наполнение повествования сведениями из других полей формы.

1. Создайте копии элемента `input` для каждого поля ввода данных и обновите в них подпись и идентификатор (`id`).
2. Для каждого поля ввода данных создайте копии инструкции JavaScript, включающей метод `getElementById()`, обновив в ней имя переменной и значение, указанное в скобках.
3. Снабдите повествование дополнительными подробностями, включив их в переменную `theStory`.

Давайте посмотрим, как реализуются эти задачи на примере следующей части повествования, в которой у пользователя запрашивается слово a verb ending in "–ing".

1. На панели HTML выделите следующий код и скопируйте его.

```
<li><input type="text" id="adj1" />
<br />Adjective</li>
```

2. Вставьте копию сразу же после оригинала.
3. Измените в копии значение атрибута `id` на `verbIng`.
4. Замените подпись в копии поля ввода данных следующей.

```
Verb (ending in "-ing")
```

5. На панели JavaScript скопируйте следующее выражение.

```
var adj1 = "<span class='replacement'>"+
   ↳document.getElementById("adj1").value + "</span>";
```

6. Вставьте скопированный код сразу же после оригинала.
7. Измените имя переменной на `verbIng`, а значение, приведенное в скобках после ключевого слова `getElementById`, — на `verbIng`.

   В результате обновленная инструкция должна выглядеть следующим образом.

```
var verbIng = "<span class='replacement'>"+
   ↳document.getElementById("verbIng").value + "</span>";
```

8. Создайте копию такой команды.

```
theStory += "One " + adj1 + " day,";
```

9. Вставьте копию в следующую после оригинала строку и измените ее так, чтобы получить показанное далее выражение.

```
theStory += " Douglas was " + verbIng;
```

10. Сохраните изменения в коде приложения, щелкнув на кнопке Update.

    В результате тестового запуска страница с игрой должна выглядеть так, как показано на рис. 10.6. Примите поздравления! Вы только что добавили в игру второй вопрос.

Повторите приведенные выше действия для всех полей ввода данных и вопросов в повествовании. По завершении вы должны наблюдать на панели HTML-код,

показанный в листинге 10.1. При этом на панели JavaScript должен содержаться код, приведенный в листинге 10.2.

Листинг 10.1. Полный HTML-код игры в слова

```
<div id="inputWords">
   <ul>
      <li><input type="text" id="adj1" /><br>Adjective</li>
      <li><input type="text" id="verbIng" /><br>Verb (ending in
         "-ing")</li>
      <li><input type="text" id="roomInHouse" /><br>Room in
         a house</li>
      <li><input type="text" id="color" /><br>Color</li>
      <li><input type="text" id="nounPlural" /><br>Plural
         noun</li>
      <li><input type="text" id="pastVerb" /><br>Verb (past
         tense)</li>
      <li><input type="text" id="beverage" /><br>Beverage</li>
      <li><input type="text" id="musicType" /><br>Type of
         music</li>
      <li><input type="text" id="diffRoom" /><br>Different room
         in a house</li>
      <li><input type="text" id="exclamation"/><br>
         Exclamation</li>
      <li><input type="text" id="pastVerb2" /><br>Verb (past
         tense)</li>
      <li><input type="text" id="adjDance" /><br>Adjective</li>
      <li><input type="text" id="animal" /><br>Animal</li>
      <li><input type="text" id="city" /><br>City</li>
      <li><input type="text" id="verb" /><br>Verb</li>
   </ul>
      <div id="buttonDiv">
      <button id="replaceButton">Replace it!</button>
</div>
</div>

<div id="story"></div>
```

Листинг 10.2. Полный JavaScript-код игры в слова

```
var replaceButton = document.getElementById("replaceButton");
replaceButton.addEventListener("click",replaceIt);

function replaceIt() {
   var storyDiv = document.getElementById("story");
   var adj1 = "<span class='replacement'>"+ document.
      ↳getElementById("adj1").value + "</span>";
   var verbIng = "<span class='replacement'>"+ document.
      ↳getElementById("verbIng").value + "</span>";
   var roomInHouse = "<span class='replacement'>"+ document.
      ↳getElementById("roomInHouse").value + "</span>";
```

```
    var color = "<span class='replacement'>"+ document.
       ↪getElementById("color").value + "</span>";
    var nounPlural = "<span class='replacement'>"+ document.
       ↪getElementById("nounPlural").value + "</span>";
    var pastVerb = "<span class='replacement'>"+ document.
       ↪getElementById("pastVerb").value + "</span>";
    var beverage = "<span class='replacement'>"+ document.
       ↪getElementById("beverage").value + "</span>";
    var musicType = "<span class='replacement'>"+ document.
       ↪getElementById("musicType").value + "</span>";
    var diffRoom = "<span class='replacement'>"+ document.
       ↪getElementById("diffRoom").value + "</span>";
    var exclamation = "<span class='replacement'>"+ document.
       ↪getElementById("exclamation").value + "</span>";
    var pastVerb2 = "<span class='replacement'>"+ document.
       ↪getElementById("pastVerb2").value + "</span>";
    var adjDance = "<span class='replacement'>"+ document.
       ↪getElementById("adjDance").value + "</span>";
    var animal = "<span class='replacement'>"+ document.
       ↪getElementById("animal").value + "</span>";
    var city = "<span class='replacement'>"+ document.
       ↪getElementById("city").value + "</span>";
    var verb = "<span class='replacement'>"+ document.
       ↪getElementById("verb").value + "</span>";

    var theStory = "<h1>Douglas's Dance Party</h1>";
    theStory += "One " + adj1 + " day,";
    theStory += " Douglas was " + verbIng;
    theStory += " in his " + roomInHouse;
    theStory += ", reading a book about " + color;
    theStory += " " + nounPlural + ".<br><br>";
    theStory += "As he " + pastVerb;
    theStory += " his " + beverage;
    theStory += ", he heard " + musicType;
    theStory += " music playing in the " + diffRoom + ".<br><br>";
    theStory += exclamation + "! he exclaimed, as he ";
    theStory += pastVerb2 + " down the stairs to join the ";
    theStory += adjDance + " party.<br><br>";
    theStory += "Douglas danced the " + animal;
    theStory += " Dance, the " + city + " Shake,";
    theStory += " and took the prize for dancing the best Electric
       ↪" + verb + ".<br><br>";

    storyDiv.innerHTML = theStory;

}
```

Завершив ввод слов в форме, щелкните на кнопке в нижней ее части и передайте управление игрой программе JavaScript. В результате выполнения кода приложения конечное повествование наполнится словами из формы и приобретет законченный вид, как показано на рис. 10.6.

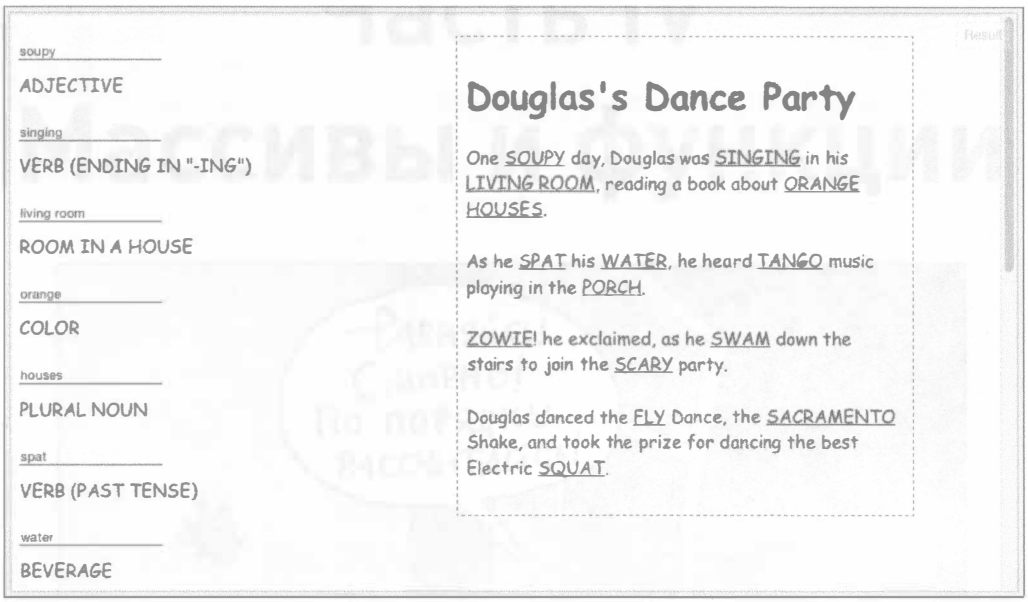


Рис. 10.6. Конечный вид веб-приложения игры в слова

Теперь, когда вы имеете в своем распоряжении рабочую копию игры, щелкните в верхней части окна на кнопке Fork и попробуйте самостоятельно усовершенствовать ее код, выдумав совершенно иную историю и запрашиваемые у пользователя слова.

Если ваша программа не работает так, как ожидается, то построчно сверьтесь с приведенным в книге вариантом. Причина, скорее всего, кроется в досадной опечатке.

Часть IV
Массивы и функции

В этой части...

- ✓ Управление массивами
- ✓ Вездесущие функции
- ✓ Список желаний

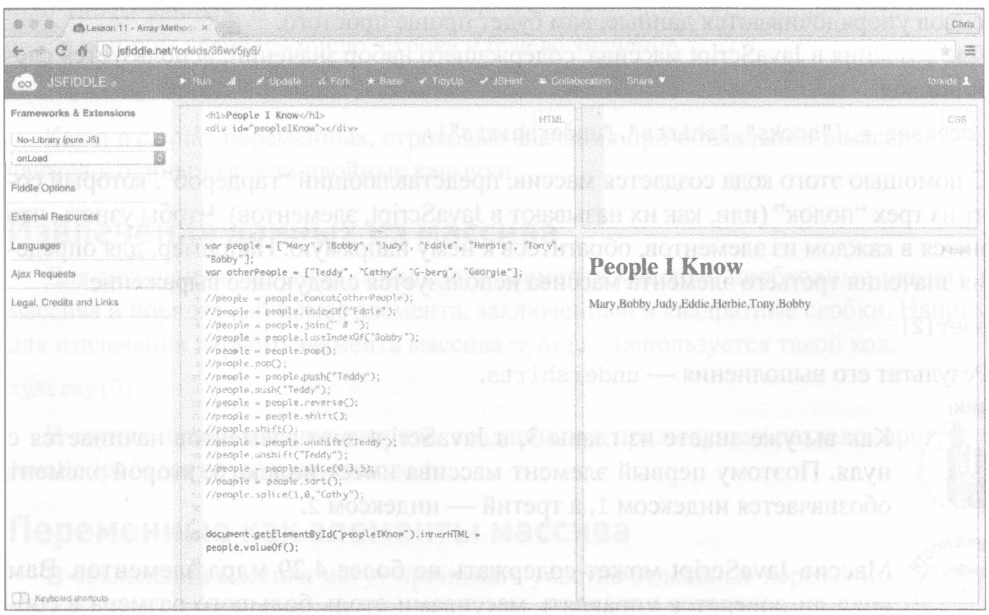
ГЛАВА

11

Управление массивами

Можно ли жить без списков? Скорее всего, нет! Мы используем их в повседневной жизни настолько часто, что даже не представляем, как без них обойтись. Перечень покупок, расписание занятий, каталог песен в альбоме любимого исполнителя, рейтинги спортивных соревнований — все эти и многие другие виды списков стали неотъемлемой частью нашей жизни.

В программировании данные, представляемые списками в повседневной жизни, хранятся в специальных структурах, называемых *массивами*. В этой главе вы узнаете о том, как создаются массивы и изменяются хранящиеся в них значения. Кроме того, вы увидите, как массивы помогают выполнять в приложениях далеко не самые тривиальные задачи.



Что такое массив

Массив — это переменная специального типа, которая позволяет хранить множество значений. Он чем-то напоминает гардероб, который состоит из огромного количества полок. На каждой полке хранятся свои вещи, но все они относятся к гардеробу и не существуют отдельно от него.

Если вам, например, понадобилась чистая майка, то зная, что она находится на второй полке снизу, достаточно открыть гардероб, найти полку и взять с нее этот предмет гардероба. Таким же образом вы поступите, когда вам потребуется свежая рубашка или носки, разве что номер полки будет другим. Поддерживая в гардеробе постоянный порядок, вы всегда будете уверены в том, что вещи в нем располагаются на своих местах, например:

- ✓ нижняя полка — носки;
- ✓ вторая полка — рубашки;
- ✓ третья полка — майки.

Если вы умеете управляться с собственным гардеробом, то понять, как с помощью массивов упорядочиваются данные, вам будет проще простого.

Для создания в JavaScript массива, содержащего набор значений, используется следующий код.

```
var dresser = ["socks","shirts","undershirts"];
```

С помощью этого кода создается массив, представляющий "гардероб", который состоит из трех "полок" (или, как их называют в JavaScript, элементов). Чтобы узнать, что хранится в каждом из элементов, обратитесь к нему напрямую. Например, для определения значения третьего элемента массива используется следующее выражение.

```
dresser[2]
```

Результат его выполнения — `undershirts`.

Как вы уже знаете из главы 3, в JavaScript счет элементов начинается с нуля. Поэтому первый элемент массива имеет индекс `0`, второй элемент обозначается индексом `1`, а третий — индексом `2`.

Массив JavaScript может содержать не более 4,29 млрд элементов. Вам вряд ли доведется управлять массивами столь большого размера в собственных проектах.

Теперь, когда вы знаете, для чего в JavaScript используются массивы, давайте посмотрим, как они объявляются и наполняются данными.

Создание массивов и доступ к ним

Объявление массива в коде программы выполняется так же, как и объявление переменной, — с помощью ключевого слова `var`. Тем не менее, чтобы отличить массив от обычной переменной, после его имени вводятся открывающая и закрывающая квадратные скобки.

Чтобы создать пустой массив, исходно не содержащий никаких данных, используйте следующую инструкцию объявления.

```
var favoriteFoods = [];
```

Если нужно создать массив, наполненный значениями, укажите их внутри квадратных скобок, разделив запятыми.

```
var favoriteFoods = ["broccoli","eggplant","tacos","mushrooms"];
```

Типы данных массива

В массиве JavaScript разрешается хранить данные любых известных типов: числа, строки, булевы значения и даже объекты.

На самом деле в одном массиве допускается хранить значения, относящиеся к разным типам данных. В частности, с помощью следующего выражения создается массив, в котором хранится число, строка и булево значение.

```
var myArray = [5, "Привет!", true];
```

Как и в случае переменных, строковые значения при объявлении в массиве заключаются в одинарные или двойные кавычки.

Извлечение данных из массива

Для получения значения отдельного элемента массива, необходимо указать имя массива и порядковый номер элемента, заключенный в квадратные скобки. Например, для извлечения первого элемента массива `myArray` используется такой код.

```
myArray[0]
```

В случае использования массива, созданного в предыдущем разделе, программой JavaScript будет возвращено число `5`.

Переменные как элементы массива

В объявлении массива могут принимать участие отдельные переменные. В приведенном далее коде в качестве элементов массива используются три переменные, представляющие имя, отчество и фамилию вымышленного человека.

```
var firstName = "Иван";
var middleName = "Иванович"
var lastName = "Иванов";
var Scientist = [firstName, middleName, lastName];
```

Поскольку переменные служат всего лишь вместилищем значений, представленный выше код равнозначен представленной ниже команде объявления массива строковых значений.

```
var Scientist = ["Иван", "Иванович", "Иванов"];
```

Для более углубленного изучения массивов и методов управления их элементами в программном коде нужно немного попрактиковаться на панели JavaScript браузера.

Изменение элементов массива

В JavaScript существует несколько способов изменения массивов.

Первый из них заключается в назначении элементам массива новых значений. Это очень простая операция, равнозначная изменению переменной. Чтобы узнать, как она выполняется, перейдите к панели разработчика (консоли JavaScript) вашего браузера.

1. С помощью следующего кода создайте новый массив.

   ```
   var people = ["Teddy","Cathy","Bobby"];
   ```

2. Выведите значения массива на экран, воспользовавшись таким кодом.

   ```
   console.log(people);
   ```

   На консоль JavaScript будет возвращен список элементов, которые объявлены в массиве в предыдущей инструкции.

3. Для изменения первого элемента массива выполните следующее выражение (введите его и нажмите клавишу <Return> или <Enter>).

   ```
   people[0] = "Georgie";
   ```

4. Снова выведите на консоль список элементов массива, введя такой код.

   ```
   console.log(people);
   ```

5. Как видите, теперь первый элемент массива представлен значением `"Georgie"`, а не `"Teddy"`.

Попробуйте самостоятельно изменить остальные элементы массива. Расширьте его, включив новые элементы. В качестве окончательного варианта рассмотрите возможность получения следующего массива имен.

```
Mary, Bobby, Judy, Eddie, Herbie, Tony
```

Создав в приложении массив, представляющий список значений, вы получаете возможность выполнять над ними самые разные действия: изменять, сортировать, сравнивать и т.п. О том, как выполнить эти и другие операции с помощью методов массива, рассказывается в следующих разделах.

Методы массива

Для работы с массивами в вашем распоряжении есть большое количество встроенных методов JavaScript. Эти методы, пожалуй, — самое лучшее из того, что применя-

ется в JavaScript для управления массивами. Научившись использовать их при разработке собственных приложений, вы сэкономите огромное количество времени и сил. К тому же работать с массивами очень увлекательно.

Описание встроенных методов управления массивами в JavaScript вы найдете в табл. 11.1.

Таблица 11.1. Методы массива в JavaScript

Метод	Описание
`concat()`	Создает новый массив, состоящий из текущего массива и других массивов и/или значений
`indexOf()`	Возвращает индекс первого экземпляра искомого значения. Если значение в массиве не найдено, то возвращается значение `-1`
`join()`	Объединяет все элементы массива в единую строку
`lastIndexOf()`	Возвращает индекс последнего экземпляра искомого значения. Если значение в массиве не найдено, то возвращается значение `-1`
`pop()`	Удаляет в массиве последний элемент
`push()`	Добавляет в конец массива новые элементы
`reverse()`	Изменяет порядок следования элементов массива на обратный
`shift()`	Удаляет первый элемент массива, возвращая его в качестве значения. Длина массива уменьшается на единицу
`slice()`	Выделяет часть текущего массива и возвращает его в виде нового массива
`sort()`	Возвращает отсортированный в определенном порядке массив (по умолчанию массив сортируется в алфавитном порядке по возрастанию)
`splice()`	Возвращает новый массив, состоящий из элементов, удаленных из текущего массива или добавленных в него
`tostring()`	Преобразует массив в строку
`unshift()`	Возвращает дополненный новыми элементами массив с измененной длиной

Практическое применение массивов

Назначение массивов зачастую становится понятным только после их использования для решения практических задач. Давайте запустим приложение JSFiddle и попробуем попрактиковаться.

1. Запустите браузер и войдите в JSFiddle под своей учетной записью.
2. Перейдите в общедоступный кабинет `http://jsfiddle.net/user/forkids/fiddles`.
3. Отыщите заголовок **Lesson 11 — Array Methods — Start** и щелкните на нем.
4. В верхней части окна щелкните на кнопке **Fork** для заимствования кода выбранного приложения в личный кабинет.
5. Измените название программы, для чего щелкните на команде **Fiddle Options**, расположенной на левой панели окна.

6. Щелкните на кнопке **Update**, а затем воспользуйтесь командой **Set as Base** для сохранения текущей версии приложения.
7. В настоящий момент код программы должен выглядеть так, как показано на рис. 11.1. На панели JavaScript содержится всего две строки кода, и столько же строк располагается на панели HTML. А вот на панели результатов выведен всего один заголовок.

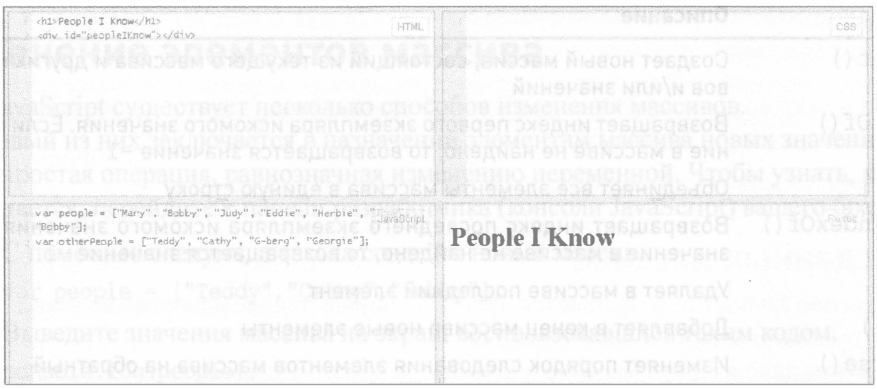


Рис. 11.1. Начальная версия приложения

Чтобы попрактиковаться на реальном примере, давайте напишем код, в котором для изменения имеющегося и создания нового массива применяются встроенные методы JavaScript. В качестве результата такая программа будет возвращать на правую нижнюю панель список элементов массива.

Методы `toString()` и `valueOf()`

Методы `toString()` и `valueOf()` выполняют одну и ту же задачу — преобразуют массив в строку, в которой элементы массива приводятся разделенными запятой.

В следующем примере метод `toString()` будет использоваться для вывода элементов массива на веб-страницу в виде списка.

Чтобы вывести элементы массива в виде строкового значения, выполните такие действия.

1. Щелкните на панели JavaScript и нажмите несколько раз клавишу <Return> или <Enter>, чтобы вставить после инструкции создания массива `otherPeople` несколько пустых строк.
2. Введите следующую строку.

   ```
   document.getElementById("peopleIKnow").innerHTML = people.toString();
   ```

   Этот код создает линейный список, состоящий из элементов массива `people`.
3. Щелкните на кнопке **Update**.

   Все элементы массива `people` будут приведены на панели результатов в виде, показанном на рис. 11.2.

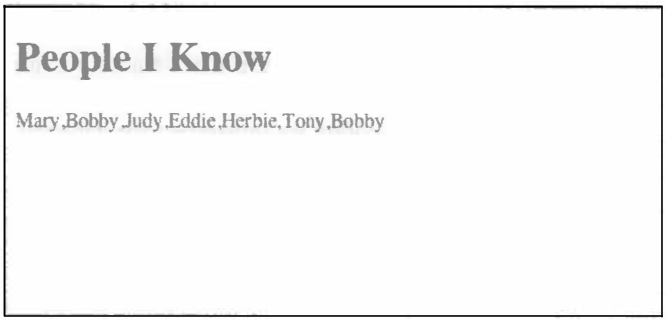


Рис. 11.2. Значения, хранящиеся в массиве `people`

4. Замените в коде JavaScript метод `toString()` методом `valueOf()`.
5. Щелкните на кнопке **Update**.

   Содержимое панели результатов останется неизменным, указывая на то, что методы `toString()` и `valueOf()` выполняют одни и те же задачи.

Метод `concat()`

Для объединения двух массивов выполните приведенные ниже операции, в которых задействуется метод `concat()`.

1. Введите следующий код, разместив его в программе под инструкциями создания массивов, но перед командой вывода данных.

   ```
   people = people.concat(otherPeople);
   ```

2. Щелкните на кнопке **Update**.

   В рассматриваемом примере метод `concat()` применяется для объединения массивов `people` и `otherPeople`. Результат этой операции (с помощью оператора присвоения =) сохраняется в массиве `people`. Панель **Results** при этом выглядит так, как показано на рис. 11.3.

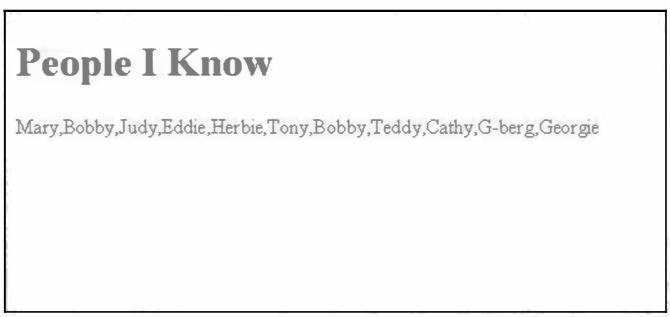


Рис. 11.3. Результат использования метода `concat()` *для конкатенации двух массивов*

Метод indexOf()

С помощью метода `indexOf()` удобно просматривать массив в поисках необходимого элемента. В качестве результата этот метод возвращает индекс искомого значения. Давайте посмотрим, на что он способен на примере уже известного вам массива `people`.

1. Закомментируйте команду, содержащую метод `concat()`, введя в начале строки две косые черты, как показано на рис. 11.4.

   Преобразовав код в комментарий, вы предотвращаете его выполнение браузером. При этом сам код не удаляется и может использоваться в дальнейшем.

```
var people = ["Mary", "Bobby", "Judy", "Eddie", "Herbie", "Tony",
"Bobby"];
var otherPeople = ["Teddy", "Cathy", "G-berg", "Georgie"];

//people = people.concat(otherPeople);

document.getElementById("peopleIKnow").innerHTML =
people.valueOf();
```

Рис. 11.4. Преобразование строки кода в комментарий

2. Выполните поиск строки `"Eddie"` в массиве `people`, воспользовавшись такой командой.

```
people = people.indexOf("Eddie");
```

3. Для сохранения изменений и запуска программы щелкните на кнопке **Update**.

   На панель результатов будет выведено числовое значение, указывающее на порядок расположения в массиве элемента `"Eddie"` (рис. 11.5).

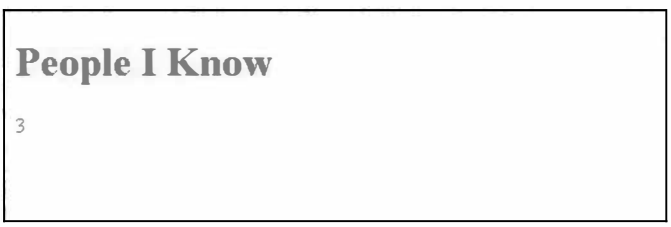


Рис. 11.5. Поиск Эдди (Eddie)

Метод join()

Встроенный метод `join()` во многом подобен методам `toString()` и `valueOf()`, в первую очередь тем, что объединяет элементы массива в единое строковое значение. Отличие этого метода от остальных заключается в возможности указания символов, которыми разделяются значения в получаемой строке.

Чтобы познакомиться с методом `join()`, выполните следующие действия.

1. Закомментируйте строку, содержащую код метода `indexOf()`.
2. Введите такую команду под только что образованным комментарием.
   ```
   people = people.join(" # ");
   ```
3. Щелкните на кнопке **Update** для сохранения и выполнения измененной программы.

   На панель **Results** будут выведены элементы массива `people`, разделенные символом `#`, как показано на рис. 11.6.

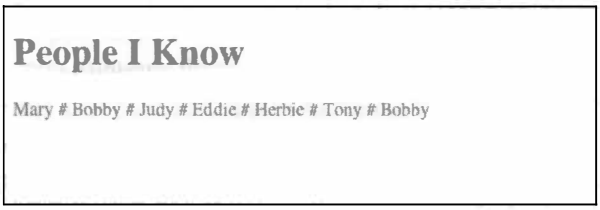


Рис. 11.6. Результат применения встроенного метода `join()`

Метод `lastIndexOf()`

С помощью метода `lastIndexOf()` определяется последний элемент массива, содержащий указанное значение.

Как несложно заметить, массив `people` содержит два значения `Bobby` — на второй и седьмой позициях (соответствуют индексам `1` и `6`). Попробуйте определить, какое значение выводится методом `lastIndexOf()` на панель результатов. Чтобы узнать наверняка, следуйте приведенным ниже инструкциям.

1. Закомментируйте строку кода, созданную в предыдущем разделе, которая содержит метод `join()`.
2. Введите под комментарием следующую инструкцию.
   ```
   people = people.lastIndexOf("Bobby");
   ```
3. Чтобы сохранить программу и выполнить ее, щелкните в верхней части окна на кнопке **Update**.

   Результат выполнения метода `lastIndexOf()` показан на рис. 11.7. На панель выводится индекс последнего экземпляра элемента, указанного в качестве аргумента (в скобках) метода.

Рис. 11.7. Индекс последнего экземпляра элемента "Bobby" в массиве

Метод pop()

Этот метод не только удаляет последний элемент массива, но и возвращает его в качестве результата. Чтобы удостовериться в этом, выполните следующее.

1. Закомментируйте созданную в предыдущем разделе строку кода, которая содержит метод `lastIndexOf()`.
2. Введите под комментарием следующую инструкцию.

   ```
   people = people.pop();
   ```

3. Чтобы сохранить программу и выполнить ее, щелкните в верхней части окна на кнопке **Update**.

   На панели результатов отобразится последний элемент, содержащийся в исходном массиве `people`.

   Легко заметить, что приведенная выше конмада не только удаляет последний элемент, но и делает его единственным в массиве. Если же вам нужно всего лишь удалить из массива последний элемент, то используйте несколько иной код.
4. Закомментируйте предыдущую инструкцию.
5. Введите под текущим комментарием такое выражение.

   ```
   people.pop();
   ```

6. Чтобы выполнить программу, щелкните на кнопке **Update**.

   На панели результатов отобразится список значений массива `people` без последнего известного элемента (рис. 11.8).

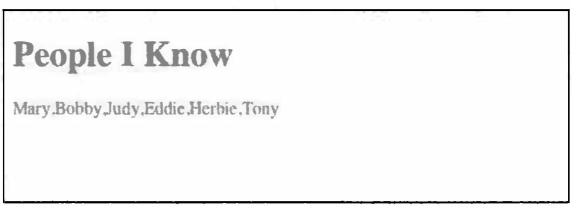


Рис. 11.8. Бобби (Bobby) теперь отсутствует в списке

Метод push()

Метод `push()` применяется для добавления в конец массива одного или нескольких новых элементов, но возвращает количество элементов модифицированного массива (другими словами, его длину).

Для знакомства с возможностями метода `push()` выполните следующие действия.

1. Закомментируйте предыдущую инструкцию.
2. Введите под текущим комментарием такое выражение.

   ```
   people = people.push("Teddy");
   ```

3. Чтобы выполнить программу, щелкните на кнопке **Update**.

На панели результатов появится число, указывающее общее количество элементов в массиве с учетом добавленного значения (8). Тем не менее, использовав приведенное выше выражение, мы удалим в массиве все старые элементы и создадим в нем только один, представленный индексом 8. Чтобы сохранить массив в виде, который он имеет после добавления в него дополнительного элемента, нужно использовать несколько иной код.

4. Закомментируйте предыдущую инструкцию.
5. Введите под текущим комментарием такое выражение.
   ```
   people.push("Teddy");
   ```
6. Чтобы увидеть результат выполнения программы, щелкните на кнопке Update.

   В качестве результата отобразится обновленный список элементов массива `people`, в конце которого находится значение `Teddy`, как показано на рис. 11.9.

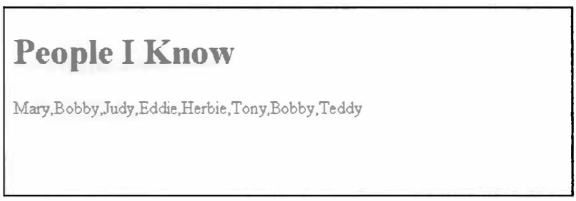


Рис. 11.9. Массив `people`, дополненный элементом `Teddy`

Метод `reverse()`

Название метода говорит само за себя. Его задача — изменить порядок следования элементов на противоположный (инвертировать массив). После его применения последний элемент массива становится первым, предпоследний — вторым и т.д.

Чтобы увидеть, как это работает, следуйте приведенным ниже инструкциям.

1. Закомментируйте предыдущую инструкцию.
2. Введите под текущим комментарием такое выражение.
   ```
   people = people.reverse();
   ```
3. Щелкните на кнопке Update.

   Результат выполнения приведенного выше выражения показан на рис. 11.10. Если сравнить преобразованный массив с исходным, то легко заметить инвертирование порядка следования элементов в нем.

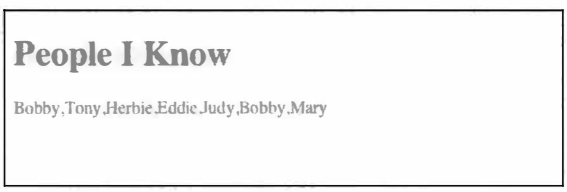


Рис. 11.10. Инвертирование массива

Методы shift() и unshift()

Метод `shift()` востребован при удалении первого элемента массива. О том, как выполнить эту задачу, речь идет в следующих инструкциях.

1. Закомментируйте предыдущий код.
2. Введите под текущим комментарием такое выражение.
   ```
   people.shift();
   ```
3. Щелкните на кнопке **Update**.

   На панели результатов отобразится массив, в котором по сравнению с исходным отсутствует первый элемент: `Mary` (рис. 11.11).

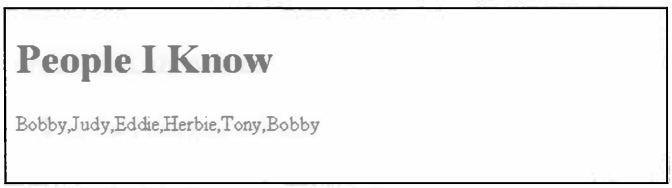


Рис. 11.11. Результат удаления первого элемента массива методом `shift()`

В качестве антипода метода `shift()` выступает метод `unshift()`. Последний добавляет новый элемент в начало массива, что продемонстрировано в следующем примере.

1. Закомментируйте предыдущий код.
2. Введите под текущим комментарием такое выражение.
   ```
   people.unshift("Teddy");
   ```
3. Щелкните на кнопке **Update**.

   На панели результатов отобразится массив, в который по сравнению с исходным добавлен новый первый элемент: `Teddy`.

Метод slice()

В задачу метода `slice()` входит образование нового массива на основе элементов старого. Чтобы создать новый массив на базе уже существующего, выполните такие инструкции.

1. Закомментируйте предыдущую инструкцию.
2. Введите под текущим комментарием такое выражение.
   ```
   people = people.slice(0,3);
   ```
3. Щелкните на кнопке **Update**.

   На панели **Results** отобразится список элементов нового массива, являющегося всего лишь частью исходного массива (рис. 11.12).

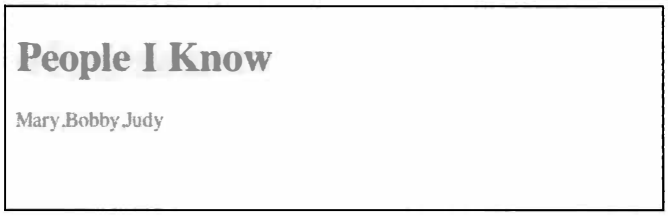


Рис. 11.12. Метод `slice()` вырезает часть элементов из исходного массива и образует новый массив

Метод `sort()`

Метод `sort()` призван обеспечить сортировку элементов в алфавитном порядке. Пример его использования приведен ниже.

1. Закомментируйте предыдущую инструкцию.
2. Введите под текущим комментарием такое выражение.

```
people = people.sort();
```

3. Щелкните на кнопке **Update**.

В качестве результата будет выведен исходный массив, но отсортированный в алфавитном порядке, что продемонстрировано на рис. 11.13.

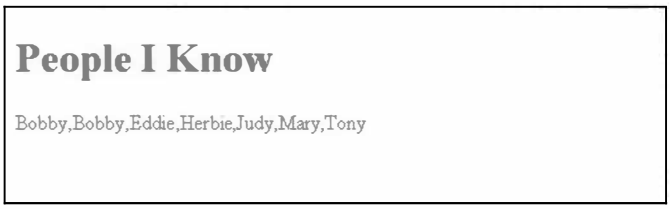


Рис. 11.13. Порядок следования элементов изменен — они располагаются в массиве в алфавитном порядке

Метод `splice()`

Последний рассматриваемый в этой главе метод, `splice()`, применяется для добавления или удаления элементов массива с определенными индексами. Чтобы узнать, как выполняется эта задача, следуйте приведенным далее инструкциям.

1. Закомментируйте предыдущий код.
2. Введите под текущим комментарием такое выражение.

```
people.splice(1,0,"Cathy");
```

В этом коде указывается добавить в массив новый элемент (`Cathy`), расположив его после первого. При этом в массиве не нужно удалять уже существующий в указанной позиции элемент (именно для этого используется второй аргумент, `0`).

3. Щелкните на кнопке Update.

   На рис. 11.14 показано, что новый элемент, `Cathy`, добавлен в массив сразу после элемента `Mary`.

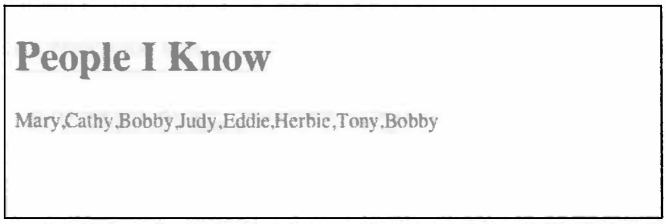


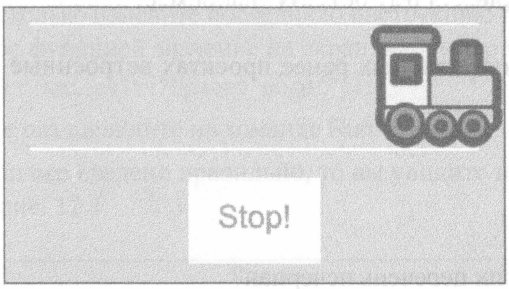
Рис. 11.14. Результат применения метода `splice()`

ГЛАВА 12

Вездесущие функции

Функции — это основные строительные блоки программ JavaScript. Они позволяют избегать ввода повторяющегося кода и делать программы проще для понимания.

В этой главе мы воспользуемся функциями при разработке игры Function Junction (Сосредоточение функций).



Что такое функция

Функция представляет собой небольшую программу внутри другой программы. Функции прекрасно справляются с выполнением задач, которые в рамках общей программы необходимо выполнить многократно. Кроме того, функции помогают правильно структурировать программу и расширяют область ее применения.

Встроенная функция

В JavaScript вы встретите большое количество встроенных функций — методов, которыми наделяются строки, числа, массивы и другие объекты. Узнать о том, что перед вами функция, очень просто. У функции после названия отображаются круглые скобки.

Метод всегда относится к определенному объекту (например, `document`), но не перестает быть функцией.

Вы уже использовали в рассматриваемых ранее проектах встроенные функции, хотя и не подозревали об этом.

```
getElementById()
toString()
addEventListener()
indexOf()
```

Вы же не думаете, что на этом их перечень исчерпан?

Пользовательская функция

В дополнение к встроенным функциям в JavaScript существует возможность написания собственных функций. О том, как это делается, речь пойдет далее.

Многим кажется, что термин *функция* (function) произошел от английского слова *веселье* (fun). В этом есть доля правды. Как нельзя обойтись без функций при написании программы на JavaScript, так невозможно представить себе этот процесс скучным.

Если вы внимательно изучали предыдущие главы, то знаете, с какой целью функции использовались в демонстрационных проектах. Ниже приведен пример еще одной простой функции, которая добавляет в текст смайлик.

```
function smileyIt(theText) {
   theText += " :)";
   return theText;
}
```

Для применения этой функции в реальном примере выполните следующие действия.

1. Запустите в браузере приложение JSFiddle.net.

   В окне должен отобразиться пустой проект. Если в приложении открыт какой-то проект, то "обнулите" его, щелкнув в левом верхнем углу на логотипе JSFiddle.

2. Введите на панель JavaScript код функции `smileyIt()`, приведенной выше.

3. Щелкните на кнопке Run.

   Легко заметить, что на панель результатов ничего не выведено. А все потому, что код функции не выполнится до тех пор, пока функция не будет вызвана в основной программе.

4. Введите под кодом функции следующую команду.

```
smileyIt("Привет всем!");
```

5. Щелкните на кнопке Run.

   На панель результатов снова ничего не выведено. Но на этот раз программа все же выполняет определенные действия. Вы просто не можете знать об этом, поскольку не снабдили программу командой вывода результата выполнения функции.

6. Несколько измените последнюю инструкцию так, чтобы она выводила возвращаемое функцией значение на экран, например в виде всплывающего сообщения.

```
alert(smileyIt("Привет всем!"));
```

7. Еще раз щелкните на команде Run.

   Если все введено правильно, то вы увидите извещение, подобное показанному на рис. 12.1.

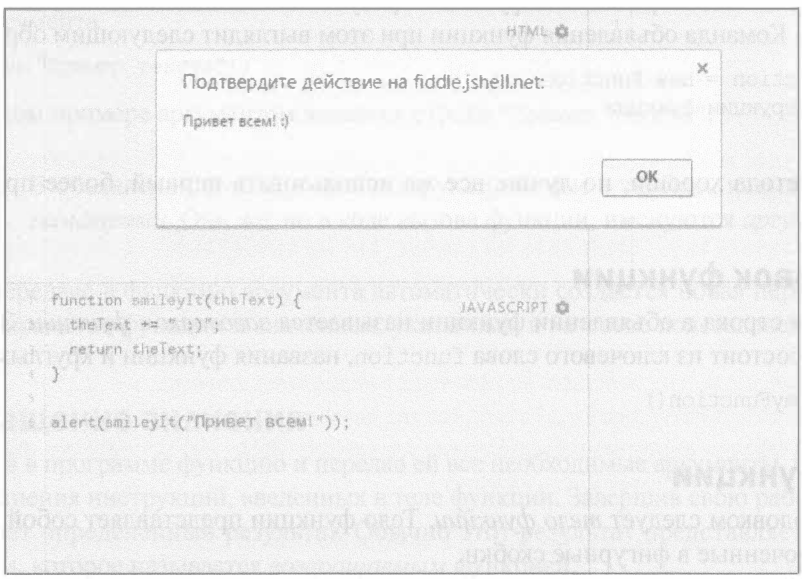


Рис. 12.1. Вывод результата выполнения функции в виде всплывающего сообщения

Команда `alert()` также относится к встроенным функциям JavaScript.

Структура функции

Код создания и вызова функции в программе строго структурирован, и при его написании необходимо придерживаться определенных правил. Чтобы научиться создавать собственные функции и эффективно применять их в веб-приложениях, вам нужно знать эти правила и неукоснительно следовать им. Давайте посмотрим, какой код лежит в основе каждой функции.

Определение функции

Написание кода функции называется ее определением. Весь код, включенный в определение функции, выполняется при вызове функции в основной программе.

Определить функцию можно несколькими способами. Самый привычный заключается во вводе ключевого слова `function`; далее указываются имя функции и круглые скобки, после которых вводятся инструкции функции, заключенные в фигурные скобки. Общая структура функции выглядит следующим образом.

```
function myFunction() {
   // инструкции функции
}
```

Второй способ определения функции требует использования ключевых слов `new Function`. Команда объявления функции при этом выглядит следующим образом.

```
var myFunction = new Function() {
   // инструкции функции
}
```

Оба метода хороши, но лучше все же использовать первый, более привычный вариант.

Заголовок функции

Первая строка в объявлении функции называется *заголовком функции*. Заголовок функции состоит из ключевого слова `function`, названия функции и круглых скобок.

```
function myFunction()
```

Тело функции

За заголовком следует *тело функции*. Тело функции представляет собой инструкции, заключенные в фигурные скобки.

```
{
   // тело функции
}
```

Вызов функции

Чтобы выполнить инструкции, заключенные в тело функции, ее необходимо *вызвать* в коде программы. Вызывается функция очень просто — достаточно ввести ее имя, сопроводив круглыми скобками.

```
myFunction();
```

Параметры функции

Параметры — это значения, указываемые в круглых скобках при вызове функции. Чтобы добавить в функцию параметр, в определении функции введите его имя, расположив в круглых скобках.

```
function myFunction(theText) {
}
```

Чтобы добавить в функцию несколько параметров, их необходимо указывать в скобках, разделенных запятыми.

Передача аргументов

При вызове функции и вводе в круглых скобках значения параметра говорят о передаче аргумента.

```
myFunction("Пример текста");
```

В данном примере аргументом является строка `"Пример текста"`.

В объявлении функции значения, указываемые в скобках, называются *параметрами*. Они же, но в коде вызова функции, именуются *аргументами*.

При передаче в функцию аргумента автоматически создается новая переменная с именем параметра функции и значением, указанным в качестве подставляемого аргумента.

Возвращение значения

Вызвав в программе функцию и передав ей все необходимые аргументы, вы ожидаете выполнения инструкций, введенных в теле функции. Завершив свою работу, функция выдает определенный результат. Обычно этот результат представляется неким значением, которое называется *возвращаемым* функцией.

В функции возвращаемое значение обозначается с помощью специальной команды, которая так и называется — *оператор возврата*. Например, следующая функция всегда возвращает значение `3000`.

```
function whatsTheNumber(){
   return 3000;
}
```

Чтобы отобразить на экране или использовать возвращаемое функцией значение, необходимо вызвать функцию в выражении, выполняющем определенное действие. Например, для проведения вычислений над возвращаемым функцией результатом она вызывается в конечном выражении как один из операндов (см. главу 8).

```
var theTotal = whatsTheNumber() + 80;
```

В приведенном выше коде для переменной `theTotal` устанавливается значение, полученное как сумма возвращаемого функцией `whatsTheNumber()` значения (`3000`) и числа `80`.

Если в функции оператор возврата не указать, то ею возвращается значение `undefined`.

Игра Function Junction

Теперь, когда вы познакомились с основными программными конструкциями, обеспечивающими управлением функциями в JavaScript, давайте используем полученные знания для написания еще одной игры. Назовем ее Function Junction (Сосредоточение функций). Цель этой игры состоит в обеспечении максимально быстрого и безостановочного движения поезда вдоль намеченного пути.

Как и ранее, прежде чем приступить к созданию новой игры, давайте посмотрим, что же у нас должно получиться в конечном итоге.

1. Запустите браузер и перейдите на страницу `http://jsfiddle.net/user/forkids/fiddles`.
2. Найдите в общедоступном кабинете заголовок **Chapter 12 — Function Junction** и щелкните на нем. Игра загрузится в приложении JSFiddle.

Чтобы начать игру, щелкните на поезде. Он начнет двигаться вдоль страницы с очень небольшой скоростью. Следующий щелчок на поезде немного увеличит скорость его движения. Чем больше раз вы щелкнете на поезде, тем быстрее он будет двигаться. Если продолжать бесконтрольно щелкать на поезде, то можно разогнать его до такой скорости, что уследить за ним будет нереально сложно.

Если не остановить поезд до того, как он достигнет конца пути, то он разобьется. Остановка поезда выполняется щелчком на кнопке **Stop**. Но успеете ли вы рвануть стоп-кран, чтобы предотвратить аварию?

Подобно многим другим, эта компьютерная игра достаточно скучная, чего не скажешь по одному только ее описанию. Но ее основная задача не в развлечении, а в обучении методам использования функций в программах JavaScript. Надеюсь, что вдоволь потренировавшись, вы самостоятельно напишете более увлекательную и полнофункциональную игру — симулятор железной дороги.

Давайте подробно рассмотрим игру Function Junction.

Начните с изучения предварительной версии игры Chapter 12 — Function Junction – Start, которая загружается из общедоступного кабинета приложения JSFiddle.

Загрузив указанную версию игры, скопируйте ее в личный кабинет, выполнив следующие действия.

1. Щелкните на кнопке Fork для создания копии игры.
2. Воспользуйтесь командой Fiddle Options, чтобы дать игре собственное название.
3. Щелкните на кнопке Update, а затем используйте команду Set as Base для сохранения программы.

Все готово к изучению кода игры!

Код HTML

Игра Function Junction представлена очень простым HTML-кодом, полностью состоящим из известных вам конструкций.

Код CSS

Перейдем к рассмотрению содержимого панели CSS. На ней приведен весь код CSS, востребованный при оформлении нашей игры. Чтобы узнать, как игра выглядит без использования CSS-стилей, взгляните на рис. 12.2. Разница более чем заметна, не правда ли?

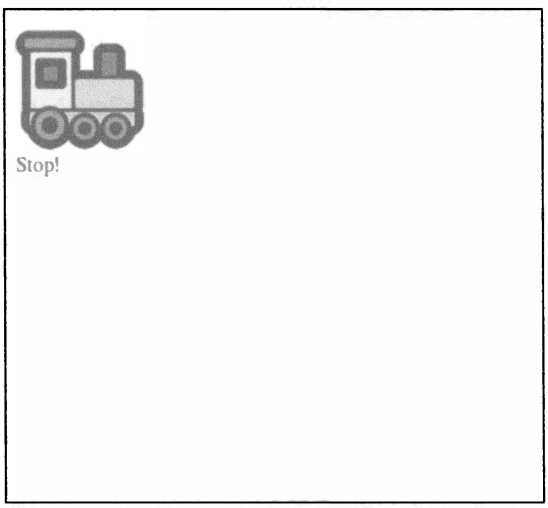


Рис. 12.2. Игра Function Junction без использования технологии CSS

Никаких сомнений — без кода CSS игра остается неполноценной! Все правила CSS, используемые при стилизации элементов игры, приведены в листинге 12.1.

Листинг 12.1. CSS-код игры Function Junction

```
body {
   font-family: Arial, sans-serif;
}
#container {
   padding: 10px;
   width: 360px;
   height: 80%;
   background-color: #00FF00;
}
#track {
   width: 340px;
   border-top: 2px solid white;
   border-bottom: 2px solid white;
   margin: 20px auto;
}
#train {
   height: 92px;
   width: 100px;
   position: relative;
   left: 0px;
}
#stopButton {
   padding-top: 15px;
   margin: 10px auto;
   background-color: white;
   width: 100px;
   height: 50px;
   color: red;
   text-align: center;
   font-size: 24px;
   line-height: 30px;
}
```

Внимательно изучите все селекторы. Обратите внимание на то, что селектор элементов назначается только элементу `body`, у которого изменяется свойство `font-family`. Во всех остальных случаях применяется селектор идентификаторов, начинающийся с символа хеш-тега (`#`).

Большинство приведенных на панели CSS свойств уже описывались нами при рассмотрении предыдущих проектов, и мы не будем лишний раз повторяться. Тем не менее некоторые правила стоят пристального изучения, особенно если вы планируете модернизировать игру в дальнейшем.

Найдите правило, определяющее форматирование элемента `container`. Оно отвечает за размер и цвет фона. Исходно устанавливается зеленый фон, но можно поэкспериментировать с другими оттенками. Чтобы сделать игру интереснее (и заставить поезд двигаться дальше), измените свойство `width` (за длину пути также отвечают некоторые другие свойства, описанные далее).

Обратите свой взор на свойства, устанавливаемые для элемента `track`. В них указывается окрасить железнодорожный путь в белый цвет. В нашем примере железнодорожный путь представляется двумя линиями, которые в свою очередь являются верхней и нижней границами широкого прямоугольника, внутри которого движется поезд. Если вы хотите удлинить путь следования поезда еще одним способом, то измените свойство `width` элемента `track`.

Проведите некоторое время, изменяя значения свойств в правилах CSS и щелкая на кнопке Run. Внесенные в код правки немедленно отразятся на панели результатов.

Если вы по неосторожности внесете в код непоправимые изменения, которые нарушают нормальный ход игры, то вернитесь в общедоступный кабинет, повторно скопируйте игру и начните тестирование с самого начала.

Код JavaScript

Вот мы и добрались до панели JavaScript!

Из начальной версии игры удалена большая часть JavaScript-кода, присутствующего в окончательной версии приложения. Удаленный код заменен комментариями, описывающими действия, которые вам предстоит запрограммировать самостоятельно. Содержимое панели JavaScript для начальной версии игры показано в листинге 12.2.

Многие комментарии начинаются с ключевого слова `todo:`. Это стандартное обозначение, используемое профессиональными программистами для указания самому себе и другим разработчикам строк кода, которые требуют завершения или улучшения.

Листинг 12.2. Панель JavaScript с инструкциями по написанию кода игры

```
/*
todo: Создать три глобальные переменные:
* trainSpeed (начальное значение = 250)
* trainPosition (начальное значение = 0)
* animation (начальное значение не устанавливается)
*/

/*
todo: Обрабатывать щелчки на поезде, вызывая функцию speedUp
*/

/*
todo: Обрабатывать щелчки на кнопке Stop, вызывая функцию stopTrain
*/

function speedUp() {
   /*
   todo: Определить, движется ли поезд с максимальной скоростью.
   Если нет, то увеличить скорость
   */
```

```
    /*
    Если поезд движется, то остановить его и перезапустить с новой
    скоростью, вызвав функцию frame()
    */

    function frame() {
        /*
        todo: Изменить положение поезда и проверить, не разбился ли он
        */

    }
}

function stopTrain() {
    /*
    todo: Проверить, не попал ли поезд в аварию. Если нет, то
    остановить поезд
    */
}

function checkPosition(currentPosition) {
    /*
    todo: Определить положение поезда и разбить его, если он
    находится в конце пути
    */
}
```

Внимательно изучите приведенные в комментариях инструкции. Попробуйте самостоятельно выполнить как можно больше из них. Если у вас что-то не получилось, то сверьтесь с приведенным ниже пошаговым описанием действий, обеспечивающих решение всех приведенных в коде задач.

Задачи, рассмотренные в следующих инструкциях, решаются в порядке их следования в листинге 12.2.

1. В первом комментарии указывается создать три переменные, поэтому введите под ним такие команды.

   ```
   var trainSpeed = 250;
   var trainPosition = 0;
   var animation;
   ```

2. Следующий комментарий требует написания обработчика события щелчка на поезде. Прослушивание события выполняется с помощью функции `addEventListener()`, уже известной вам по предыдущим главам. Обработчик события привязывается к элементу с идентификатором `id = "train"` с помощью такого кода.

   ```
   var train = document.getElementById("train");
   train.addEventListener("click", speedUp);
   ```

3. Еще один обработчик события привязывается к кнопке Stop. Его код во многом подобен предыдущему.

```
var stopButton = document.getElementById("stopButton");
stopButton.addEventListener("click", stopTrain);
```

4. Теперь можно переходить к написанию кода функций. В функции `speedUp()` сначала проверяется, не едет ли поезд с максимальной скоростью. Для этого применяется такая условная конструкция.

```
if (trainSpeed > 10) {
   trainSpeed -= 10;
}
```

В этой конструкции значение переменной `trainSpeed` сравнивается с `10`. Если скорость поезда больше 10, то ее все еще можно увеличивать, поэтому в следующей строке из значения переменной `trainSpeed` вычитается `10`.

Скорость анимации определяется вторым параметром функции, `setInterval()`. Он указывает время (в миллисекундах), через которое выполняется следующий шаг анимации. Чем меньше значение этого параметра, тем быстрее выполняются шаги анимации, и поезд движется быстрее.

5. Вторая задача, решаемая функцией `speedUp()`, — это перезапуск анимации, но уже с новой скоростью. Она выполняется с помощью следующих двух строк кода.

```
clearInterval(animation);
animation = setInterval(frame, trainSpeed);
```

Первое выражение приостанавливает анимацию, для чего вызывается функция `clearInterval()`. Во втором выражении анимация возобновляется (с помощью функции `setInterval()`, которой передается обновленный параметр `trainSpeed`). Функция `frame()` вызывается внутри функции `setInterval()`.

6. Теперь можно приступать к написанию функции `frame()`. Она во многом подобна одноименной функции `frame()`, используемой нами для анимации робота Дугласа (см. главу 7). В нее нужно внести лишь незначительные изменения, учитывающие условия новой задачи. В результате модифицированная функция `frame()` для игры Function Junction принимает следующий вид.

```
function frame() {
   trainPosition += 2;
   train.style.left = trainPosition + 'px';
   checkPosition(trainPosition);
}
```

Сначала функция увеличивает значение переменной `trainPosition`, а затем обновляет согласно ему текущее положение поезда. Как только положение поезда изменяется, вызывается функция `checkPosition()`, которой передается значение переменной `currentPosition`.

7. Функция `checkPosition()` имеет следующий вид.

```
function checkPosition(currentPosition) {
   if (currentPosition === 260) {
      alert("Crash!");
      console.log("Crash!");
      clearInterval(animation);
   }
}
```

Функция принимает единственный аргумент: `currentPosition`. Она проверяет, равен ли этот аргумент значению `260` (именно на таком расстоянии от левого края пути, измеренном в пикселях, поезд попадает в аварийную ситуацию). Как нетрудно заметить, это значение определяет общую длину железнодорожного пути. Если вы хотите увеличить его, то самое время сделать это здесь.

8. Последней создается функция `stopTrain()`. Она имеет такой вид.

```
function stopTrain() {
   if (trainPosition < 260) {
      clearInterval(animation);
   }
}
```

Данная функция запускается после щелчка на кнопке **Stop**. В ней предварительно проверяется, не разбился ли поезд, для чего значение переменной `trainPosition` сравнивается с длиной пути (`260`).

Это последнее место в коде, где можно изменить общую длину пути, по которому движется наш поезд.

9. Щелкните на кнопке **Update**, чтобы сохранить код приложения и запустить игру для тестирования.

Если вы все сделали правильно, то поезд начнет двигаться. Если этого не происходит, то внимательно проверьте введенный ранее код. Проверьте наличие сообщений об ошибках на консоли JavaScript. Очень часто в них указываются точные причины возникших неполадок.

Дополнительное задание: удлинение пути

Описывая код игры, мы указывали на места, в которых можно изменить общую длину пути, по которому движется поезд. Попробуйте воспользоваться каждым из способов.

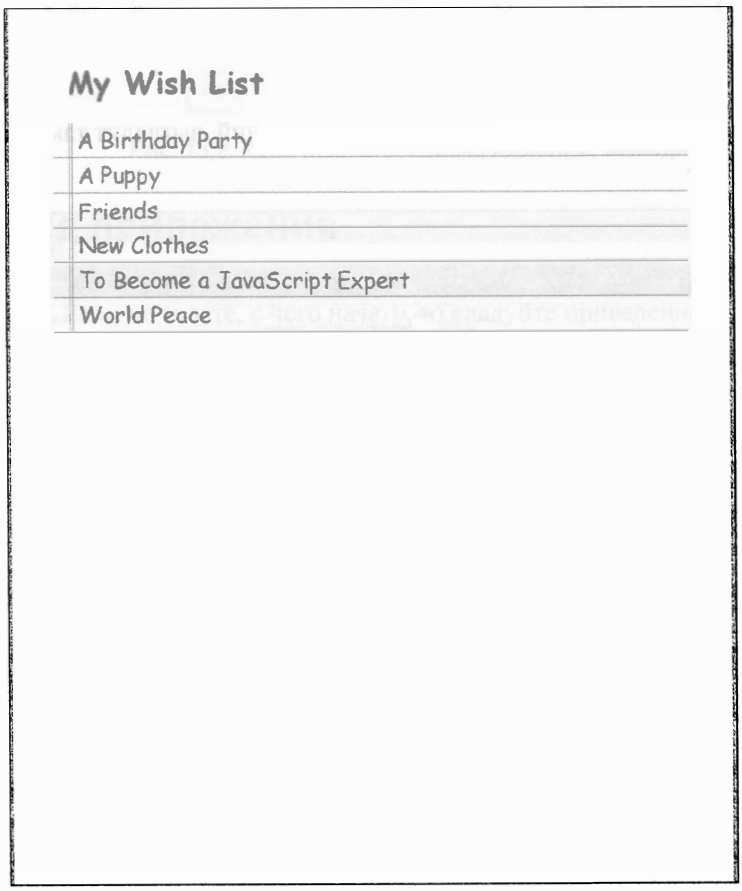
Просмотрите код, представленный на панелях CSS и JavaScript, и увеличьте значения, которые отвечают за длину железнодорожного пути. По завершении щелкните на кнопке **Update**, чтобы ознакомиться с результатами изменения приложения. Попадает ли сейчас поезд в аварию на границе фона? Если нет, то что нужно изменить, чтобы исправить ситуацию?

ГЛАВА

13

Список желаний

Каждый из нас хоть раз в жизни мечтал поймать рыбку, которая умеет выполнять три желания. Но прежде чем отправиться на заветную рыбалку, нужно составить список желаний, чтобы в самый ответственный момент не спасовать перед лицом неожиданно свалившегося на голову счастья. В этой главе мы воспользуемся массивами для составления списка желаний на случай непредвиденной встречи с любым сказочным персонажем, умеющим их исполнять.



Составление списка желаний

В нашем приложении, отвечающем за создание списка желаний, данные у пользователя запрашиваются с помощью HTML-формы. Впоследствии они сохраняются в виде массива, а отображаются на экране в виде списка HTML. Для вывода списка желаний в отсортированном, предельно понятном для изучения виде наше приложение необходимо снабдить кнопкой Print.

Просмотр готового приложения

Чтобы понять, как должно выглядеть и функционировать приложение для составления списка желаний, выполните следующие действия.

1. Перейдите к общедоступному кабинету приложения JSFiddle, открыв в браузере следующую страницу:

   `http://jsfiddle.net/user/forkids/fiddles`

2. В списке доступных программ найдите заголовок **Chapter 13 — Wish List — Finished** и щелкните на нем.

   Конечная редакция приложения, создающего список желаний, выглядит так, как показано на рис. 13.1.

Рис. 13.1. Программа создания списка желаний в конечном виде

3. Введите произвольный текст в поле HTML-формы, например `Всем добра!`, и щелкните на кнопке **Add it** (Добавить).

   Введенное вами значение будет помещено в список и удалено из текстового поля.

4. Наполните список другими желаниями.

   Получив список из 5–6 элементов, переходите к следующему пункту.

5. Щелкните на кнопке Print Your List (Вывести список).

   На экране появится список запрошенных вами желаний, приятно отформатированный и упорядоченный по алфавиту (рис. 13.2).

Рис. 13.2. Конечный вид списка желаний, выводимого приложением

Код приложения

Вы знаете достаточно, чтобы приступить к самостоятельной разработке приложения. Если не знаете, с чего начать, то следуйте приведенным ниже инструкциям.

1. Откройте в приложении JSFiddle общедоступный кабинет, найдите в списке приложений заголовок Chapter 13 — Wish List — Start и щелкните на нем.

   В качестве отправной точки вы получаете приложение, имеющее вид, показанный на рис. 13.3.

2. Щелкните в верхней части окна на кнопке Fork, чтобы скопировать код приложения.

3. Воспользовавшись командой Fiddle Options, дайте своей копии программы новое название.

4. Для сохранения копии программы в личном кабинете щелкните на кнопке Update, а затем выполните команду Set as Base.

Ознакомившись с панелями JSFiddle, вы заметите, что начальная версия приложения снабжена всем необходимым кодом HTML и CSS, а вместо кода JavaScript приведены комментарии с инструкциями по его написанию.

Приняв вызов, попробуйте написать код JavaScript самостоятельно, ориентируясь только на подсказки в комментариях и не заглядывая в приведенный далее материал. В своих экспериментах используйте все известные вам методики — их более чем достаточно, чтобы решить поставленную

задачу. Протестируйте свое приложение перед тем, как перейти к изучению оставшейся части главы. Только после этого сравните полученный результат с нашей версией приложения, описанной далее.

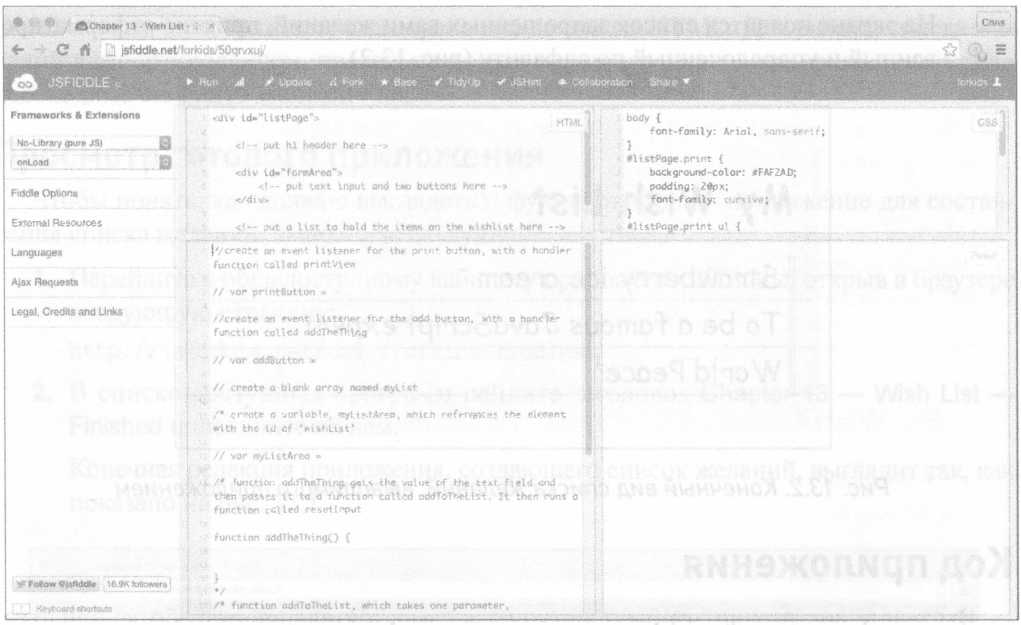


Рис. 13.3. Начальная версия приложения, создающего список желаний

Код HTML

Подобно остальным уже завершенным проектам, создавать настоящее приложение мы начнем с написания кода HTML. В первой строке, добавляемой на панель HTML, вводится код открывающего тега элемента `div`, выступающего контейнером всего, что выводится на веб-странице.

```
<div id="listPage">
```

В начальной версии приложения этот код уже присутствует, и вам не нужно вводить его повторно. А вот дальнейшие действия потребуют вашего непосредственного участия.

1. Добавьте на страницу крупный заголовок, воспользовавшись элементом `h1`.

   ```
   <h1>My Wish List</h1>
   ```

   Обратите внимание, что на панели HTML уже содержится код формы, применяемой для запроса у пользователя необходимых данных. Представлен он следующим тегом.

   ```
   <div id="formArea">
   ```

2. Нам нужно добавить в эту форму поле, в которое вводятся пользовательские желания. Элемент поля заключается в открывающий и закрывающий теги элемента `label`, что позволяет добавить слева от поля надпись, в нашем случае — **What I want** (Чего я хочу).

```
<label>What I want:
   <input type="text" id="iWant" />
</label>
```

3. Создайте первую кнопку формы с надписью **Add It!** (Добавить!) на ней, введя такой код.

```
<button type="button" id="addIt">Add It!</button>
```

Кнопка будет размещена в одной строке с полем ввода данных, добавленным на страницу ранее.

4. С помощью следующего кода на форму добавляется вторая кнопка — **Print Your List** (Вывести список).

```
<button type="button" id="printable">Print Your List</button>
```

5. Чтобы поместить кнопку **Print Your List** под кнопкой **Add it!**, создайте два элемента `br`, расположив их код между тегами создания кнопок.

```
<br /><br />
```

6. Теперь создайте пустой элемент `ul`, снабдив его идентификатором `id="wishList"`.

```
<ul id="wishList"></ul>
```

7. Наконец, закройте элемент `div`, открытый в п. 1 и содержащий все отображаемые на странице данные.

```
</div>
```

8. Щелкните на кнопке **Update**, чтобы сохранить весь созданный ранее код и просмотреть результат его выполнения на панели **Results**.

Приложение должно выглядеть так, как показано на рис. 13.4.

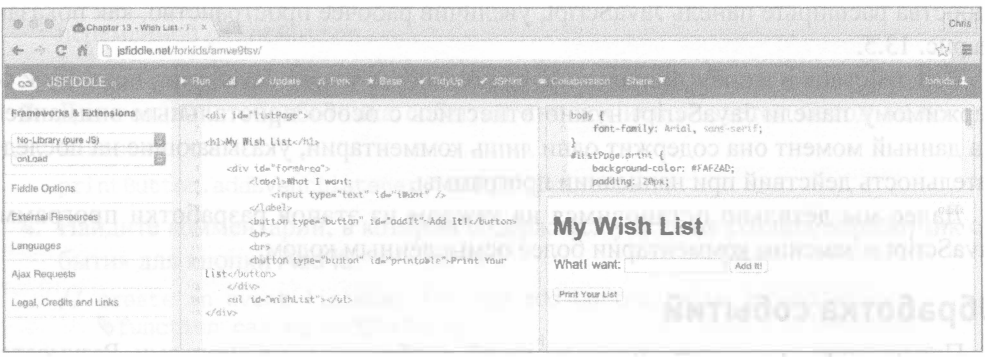


Рис. 13.4. Завершенный HTML-код приложения, создающего список желаний

При желании можете удалить с панели HTML все комментарии, исходно используемые как заполнители создаваемого вручную кода. Но лучше просто изменить их, снабдив себя подсказками о назначении каждой из частей страницы, чтобы иметь возможность в будущем использовать ее в других проектах.

Конечный HTML-код нашего приложения приведен в листинге 13.1.

Введенный вами код может отличаться от приведенного ниже, но это не означает, что он неработоспособный. Часто иной внешний вид вызван использованием других интервалов и отступов. Чтобы "подчистить" свой HTML-код и привести его к общепринятому виду, щелкните в верхней части окна приложения JSFiddle на кнопке TidyUp.

Листинг 13.1. HTML-код приложения, создающего список желаний

```
<div id="listPage">

   <h1>My Wish List</h1>

   <div id="formArea">
      <label>What I want:
         <input type="text" id="iWant" />
      </label>
      <button type="button" id="addIt">Add It!</button>
      <br /><br />
      <button type="button" id="printable">Print Your List</button>
   </div>
   <ul id="wishList"></ul>
</div>
```

Код JavaScript

После HTML-кода нужно переходить к написанию кода JavaScript. Для большего удобства расширьте панель JavaScript, увеличив рабочее пространство, как показано на рис. 13.5.

Все действия в нашем приложении выполняются в коде JavaScript, поэтому к содержимому панели JavaScript нужно отнестись с особо пристальным вниманием. На данный момент она содержит одни лишь комментарии, указывающие на последовательность действий при написании программы.

Далее мы детально остановимся на каждом из этапов разработки программы JavaScript и заменим комментарии более осмысленным кодом.

Обработка событий

Приложение, создающее список желаний, снабжено двумя кнопками. Разумеется, каждую из них необходимо снабдить обработчиком такого события, как щелчок. Для выполнения этой задачи следуйте приведенным ниже инструкциям.

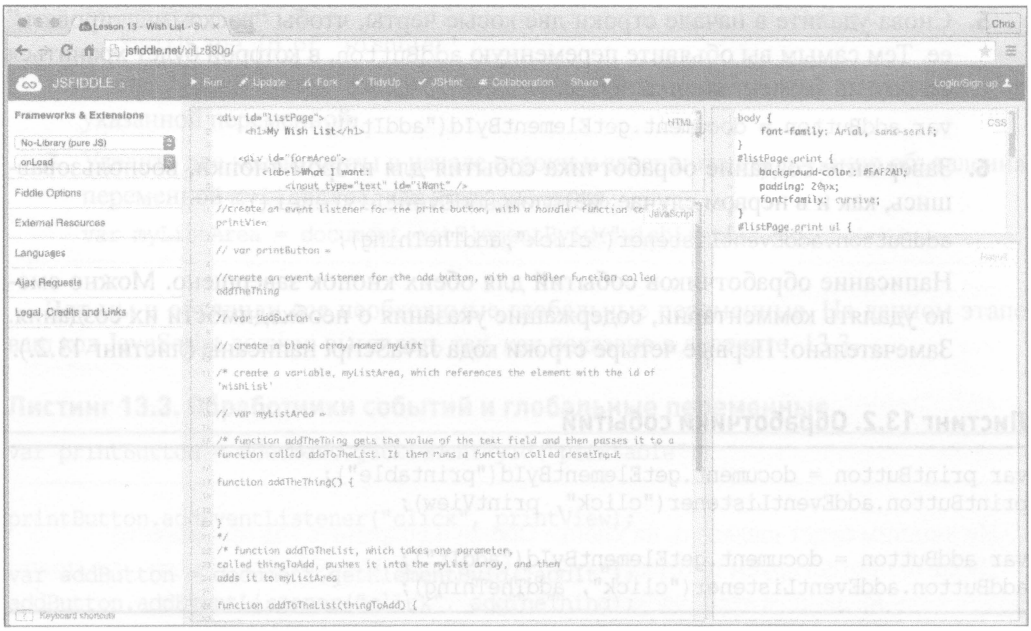


Рис. 13.5. Панель JavaScript увеличенного размера

1. Найдите комментарий, указывающий создать обработчик события для кнопки Print Your List.

   ```
   // create an event listener for the print button, with a handler
      ↳function called printView
   ```

   Обратите внимание на вторую строку комментария.

   ```
   // var printButton =
   ```

2. Удалите в этой строке признак комментария (две косые черты в самом начале), образовав код объявления переменной. В его правую часть добавьте ссылку на элемент кнопки Print Your List. Таким образом вы сохраните ссылку на кнопку в переменной `printButton`.

   ```
   var printButton = document.getElementById("printable");
   ```

3. Для создания обработчика событий кнопки используется метод `addEvent Listener()`.

   ```
   printButton.addEventListener("click",printView);
   ```

4. Найдите комментарий, в котором содержится указание создать обработчик события для кнопки Add It.

   ```
   // create an event listener for the add button, with a handler
      ↳function called addTheThing
   ```

   Обратите внимание на следующую строку.

   ```
   // var addButton =
   ```

5. Снова удалите в начале строки две косые черты, чтобы "раскомментировать" ее. Тем самым вы объявите переменную `addButton`, в которой будет храниться ссылка на элемент кнопки `addIt`.

   ```
   var addButton = document.getElementById("addIt");
   ```

6. Завершите создание обработчика события для второй кнопки, воспользовавшись, как и в первом случае, методом `addEventListener()`.

   ```
   addButton.addEventListener("click",addTheThing);
   ```

   Написание обработчиков событий для обеих кнопок завершено. Можно смело удалять комментарии, содержащие указания о необходимости их создания. Замечательно! Первые четыре строки кода JavaScript написаны (листинг 13.2.).

Листинг 13.2. Обработчики событий

```
var printButton = document.getElementById("printable");
printButton.addEventListener("click", printView);

var addButton = document.getElementById("addIt");
addButton.addEventListener("click", addTheThing);
```

Объявление глобальных переменных

Покончив с написанием обработчиков событий, давайте создадим несколько переменных, которые будут использоваться в дальнейшем коде приложения.

Когда вы создаете переменные вне кода функций, они называются *глобальными*. Созданные таким образом переменные можно использовать в любом месте программы.

Переменные, объявленные внутри функции, называются *локальными*. Они могут использоваться только внутри текущей функции и нигде больше.

Чтобы создать глобальные переменные для приложения, создающего список желаний, выполните следующие действия.

1. Отыщите комментарий, предлагающий создать пустой массив `myList`.

   ```
   // create a blank array named myList
   ```

   Замените его следующим выражением.

   ```
   var myList = [];
   ```

   Присвоив переменной квадратные скобки, внутри которых отсутствуют значения, вы создадите пустой массив, не содержащий элементов. Вам еще только предстоит его заполнить.

2. Перейдите к комментарию, в котором указывается создать переменную `myListArea`.

```
/* create a variable, myListArea, which references the element
   ↳with the id of 'wishList'
```

В следующей строке содержится комментарий, включающий код объявления указанной переменной.

3. Удалите две косые черты в начале строки и завершите инструкцию объявления переменной `myListArea`, как показано ниже.

```
var myListArea = document.getElementById("wishList");
```

Вот мы и объявили все необходимые глобальные переменные. На данном этапе ваш код JavaScript должен выглядеть так, как показано в листинге. 13.3.

Листинг 13.3. Обработчики событий и глобальные переменные

```
var printButton = document.getElementById("printable");

printButton.addEventListener("click", printView);

var addButton = document.getElementById("addIt");
addButton.addEventListener("click", addTheThing);

var myList = [];
var myListArea = document.getElementById("wishList");
```

Функции

Остальная часть программы JavaScript отводится для кода функций, отвечающих за выполнение в приложении всех необходимых действий. Среди них — добавление в список новых элементов, очистка содержимого поля ввода данных и форматирование конечного документа.

Рассмотрим функцию `addTheThing()`, которая запускается после щелчка на кнопке Add It. Эта функция создает ссылку на поле ввода данных и передает ее в качестве аргумента двум другим функциям, представленным в программе.

Для написания кода функции `addTheThing()` выполните такие действия.

1. Создайте переменную `theThing` и назначьте ей в качестве значения ссылку на элемент поля ввода данных.

```
var theThing = document.getElementById("iWant");
```

Не забывайте о том, что эта переменная хранит всего лишь ссылку на поля, и не представляет значение, введенное в поле. С ее помощью можно обращаться непосредственно к элементу поля, что используется далее в программе.

2. Передайте переменную `theThing` как аргумент функции `addToTheList()`.

```
addToTheList(theThing);
```

Эта функция получает значение и добавляет в список новый элемент.

3. Передайте переменную `theThing` как аргумент функции `resetInput()`, которая обнуляет содержимое поля ввода данных.

   ```
   resetInput(theThing);
   ```

   Функция `addTheThing()` представлена только этими тремя инструкциями. Конечный ее вид показан в листинге 13.4.

Листинге 13.4. Код функции `addTheThing()`

```
function addTheThing() {

   var theThing = document.getElementById("iWant");

   addToTheList(theThing);
   resetInput(theThing);
}
```

Не лишним будет щелкнуть на кнопке **Update** и сохранить код приложения.

Давайте протестируем программу в текущем виде. Попробуйте угадать, что произойдет, если на данном этапе запустить приложение? Правильно, ничего!

По правде говоря, и это не совсем верно, ведь в результате выполнения кода возникает ошибка.

Чтобы удостовериться в этом, откройте в браузере консоль JavaScript. Для этого выполните команду **Главное меню Chrome⇨Дополнительные инструменты⇨Инструменты разработчика**. Проследите за содержимым консоли после щелчка в приложении JSFiddle.net на кнопке **Update** или **Run**. Не заметить ошибку просто невозможно (рис. 13.6).

В сообщении об ошибке указывается, что в программе отсутствует функция `printView()`, запрашиваемая одним из обработчиков событий.

Сейчас не время заниматься кодом функции `printView()`, поэтому давайте создадим ее "пустышку", состоящую из одной только инструкции объявления.

1. На панели JavaScript приложения JSFiddle найдите строку с комментарием, содержащим объявление функции `printView()`.

   ```
   /*function printView, which outputs a nicely formatted view of the list
   function printView() {

   }
   */
   ```

2. В начале и конце этого кода удалите ключевые слова, делающие его комментарием (`/*` и `*/`).

   Вы получите следующее.

   ```
   function printView() {

   }
   ```

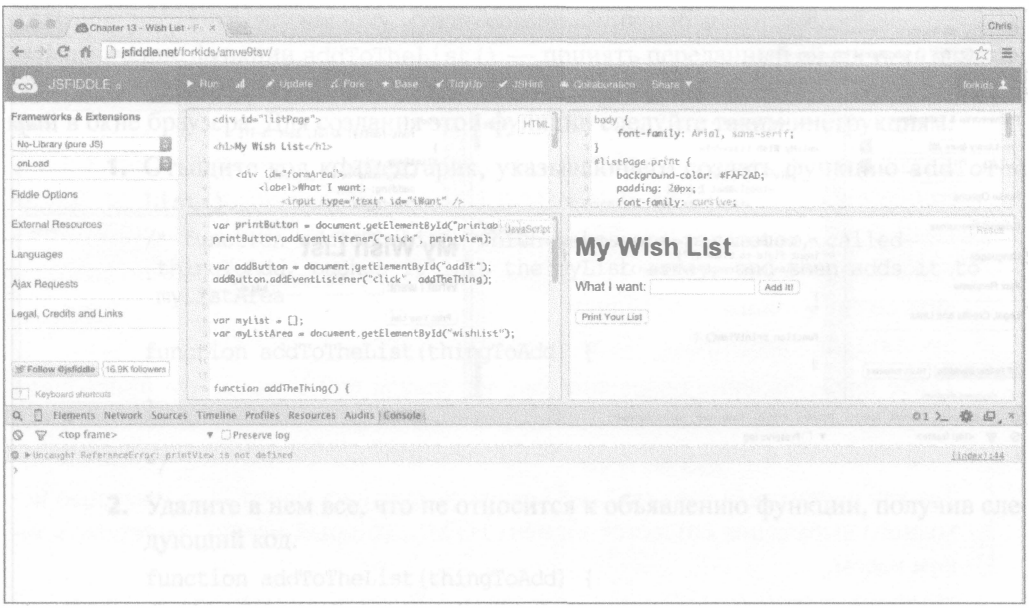


Рис. 13.6. Запуск приложения, создающего список желаний, при незавершенных функциях чреват возникновением ошибок

С формальной точки зрения функция `printView()` теперь существует, хотя и не выполняет никаких действий. Но этого вполне достаточно, чтобы не вызывать ошибку выполнения программы.

3. На консоли JavaScript браузера Chrome щелкните на кнопке **Clear Console Log** (Очистить журнал консоли), представленной значком с изображением перечеркнутого круга. Все сообщения на консоли удаляются.
4. Щелкните в приложении JSFiddle.net на кнопке **Update**, чтобы проверить, будет ли теперь отображаться сообщение об ошибке.

   Если вы ничего не напутали, выполняя пп. 1–3 предыдущих инструкций, то консоль будет оставаться чистой, как показано на рис. 13.7.
5. Не закрывая консоль JavaScript, наберите в поле ввода данных некий текст, а затем щелкните на кнопке **Add It**.
6. На консоли снова появится сообщение об ошибке, как показано на рис. 13.8.

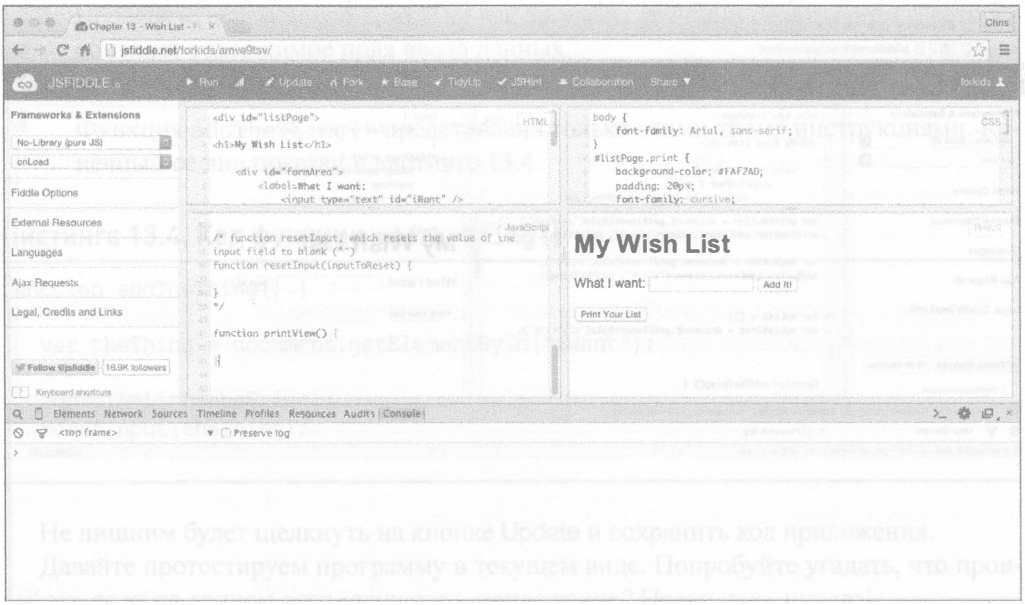


Рис. 13.7. Сообщение об ошибке при запуске приложения, создающего список желаний, не выводится

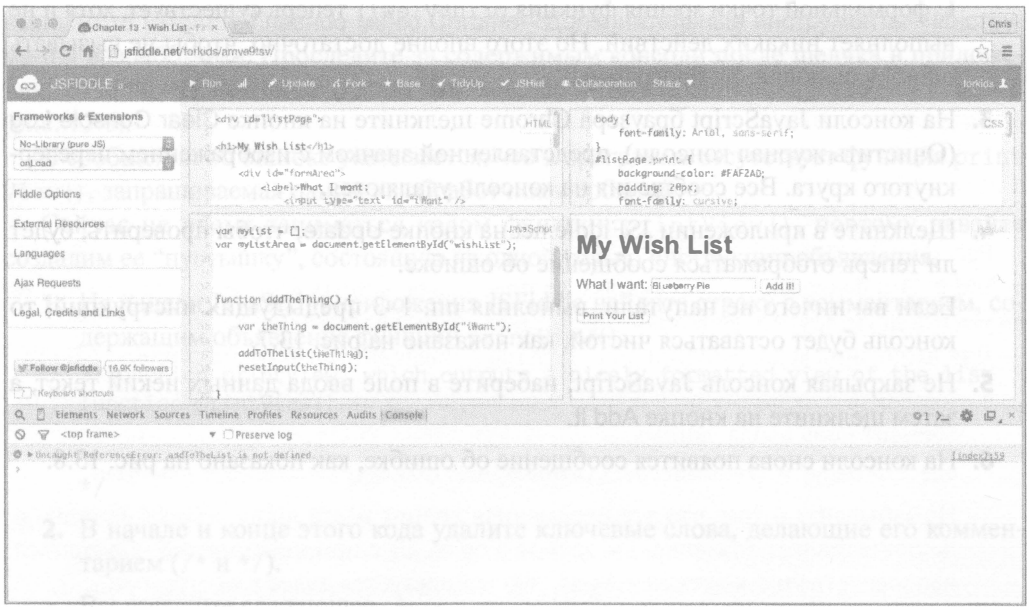


Рис. 13.8. На этот раз отсутствует функция `addToTheList()`

Чтобы устранить эту ошибку, нужно создать функцию `addToTheList()`

Назначение функции `addToTheList()` — принять переданный ей аргумент, сохранить его в массиве и добавить в виде элемента в маркированный список, отображаемый в окне браузера. Для создания этой функции следуйте таким инструкциям.

1. Отыщите код комментария, указывающего создать функцию `addToTheList()`.

```
/* function addToTheList, which takes one parameter, called
 thingToAdd, pushes it into the myList array, and then adds it to
 myListArea

function addToTheList(thingToAdd) {

}

*/
```

2. Удалите в нем все, что не относится к объявлению функции, получив следующий код.

```
function addToTheList(thingToAdd) {

}
```

3. Добавьте такую команду в тело функции.

```
myList.push(thingToAdd.value);
```

В ней с помощью метода `push()` значение, введенное в поле, заносится в массив `myList`.

Как вы уже знаете, метод `push()` добавляет новое значение в конец массива.

4. Введите в теле функции вторую строку кода, отвечающую за создание нового элемента `li`.

```
var newListItem = document.createElement("li");
```

Метод `createElement()` применяется для создания нового элемента в текущем окне браузера. Сам элемент нигде не отображается — он пустой и хранится в переменной `newListItem`.

5. Используя следующий код, вы назначите свойству `innerHTML` нового элемента значение, введенное в поле формы.

```
newListItem.innerHTML = myList[myList.length - 1];
```

Внимательно изучите последнее выражение. Конструкция несколько хитроумная и поэтому требует детального рассмотрения. Начнем ее анализ с выражения, размещенного внутри квадратных скобок после названия массива `myList`.

```
myList.length - 1
```

Это выражение возвращает длину массива `myList`, из которой вычтено значение 1. Такая операция позволяет определить индекс последнего (только что созданного) элемента массива. Единица вычитается из длины массива потому, что нумерация его элементов ведется с нуля.

Давайте посмотрим, как работает эта формула, когда в массиве имеется всего один элемент. В таком случае выражение `myList[myList.length - 1]` соответствует коду `yList[0]` и представляет значение единственного элемента массива.

Если в поле формы ввести текст `Всем добра!`, то полный код можно сопоставить со следующим выражением.

```
newListItem.innerHTML = "Всем добра!";
```

Такое представление более знакомо, не правда ли? Учтите, что присвоение свойству `innerHTML` значения приводит к изменению всего находящегося между открывающим и закрывающим тегами элемента.

Так как переменная `newListItem` хранит ссылку на элемент `li`, с помощью исходного выражения создается элемент HTML, который можно представить таким кодом.

```
<li>Всем добра!</li>
```

Как ни странно, указанный элемент не хранится в документе HTML, и для его вывода на экран нужно добавить в функцию `addToTheList()` еще одну инструкцию.

6. В новой строке введите следующий код.

   ```
   myListArea.appendChild(newListItem);
   ```

   Эта инструкция добавляет только что созданный элемент `li` в конец содержимого элемента, на который ссылается переменная `myListArea`. Для этого в нем используется еще неизвестный вам метод `appendChild()`.

   Вернувшись к предыдущему коду, вы обнаружите, что глобальная переменная `myListArea` предназначается для хранения ссылки на элемент `ul` с идентификатором `"wishList"`.

   Таким образом, последняя команда добавляет в список (элемент `ul`), отображаемый в окне браузера, только что созданный элемент `li`.

7. Щелкните на кнопке **Update** для сохранения введенного выше кода программы.

   Полный код функции `addToTheList()` приведен в листинге 13.5.

Листинг 13.5. Код функции `addToTheList()`

```
function addToTheList(thingToAdd) {
   myList.push(thingToAdd.value);
   var newListItem = document.createElement("li");
   newListItem.innerHTML = myList[myList.length - 1];
   myListArea.appendChild(newListItem);
}
```

1. Наберите в поле ввода данных некий текст и щелкните на кнопке Add It.

   В список, расположенный под полем ввода данных, будет добавлен новый элемент.

2. Вводите в поле другие значения и следите за тем, как список пополняется новыми элементами (рис. 13.9).

Рис. 13.9. Новые элементы добавляются в конец списка

Форма все еще не приведена к финальному виду. Легко заметить, что при вводе в поле нового текста старый текст из него автоматически не удаляется.

Чтобы исправить ситуацию, необходимо написать функцию `resetInput()`, которая будет вызываться функцией `addTheThing()` после каждого добавления в список нового элемента. Задача функции `resetInput()` всего одна — удалять из поля текущее значение, подготавливая его для ввода нового текста.

Напишем функцию `resetInput()`.

1. Найдите на панели JavaScript закомментированный код объявления функции `resetInput()`.

```
/* function resetInput, which resets the value of the input field
   to blank ("")
function resetInput(inputToReset) {

}
*/
```

2. Удалите комментарии, оставив "чистый" код функции.

```
function resetInput(inputToReset) {

}
```

3. Введите в фигурных скобках следующий код.

```
inputToReset.value = "";
```

И это все! Функция `resetInput()` содержит всего одну инструкцию, заменяющую значение свойства элемента (поле ввода данных), ссылка на который хранится в переменной `inputToReset`, пустой строкой. В результате поле очищается от введенных перед этим данных.

4. Для сохранения кода функции в приложении щелкните на кнопке Update.

Конечный вид функции `resetInput()` приведен в листинге 13.6.

Листинг 13.6. Код простой функции `resetInput()`

```
function resetInput(inputToReset) {
   inputToReset.value = "";
}
```

5. Протестируйте приложение, последовательно набирая в поле ввода данных несколько значений и щелкая на кнопке Add It.

На этом базовая структура приложения, создающего ваш список желаний, завершена. Теперь все вводимые в поле формы значения будут добавляться в массив и отображаться в виде списка в нижней части веб-страницы.

Последняя функция, которую нам необходимо создать, применяется в обработчике события для кнопки Print Your List. Назначение функции `printView()` — скрыть форму и представить каждый элемент массива `myList` в удобочитаемом формате, который не стыдно предъявить даже избалованной золотой рыбке.

Ниже приведены инструкции, следуя которым, вы напишете код тела функции `printView()`.

1. Найдите на панели JavaScript "заглушку" функции, имеющую такой вид.

```
function printView() {
}
```

2. Введите внутри функции инструкцию объявления переменной, которая хранит ссылку на элемент всей страницы.

```
var listPage = document.getElementById("listPage");
```

3. Создайте еще одну переменную, в которой будет храниться ссылка на элемент формы ввода данных.

```
var formArea = document.getElementById("formArea");
```

4. Скройте форму, изменив в правиле CSS значение свойства `display` на `none`.

```
formArea.style.display = "none";
```

5. Добавьте в элемент `listPage` новый атрибут `class` со значением `"print"`.

```
listPage.className = "print";
```

Последняя команда всего лишь изменяет первый элемент `div` документа с

```
<div id="listPage">
```

на

```
<div id="listPage" class="print">
```

Класс `print` применяется в CSS для переключения форматирования страницы. Другими словами, после отнесения фрейма `listPage` к классу `print` к нему применяется форматирование, указанное свойствами правила `print`.

6. Введите команду очистки списка ото всех его элементов.

```
myListArea.innerHTML = "";
```

7. Выполните сортировку массива с помощью встроенного метода `sort()`.

```
myList.sort()
```

Данное выражение указывает выполнять сортировку в алфавитном порядке.

8. Вывод всех элементов массива на страницу выполняется с помощью циклической структуры. Она имеет следующий вид.

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

Подробно циклы описаны в главе 17. На данный момент вам достаточно знать, что в приведенном выше коде инструкции, заданные в фигурных скобках, последовательно применяются к каждому элементу массива.

9. Для вывода на экран текущего элемента массива введите следующее выражение.

```
wishList.innerHTML += "<li>" + myList[i] + "</li>";
```

С помощью последнего кода новый элемент добавляется в массив `wishList`, а затем выводится в окне браузера.

10. Завершите цикл, введя с новой строки закрывающую фигурную скобку.

```
}
```

11. Сохраните введенный код, щелкнув на кнопке **Update**.

Полный код функции `printView()` показан в листинге 13.7.

Листинг 13.7. Код функции `printView()`

```
function printView() {
   var listPage = document.getElementById("listPage");
   var formArea = document.getElementById("formArea");

   formArea.style.display = "none";

   listPage.className = "print";
   myListArea.innerHTML = "";
   myList.sort();

   for (var i = 0; i < myList.length; i++) {
      wishList.innerHTML += "<li>" + myList[i] + "</li>";
   }
}
```

12. Добавьте в список желаний несколько элементов и щелкните на кнопке Print Your List.

    На веб-странице отобразится список ваших желаний, представленный в предельно удобочитаемом виде (рис. 13.10).

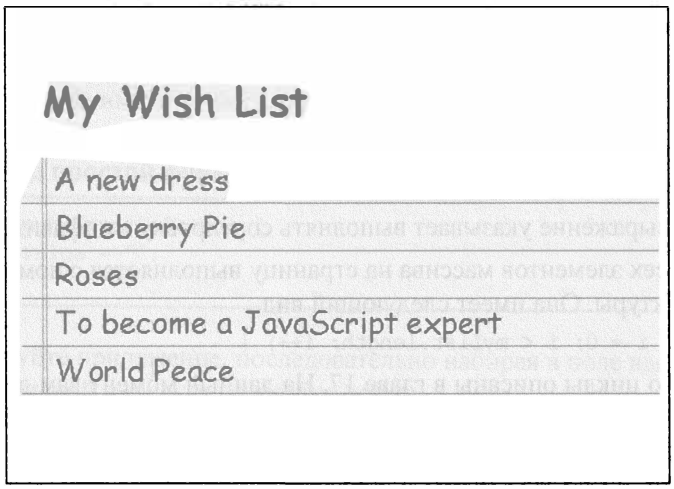


Рис. 13.10. Отформатированный список желаний

Диалоговое окно Печать

Добившись вывода списка желаний на веб-страницу в нужном формате, давайте снабдим наше приложение командой вызова диалогового окна Печать. Рассмотрим вариант, когда такое окно появляется на экране в результате щелчка на кнопке Print Your List.

Для снабжения приложения командой печати списка желаний выполните следующие действия.

1. Введите следующую инструкцию, поместив ее в коде функции `printView()` внутри цикла.

```
window.print();
```

В этом выражении задействован метод `print()` объекта `window`. Назначение этого метода понятно из его названия: вызов диалогового окна настроек печати браузера, в котором заданы параметры по умолчанию.

Объект `window` в JavaScript представляет текущее окно браузера.

2. Чтобы сохранить изменения в коде, щелкните на кнопке Update.
3. Добавьте в список побольше желаний и щелкните на кнопке Print Your List.

    На экране отобразится диалоговое окно настроек печати, подобное показанному на рис. 13.11.

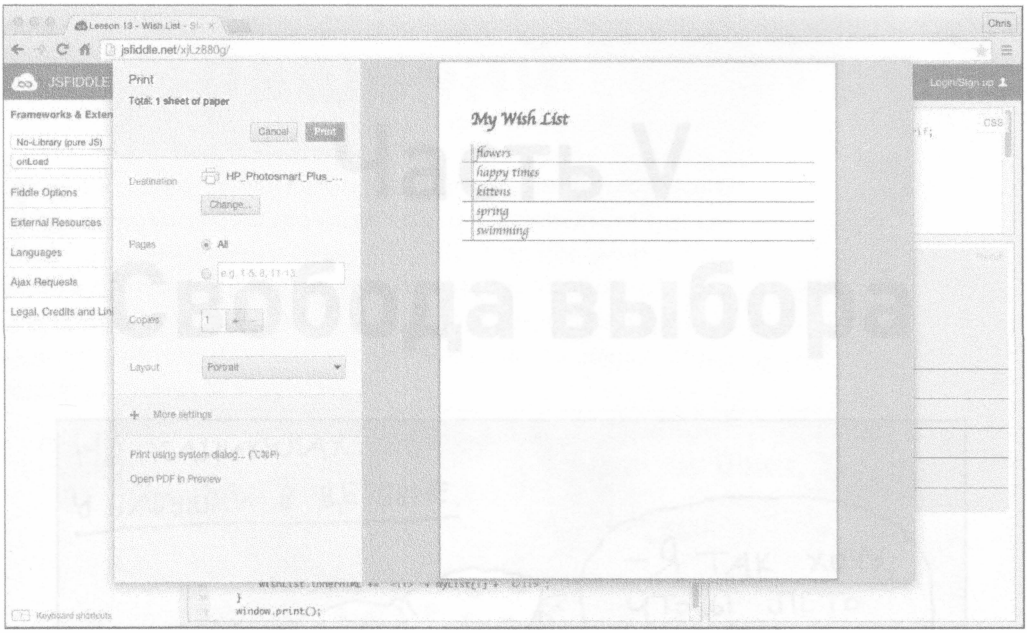


Рис. 13.11. Диалоговое окно настроек печати браузера Chrome

4. Если к вашему компьютеру подключен хотя бы один принтер, попробуйте распечатать свой список желаний.

Улучшение списка желаний

Вот мы и завершили создание приложения, использующего массивы и функции для вывода на страницу и печати данных, запрашиваемых у пользователя. Несмотря на то что конечный вариант списка желаний выглядит вполне презентабельно, приложение, его создающее, все еще можно улучшить. Ниже приведены рекомендации по дальнейшей модификации программы.

- ✓ Добавьте к ссылкам на элементы списка отдельные поля ввода данных, а сами ссылки сохраните в специально созданном массиве.
- ✓ Сохраните желания, представленные в списке, в файл, который расположен на текущем компьютере.
- ✓ Снабдите приложение командой отправки списка желаний произвольному адресату.

А каковы ваши идеи по усовершенствованию приложения?

Часть V
Свобода выбора

В этой части...

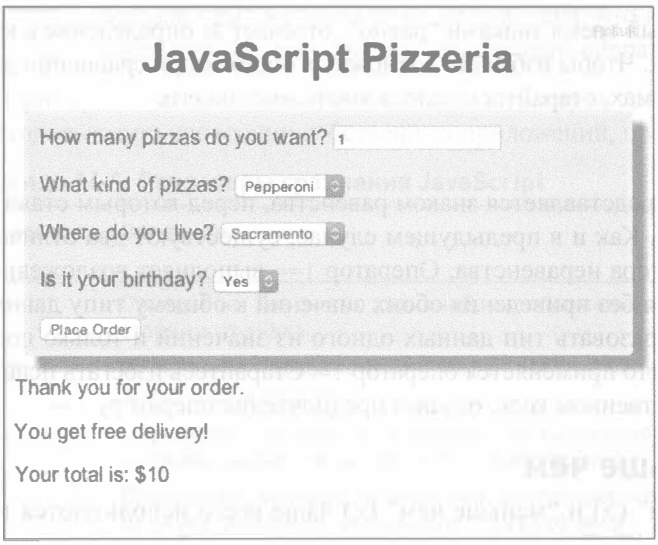
- ✓ Принятие решений
- ✓ Ветвление программы
- ✓ Написание повествования

ГЛАВА

14

Принятие решений

В ситуациях, требующих принятия решений, чаще всего приходится соглашаться (и/или отказываться) с предложенным вариантом действий. В программировании данная ситуация моделируется с помощью условной конструкции `if...else`. Она позволяет динамически реагировать на требования пользователей и выполнять операции согласно их текущим запросам.



Булева логика

В главе 9 мы упоминали об операторах сравнения, поддерживаемых в JavaScript, и воспользовались ими для написания программы суперкалькулятора. Давайте вкратце вспомним, чем характеризуются эти операторы и каковы их возможности.

Равно

Существуют два оператора, с помощью которых определяется равенство двух значений: == и ===. В первом случае операция сравнения выполняется после обязательного автоматического приведения обоих значений к одному типу данных. Например, при сравнении с помощью оператора == значений `3` и `"3"` возвращается значение `true` (истина). В случае применения оператора === результат будет `false` (ложь), поскольку автоматическое приведение типов данных при этом не выполняется.

Оператор, представленный тремя знаками "равно", отвечает за определение в коде *строгого равенства* данных. Чтобы избежать возможных ошибок при сравнении данных в собственных программах, старайтесь использовать именно его.

Не равно

Операция неравенства представляется знаком равенства, перед которым ставится восклицательный знак (!=). Как и в предыдущем случае, существуют два отличающихся друг от друга оператора неравенства. Оператор `!==` выполняет возложенную на него операцию сравнения без приведения обоих значений к общему типу данных. Если нужно сначала преобразовать тип данных одного из значений и только после этого выполнить сравнение, то применяется оператор `!=`. Старайтесь избегать использования оператора `!=` в собственном коде, отдавая предпочтение оператору `!==`.

Больше чем и меньше чем

Операторы "больше чем" (>) и "меньше чем" (<) чаще всего используются при сравнении числовых значений. Для определения направления действия операторов достаточно знать, что большее число всегда располагается у расширенного конца "стрелки", а ее "носик" всегда указывает в сторону меньшего числа. В ситуациях, когда меньшее число оказывается со стороны расширения стрелки, оператором возвращается значение `false` (ложь).

Больше или равно и меньше или равно

Операторы "больше или равно" и "меньше или равно" во многом напоминают операторы, рассмотренные в предыдущем разделе. Разница между ними заключается в том, что при использовании операторов "больше или равно" и "меньше или равно" значение `true` (истина) возвращается также в случае равенства сравниваемых значений.

В коде эти операторы представляются теми же знаками "больше" (>) и "меньше" (<), но с символом "равно" (=) в конце. Ниже приведены примеры их использования в коде JavaScript.

```
3 >= 4 // возвращает false
4 >= 4 // возвращает true
4 >= 3 // возвращает true
3 <= 3 // возвращает true
3 <= 4 // возвращает true
4 <= 3 // возвращает false
```

Не больше чем и не меньше чем

Добавив символ восклицательного знака перед знаком "больше" (>) или "меньше" (<), вы получите оператор "не больше чем" (!>) или "не меньше чем" (!>).

Многим (в том числе и нам) операторы "не больше чем" и "не меньше чем" кажутся избыточными. Старайтесь всячески избегать их применения в собственных программах JavaScript. Если подумать, то оператор "не больше чем" выполняет те же действия, что и оператор "меньше чем", а оператору "не меньше чем" сопоставим оператор "больше чем".

Примеры использования и назначение операторов сравнения, которые вам могут пригодиться при написании собственных приложений, приведены в табл. 14.1.

Таблица 14.1. Операторы сравнения JavaScript

Оператор	Описание	Пример результата `true`
`===`	Определяет равенство значений, расположенных по обе стороны от знака	`14 === 14`
`!==`	Определяет неравенство значений, расположенных по обе стороны от знака	`14 !== 15`
`>`	Определяет, больше ли значение, расположенное слева от знака, значения, указанного справа от него	`15 > 14`
`<`	Определяет, меньше ли значение, расположенное слева от знака, значения, указанного справа от него	`14 < 15`
`>=`	Определяет, больше или равно значение, расположенное слева от знака, значения, указанного справа от него	`14 >= 14`
`<=`	Определяет, меньше или равно значение, расположенное слева от знака, значения, указанного справа от него	`14 <= 14`

Условная конструкция `if...else`

Операторы сравнения сами по себе не имеют никакой практической ценности. Чаще всего они применяются как неотъемлемая часть условной конструкции `if...else`.

Ниже приведен общий вид условного оператора `if...else`.

```
if ([условие]) {
   // инструкции, выполняемые, если условие истинно (true)
} else {
   // инструкции, выполняемые, если условие ложно (false)
}
```

Вторая часть условной конструкции, начинающаяся с ключевого слова `else`, не обязательна. В большинстве случаев можно вполне обходиться одним только ключевым словом `if`, после которого указывается операция сравнения, а также инструкцией, выполняемой в случае истинности условия.

Простой пример использования условной конструкции в коде JavaScript показан в листинге 14.1.

Листинг 14.1. Пример условной конструкции `if...else`

```
var language = prompt("На каком языке Вы разговариваете?");
if (language === "JavaScript") {
   alert("Замечательно! Давайте общаться на JavaScript!");
} else {
   alert("Я Вас не понимаю!");
}
```

Чтобы познакомиться с действиями, выполняемыми этой программой, следуйте приведенным ниже инструкциям.

1. Запустите в браузере приложение JSFiddle.
2. Создайте новое "пустое" приложение, щелкнув на логотипе JSFiddle.
3. Введите на панель JavaScript код листинга 14.1.
4. Для выполнения кода программы щелкните на кнопке **Update** или **Run**.

   На экране появится всплывающее окно, запрашивающее данные.
5. Введите в текстовое поле **`JavaScript`** (учитывая регистр символов) и щелкните на кнопке **OK**.

   Первое всплывающее окно будет скрыто, а на экране появится сообщение `Замечательно! Давайте общаться на JavaScript!`, как показано на рис. 14.1.

Условия без операторов сравнения

Время от времени возникают ситуации, в которых определенные действия нужно выполнить только потому, что некая переменная принимает значение или просто объявляется. В подобных случаях достаточно поместить имя этой переменной (заключенное в скобки) в качестве условия конструкции `if...else`. Если указанная переменная не существует или не имеет значения, то условным оператором возвращается булево значение `false`.

В листинге 14.2 приведен код программы, равнозначной представленному в листинге 14.1. Теперь, когда вы набираете в поле ввода данных значение `JavaScript`, в программе создается специальная переменная: `speaksJavaScript`.

Логика следующей программы очень проста. Если вы вводите в точности `JavaScript`, то выполняются инструкции, указанные внутри блока `if`, что приводит к выводу на экран специального приглашения пообщаться на JavaScript. Если ввести в поле запроса значение, отличное от `JavaScript`, то выполняются операторы блока `else`, приводящие к выводу на экран совершенно иного извещения.

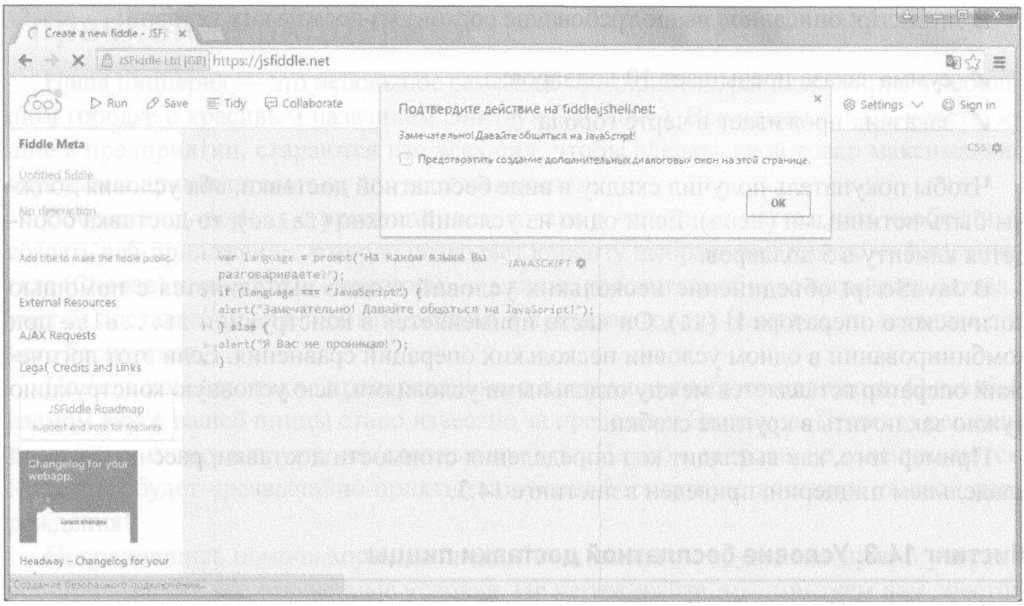


Рис. 14.1. Условная конструкция `if...else` позволяет принимать решения о дальнейших действиях

Листинг 14.2. Условия без операторов сравнения

```
var language = prompt("На каком языке Вы разговариваете?");

if (language === "JavaScript") {
   alert("Замечательно! Давайте общаться на JavaScript!");
   var speaksJavaScript = true;
} else {
   alert("Я Вас не понимаю!");
}

if (speaksJavaScript) {
   alert("Рад Вас видеть!");
}
```

Операторы сравнения и логические операторы

Логические операторы позволяют комбинировать в одном условии несколько операций сравнения. Представьте, что вы владелец пиццерии. В своем рекламном проспекте вы указываете, что покупатели, заказывающие пиццу на сумму, превышающую 10 долларов и живущие в пределах города, имеют право на бесплатную доставку.

В JavaScript описанное выше требование состоит из нескольких условий:

- ✓ сумма заказа превышает 10 долларов;
- ✓ заказчик проживает в черте города.

Чтобы покупатель получил скидку в виде бесплатной доставки, оба условия должны быть истинными (`true`). Если одно из условий ложно (`false`), то доставка обойдется клиенту в 5 долларов.

В JavaScript объединение нескольких условий в одно выполняется с помощью логического оператора И (`&&`). Он часто применяется в конструкции `if...else` при комбинировании в одном условии нескольких операций сравнения. Если этот логический оператор вставляется между отдельными условиями, всю условную конструкцию нужно заключить в круглые скобки.

Пример того, как выглядит код определения стоимости доставки, рассчитываемой владельцем пиццерии, приведен в листинге 14.3.

Листинг 14.3. Условие бесплатной доставки пиццы

```
if ((deliveryCity === "Anytown") && (orderPrice > 10)) {
   var deliveryPrice = 0;
} else {
   var deliveryPrice = 5;
}
```

Пусть в качестве специального предложения бесплатная доставка осуществляется заказчикам, которые празднуют свой день рождения (независимо от стоимости заказа и региона проживания). Чтобы добавить в код определения стоимости доставки еще и это условие, вам придется воспользоваться оператором логического ИЛИ (`||`). Символ вертикальной черты вводится при нажатии клавиши </> одновременно с клавишей <Shift> (в английской раскладке).

Конечный вид условной конструкции, определяющей стоимость доставки пиццы, приведен в листинге 14.4.

Листинг 14.4. Бесплатная доставка на день рождения

```
if (((deliveryCity === "Anytown") && (orderPrice > 10)) ||
      (birthday === "yes")) {
   var deliveryPrice = 0;
} else {
   var deliveryPrice = 5;
}
```

В следующем разделе мы воспользуемся последней условной конструкцией в программе вычисления стоимости пиццы с учетом ее доставки. Она также станет основой для принятия некоторых других решений.

Управление пиццерией

Наша пиццерия — это небольшое семейное предприятие, расположенное в небольшом городке с красивым названием Энитаун (Anytown). Все члены семьи, участвующие в предприятии, стараются изо всех сил, чтобы сделать свой товар максимально вкусным и привлекательным для всех желающих.

Собравшись помочь с продвижением товара в народные массы, нам необходимо создать веб-приложение, которое позволяет клиенту выбрать вид пиццы — Классическая (Cheese) или Острая (Pepperoni), — а также рассчитать общую стоимость заказа с учетом доставки. Если клиент проживает в пределах города Энитаун, то доставка для него бесплатная.

Но ведь клиенты такие привередливые и постоянно требуют новшеств. О незабываемом вкусе вашей пиццы стало известно за пределами Энитауна. Принято решение доставлять пиццу в другие населенные пункты. На этом запросы не заканчиваются. Клиентам будет чрезвычайно приятно получить бесплатную доставку на свой день рождения!

Подрядившись помочь хорошим людям в их нелегком труде, вам нужно учесть в веб-приложении все озвученные условия. Не переживайте, мы поможем вам советом в самую трудную минуту!

Знакомство с приложением

Для тестирования начальной версии приложения, формирующего заказ в пиццерии, выполните следующие действия.

1. Откройте в браузере приложение JSFiddle и перейдите к общедоступному кабинету.

   `http://jsfiddle.net/user/forkids/fiddles`

2. Найдите в списке заголовок Chapter 14 — JavaScript Pizzeria — Start и щелкните на нем.

3. Укажите количество заказываемых пицц и их названия, после чего щелкните на кнопке Place Order (Заказать).

   Под формой заказа появится счет к оплате (10 долларов за пиццу).

Приложение очень простое; вы только что ознакомились со всеми его возможностями. В следующих разделах вы узнаете, как модифицировать программу, снабдив ее дополнительными функциями.

Заимствование приложения

Чтобы получить в свое распоряжение копию приложения, которая станет основной для дальнейшего улучшения, выполните такие операции.

1. Запустите приложение Chapter 14 — JavaScript Pizzeria — Start.

2. В верхней части приложения щелкните на кнопке Fork.

3. Воспользовавшись командой Fiddle Options (на левой панели), переименуйте свою копию программы.
4. Сохраните изменения в приложении, щелкнув на кнопке Update, а затем выполните команду Set as Base.

Готово! Можно приступать к улучшению кода.

Улучшение приложения

Ниже описаны все изменения, которые будут внесены в имеющуюся программу:

- ✓ расширить ассортимент новым видом пиццы;
- ✓ добавить в область обслуживания другие населенные пункты;
- ✓ рассчитать отдельно плату за доставку;
- ✓ обеспечить бесплатную доставку для всех именинников.

Для реализации каждого из пунктов плана нужно будет использовать условную конструкцию `if...else`, а также вводить дополнительный код HTML.

Расширение ассортимента

Вначале сконцентрируемся на меню пиццерии. Шеф-повар (по совместительству — глава семейства) совершенно случайно изобрел новый вид пиццы, которая включает все, что в понедельник утром можно было найти в холодильнике: колбаса, сыры пяти сортов, яблоки, лук, чеснок и немного собачьего корма. Новый рецепт приобрел отличительное название: “Убийственная” (Supreme).

Проблема в том, что новый вид пиццы оказался дороговатым для производства. Конечно, виноват в этом секретный ингредиент — корм для собак. А все потому, что собачий корм более сбалансированный, чем продукты для людей! Как бы там ни было, убийственная пицца стоит на два доллара дороже, чем остальные виды.

Выяснив подробности, нам необходимо добавить пиццу Supreme в меню и снабдить ее собственным ценником.

1. Найдите на панели HTML раздел, в котором составляется список названий пицц.

   Исходно он имеет такой вид.

```
<label>What kind of pizzas?
   <select id="typePizza">
      <option value="cheese">Cheese</option>
      <option value="pepperoni">Pepperoni</option>
   </select>
</label>
```

2. Внутри элемента `select` создайте еще один элемент `option`, который будет представлять в меню убийственную пиццу.

В качестве атрибута `value` этого элемента используйте значение `"supreme"`, а вместо подписи (вводится между тегами `<option>` и `</option>`) — `Supreme`.

3. Чтобы сохранить изменения в коде, щелкните на кнопке **Update**. Убедитесь в том, что в меню выбора пиццы добавлен элемент **Supreme**, как показано на рис. 14.2.

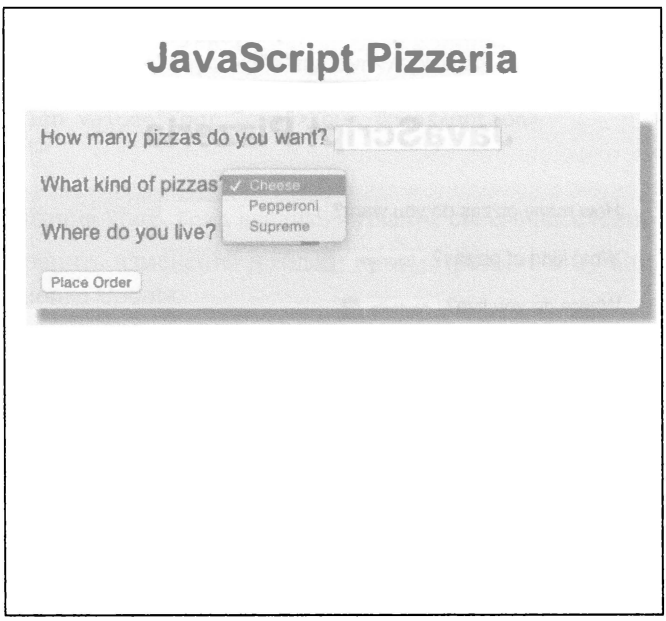


Рис. 14.2. В меню добавлен новый элемент

4. На панели JavaScript найдите код функции `calculatePrice()`.

   Выглядит он следующим образом.

```
function calculatePrice(numPizzas, typePizza) {
   var orderPrice = Number(numPizzas) * 10;
   var extraCharge = 0;

   // вычисление надбавки extraCharge за секретный ингредиент

   orderPrice += extraCharge;
   return orderPrice;
}
```

5. Сразу под комментариемвведите код следующего условного выражения `if... else`.

```
if (typePizza === "supreme") {
   extraCharge = Number(numPizzas) * 2;
}
```

В этой программной конструкции проверяется, выбрана ли в меню пицца Supreme. Если выбрана, то для вычисления общей надбавки указанное количество единиц товара умножается на два (доллара).

6. Сохраните измененный в приложении код, щелкнув на кнопке Update, после чего проверьте, как он выполняется.

   При выборе в меню пиццы Supreme каждая ее единица в конечном счете должна обходиться клиенту в 12 долларов, что проиллюстрировано на рис. 14.3.

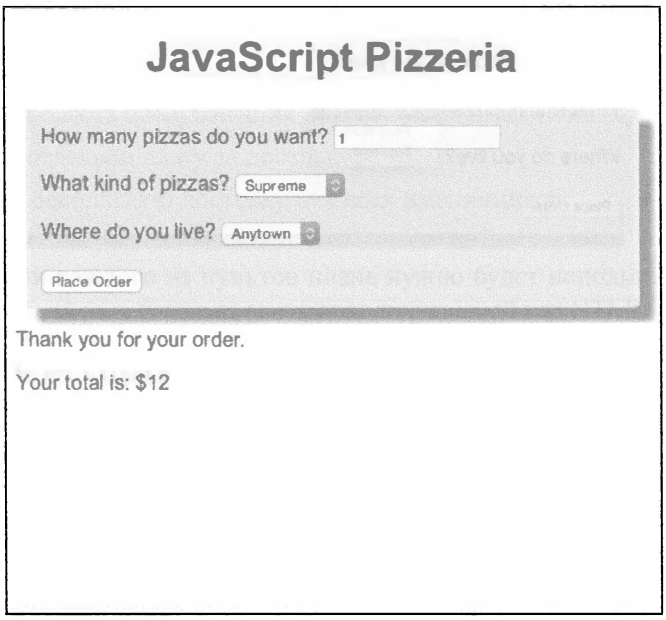


Рис. 14.3. Выбран новый вид пиццы

Доставка в другие города

Дело идет! Пиццерия становится популярной за пределами нашего городка, да и население не в силах одолеть всего, что предлагают трудолюбивые пекари. Настало время расширяться, предлагая доставку в соседние населенные пункты. Вам нужно обеспечить возможность выбора пункта доставки товара в форме заказа.

Перед нами стоит дилемма, требующая немедленного разрешения. Совершенно очевидно, что бесплатная доставка одной единицы товара в пределах нашего городка убыточна, равно как и бесплатная доставка любого количества товара в другие регионы. Именно поэтому при заказе на сумму, не превышающую 10 долларов, или при указании в качестве места доставки населенного пункта, отличного от `Anytown`, в сумму заказа необходимо включать дополнительные 5 долларов на логистику.

Новые правила ценообразования включаются в приложение следующим образом.

1. На панели HTML найдите код, отвечающий за создание раскрывающего списка для выбора места доставки товара.

   Исходно в списке представлен только один пункт: `Anytown`.

2. Добавьте в раскрывающийся список по крайней мере два новых пункта.

   Конечный код, отражающий изменения в содержимом списка, имеет такой вид.

```
<label>Where do you live?
   <select id="deliveryCity">
      <option value="Anytown">Anytown</option>
      <option value="Sacramento">Sacramento</option>
      <option value="Your Town">Your Town</option>
   </select>
</label>
```

   Вместо названия `Your Town` введите название своего населенного пункта.

3. Чтобы сохранить изменения в коде и просмотреть, к чему они приведут, щелкните на кнопке **Update**.

4. На панели JavaScript отыщите код функции `calculateDelivery()`.

   Исходно в ней содержится всего одна инструкция, устанавливающая для переменной `deliveryPrice` значение `0`.

5. Под комментарием `calculate delivery price, if there is one` вставьте следующее условное выражение.

```
if ((deliveryCity === "Anytown") && (orderPrice > 10))
{
   deliveryPrice = 0;
} else {
   deliveryPrice = 5;
}
```

6. Сохраните результаты работы, щелкнув на кнопке **Update**, и проверьте работоспособность формы на панели **Results**.

   При выборе города доставки, отличного от `Anytown`, или при указании заказа на сумму всего 10 долларов к чеку итоговой стоимости товара будет прибавлено 5 долларов.

Отображение стоимости доставки

Нам предстоит вывести цену доставки отдельной строкой в чеке, ведь клиенты вправе знать, как формируется полная стоимость выбранного ими товара.

Для включения в чек стоимости доставки товара выполните следующие действия.

1. В функции `placeOrder()` отыщите комментарий, содержащий текст `todo: output the delivery price, if there is one`.

2. Сразу под указанным комментарием введите следующую условную конструкцию `else...if`.

```
if (deliveryPrice === 0) {
   theOutput += "<p>You get free delivery!</p>";
} else {
   theOutput += "<p>Your delivery cost is: $" + deliveryPrice;
```

Если стоимость доставки (значение переменной `deliveryPrice`) равна `0`, то на экран будет выведено специальное извещение. В противном случае оно будет заменено суммой доставки.

3. Для сохранения изменений в коде щелкните на кнопке **Update**, расположенной в верхней части окна. Ознакомьтесь с формой, представленной на панели результатов.

   Извещение о бесплатной доставке показано на рис. 14.4.

Рис. 14.4. Ничто так не способствует росту объемов продаж, как бесплатная доставка товара

Специальное предложение для именинников

Последнее изменение, затрагивающее код JavaScript приложения, связано с предоставлением специального предложения для клиентов, которые делают заказ в свой день рождения.

Чтобы порадовать именинников бесплатной доставкой, выполните следующие действия.

1. На панели HTML после кода поля, в котором выбирается город доставки пиццы, создайте элемент поля, в котором указывается, что у клиента сегодня день рождения.

```
<label>Is it your birthday?
   <select id="birthday">
      <option value="yes">Yes</option>
      <option value="no">No</option>
   </select>
</label>
```

2. Сохраните результаты работы, щелкнув на кнопке **Update**, и проверьте работоспособность формы на панели **Results**.

   Если содержимое панели результатов отличается от показанного на рис. 14.5, то внимательно проверьте код на наличие ошибок. Вам может понадобиться добавить в код формы несколько элементов `br`, чтобы обеспечить одинаковые интервалы между отдельными полями.

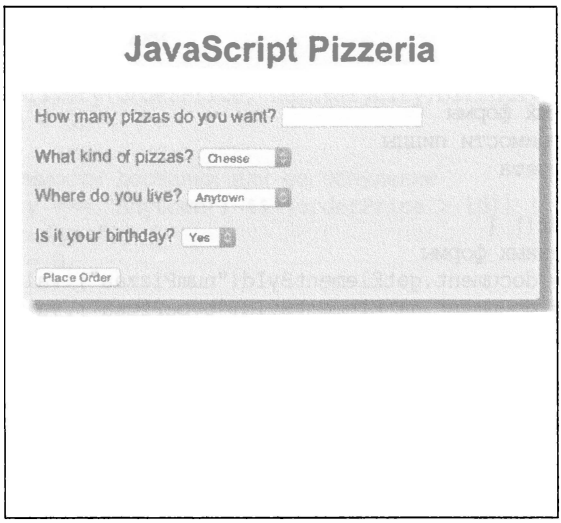


Рис. 14.5. Панель результатов с формой, позволяющей именинникам получить скидку

3. Перейдите к панели JavaScript и добавьте в функцию `placeOrder()` следующий код. Разместите его под инструкциями вызова метода `getElementById()`.

   ```
   var birthday = document.getElementById("birthday").value;
   ```

4. Снабдите функцию `calculateDelivery()` третьим параметром, представленным переменной, в которой хранятся сведения о дне рождения клиента.

   ```
   function calculateDelivery(orderPrice, deliveryCity, birthday)
   ```

5. В функции `calculateDelivery()` добавьте в условную конструкцию `if...else` оператор `or` и дополнительное условие, чтобы проверить, установлена ли переменная `birthday` равной значению `yes`.

   ```
   if (((deliveryCity === "Anytown") && (orderPrice > 10)) ||
      ↳(birthday === "yes")) {
   ```

6. В функции `placeOrder()` измените инструкцию вызова функции `calculateDelivery()`, обеспечив в ней поддержку параметра `birthday`.

```
var deliveryPrice = calculateDelivery(orderPrice,deliveryCity,
   ↳birthday);
```

7. Для сохранения измененного кода JavaScript щелкните на кнопке Update.

Полный JavaScript-код приложения обработки заказов в пиццерии показан в листинге 14.5.

Листинг 14.5. Полный JavaScript-код приложения, обрабатывающего заказы в пиццерии

```
// Обработка щелчков мышью
document.getElementById("placeOrder").addEventListener("click",
   ↳placeOrder);

/**
* - получение данных формы
* - определение стоимости пиццы
* - формирование счета
*/
function placeOrder() {
   // получение данных формы
   var numPizzas = document.getElementById("numPizzas").value;
   var typePizza = document.getElementById("typePizza").value;
   var deliveryCity = document.getElementById("deliveryCity").value;
   var birthday = document.getElementById("birthday").value;

   // определение стоимости пиццы
   var orderPrice = calculatePrice(numPizzas, typePizza);
   // определение стоимости доставки
   var deliveryPrice = calculateDelivery(orderPrice, deliveryCity,
      ↳birthday);

   // формирование счета
   var theOutput = "<p>Thank you for your order.</p>";

   // вывод стоимости доставки
   if (deliveryPrice === 0) {
      theOutput += "<p>You get free delivery!</p>";
   } else {
      theOutput += "<p>Your delivery cost is: $" + deliveryPrice;
   }

   theOutput += "<p>Your total is: $" + (orderPrice + deliveryPrice);

   // вывод счета
   document.getElementById("displayTotal").innerHTML = theOutput;
}
```

```
/**
* Вычисление стоимости пиццы
*/
function calculatePrice(numPizzas, typePizza) {
   var orderPrice = Number(numPizzas) * 10;
   var extraCharge = 0;
   // вычисление надбавки к стоимости, extraCharge
   if (typePizza === "supreme") {
      extraCharge = Number(numPizzas) * 2;
   }

   orderPrice += extraCharge;
   return orderPrice;
}

/**
* Вычисление стоимости доставки
*/
function calculateDelivery(orderPrice, deliveryCity,birthday) {
   var deliveryPrice = 0;

   // Вычисление стоимости доставки или ее обнуление
   if (((deliveryCity === "Anytown") && (orderPrice > 10)) ||
      ↳(birthday === "yes")) {
      deliveryPrice = 0;
   } else {
      deliveryPrice = 5;
   }
   return deliveryPrice;
}
```

Полный HTML-код приложения обработки заказов в пиццерии приведен в листинге 14.6.

Листинг 14.6. Полный HTML-код приложения, обрабатывающего заказы в пиццерии

```
<h1>JavaScript Pizzeria</h1>

<div id="orderForm">
   <label>How many pizzas do you want?
   <input type="number" id="numPizzas" />
   </label>
   <br />
   <br />
   <label>What kind of pizzas?
      <select id="typePizza">
         <option value="cheese">Cheese</option>
         <option value="pepperoni">Pepperoni</option>
         <option value="supreme">Supreme</option>
```

```
      </select>
   </label>
   <br />
   <br />
   <label>Where do you live?
      <select id="deliveryCity">
         <option value="Anytown">Anytown</option>
         <option value="Sacramento">Sacramento</option>
         <option value="Beverly Hills">Beverly Hills</option>
      </select>
   </label>
   <br />
   <br />
   <label>Is it your birthday?
      <select id="birthday">
         <option value="yes">Yes</option>
         <option value="no">No</option>
      </select>
   </label>
   <br />
   <br />
   <button type="button" id="placeOrder">Place Order</button>
</div>
<div id="displayTotal"></div>
```

На рис. 14.6 показано содержимое панели **Results** после размещения заказа в полностью завершенном приложении.

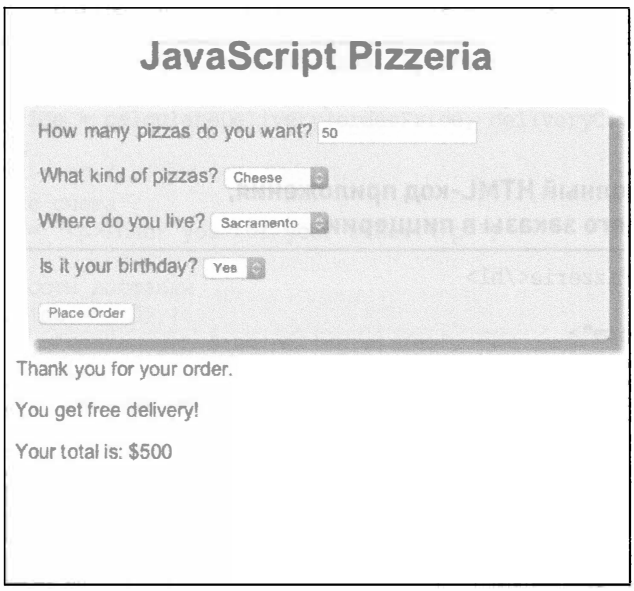


Рис. 14.6. Заказ пиццы на сегодняшний обед

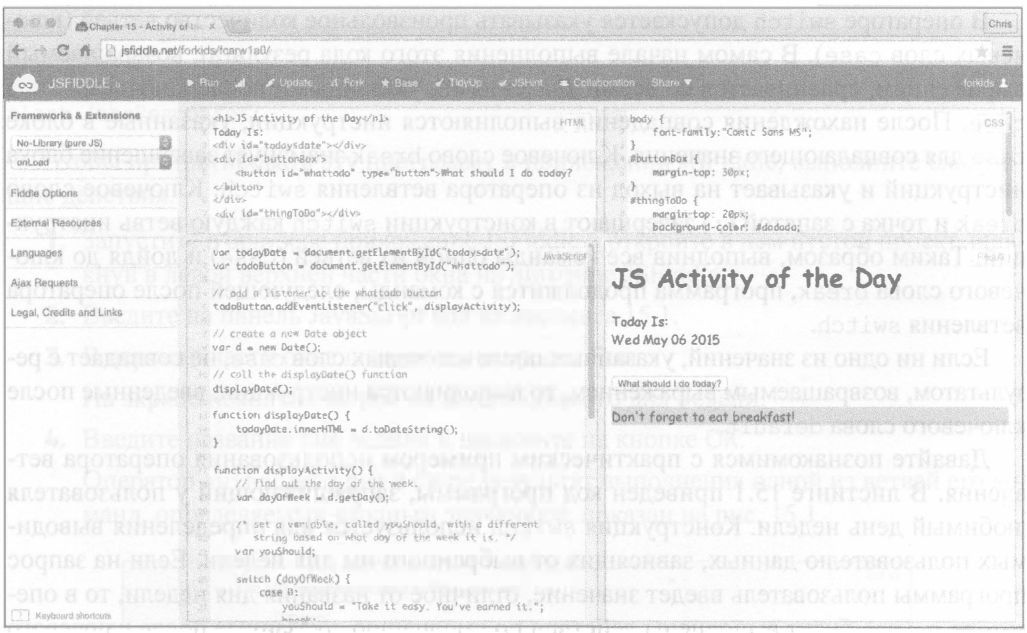
ГЛАВА

15

Ветвление программы

Команда ветвления позволяет направить выполнение программы по нескольким путям. В операторе `switch` решение о переходе к выполнению тех или иных инструкций принимается на основе значения, полученного в результате вычисления специального выражения. Сравнивая его с одним из значений, указанных после ключевого слова `case`, программа определяет, что ей делать дальше. Каждый из вариантов действий в операторе `switch` называется *ветвью*.

В этой главе вы познакомитесь с оператором `switch` и используете его для создания программы, выводящей на экран список текущих дел для разных дней недели.



Принятие решений

Оператор ветвления начинается с ключевого слова `switch`, после которого в скобках указывается некое выражение, результат выполнения которого определяет дальнейшие операции. Каждый из возможных вариантов действий (ветвь) задается после ключевого слова `case`.

В самом общем виде конструкция `switch` имеет следующий синтаксис.

```
switch (выражение) {
   case значение1:
      //инструкции1
   break;
   case значение2:
      //инструкции2
   break;
   default:
     //инструкции
   break;
}
```

В операторе `switch` допускается указывать произвольное количество ветвей (ключевых слов `case`). В самом начале выполнения этого кода результат, возвращаемый выражением, сравнивается со всеми значениями, указанными после ключевых слов `case`. После нахождения совпадения выполняются инструкции, указанные в блоке `case` для совпадающего значения. Ключевое слово `break` знаменует завершение блока инструкций и указывает на выход из оператора ветвления `switch`. Ключевое слово `break` и точка с запятой (`;`) завершают в конструкции `switch` каждую ветвь инструкций. Таким образом, выполнив все команды одной из ветвей (`case`) и дойдя до ключевого слова `break`, программа продолжится с команды, следующей после оператора ветвления `switch`.

Если ни одно из значений, указанных после ключевых слов `case`, не совпадает с результатом, возвращаемым выражением, то выполняются инструкции, введенные после ключевого слова `default`.

Давайте познакомимся с практическим примером использования оператора ветвления. В листинге 15.1 приведен код программы, запрашивающий у пользователя любимый день недели. Конструкция `switch` используется для определения выводимых пользователю данных, зависящих от выбранного им дня недели. Если на запрос программы пользователь введет значение, отличное от названия дня недели, то в операторе `switch` будут выполнены действия по умолчанию, указанные после ключевого слова `default`.

Листинг 15.1. Вывод данных, зависящих от входных значений

```
var myNumber = prompt("Введите день недели!");
var theResponse;

switch (myNumber) {
   case "Понедельник":
```

```
      theResponse = "Бездельник!";
      break;
   case "Вторник":
      theResponse = "Повторник!";
      break;
   case "Среда":
      theResponse = "Бреда!";
      break;
   case "Четверг":
      theResponse = "Чёртегдерг!";
      break;
   case "Пятница":
      theResponse = "Расслабильница!";
      break;
   case "Суббота":
      theResponse = "Клуббота!";
      break;
   case "Воскресенье":
      theResponse = "Высплюсенье!";
      break;
   default:
      theResponse = "Таких дней не бывает!";
      break;
}
alert (theResponse);
```

Чтобы протестировать эту программу в приложении JSFiddle, выполните следующие действия.

1. Запустите в браузере приложение JSFiddle и откройте в нем пустой проект, щелкнув в левой верхней части окна на логотипе JSFiddle.
2. Введите на панель JavaScript код из листинга 15.1.
3. В верхней части окна щелкните на кнопке Run.

   На экране появится запрос на ввод любимого дня недели.
4. Введите название дня недели и щелкните на кнопке OK.

   Оператор `switch` вступит в силу; результат выполнения одной из ветвей его команд, определяемых входным значением, показан на рис. 15.1.

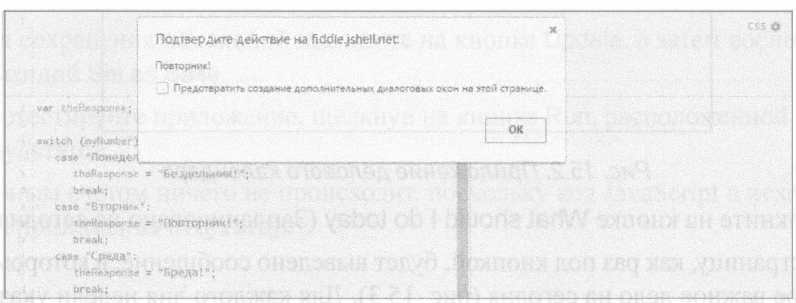


Рис. 15.1. Ответные действия на сделанный вами выбор

Деловой календарь

Просыпаясь рано утром, каждый занятой человек первым делом старается вспомнить, какой сегодня день недели. Впереди столько неотложных дел, но какое из них самое важное? В отдельные дни вспомнить, что же запланировано на сегодня, далеко не так просто, как кажется. Встав не с той ноги, да еще и не с той стороны кровати, вы обречены на неудачу даже в таком простом вопросе, как планирование неотложных дел.

Помня о неудачных днях, вы решаете создать специальную веб-страницу или приложение, которое будет выводить на экран текущую дату и сведения о самом важном деле, запланированном заранее. Начав пользоваться подобным приложением со следующего понедельника, вы начнете новую жизнь, насыщенную приятными событиями и лишенную досадных провалов памяти. Проверено на авторах!

Знакомство с приложением делового календаря

Уже традиционно, прежде чем приступить к самостоятельному написанию приложения календаря, в котором указываются важные дела, давайте познакомимся с возможностями готовой программы.

1. Откройте в браузере общедоступный кабинет приложения JSFiddle, перейдя по такому адресу:

   `http://jsfiddle.net/user/forkids/fiddles`

2. Найдите в списке приложений название Chapter 15 — Activity of the Day и щелкните на нем.

   В окне браузера отобразятся все четыре панели: три — с кодом приложения и одна — с результатом его выполнения (рис. 15.2). На последней исходно указывается текущая дата.

Рис. 15.2. Приложение делового календаря

3. Щелкните на кнопке What should I do today (Запланировано на сегодня).

   На страницу, как раз под кнопкой, будет выведено сообщение, в котором указано самое важное дело на сегодня (рис. 15.3). Для каждого дня недели указываются свои важные дела.

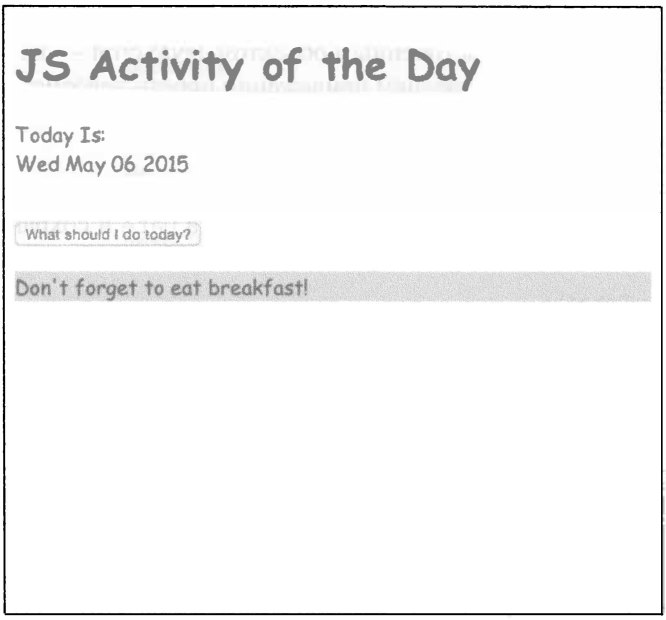


Рис. 15.3. Самое важное дело на сегодня

Копирование приложения делового календаря в личный кабинет

Чтобы приступить к работе над приложением делового календаря, выполните следующие действия.

1. Посетите общедоступный кабинет JSFiddle по такому адресу:
   `http://jsfiddle.net/user/forkids/fiddles`

   В списке приложений найдите заголовок Chapter 15 — Activity of the Day — Start.
2. Щелкните на заголовке приложения, чтобы отобразить его код в JSFiddle.
3. На левой панели щелкните на команде Fiddle Options и измените название приложения на более благозвучное (исходный вариант далеко не самый подходящий).
4. Для сохранения изменений щелкните на кнопке Update, а затем воспользуйтесь командой Set as Base.
5. Протестируйте приложение, щелкнув на кнопке Run, расположенной на панели результатов.

   Ровным счетом ничего не происходит, поскольку код JavaScript в исходной версии приложения отсутствует.

Перед тем как углубиться в написание кода программы делового календаря, нам необходимо познакомиться со встроенным объектом JavaScript — `Date`. Без него добиться желаемого результата в текущем приложении просто невозможно.

Объект `Date`

В коде JavaScript встроенный объект `Date` представляет текущий момент времени. Чтобы создать в своем приложении экземпляр объекта `Date` и сохранить его в переменной, примените ключевое слово `new`, как показано ниже.

```
var myDate = new Date();
```

В дальнейшем коде только что созданный объект представляется переменной с именем `myDate`.

Чтобы протестировать новый объект, выполните следующие действия.

1. В браузере Google Chrome отобразите консоль разработчика.
2. Введите на консоли следующее выражение и нажмите <Return> (в Mac) или <Enter> (в Windows).

   ```
   var myDate = new Date();
   ```

   На консоль будет возвращено значение `undefined`, указывающее на успешное выполнение текущей команды.
3. Введите приведенный ниже код и снова нажмите <Return> (в Mac) или <Enter> (в Windows).

   ```
   myDate
   ```

   На консоли отображаются текущие дата и время, представляемые объектом `Date`.

Как и другие объекты JavaScript, рассмотренные ранее, объект `Date` снабжен встроенными функциями (известными как методы), позволяющими выполнять над значением даты и времени определенные манипуляции.

Описание методов, применяемых для управления объектом `Date`, приведено в табл. 15.1. Метод, который используется для извлечения данных объекта, называется получающим (*getter method*).

Таблица 15.1. Получающие методы объекта `Date`

Метод	Описание
`getDate()`	Извлекает из объекта день месяца (1-31)
`getDay()`	Извлекает из объекта порядковый номер дня недели (0-6)
`getFullYear()`	Извлекает из объекта год (yyyy)
`getHours()`	Извлекает из объекта часы указанной даты по местному времени (0-23)
`getMilliseconds()`	Извлекает из объекта миллисекунды указанной даты по местному времени (0-999)

Окончание табл. 15.1

Метод	Описание
`getMonth()`	Извлекает из объекта месяц (0–11)
`getSeconds()`	Извлекает из объекта секунды указанной даты по местному времени (0–59)
`getTime()`	Извлекает из объекта время, представленное в формате Unix (количество миллисекунд, прошедших с 1 января 1970 года)

Для применения получающего метода укажите его название после экземпляра объекта `Date`, предварив его точкой.

Например, после сохранения объекта `Date` в переменной, воспользовавшись для этих целей консолью разработчика браузера Chrome, можно применять к ней все описанные выше получающие методы.

1. Для получения дня недели, представленного порядковым номером, используется следующий код.

   ```
   myDate.getDay()
   ```

   На консоль разработчика выводится числовое значение из диапазона 0–6, где 0 соответствует воскресенью, а 6 — субботе.

2. Чтобы извлечь из даты день текущего месяца, представленного порядковым номером, введите на консоли такой код.

   ```
   myDate.getDate();
   ```

3. Для получения текущего месяца используйте следующую команду.

   ```
   myDate.getMonth();
   ```

   Обратите внимание на то, что отсчет значений, возвращаемых методами `getMonth()` и `getDay()`, ведется с нуля (в JavaScript номером 0 представляется январь).

   Наряду с этим методы `getDate()` и `getFullYear()` возвращают вполне ожидаемые числовые значения. В частности, второй день месяца представляется числом 2, а 2020 год — числом `2020`.

В дополнение к методам, извлекающим определенные значения из объекта `Date`, в JavaScript содержатся методы, назначающие их. В табл. 15.2 описаны наиболее распространенные методы изменения данных в объекте `Date`. Метод, с помощью которого информация заносится в объект, называется устанавливающим (*setter method*).

Таблица 15.2. Устанавливающие методы объекта `Date`

Метод	Описание
`setDate()`	Устанавливает в объекте день месяца (1–31)
`setDay()`	Сохраняет в объекте день недели в виде порядкового номера (0–6)
`setFullYear()`	Устанавливает в объекте год (yyyy)

Окончание табл. 15.2

Метод	Описание
`setHours()`	Сохраняет в объекте часы указанной даты по местному времени (0-23)
`setMilliseconds()`	Сохраняет в объекте миллисекунды указанной даты по местному времени (0-999)
`setMonth()`	Устанавливает в объекте месяц (0-11)
`getSeconds()`	Сохраняет в объекте секунды указанной даты по местному времени (0-59)
`setTime()`	Сохраняет в объекте время, представленное в формате Unix (количество миллисекунд, прошедших с 1 января 1970 года)

Чтобы получше познакомиться с устанавливающими методами, воспользуйтесь консолью разработчика браузера Chrome.

1. Создайте объект `Date`, введя следующий код.

   ```
   var myNewDate = new Date();
   ```

2. Определите исходное значение, содержащееся в объекте `Date`, введя на консоли имя переменной, в которой он хранится.

   ```
   myNewDate
   ```

   На консоль будет выведено текущее значение объекта `myNewDate`, представленное в строковом формате.

3. Измените месяц текущей даты на август.

   ```
   myNewDate.setMonth(7);
   ```

   Консоль ответит гигантским числом. Несмотря на монстроподобность, это число представляет время, правда, выраженное в формате Unix. Именно в таком формате даты представляются в JavaScript исходно. В данном случае выведенное на консоль значение равняется количеству миллисекунд, отсчитанных от 1 января 1970 года до измеряемого момента времени.

4. Чтобы представить дату в привычном формате, введите на консоли название объекта.

   ```
   myNewDate
   ```

Теперь, когда вы знаете об объекте `Date` все самое важное, давайте применим его в операторе `switch`, используемом при написании кода нашего делового календаря.

Код делового календаря

В начальной версии программы делового календаря, загружаемой в приложении JSFiddle, уже есть некий код и комментарии, содержащие инструкции по его дальнейшему улучшению. В листинге 15.2 показано, с чего нужно начинать создание собственной версии делового календаря.

Листинг 15.2. Начальный код делового календаря

```
var todayDate = document.getElementById("todaysdate");
var todoButton = document.getElementById("whattodo");

// Обработчик событий для кнопки
todoButton.addEventListener("click", displayActivity);

// Создание объекта Date
var d = new Date();

// Вызов функции displayDate()
displayDate();

function displayDate() {
   // todo: отображение текущей даты во фрейме todaysdate
}

function displayActivity() {
   // todo: определить день недели

   /* todo: сохранение в переменной youShould строки, зависящей
            от дня недели */

   // todo: вывод значения переменной youShould во фрейме thingToDo
}
```

Давайте посмотрим, как устроена программа с таким кодом. Последовательно изучите включенные в нее команды, отвечающие за выполнение следующих действий:

- ✓ объявление двух переменных, в которых хранятся ссылки на элементы HTML, обрабатываемые в дальнейшей программе;
- ✓ создание обработчика для события щелчка на кнопке;
- ✓ создание экземпляра объекта `Date`, хранящего текущую дату;
- ✓ вызов функции, отображающей текущую дату на экране.

В результате выполнения приведенного выше кода ничего не происходит, поскольку программа ожидает, когда пользователь щелкнет на кнопке What should I do today?. Распознав щелчок на кнопке, программа вызывает функцию `displayActivity()`, указанную в обработчике этого события.

Ваша задача заключается в написании кода остальных двух функций программы.

Перед тем как приступить к изучению следующих пошаговых инструкций, попробуйте решить поставленную задачу самостоятельно. Завершив написание кода обеих функций, сверьтесь с приведенными ниже рекомендациями.

1. Отыщите в коде приложения функцию `displayDate()` и добавьте в нее следующую строку, поместив ее непосредственно под комментарием.

   ```
   todayDate.innerHTML = d;
   ```

   В этой инструкции значение переменной `d`, представляющей объект `Date`, назначается свойству `innerHTML` элемента `div` (сохраненного в переменной `todayDate`).

2. Чтобы увидеть результат изменения кода функции, щелкните на кнопке **Update**.

3. Представьте дату, выводимую на панель результатов, в более привлекательном виде.

   ```
   todayDate.innerHTML = d.toDateString();
   ```

   Теперь при выполнении программы дата на панели будет выводиться в более знакомом виде: день недели, месяц, день месяца, год.

4. Отыщите в приложении функцию `displayActivity()` и добавьте в нее приведенный ниже код, отвечающий за извлечение из объекта `Date` текущего дня недели.

   ```
   var dayOfWeek = d.getDay();
   ```

5. Объявите переменную, в которой хранится сообщение, выводимое пользователю.

   ```
   var youShould;
   ```

6. Введите первую строку оператора `switch`, определяющего различные действия в зависимости от значения переменной `dayOfWeek`, не забыв об открывающей фигурной скобке в конце.

   ```
   switch (dayOfWeek) {
   ```

7. Напишите первую ветвь инструкции, соответствующую значению `0` или воскресенью.

   ```
   case 0:
   ```

8. Создайте сообщение, выводящееся для пользователя в воскресенье.

   ```
   youShould = "Take it easy. You've earned it!";
   ```

9. Добавьте команду завершения ветви.

   ```
   break;
   ```

10. Введите код ветвей для всех остальных дней недели.

11. После ветви для случая, в котором переменная `dayOfWeek` принимает значение 6, укажите действия по умолчанию, которые выполняются, когда не справедливо ни одно из указанных выше условий.

    ```
    default:
       youShould = "Hmm. Something has gone wrong.";
       break;
    ```

12. Завершите программную конструкцию выбора действия для каждого из дней недели, введя в ее конце закрывающую фигурную скобку.

    ```
    }
    ```

13. Под кодом оператора `switch` введите выражение вывода на экран сообщения (значения переменной `youShould`), заключенного в элемент `div`, который представлен идентификатором `thingToDo`.

```
document.getElementById("thingToDo").innerHTML = youShould
```

Полный код программы нашего делового календаря, включающий все указанные выше изменения, показан в листинге 15.3.

Листинг 15.3. JavaScript-код программы делового календаря

```
var todayDate = document.getElementById("todaysdate");
var todoButton = document.getElementById("whattodo");

// Обработчик событий для кнопки
todoButton.addEventListener("click", displayActivity);

// Создание объекта Date
var d = new Date();

// Вызов функции displayDate()
displayDate();

function displayDate() {
   // todo: отображение текущей даты во фрейме todaysdate
}
function displayActivity() {
   // todo: определить день недели
   var dayOfWeek = d.getDay();

   /* todo: сохранение в переменной youShould строки, зависящей
            от дня недели */

   var youShould;

   switch (dayOfWeek) {
      case 0:
         youShould = "Take it easy. You've earned it.";
         break;
      case 1:
         youShould = "Gotta do what ya gotta do!";
         break;
      case 2:
         youShould = "Take time to smell the roses!";
         break;
      case 3:
         youShould = "Don't forget to eat breakfast!";
         break;
      case 4:
         youShould = "Learn something new today!";
         break;
```

```
      case 5:
        youShould = "Make a list of things you like to do.";
        break;
      case 6:
         youShould = "Do one thing from your list of things
            ↳you like to do.";
         break;
      default:
         youShould = "Hmm. Something has gone wrong.";
         break;
      }

   // todo: вывод значения переменной youShould во фрейме
   //       с идентификатором thingToDo
   document.getElementById("thingToDo").innerHTML = youShould;
}
```

Завершив ввод кода, запустите приложение и щелкните в нем на единственной кнопке. На панели **Results** отобразится результат для текущего дня недели, подобный показанному на рис. 15.4.

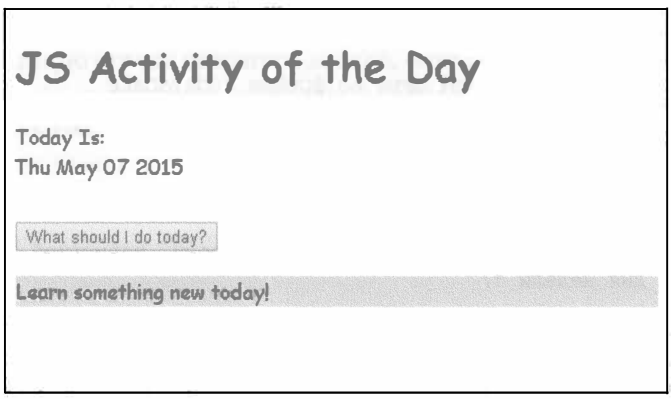


Рис. 15.4. Результат, возвращаемый программой делового календаря

Рассмотрите возможность дальнейшей модификации приложения.

- ✓ Укажите в качестве выводимых сообщений собственные деловые потребности.
- ✓ Рассмотрите необходимость указания отдельных дел для каждого дня месяца, а не недели.
- ✓ Выводите для пользователя сразу несколько сообщений: для каждого дня недели, каждого дня месяца, каждого месяца и каждого года.
- ✓ Измените внешний вид приложения календаря, воспользовавшись правилами CSS.

Обязательно реализуйте в приложении собственные идеи, отсутствующие в списке.

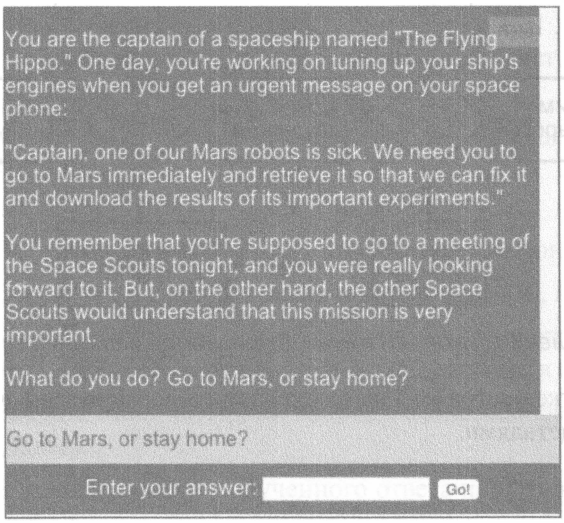
ГЛАВА

16

Написание повествования

Представьте, что вы карабкаетесь на широкое ветвистое дерево. Забравшись на одну из его ветвей, вы видите воронье гнездо или воздушный шарик, принесенный ветром с ближайшей вечеринки. Если вы переберетесь на соседнюю ветку, то вам откроется прекрасный вид на соседский бассейн. Аналогия с ветвями не случайна. В JavaScript выбор действия с помощью конструкций `if... else` и `switch` называют *ветвлением* программы.

В этой главе мы воспользуемся методикой ветвления программного кода для создания интерактивного повествования, в наполнении которого пользователю отводится далеко не последняя роль.



Повествование

У каждого художественного произведения есть свой сюжет. Это ряд последовательных событий, происходящих в любом повествовании. При написании истории, отдельные части которой согласуются с пользователем приложения, ее сюжет нужно продумывать как можно тщательнее. Каждая возможная сюжетная линия имеет общее начало, а вот середина повествования и концовка у них разные.

Алгоритм построения сюжета

Реализация всех заранее продуманных сюжетов в одном повествовании — вот задача для истинного разработчика, в совершенстве владеющего методикой ветвления программы.

В начале повествования пользователю предлагается ответить на вопрос. Каждый из двух возможных вариантов ответа запускает свою сюжетную ветвь. В каждой из ветвей повествования пользователю нужно ответить еще на один вопрос, предопределяющий дальнейшее развитие событий. К счастью, концовок у повествования всего две, и к ним ведут все возможные сюжетные ветви.

Для получения более наглядного представления о ветвлении сюжета рассмотрите блок-схему, показанную на рис. 16.1.

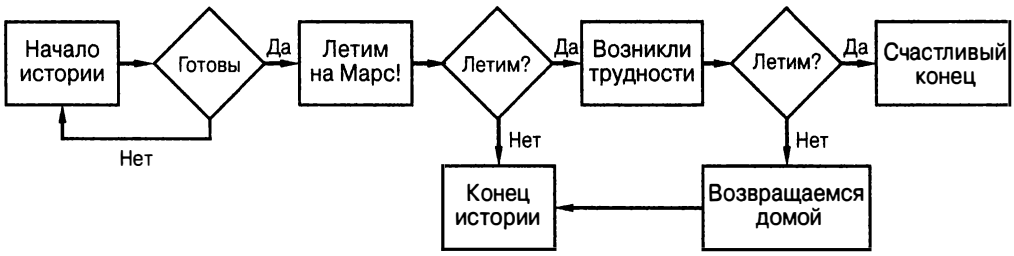


Рис. 16.1. Блок-схема ветвления сюжетной линии, выбираемой пользователем

Следующий этап в разработке программы написания повествования заключается в наполнении сюжетной линии деталями.

Сюжет

Наша история приключилась на космическом челноке в недалеком будущем. Вы как капитан корабля должны успешно довести его до Марса, забрать с поверхности планеты старого, уже не функционирующего робота, а затем вернуться на Землю, предоставив накопленную роботом информацию для дальнейшего изучения светилами науки.

Старт с Земли прошел успешно, и в первой половине пути не произошло сколь-нибудь значимых происшествий. Но на 260-й день полета вы внезапно обнаружили на корабле своего кота. Лишенные возможности пополнить запасы, вы обречены на нехватку провизии, чтобы успешно завершить путешествие. Перед вами стоит мучительный выбор: развернуть космический корабль обратно или продолжить путешествие к Марсу, надеясь на чудо.

Если повернуть назад, то кот будет спасен, но ваша миссия потерпит неудачу, что скажется не только на вашем финансовом благосостоянии, но и на дальнейшей карьере.

С другой стороны, продолжив путешествие, вам с котом придется обходиться половиной суточного пайка. Играя роль положительного персонажа, вы не позволите себе избавиться от животного, полностью зависящего от человека. Добравшись до поселения на Марсе, вы обнаруживаете, что последние посетившие его колонисты оставили в хранилище огромное количество съестных припасов. Подобрав робота и контейнер с провизией, вы успешно направляетесь к Земле, где вас встречают как героя! Счастливый конец!

Игра в писателя

Самая интересная часть приложения — это возможность непосредственно воздействовать на сюжетную линию. В зависимости от сделанного пользователем выбора в ходе повествования перед главным героем открываются новые возможности. Наша интерактивная история не будет длинной, поскольку для принятия решения пользователю нужно изучить уже выведенный в форму текст, что весьма утомительно. К тому же от нас требуется снабдить текстом все возможные сюжетные ветви, на что уходит много времени и сил.

Чтобы выяснить, как приложение будет писать историю о спасении робота из марсианского плена, выполните следующие действия.

1. Запустите браузер и отобразите в нем общедоступный кабинет приложения JSFiddle.

   `http://jsfiddle.net/user/forkids/fiddles`

   На экране появится список проектов, рассматриваемых в книге.

2. Откройте проект **Chapter 16 — Martian Rescue!**, щелкнув на его заголовке.

   Содержимое панелей кода для текущего проекта показано на рис. 16.2. Панель результатов содержит вопрос к пользователю и поле ввода ответа на него.

3. Введите ответ на первый вопрос в единственном поле и щелкните на кнопке **Go** (Далее).

   В зависимости от полученного ответа на панели **Results** появится следующий вопрос.

4. Ответьте на второй вопрос.

   Снова-таки, ваш ответ определяет продолжение сюжетной линии в повествовании.

5. Отвечайте на вопросы и просматривайте результаты своих действий до конца игры в сценариста.

6. Чтобы запустить приложение еще раз, щелкните на кнопке **Run**, расположенной в верхней части окна JSFiddle.

   Текст повествования, сформированный при первом запуске программы, удаляется, а на панели **Results** отображается исходный вопрос.

7. Сыграйте в игру еще раз, теперь отвечая на вопросы совершенно по-иному. Тем самым вы проверите доступность всех возможных сюжетных линий.

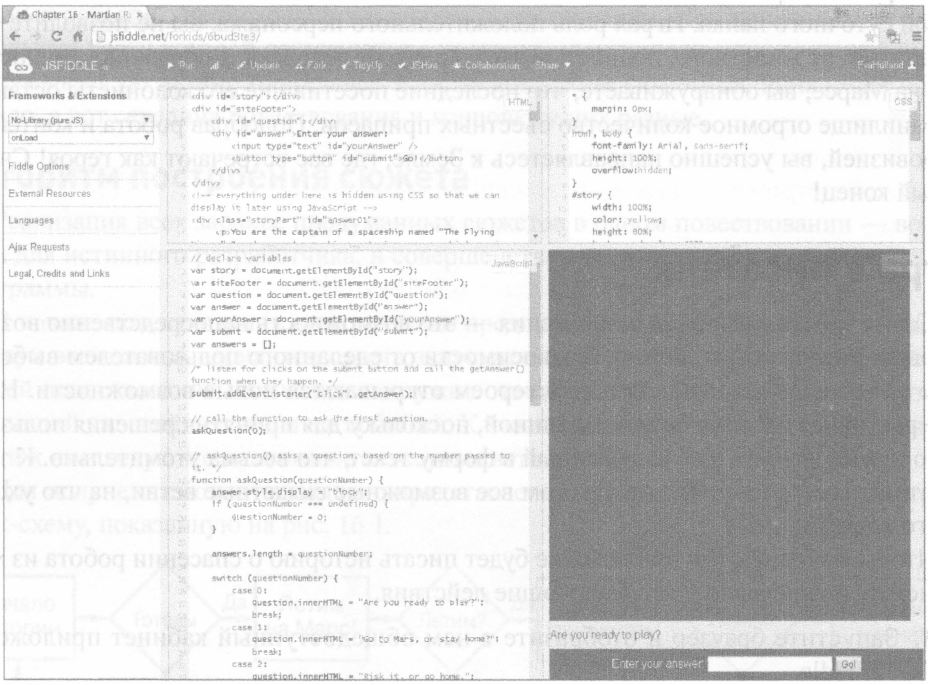


Рис. 16.2. Программа написания истории о спасении робота

Код приложения

Исходная версия нашего приложения, которое можно смело называть игрой, включает полный код HTML и CSS, чего не скажешь о коде JavaScript — он далек от завершения. Чтобы создать копию приложения, отвечающего за написание истории о чудесном спасении робота на Марсе, выполните в JSFiddle следующие действия.

1. Зайдите в JSFiddle под собственной учетной записью.
2. Перейдите в общедоступный кабинет по следующему адресу:
   `http://jsfiddle.net/user/forkids/fiddles`
   Отыщите в списке название **Chapter 16 — Martian Rescue — Start**.
3. Запустите исходную версию проекта, щелкнув на ней.
4. На верхней панели инструментов щелкните на кнопке **Fork**, чтобы скопировать код приложения в личный кабинет JSFiddle.
5. Измените название проекта на более привлекательное.
6. Чтобы сохранить копию приложения, щелкните на кнопке **Update**, а затем выполните команду **Set as Base**.

Код HTML и CSS

Как уже упоминалось, код HTML и CSS уже содержится в исходной версии нашего приложения. Нам необходимо внимательно изучить его перед тем, как приступить к написанию кода JavaScript. Начнем экскурс с содержимого панели HTML.

Документ HTML состоит из двух частей, разделенных комментарием. В первой части кода создается верхняя часть страницы, на которую выводится скомпонованное из отдельных частей повествование. В нижней части страницы отображаются вопросы и содержится поле, в которое пользователь вводит ответы.

Код создания верхней части веб-страницы приведен в листинге 16.1.

Листинг 16.1. Начальная часть кода HTML

```
<div id="story"></div>
<div id="siteFooter">
   <div id="question"></div>
   <div id="answer">Enter your answer:
      <input type="text" id="yourAnswer" />
      <button type="button" id="submit">Go!</button>
   </div>
</div>
```

При первом запуске начальной версии приложения содержимое панели результатов определяется приведенными выше строками HTML-кода, а его форматирование устанавливается с помощью кода CSS.

До выполнения кода JavaScript панель **Results** выглядит так, как на рис. 16.3.

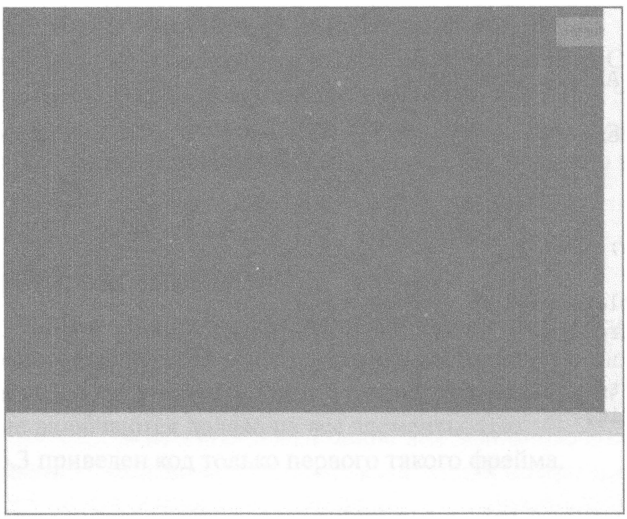


Рис. 16.3. Содержимое панели результатов при запуске начальной версии программы

Обратите внимание на то, что веб-страница состоит из трех областей, имеющих разные фоны.

- ✓ Верхняя, самая темная, область предназначена для вывода скомпонованного программой повествования.
- ✓ На светло-серой области выводятся вопросы, задаваемые пользователю.
- ✓ Белая область отведена для поля ввода вариантов ответов на поставленные вопросы.

Если вы внимательно изучите приведенный выше HTML-код и содержимое панели **Results**, то легко заметите несоответствие. В HTML-коде четко указывается создать на веб-странице кнопку и поле ввода данных, но на панели результатов они не видны. Давайте разберемся, почему.

Отключение элементов

Кнопку и поле ввода данных нужно отображать в HTML-документе только тогда, когда программа запрашивает данные у пользователя. В остальных случаях эти элементы скрываются, для чего используется код CSS.

Полный CSS-код приложения написания истории о спасении робота приведен в листинге 16.2.

Листинг 16.2. CSS-код программы написания истории

```
* {
   margin: 0px;
}
html, body {
   font-family: Arial, sans-serif;
   height: 100%;
   overflow:hidden;
}
#story {
   width: 100%;
   color: yellow;
   height: 80%;
   background-color: #333;
   overflow-y:scroll;
}
#site-footer, .story:after {
   position:static;
   bottom: 0;
   height: 20%;
}
#question {
   padding: 10px 0;
   width: 100%;
```

```
   background-color: #CCC;
   color: #333;
}
#answer {
   padding: 10px 0;
   width: 100%;
   background-color: #333;
   color: #FFF;
   text-align: center;
   display: none;
}
.storyPart {
   display: none;
}
p {
   margin-top: 1em;
}
```

Вернемся к HTML-коду документа. Вы видите, что элементы `input` и `button` располагаются внутри элемента `div` с идентификатором `answer`.

Чтобы узнать, какие свойства элемента `div` изменяются, отыщите на панели CSS правило с селектором, содержащим ключевое слово `answer`.

В первых пяти свойствах указанного правила устанавливаются цвет фона, цвет текста, отступ, ширина поля и выравнивание текста в нем. А вот последнее свойство задает режим видимости элемента.

```
display: none;
```

Если свойство `display` имеет значение `none`, соответствующий элемент не отображается на странице, хотя и присутствует на ней.

Чаще всего программисты пользуются этой особенностью кода CSS для сокрытия и отображения элементов HTML-документов с помощью JavaScript. Чтобы отобразить скрытый с помощью CSS-кода элемент, необходимо в коде JavaScript изменить значение свойства `display` на противоположное, например, как показано ниже.

```
document.getElementById("answer").style.display = "block";
```

Компоненты повествования

Вторая часть HTML-кода предназначена для создания нескольких элементов `div`, каждый из которых содержит отдельную часть повествования, включаемую в общий сюжет только при вводе пользователем определенного ответа. Таким образом, в конечное повествование включаются далеко не все элементы `div`.

В листинге 16.3 приведен код только первого такого фрейма.

Листинг 16.3. Фрейм начальной части повествования

```
<div class="storyPart" id="answer01">
   <p>You are the captain of a spaceship named "The
      Flying Hippo." One day, you're working on
      tuning up your ship's engines when you get an
      urgent message on your space phone:</p>
   <p>"Captain, one of our Mars robots is sick. We need
      you to go to Mars immediately and retrieve it
      so that we can fix it and download the results
      of its important experiments."</p>
   <p>You remember that you're supposed to go to a meeting
      of the Space Scouts tonight, and you were
      really looking forward to it. But, on the other
      hand, the other Space Scouts would understand
      that this mission is very important.</p>
   <p>What do you do? Go to Mars, or stay home?</p>
</div>
```

Все части повествования относятся к классу `storyPart`, но каждая из них имеет уникальный идентификатор (атрибут `id`).

В документе может быть несколько элементов с одним и тем же значением атрибута `class`, но каждый из них обладает своим, уникальным атрибутом `id`.

Попробуйте угадать, почему при запуске приложения ни один из элементов `div` с атрибутом `class=="storyPart"` не отображается на панели результатов. Конечно! Ко всем ним применяется CSS-свойство `display`, исходно установленное равным значению `none`.

Взгляните на содержимое панели CSS. Найдите на ней селектор `.storyPart` и убедитесь, что для него указано только одно свойство, `display: none;`. Таким образом, в документе скрываются сразу все элементы, относящиеся к классу `storyPart`. В процессе дальнейшего выполнения приложения управление режимом отображения отдельных элементов осуществляется в коде JavaScript.

На этом мы завершаем рассмотрение кода HTML и CSS нашей программы и переходим к самому интересному — анализу и написанию кода JavaScript.

Код JavaScript

При запуске исходной версии приложения, отвечающего за написание истории, его код JavaScript весьма скромный, как видно из листинга 16.4.

Давайте проанализируем имеющиеся команды и дополним их всеми необходимыми инструкциями.

Листинг 16.4. Начальный JavaScript-код программы написания истории

```
// Объявление переменных
var story = document.getElementById("story");
var siteFooter = document.getElementById("siteFooter");
var question = document.getElementById("question");
var answer = document.getElementById("answer");
var yourAnswer = document.getElementById("yourAnswer");
var submit = document.getElementById("submit");

// todo: создать пустой массив с именем answers

/* todo: добавить обработчик событий, вызывающий функцию
   getAnswer() после щелчка на кнопке */

// todo: вызвать функцию, отвечающую за вывод первого вопроса

/*  Вопрос, формируемый функцией askQuestion(), зависит от
    переданного ей аргумента */
function askQuestion(questionNumber) {
}

/* Функция getAnswer() извлекает ответ из текстового поля
   и заносит его в специальный массив, а затем вызывает функцию
   continueStory() */
function getAnswer() {
}

/* Функция continueStory() в зависимости от полученного из массива
   значения выводит часть повествования или сообщение */
function continueStory(answerNumber) {
}

/* Функция theEnd() завершает повествование и скрывает поле */
function theEnd() {
}
```

Ссылки на элементы

В первой части кода JavaScript объявляются глобальные переменные, активно используемые в дальнейшем коде. Все они уже созданы нами, поскольку представляют в приложении отдельные элементы HTML. Эти переменные применяются исключительно удобства ради, позволяя избегать постоянного ввода длинных конструкций, подобных `document.getElementById()`.

Разумеется, заменив длинные инструкции более компактным выражением (именем переменной), вы сэкономите много времени и сил. В общем случае переменная объявляется согласно следующему шаблону.

```
var myElement = document.getElementById("myElement");
```

Код, в котором обращение к элементу осуществляется по названию переменной, выглядит компактнее и понятнее, а потому такой метод ссылки на элементы использовать предпочтительнее.

Пустой массив

После объявления переменных приведен комментарий, в котором указывается создать пустой массив.

Как рассказывалось в главе 11, для создания пустого массива (не содержащего элементов) необходимо объявить новую переменную, в качестве значения которой указываются открывающая и закрывающая квадратные скобки. Таким образом, чтобы создать массив `answers`, не содержащий назначенных элементов, используется следующий код.

```
var answers = [];
```

Этот пустой массив создается вне функции, поэтому к нему можно обращаться в любом месте программы.

Переменная, которая доступна для использования в любом месте программы, называется *глобальной.*

Обработчик события

В следующем комментарии, содержащем инструкцию `todo:`, указывается создать в коде JavaScript обработчик события щелчка на кнопке. За прослушивание события отвечает специальный встроенный метод, который в нашем случае называется `addEventListener()`. Конечно, вы все это уже знаете из предыдущих глав.

Чтобы написать код необходимого обработчика события, выполните такие инструкции.

1. Сразу под комментарием, с новой строки введите название переменной, представляющей в коде JavaScript элемент кнопки. В конце имени введите точку.

   ```
   submit.
   ```

2. Добавьте после точки название метода, прослушивающего целевое событие, сопроводив его открывающей и закрывающей скобками.

   ```
   submit.addEventListener()
   ```

   Внутри скобок указываются аргументы метода `addEventListener()`: их всего два. Первый определяется прослушиваемое событие, а второй — функцию, которая выполняется при его возникновении.

   ```
   submit.addEventListener("click",getAnswer);
   ```

Замечательно! Вы получили массив, в котором будут храниться ответы пользователя на поставленные программой вопросы, и обработчик события для элемента кнопки. Но если запустить программу прямо сейчас, то ничего из продемонстрированного в

готовой программе происходить не будет. На панели результатов отображаются три статические области с разными фонами.

Чтобы заставить программу выполнить некое полезное действие, необходимо вызвать в ней правильную функцию. В нашем случае это функция `askQuestions()` — на ее важность указывает следующий комментарий с ключевым словом `todo:`, приведенный на панели JavaScript.

Вызов функции `askQuestions()`

Функция `askQuestions()` снабжена единственным параметром, `questionNumber`, представляющим номер вопроса, задаваемого пользователю программой. Помня о специфике счета в JavaScript, можно смело утверждать, что первый вопрос в программе имеет номер 0.

Исходя из вышесказанного, чтобы вызвать функцию и задать пользователю вопрос, необходимо сразу после соответствующего комментария ввести такой код.

```
askQuestion(0);
```

После выполнения требований всех комментариев, содержащих ключевые слова `todo:`, их лучше удалить, дабы в дальнейшем не вводить себя в заблуждение.

Поздравляем! Вы написали основной код JavaScript, не относящийся к функциям. В листинге 16.5 приведен его полный вариант.

Листинг 16.5. Код JavaScript, не относящийся к функциям

```
// Объявление переменных
var story = document.getElementById("story");
var siteFooter = document.getElementById("siteFooter");
var question = document.getElementById("question");
var answer = document.getElementById("answer");
var yourAnswer = document.getElementById("yourAnswer");
var submit = document.getElementById("submit");
var answers = [];

/* Создание обработчика событий, вызывающего функцию getAnswer()
   по щелчку на кнопке */

submit.addEventListener("click", getAnswer);

// Вызов функции, выводящей первый вопрос пользователю
askQuestion(0);
```

Как это ни странно, при запуске в таком виде эта программа снова не выполняет никаких действий, представляемых на панели **Results** значимыми данными.

Чтобы наконец заставить программу выводить некий результат, нам осталось наполнить кодом используемые в ней функции.

Код функций

Первая функция, кодом которой мы займемся, — `askQuestion()`; она задает вопросы пользователю.

Чтобы наполнить функцию `askQuestion()` содержимым, выполните такие действия.

1. Измените значение свойства `display` элемента `div`, в котором выводится вопрос, на противоположное, чтобы отобразить на экране кнопку и поле ввода данных.

   ```
   answer.style.display = "block";
   ```

   На панели результатов появится форма ввода данных.

2. Измените длину массива `answers`, приравняв ее к номеру задаваемого пользователю вопроса.

   ```
   answers.length = questionNumber;
   ```

   В этой команде передаваемый функции аргумент устанавливается равным длине массива `answers`. Это делается для того, чтобы ответы всегда сохранялись в массиве вместе с вопросами. В случае, если пользователь ответит неправильно, приравнивание длины массива к номеру текущего вопроса позволит переписать неправильный ответ и задать вопрос повторно.

   Если свойству `length` присвоить значение, меньшее реальной длины массива, то элементы, имеющие больший индекс, попросту удаляются.

3. Напишите код конструкции ветвления `switch`, в которой выбор задаваемого вопроса проводится по аргументу, передаваемому в функцию.

   Код команды `switch` имеет следующий вид.

   ```
   switch (questionNumber) {
      case 0:
         question.innerHTML = "Are you ready to play?";
         break;
      case 1:
         question.innerHTML = "Go to Mars, or stay home?";
         break;
      case 2:
         question.innerHTML = "Risk it, or go home.";
         break;
      default:
         break;
   }
   ```

Строго говоря, в блоке действий по умолчанию инструкции `switch` использовать ключевое слово `break` совсем не обязательно. По его завершении выполнение программы все равно будет продолжено инструкциями, которые указаны после закрывающей фигурной скобки, завершающей конструкцию `switch`. В нашем случае оператор `break` приводится исключительно для того, чтобы соблюсти формальности и сохранить единую структуру кода.

4. Завершите текущую функцию, введя после команды ветвления `switch` закрывающую фигурную скобку.

```
}
```

5. Сохраните код функции, щелкнув на кнопке **Update**.

Завершенная функция `askQuestion()` имеет код, показанный в листинге 16.6.

Листинг 16.6. Полный код функции `askQuestion()`

```
/*  Вопрос, формируемый функцией askQuestion(), зависит от
    переданного ей аргумента */
function askQuestion(questionNumber) {
   answer.style.display = "block";

   // Проверка длины массива
   answers.length = questionNumber;
   switch (questionNumber) {
      case 0:
         question.innerHTML = "Are you ready to play?";
         break;
      case 1:
         question.innerHTML = "Go to Mars, or stay home?";
         break;
      case 2:
         question.innerHTML = "Risk it, or go home.";
         break;
      default:
         break;
   }
}
```

Выполнение приложения с завершенной функцией `askQuestion()` приводит к выводу на панель **Results** ожидаемого результата. В нижней части страницы отображается первый вопрос, ответ на который нужно ввести в поле, а подтвердить — щелчком на кнопке, как показано на рис. 16.4.

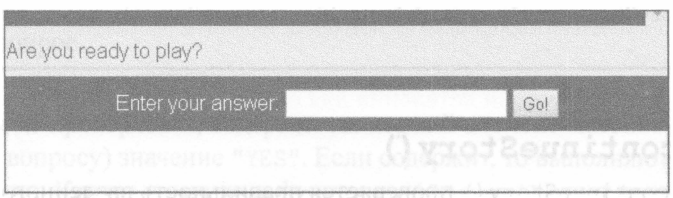


Рис. 16.4. Наконец-то! Первый вопрос пользователю

Тем не менее после ввода в поле произвольного значения и щелчка на кнопке ровным счетом ничего не происходит. Чтобы обеспечить реакцию программы на ответ пользователя, нужно написать код следующих двух функций.

Выполните следующие действия для наполнения кодом функции `getAnswer()`.

1. Извлеките значение из поля ввода данных и приведите его символы к верхнему регистру.

   ```
   cleanInput = yourAnswer.value.toUpperCase();
   ```

2. Воспользуйтесь методом `push()` объекта массива, чтобы представить ответ пользователя отдельным элементом и добавить его в конец массива.

   ```
   answers.push(cleanInput);
   ```

3. Сбросьте поле ввода данных, удалив из него текущее значение.

   ```
   yourAnswer.value = "";
   ```

4. Вызовите функцию `continueStory()`, подставив в нее индекс последнего элемента в массиве `answers`.

   ```
   continueStory(answers.length - 1);
   ```

Поскольку индексация элементов массива начинается с нуля, *длина* массива (количество его элементов) всегда больше на единицу, чем индекс последнего элемента. Именно поэтому в приведенном выше аргументе функции `continueStory()` из длины массива вычитается `1`.

5. Завершите функцию `getAnswer()`, введя с новой строки символ закрывающей фигурной кавычки.

   ```
   }
   ```

Полный код функции `getAnswer()` представлен в листинге 16.7.

В следующем разделе речь идет о коде функции `continueStory()`.

Листинг 16.7. Окончательный вид функции `getAnswer()`

```
/* Функция getAnswer() получает введенный в поле ответ
   пользователя и заносит его в массив answers, после чего
   вызывает функцию continueStory() */
function getAnswer() {
   cleanInput = yourAnswer.value.toUpperCase();
   answers.push(cleanInput);
   yourAnswer.value = "";
   continueStory(answers.length - 1);
}
```

Функция `continueStory()`

В функции `continueStory()` проверяется правильность введенного пользователем ответа, для чего применяется условная конструкция `if...else`. После этого в повествование добавляется та часть текста, которая соответствует выбранной сюжетной линии.

Чтобы написать код функции `continueStory()`, выполните следующие действия.

1. Используйте команду `switch` для определения номера задаваемого вопроса.

   Базовая конструкция `switch` (без условных операторов `if...else` для каждого вопроса) имеет следующий вид.

```
switch (answerNumber) {
   case 0:
      //добавьте инструкции
      break;
   case 1:
      // добавьте инструкции
      break;
   case 2:
      // добавьте инструкции
      break;
   default:
      // добавьте инструкции
      break;
}
```

2. Сформулируйте условную конструкцию `if...else` для первого вопроса: "Are you ready to play?" (Готовы играть?).

   После ее добавления в команду `switch` вы должны получить такой код.

```
case 0:
   if (answers[0] === "YES") {
      story.innerHTML = document.getElementById("answer01").innerHTML;
      askQuestion(1);
   } else if (answers[0] === "NO") {
      story.innerHTML = document.getElementById("answer02").innerHTML;
      askQuestion(0);
   } else {
      story.innerHTML = document.getElementById("err0").innerHTML;
      askQuestion(0);
   }
   break;
```

   Давайте проанализируем этот код построчно.

```
case 0:
```

   Эта строка указывает выполнить следующий код, если пользователь ответил на первый вопрос.

```
if (answers[0] === "YES") {
```

   В этом коде проверяется, содержит ли первый элемент массива (соответствует первому вопросу) значение `"YES"`. Если содержит, то выполняются дальнейшие инструкции. Как вы помните, в функции `getAnswer()` ответ пользователя перед помещением в массив конвертировался в символы верхнего регистра. Именно поэтому при вводе в поле `yes`, `Yes` или даже `yeS` условный оператор возвращает значение `true`.

```
story.innerHTML = document.getElementById("answer01").innerHTML;
```

В этом выражении содержимое элемента `div` с идентификатором `answer01` заменяет содержимое элемента `div` с идентификатором `story`. Если найти элемент `div` с идентификатором `answer01` на панели HTML, то можно удостовериться, что он содержит начало повествования.

Ответ `"YES"` на вопрос "Are you ready to play?", должен приводить к отображению первой части повествования.

```
askQuestion(1);
```

Текущая команда вызывает функцию `askQuestion()`, которой в качестве аргумента передается число `1`. На панель результатов при этом выводится второй вопрос из массива: "Go to Mars, or stay home?" (Летим на Марс или остаемся дома?).

```
} else if (answers[0] === "NO") {
```

Если ответ пользователя отличается от `"YES"`, то запускается ветвь инструкций, указанная после ключевого слова `else`. Она начинается с еще одной условной конструкции `if`. Таким образом, ответ `"NO"` проверяется только в случае, если первый ответ отличался от `"YES"`.

```
story.innerHTML = document.getElementById("answer02").innerHTML;
```

Получив отрицательный ответ, программа устанавливает свойство `innerHTML` элемента `div` равным исходному значению.

```
askQuestion(0);
```

Так как пользователь отказался играть, для него выводится первый вопрос. Только при утвердительном ответе на него повествование начинает наполняться текстом.

```
} else {
```

В случае получения ответа, не совпадающего с основными значениями, `"YES"` и `"NO"`, выполняется следующая команда.

```
story.innerHTML = document.getElementById("err0").innerHTML;
```

Здесь указывается вывести в элементе `div` с идентификатором `story` сообщение об ошибке, содержащее просьбу отвечать на вопросы кратко: `Yes` или `No`.

```
askQuestion(0);
```

Программа вновь переходит к первому вопросу в надежде получить один из правильных ответов.

3. Напишите конструкции ветвления для оставшихся двух вопросов.

```
case 1:
   if (answers[1] === "GO TO MARS") {
      story.innerHTML = document.getElementById("answer11").innerHTML;
      askQuestion(2);
   } else if (answers[1] === "STAY HOME") {
      story.innerHTML = document.getElementById("answer12").innerHTML;
      theEnd();
   } else {
      story.innerHTML = document.getElementById("err1").innerHTML;
```

```
            askQuestion(1);
         }
         break;
      case 2:
         if (answers[2] === "RISK IT") {
            story.innerHTML = document.getElementById("answer21").innerHTML;
            theEnd();
         } else if (answers[2] === "GO HOME") {
            story.innerHTML = document.getElementById("answer22").innerHTML;
            theEnd();
         } else {
            story.innerHTML = document.getElementById("err2").innerHTML;
            askQuestion(2);
         }
         break;
         default:
            story.innerHTML = "The story is over!";
         break;
   }
```

4. Завершите функцию закрывающей фигурной кавычкой.

```
   }
```

5. Сохраните код JavaScript, щелкнув на кнопке Update.

Полный код функции `continueStory()` приведен в листинге 16.8.

Листинг 16.8. Код функции `continueStory()`

```
/* Функция continueStory() выводит повествование или сообщение
   об ошибке в зависимости полученного из массива answers значения */
function continueStory(answerNumber) {
   switch (answerNumber) {
      case 0:
         if (answers[0] === "YES") {
            story.innerHTML = document.getElementById("answer01").
               ↳innerHTML;
            askQuestion(1);
         } else if (answers[0] === "NO") {
            story.innerHTML = document.getElementById("answer02").
               ↳innerHTML;
            askQuestion(0);
         } else {
            story.innerHTML = document.getElementById("err0").
               ↳innerHTML;
            askQuestion(0);
         }
         break;
      case 1:
         if (answers[1] === "GO TO MARS") {
            story.innerHTML = document.getElementById("answer11").
               ↳innerHTML;
            askQuestion(2);
```

```
        } else if (answers[1] === "STAY HOME") {
            story.innerHTML = document.getElementById("answer12").
                ↳innerHTML;
            theEnd();
        } else {
            story.innerHTML = document.getElementById("err1").
                ↳innerHTML;
            askQuestion(1);
        }
        break;
    case 2:
        if (answers[2] === "RISK IT") {
            story.innerHTML = document.getElementById("answer21").
                ↳innerHTML;
            theEnd();
        } else if (answers[2] === "GO HOME") {
            story.innerHTML = document.getElementById("answer22").
                ↳innerHTML;
            theEnd();
        } else {
            story.innerHTML = document.getElementById("err2").
                ↳innerHTML;
            askQuestion(2);

        }
        break;
    default:
        story.innerHTML = "The story is over!";
        break;
  }
}
```

Нам осталось рассмотреть код всего одной функции, которая вызывается, когда нужно завершить повествование.

Функция `theEnd()`

Функция `theEnd()` отвечает за вывод на веб-страницу последней части повествования и сокрытие элемента `div`, включающего поле ввода данных и кнопку отправки ответа. Чтобы правильно написать код этой функции, следуйте приведенным далее инструкциям.

1. Введите в теле функции следующее выражение, которое отвечает за отображение в конце повествования, выводимого в элементе `div` с идентификатором `story`, сообщения "The End".

   ```
   story.innerHTML += "<p>The End.</p>";
   ```

2. Удалите из элемента `div` с идентификатором `question` последний вопрос к пользователю.

   ```
   question.innerHTML = "";
   ```

3. Скройте поле ввода данных и кнопку отправки ответа.

```
answer.style.display = "none";
```

4. Щелкните на кнопке **Update**, чтобы сохранить код функции.

Полный код функции `theEnd()` показан в листинге 16.9.

Листинг 16.9. Код функции `theEnd()`

```
/* Функция theEnd() завершает повествование и скрывает форму */
function theEnd() {
   story.innerHTML += "<p>The End.</p>";
   question.innerHTML = "";
   answer.style.display = "none";
}
```

Создание программы интерактивного повествования о покорителях космоса завершено. Чтобы протестировать его, щелкните на кнопке **Update**, а затем выполните команду **Set as Base**.

Если вы умудрились не наделать ошибок, то можете насладиться игрой, отвечая на задаваемые вопросы как вам заблагорассудится. Содержимое панели результатов в процессе игры проиллюстрировано на рис. 16.5.

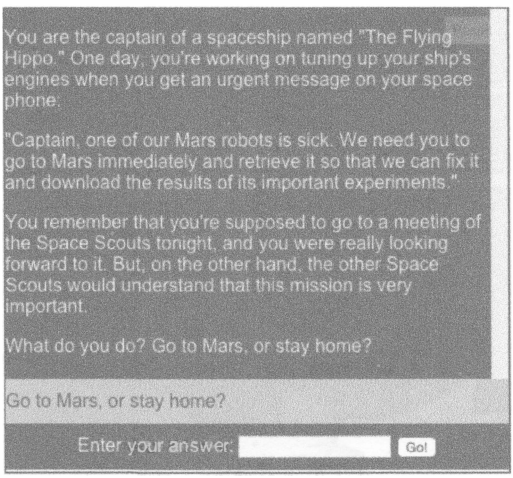


Рис. 16.5. Частично скомпонованное повествование

А у вас есть собственные идеи по созданию интерактивных историй? Не хотите сделать повествование более разветвленным и длинным? Проанализируйте программу и попробуйте преобразовать ее в нечто более совершенное.

Часть VI
Циклы

В этой части...

- ✓ Циклы в JavaScript
- ✓ Цикл `While`
- ✓ Лимонадная лавка

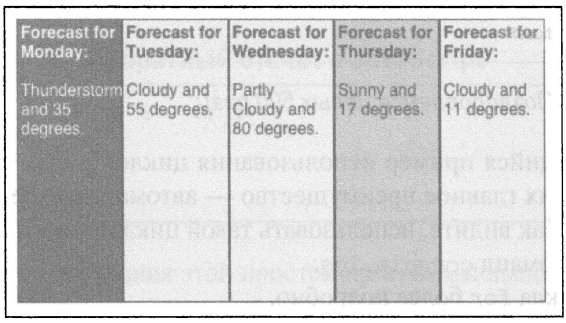
ГЛАВА

17

Повторяющиеся действия в JavaScript

Циклически выполнять одни и те же операции имеет смысл только в случае, когда известно, сколько раз это нужно сделать. В коде JavaScript эта задача решается с помощью циклической структуры `for`, позволяющей реализовать повторение одних и тех же действий произвольное количество раз.

В этой главе мы поговорим об одной и самых распространенных программных конструкций JavaScript — о цикле `for`. В процессе изучения вы увидите, как воспользоваться ею при написании приложения прогноза погоды.



Знакомство с циклом for

Сегодня самым распространенным в JavaScript является цикл `for`. Ниже приведен пример циклической структуры, которая выводит сообщение `Привет, JavaScript!` на экран ни много ни мало — 500 раз.

```
for (var counter = 0; counter < 500; counter++) {
   console.log(counter + ": Привет, JavaScript!");
}
```

Результат выполнения этого кода на панели разработчика в браузере Chrome показан на рис. 17.1.

Рис. 17.1. Поздороваемся целых 500 раз!

Это далеко не самый выдающийся пример использования циклов в программах JavaScript, но он демонстрирует их главное преимущество — автоматическое повторение одних и тех же операций. Как видите, использовать такой цикл намного проще, чем добавлять в программу 500 команд `console.log`.

Давайте изучим структуру цикла `for` более подробно.

Структура цикла

Цикл `for` состоит из трех основных частей.

- ✓ **Инициализация.** В выражении инициализации объявляется переменная, значение которой определяет, сколько раз в теле цикла будут выполняться операции.
- ✓ **Условие.** Булево выражение, выполняемое в начале каждой итерации и определяющие необходимость продолжения цикла.
- ✓ **Операции.** Инструкции, повторно выполняемые на каждой итерации цикла.

Если взглянуть на цикл, отвечающий за вывод на панель разработчика сообщений `Привет, JavaScript!`, то разобраться в том, как он работает, не составит большого труда.

1. В начале цикла инициализируется новая переменная `counter` (счетчик цикла) со значением `0`.
2. В условии проверяется, меньше ли значение переменной числа 500.

   Если меньше, то выполняются операции, указанные в теле цикла. Именно они отвечают за вывод на экран приветствия `Привет, JavaScript!`.
3. В последней инструкции значение счетчика цикла увеличивается на единицу.
4. Условие проверяется снова, чтобы получить разрешение на продолжение цикла.

   Если условие истинно, то инструкции тела цикла выполняются еще раз.
5. Счетчик цикла снова увеличивается на единицу.
6. Операции из пп. 2 и 3 повторяются до тех пор, пока сохраняется истинность условия (`counter < 500`).

Назначение цикла

Самое очевидное применение цикла `for` заключается в использовании счетчика цикла для изменения содержимого, выводимого на экран.

Простейший пример такого поведения программы приведен в листинге 17.1. В нем команда `alert()` используется для проведения обратного отсчета.

Листинг 17.1. Обратный отсчет в JavaScript

```
for (var i = 10; i > 0; i--) {
   alert (i);
}
alert ("Пуск!");
```

Для тестирования этой простой программы выполните следующие действия.

1. Посетите сайт `http://jsfiddle.net` и войдите в приложение под собственной учетной записью.
2. Запустите новый проект, щелкнув на логотипе **JSFiddle**.
3. Введите на панель JavaScript код из листинга 17.1.
4. Для выполнения программы щелкните на кнопке **Run**.

   На экране появится окно сообщения, содержащее число 10. Щелкнув на кнопке **OK**, вы увидите, что на экран выводится новое сообщение, в котором указано число 9. Новые сообщения появляются на экране до тех пор, пока значение счетчика (`i`) больше нуля. Как только условие нарушается, программа выходит из цикла, и выполняется инструкция, следующая после него, — на экране появляется сообщение `Пуск!`.

Использовать цикл для проведения одного только обратного отсчета как минимум непрактично. У циклов есть более важное применение, заключающееся в обработке элементов массива.

В листинге 17.2 показана программа, создающая массив, в котором хранятся имена пользователей. В ней цикл `for` используется для вывода шаблонного предложения, в котором имя пользователя заменяется одним из указанных в массиве значений.

Листинг 17.2. Вывод массива значений с помощью цикла `for`

```
var myFriends = ["Андрей", "Александр", "Алексей", "Анатолий"];

for (var i = 0; i < myFriends.length; i++){
   alert(myFriends[i] + " мой друг.");
}
```

Чтобы обеспечить включение в сообщение всех элементов массива, количество итераций цикла приравнивается к длине массива (значение свойства `length`). Команда `alert()` выполняется до тех пор, пока длина массива больше значения счетчика цикла (`i`).

Для извлечения имени, выводимого в сообщении, в качестве индекса элемента массива используется текущее значение счетчика.

Теперь, когда вы знаете, как использовать цикл `for` для управления элементами массива, можно применить его в приложении, выполняющем некую прикладную задачу. В последующем материале главы рассматривается программа прогноза погоды на пять дней, в коде которой циклической структуре `for` отводится центральное место.

Приложение прогноза погоды

Ничто так не изменчиво, как погода. Выходя с утра на улицу, вы никогда не угадаете, как правильно одеться. В этом есть нечто мистическое. Если одеться легко, то обязательно похолодает. Надев теплый свитер, вы обязательно попадете под дождь, а взяв с собой зонт, будете целый день наблюдать безоблачное небо. Решение рядом! Только точный прогноз погоды спасет наш бренный мир!

Ваша следующая должность — метеоролог. Теперь вы следите погодой и предсказываете ее поведение в ближайшие дни. О результатах вашей работы рассказывают по телевизору и в газетах!

Конечно, мы несколько преувеличиваем. Все не настолько серьезно, как кажется. Но никогда не знаешь наверняка, где продолжится карьера.

Конечно, у нас нет профессионального образования в области метеорологии, поэтому наша программа в качестве прогноза погоды будет выводить случайные значения. О том, как получить в JavaScript случайные числа, рассказывается в следующем разделе.

Метод `Math.random()`

Для создания произвольных числовых значений в JavaScript используется специальная функция. Она называется `Math.random()`.

При каждом запуске функция `Math.random()` возвращает произвольное дробное число в диапазоне от 0 до 1. Используя это значение, вы получаете возможность выполнять в программах огромное количество задач, требующих принятия произвольных решений — например, обеспечивать хаотичность перемещения монстров в игре или выбирать случайный элемент из массива. Последнее будет применяться нами при создании самого необычного прогноза погоды в вашей жизни.

В листинге 17.3 показан код, обеспечивающий вывод на экран сообщения, в котором содержится случайное число. Запустите эту программу в приложении JSFiddle несколько раз и убедитесь, что дважды получить с ее помощью одно и то же число практически невозможно.

Листинг 17.3. Вывод случайного числа

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

Пример случайного значения, выводимого на экран при выполнении кода из листинга 17.3, показан на рис. 17.2.

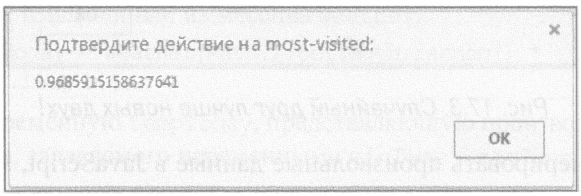


Рис. 17.2. Произвольное числовое значение

Впоследствии выданное методом `Math.random()` десятичное число с огромным количество цифр после запятой с помощью специальных функций приводится к произвольному значению из диапазона, которым оперирует конечная программа.

Например, если в программе нужно вывести произвольное число в диапазоне от 0 до 10, то результат выполнения функции `Math.random()` необходимо умножить на 11.

```
alert(Math.random() * 11);
```

Для удаления из произвольного числа десятичной части используется еще одна встроенная функция: метод `Math.floor()`.

```
alert(Math.floor(Math.random() * 11);
```

Чтобы получить случайное число в диапазоне от 10 до 1000, следует умножить значение, возвращаемое функцией `Math.random()`, на результат вычитания наименьшего числа диапазона из наибольшего, к которому прибавлено наименьшее число.

```
alert(Math.floor(Math.random() * (1000 - 100) + 100));
```

Для выбора произвольного элемента из массива нужно поступать так же, как и при генерации случайного числа из диапазона, начинающегося со значения 0. Умножив

произвольное значение на длину массива и округлив полученный результат, вы получите индекс случайного элемента.

Например, в листинге 17.4 приведен код программы, в которой сначала создается массив `myFriends`, а из него впоследствии (с помощью метода `Math.random()`) извлекается произвольный элемент.

Листинг 17.4. Произвольный элемент массива

```
var myFriends = ["Андрей", "Александр", "Алексей", "Анатолий"];
var randomFriend = Math.floor(Math.random() * myFriends.length);
alert(myFriends[randomFriend]);
```

При запуске этой программы в приложении JSFiddle на экране появится сообщение, в котором указывается имя произвольного друга, как показано на рис. 17.3.

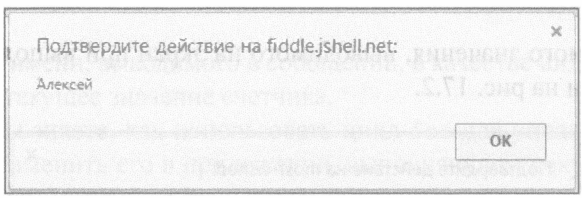


Рис. 17.3. Случайный друг лучше новых двух!

Научившись генерировать произвольные данные в JavaScript, можете смело приступать к написанию приложения, в котором они играют важную роль.

Приложение, выдающее случайный прогноз погоды

Для написания кода приложения, выдающего произвольный прогноз погоды, следуйте приведенным ниже инструкциям.

1. Перейдите в браузере на сайт `http://jsfiddle.net` и войдите в него под собственной учетной записью.
2. Создайте новый проект, щелкнув на логотипе **JSFiddle**.
3. Воспользуйтесь командой **Fiddle Options**, расположенной на левой панели, чтобы ввести имя новой программы.
4. На панели инструментов, находящейся в верхней части окна, щелкните на кнопке **Save** для сохранения проекта и его размещения в общедоступном кабинете.
5. На панели HTML создайте элемент `div` с атрибутом `id`, равным `5DayWeather`.
   ```
   <div id="5DayWeather"></div>
   ```
6. Код, добавляемый на панель JavaScript, начните с команды создания массива, хранящего названия пяти дней недели.
   ```
   var days = ["Monday","Tuesday","Wednesday","Thursday","Friday"];
   ```
7. Создайте еще один массив, `weather`.

Элементы этого массива содержат описание самых разных погодных условий. Постарайтесь наполнить массив всеми известными вам словами, услышанными от диктора прогноза погоды на англоязычном телеканале.

```
var weather = ["Sunny", "Partly Sunny", "Partly Cloudy",
   ↳"Cloudy", "Raining", "Snowing", "Thunderstorm", "Foggy"];
```

8. Объявите две переменные, `minTemp` и `maxTemp`, в которых хранятся минимальная и максимальная температуры, отображаемые в вашем невероятном прогнозе.

   Например, мы выбрали такие значения.

```
minTemp = 0;
maxTemp = 100;
```

9. Создайте функцию `generateWeather()`.

```
function generateWeather() {
```

10. Первая строка в теле функции содержит оператор `for`, позволяющий просматривать элементы (дни недели) массива `days`.

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

11. Объявите в функции новую переменную, `weatherToday`, хранящую произвольный элемент, извлеченный из массива `weather`.

```
var weatherToday = weather[Math.floor(Math.random() *
   ↳weather.length)];
```

12. Создайте переменную `tempToday`, представляющую произвольную температуру из диапазона, задаваемого переменными `minTemp` и `maxTemp`.

```
var tempToday = Math.floor(Math.random() * (maxTemp - minTemp) +
minTemp);
```

13. Используя свойство `innerHTML`, поместите значения переменных `weatherToday` и `tempToday` в элемент `div`, созданный нами ранее на панели HTML.

```
document.getElementById("5DayWeather").innerHTML += "<div id='"
   ↳+ days[i] + "' class='" + weatherToday + "'><b>Forecast for "
   ↳+ days[i] + ":</b><br><br>" + weatherToday + " and " +
   ↳ tempToday + " degrees.</div>";
```

   Обратите внимание на то, что в приведенном выше коде обращение к элементу дня недели осуществляется через идентификатор (атрибут `id`), а к элементу погодных условий — через атрибут `class`. В дальнейшем это позволит нам применить к указанным элементам форматирование, устанавливаемое правилами CSS.

14. Завершите функцию и цикл, введя две закрывающие фигурные скобки.

```
    }
}
```

15. Наконец, вызовите функцию `generateWeather()` сразу после объявления переменных и перед ее кодом.

На данный момент код, содержащийся на панели JavaScript, должен выглядеть, как в листинге 17.5.

Листинг 17.5. Полный JavaScript-код приложения, выдающего случайный прогноз погоды

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
var weather = ["Sunny", "Partly Sunny", "Partly Cloudy",
   ↳"Cloudy", "Raining", "Snowing", "Thunderstorm", "Foggy"];
var maxTemp = 100;
var minTemp = 0;

generateWeather();

function generateWeather() {
   for (var i = 0; i < days.length; i++) {
      var weatherToday = weather[Math.floor(Math.random() *
         ↳weather.length)];
      var tempToday = Math.floor(Math.random() * (maxTemp -
         ↳minTemp) + minTemp);

      document.getElementById("5DayWeather").innerHTML += "<div
         ↳id='" + days[i] + "' class='" + weatherToday + "'><b>
         ↳Forecast for " + days[i] + ":</b><br><br>" +
         ↳weatherToday + " and " + tempToday + " degrees.</div>";
   }
}
```

При запуске приведенной выше программы (после щелчка на кнопке Run в верхней части окна приложения JSFiddle) на панель результатов выводится прогноз погодных условий для всех пяти дней, названия которых хранятся в массиве `days`, как показано на рис. 17.4.

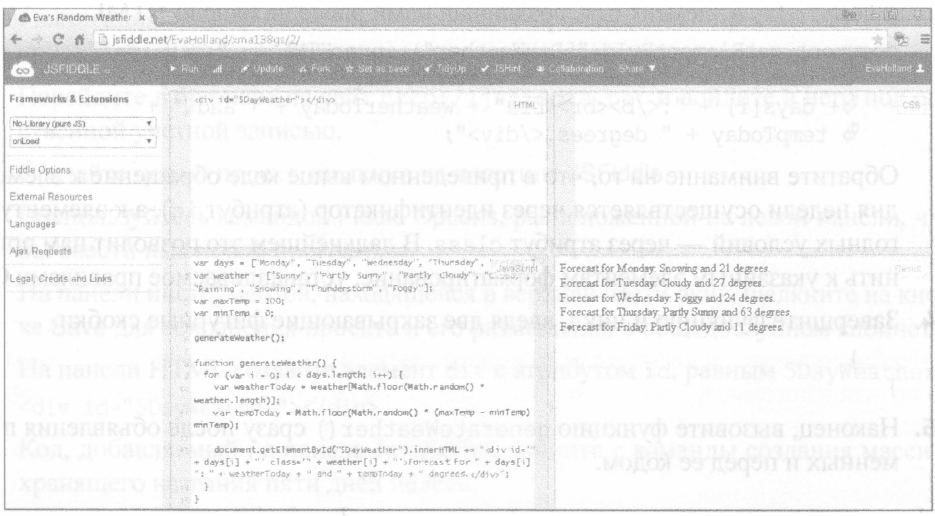


Рис. 17.4. Результат выполнения программы, код которой приведен в листинге 17.5

Анализ результатов

Получив от своего приложения прогноз погоды, вы заметите, что данные выводятся в формате, малопригодном для изучения. К счастью, мы предусмотрительно снабдили каждый выводящийся на веб-странице элемент `div` атрибутами `id` и `class`.

Давайте рассмотрим содержимое панели **Result** и попробуем разобраться в том, какими элементами HTML и атрибутами JavaScript оно представляется.

1. В верхней части окна JSFiddle щелкните на команде **Update** или **Run**.

   На панели результатов отобразится произвольный прогноз погоды.

2. Выполните команду **Chrome⇨Дополнительные инструменты⇨Инструменты разработчика**.

   В нижнюю часть окна браузера добавляется панель разработчика.

3. На панели разработчика перейдите на вкладку **Elements** (Элементы).

   Панель элементов показана на рис. 17.5.

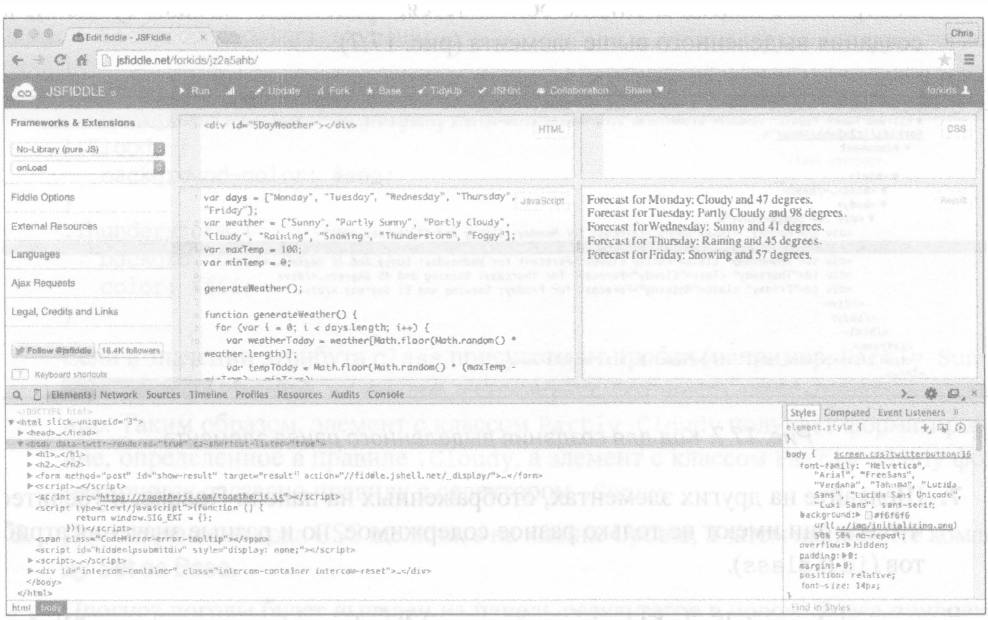


*Рис. 17.5. Вкладка **Elements** панели разработчика браузера Chrome*

4. Выберите инструмент **Inspector** (Просмотр), представленный значком с изображением увеличительного стекла (расположен в левом верхнем углу панели разработчика).

5. Переместите указатель мыши на панель результатов приложения JSFiddle.

   Элементы HTML, на которые наводится указатель мыши, выделяются, как показано на рис. 17.6.

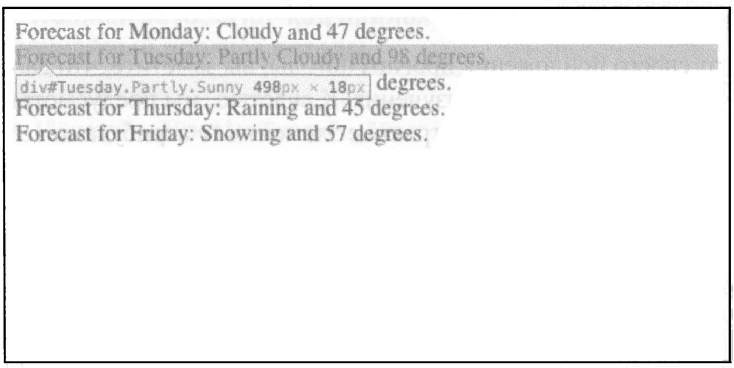


Рис. 17.6. Выделение элементов HTML на панели результатов

6. Наведя мышь на одну из строк с прогнозом погоды (для одного из дней рабочей недели), щелкните на нем.

   Содержимое панели **Elements** обновляется, представляя на ваше суждение код создания выделенного выше элемента (рис. 17.7).

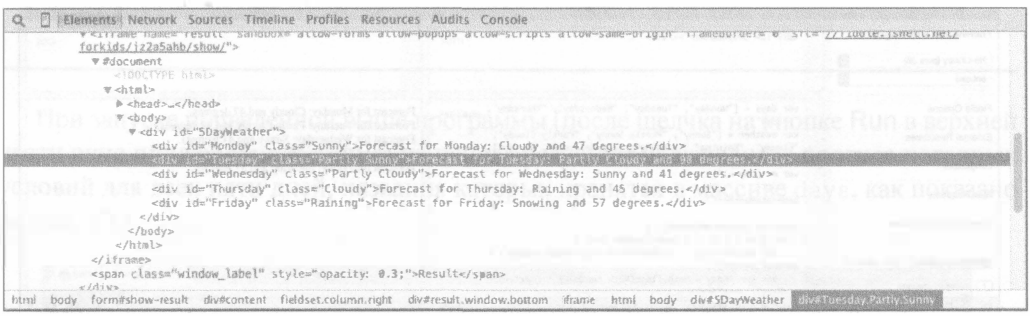


Рис. 17.7. Код для создания выделенного ранее элемента

7. Щелкайте на других элементах, отображенных на панели **Results**, и убедитесь, что все они имеют не только разное содержимое, но и разные значения атрибутов (`id` и `class`).

В последнем разделе вы узнаете о том, как атрибуты `id` и `class` применяются для форматирования результата, выводимого для пользователя.

Стилизация выводимого результата

Заранее назначив каждому выводимому на экран элементу атрибуты `id` и `class`, можно добиться специального форматирования прогноза для каждого дня согласно ожидаемой погоде.

Стилизация выводимого на экран прогноза погоды выполняется следующим образом.

1. Создайте на панели CSS общее правило форматирования, справедливое для всех дней недели.

```
#Monday, #Tuesday, #Wednesday, #Thursday, #Friday {
   width: 18%;
   height: 200px;
   float: left;
   border: 1px solid black;
   padding: 2px;
   font-family: sans-serif;
   font-size: 12px;
}
```

   В этом правиле настраиваются ширина и высота элемента, его граница, отступы, шрифт и его размер. Свойство `float`, установленное равным `left`, указывает располагать результаты прогноза погоды для отдельных дней рядом друг с другом, а не один под другим.

2. Создайте правила форматирования для отдельных погодных условий.

```
.Sunny {
   background-color: skyblue;
}
.Raining {
   background-color: lightgrey;
}.Cloudy {
   background-color: #eee;
}
.Thunderstorm {
   background-color: #333;
   color: #fff;
}
```

   Если в значении атрибута `class` присутствует пробел (например, `Partly Sunny` или `Partly Cloudy`), то каждое из слов рассматривается как отдельное значение. Таким образом, элемент с классом `Partly Cloudy` получает форматирование, определенное в правиле `.Cloudy`, а элемент с классом `Partly Sunny` форматируется согласно правилу с селектором `.Sunny`.

3. Для сохранения кода CSS щелкните на кнопке **Update**, а затем выполните команду **Set as Base**.

   Прогноз погоды будет выведен на панель результатов в новом, более привлекательном виде, показанном на рис. 17.8.

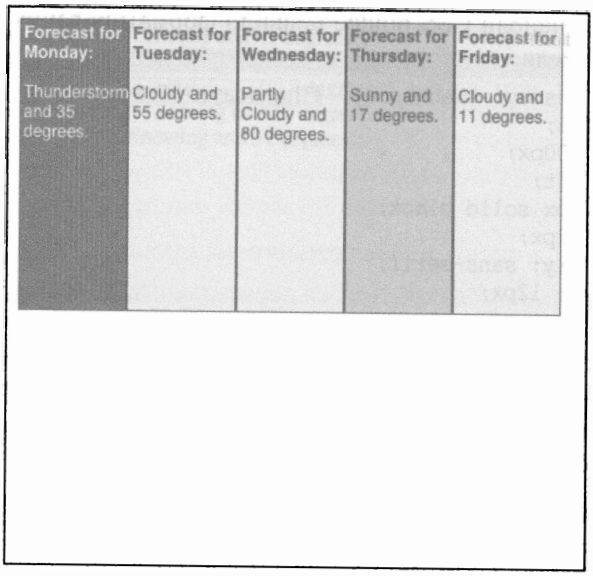


Рис. 17.8. Конечный результат, возвращаемый приложением прогноза погоды

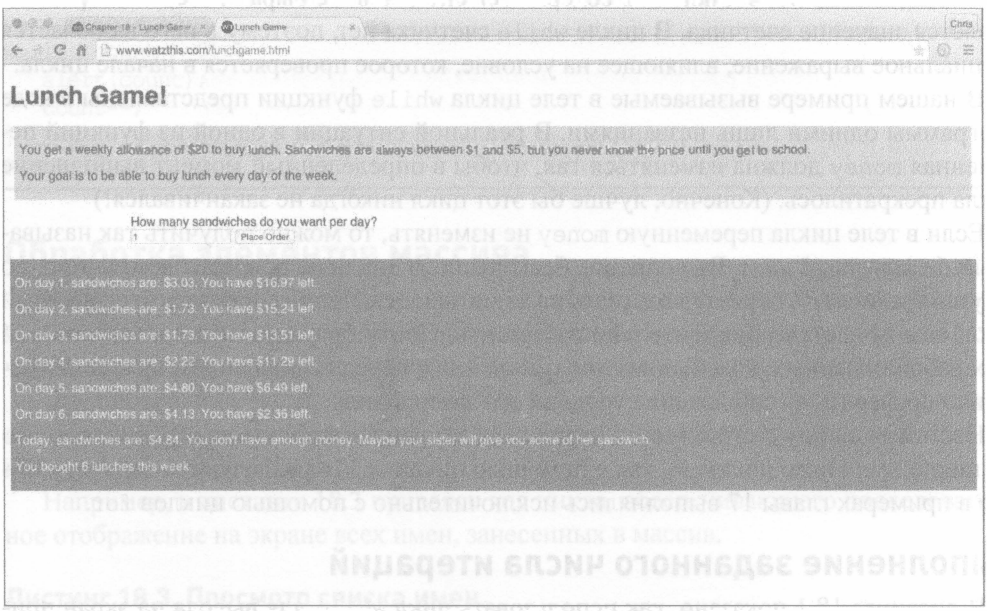
ГЛАВА

18

Цикл While

В цикле `while` инструкции продолжают выполняться до тех пор, пока справедливо определенное условие. Если условие справедливо всегда, то цикл выполняется бесконечно!

В этой главе мы воспользуемся циклом `while` для написания игры, в которой сэндвичи можно покупать до тех пор, пока не закончатся деньги. Цель игры заключается в экономном расходовании наличных средств так, чтобы их хватило до конца недели.



Структура цикла while

Структурно цикл while намного проще уже известного вам цикла for. Кроме ключевого слова while, в его первой строке вводится всего одно булево выражение, определяющее, будут ли выполняться инструкции, указанные в теле цикла.

Пример цикла while приведен ниже.

```
while (money > 0) {
   buyThings();
   saveMoney();
   payTaxes();
}
```

В этом цикле вызываются три функции: buyThings(), saveMoney() и payTaxes(). Это действие продолжается до тех пор, пока значение переменной money не станет меньше нуля.

Как вам известно, в цикле for содержится специальное выражение, в котором изменяется значение счетчика. В цикле while счетчика нет, поэтому в него включается специальное выражение, влияющее на условие, которое проверяется в начале цикла.

В нашем примере вызываемые в теле цикла while функции представлены в коде программы одними лишь названиями. В реальной ситуации в одной из функций переменная money должна изменяться так, чтобы в определенный момент выполнение цикла прекратилось. (Конечно, лучше бы этот цикл никогда не заканчивался!)

Если в теле цикла переменную money не изменять, то можно получить так называемый *бесконечный цикл*. Выполнение бесконечного цикла не повредит компьютер, но браузер "зависнет", перестав отвечать на ваши запросы. Чтобы прервать бесконечный цикл, вам придется принудительно завершить работу браузера, что грозит потерей всех рабочих данных. Именно поэтому цикл while перед выполнением нужно внимательно проверять на соблюдение условий его завершения.

Несмотря на иную структуру, в цикле while можно выполнять те же действия, что и в цикле for. Ниже показано, как с помощью цикла while реализуются задачи, которые в примерах главы 17 выполнялись исключительно с помощью циклов for.

Выполнение заданного числа итераций

В листинге 18.1 показано, как использовать цикл while для вывода на экран приветствия Привет, JavaScript! 500 раз.

Листинг 18.1. Вывод сообщения Привет, JavaScript!

```
var i = 0;
while (i < 500) {
   console.log(i + ": Привет, JavaScript!");
   i++;
}
```

Легко заметить, что программа из листинга 18.1 состоит из тех же трех составных частей, которые использовались в цикле `for` (инициализация, условие и операции), только в нашем случае условие вводится в скобках после ключевого слова `while`. Инициализация (`var i = 0;`) проводится перед самим циклом, а выражение изменения счетчика (`i++`) — в конце тела цикла.

Обратный отсчет

Чтобы создать цикл, умеющий считать в обратном порядке, нужно в каждой итерации (в теле цикла) изменять значение одной из переменных, которая задействуется при проверке условия.

В листинге 18.2 приведен простой пример цикла `while`, проводящего обратный отсчет. Сравните его с аналогичным кодом, описанным в главе 17.

Листинг 18.2. Обратный отсчет с помощью цикла `while`

```
var count = 10;
while (count > 0) {
   alert(count);
   count--;
}
alert("Пуск!");
```

Обработка элементов массива

Обработка элементов массива в цикле `while` реализуется намного проще, чем в цикле `for`. Чтобы выполнить эту задачу, условие в цикле нужно изменять таким образом, чтобы в начале каждой итерации проверялось наличие следующего элемента массива.

Для проверки существования элемента массива необходимо ввести его название после ключевого слова `while` (в круглых скобках), указав в квадратных скобках переменную счетчика.

Например, в листинге 18.3 приведен код цикла, обеспечивающего последовательное отображение на экране всех имен, занесенных в массив.

Листинг 18.3. Просмотр списка имен

```
var people = ["Deborah","Carla","Mary","Suzen"];
var i = 0;
   while (people[i]) {
   alert(people[i]);
i++;
}
```

Условия, вводимые в скобках как в цикле `for`, так и в цикле `while`, представляются булевыми выражениями, возвращающими одно из двух возможных значений: `true` или `false`. Обращаясь к отдельному элементу массива, например к `people[5]`, вы получите ответ `true`, если указанный элемент существует.

Игра на выживание

Рассмотренное далее приложение Lunch Game совмещает в себе случайность и строгий математический расчет. Цель игры состоит в том, чтобы правильно распорядиться денежными средствами, обеспечив себя сэндвичами на каждый день.

В чем же загвоздка? В дополнительных условиях! Во-первых, вы покупаете сэндвичи в очень странном заведении: их стоимость становится известной только после приготовления. Во-вторых, сэндвичи нужно выкупить наперед на всю неделю.

Вам известно, что цена на сэндвичи колеблется в диапазоне от 1 до 5 долларов. Поэтому, если повезет, вы сможете сразу приобрести от четырех до двадцати сэндвичей.

Готовы ли вы рискнуть и насколько серьезно? Придет ли вам кто-нибудь на помощь, если у вас закончатся наличные средства до конца недели? Сколько сэндвичей вы можете позволить себе купить на неделю?

В процессе игры вам придется ответить на эти и некоторые другие вопросы.

Код игры

Чтобы приступить к рассмотрению приложения Lunch Game, выполните следующие действия.

1. Откройте в браузере общедоступный кабинет JSFiddle по такому адресу:

   `http://jsfiddle.net/user/forkids/fiddles`

   На экране отобразится список доступных для изучения приложений.

2. Найдите заголовок Chapter 18 — Lunch Game – Start и щелкните на нем.

   Начальная версия программы показана на рис. 18.1.

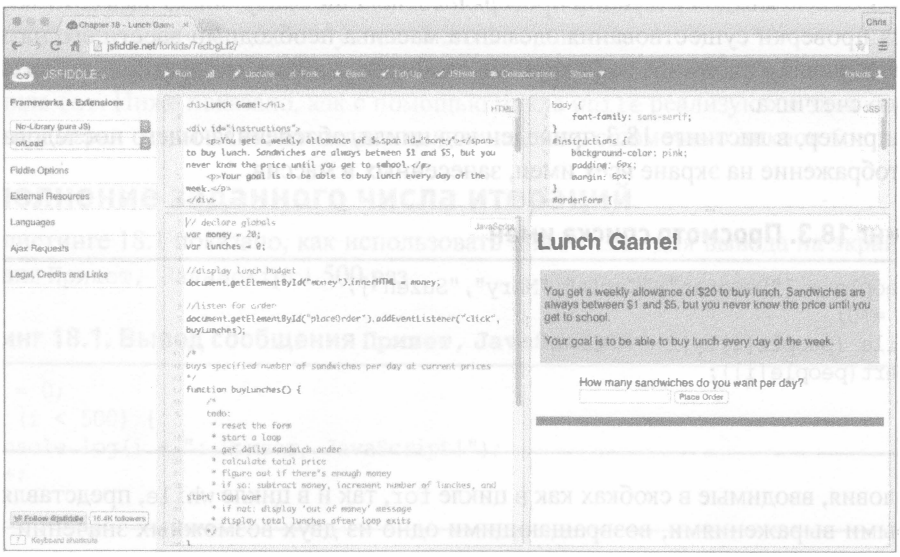


Рис. 18.1. Программа, запускающая игру Lunch Game

Как видите, весь код HTML, CSS и частично JavaScript уже создан. Нам осталось написать только код функции `buyLunches()`. Этим мы и займемся в следующем разделе.

Функция `buyLunches()`

Исходный код функции `buyLunches()`, содержащий комментарии по наполнению ее командами, приведен в листинге 18.4.

Листинг 18.4. Начальный код функции `buyLunches()`

```
/*
Приобрести определенное количество сэндвичей на день по текущим ценам
*/
function buyLunches() {
/*
todo:
* сброс формы
* запуск цикла
* получение заказа на день
* вычисление общей суммы
* определение доступности наличных денежных средств
* денег достаточно: вычитание необходимой суммы, увеличение
    количества сэндвичей и запуск следующей итерации
* денег недостаточно: вывод сообщения о нехватке денег
* вывод количества дней, для которых приобретены сэндвичи
*/
}
```

Чтобы правильно написать код тела функции, следуйте приведенным ниже инструкциям.

1. После заголовка функции `buyLunches()` введите строку, в которой вызывается функция `resetForm()`, а затем объявите переменную, которая применяется для указания текущего дня недели.

   ```
   resetForm();
   var day = 0;
   ```

2. Создайте цикл, отвечающий за покупку сэндвичей до тех пор, пока не закончатся деньги.

   ```
   while (money > 0) {
   ```

3. Получите текущую цену на сэндвичи, вызвав функцию `getSandwichPrice()`, а возвращаемый ею результат сохраните в переменной `priceToday`.

   ```
   var priceToday = getSandwichPrice();
   ```

   Всесторонне изучите действия, выполняемые функцией `getSandwichPrice()`. Ее назначение — сгенерировать произвольное число в диапазоне от 1 до 5 и возвратить его в качестве результата.

4. Получите количество сэндвичей, которые пользователь указал в поле ввода данных формы.

```
var numberOfSandwiches = document
   ↳.getElementById("numSandwiches").value;
```

5. Вычислите общую сумму, которую придется заплатить за сэндвичи, умножив их количество на цену одного сэндвича.

```
var totalPrice = priceToday * numberOfSandwiches;
```

6. Определите, достаточно ли у вас денег для приобретения указанного количества сэндвичей.

```
if(money >= totalPrice) {
```

7. Если денег достаточно, то вычтите общую сумму сделки из имеющихся у вас наличных средств.

```
money = money - totalPrice;
```

Замечательно! Вы только что обеспечили себя пропитанием.

8. Увеличьте значение переменной `lunches`, указывающей количество дней, на которые вы приобрели сэндвичи.

```
lunches++;
```

9. Выведите на экран сообщение, в котором укажите стоимость сэндвичей и остаток на вашем счету после их покупки.

```
document.getElementById("receipt").innerHTML += "<p>On day " +
   ↳day + ", sandwiches are: $" + priceToday + ". You have $" +
   ↳money.toFixed(2) + " left.</p>";
```

Обратите внимание на то, что к переменной `money` применяется метод `toFixed()`. Он преобразует число в строковый тип данных, сохраняя в нем указанное в скобках количество десятичных разрядов. В нашем случае речь идет о деньгах, поэтому при конвертации в числе сохраняются только две цифры после запятой.

10. Теперь введите команды в блоке `else` условной конструкции, обыграв ситуацию, в которой у пользователя недостаточно денег для приобретения сэндвичей.

```
} else {
```

11. В выводимом в подобном случае сообщении попробуйте утешить пользователя, предложив ему одолжить несколько сэндвичей у близкого человека.

```
document.getElementById("receipt").innerHTML += "<p>Today,
   ↳sandwiches are: $" + priceToday + ". You don't have enough
   ↳money. Maybe your sister will give you some of her sandwich.</p>";
```

12. Здесь же необходимо установить для переменной `money` значение 0, чтобы предотвратить дальнейшее выполнение цикла.

```
money = 0;
```

13. Завершите условную конструкцию `if...else` и цикл `while`, введя две закрывающие фигурные скобки.

```
   }
}
```

14. Завершив цикл, добавьте в функцию код вывода сообщения, в котором указывается, на сколько дней вам хватит приобретенных сэндвичей.

```
document.getElementById("reciept").innerHTML += "<p>You bought " +
    ↳lunches + " lunches this week.</p>";
```

15. Завершите функцию закрывающей фигурной скобкой.

```
}
```

16. В верхней части окна приложения JSFiddle щелкните на кнопке **Update** и выполните команду **Set as Base** для сохранения введенного выше кода функции `buyLunches()`.

Тестирование игры

После запуска полностью завершенной программы вы увидите на экране форму ввода данных, показанную на рис. 18.2.

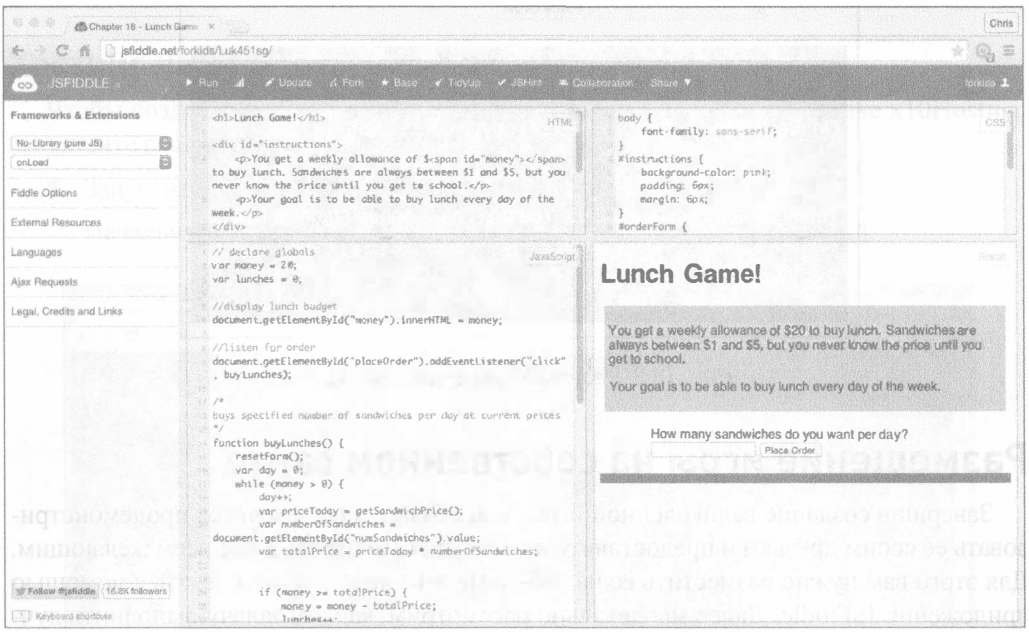


Рис. 18.2. Игра Lunch Game в конечном виде

После указания количества сэндвичей, которые пользователь съедает за день, в поле формы и щелчка на кнопке **Place Order** (Заказать) программа рассчитает, на сколько дней пользователю хватит наличных средств для обеспечения себя пропитанием. ***Важно знать:*** цена одного сэндвича генерируется произвольным образом, а за день пользователь потребляет строго указанное им количество сэндвичей.

Запустите программу несколько раз, каждый раз вводя в поле формы другие значения. Действительно, цена сэндвича формируется случайным образом, и поэтому количество дней, на которые хватает денег, невозможно предугадать.

На рис. 18.3 показан один из возможных вариантов завершения игры Lunch Game.

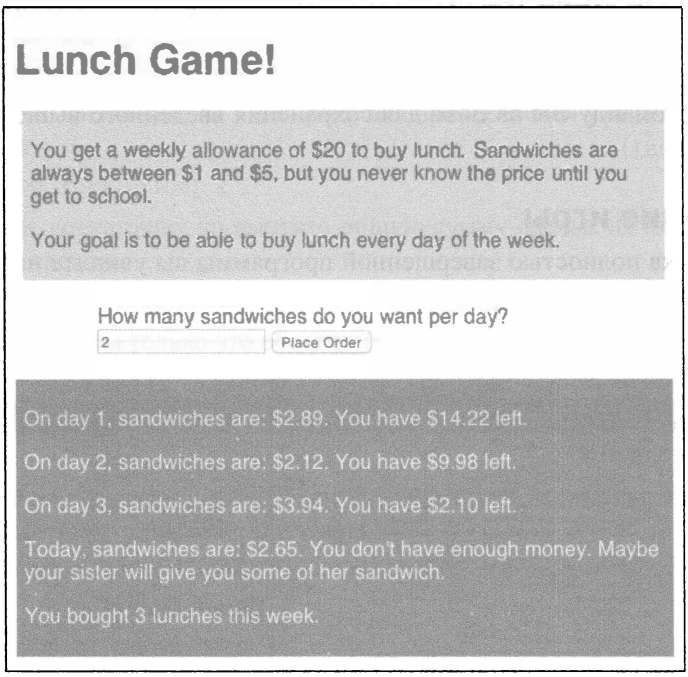


Рис. 18.3. Результат завершения игры

Размещение игры на собственном сайте

Завершив создание великолепной игры, вам обязательно захочется продемонстрировать ее своим друзьям и предоставить возможность поиграть в нее всем желающим. Для этого вам нужно разместить ее на веб-сайте, что невозможно сделать с помощью приложения JSFiddle. Далее мы детально рассмотрим, как эту задачу выполнить лучше всего.

Веб-хостинг

Каждый сайт в Интернете имеет уникальный адрес, по которому пользователи его посещают. Чтобы получить адрес для собственного веб-сайта, вам нужно заключить договор со специальной компанией, занимающейся хостингом. К счастью, JSFiddle — это не только название приложения, но и компания, занимающаяся бесплатным размещением приложений, написанных начинающими разработчиками на языках HTML, CSS и JavaScript.

Разумеется, согласившись на бесплатный хостинг, вы автоматически разрешаете другим посетителям сайта JSFiddle.net копировать и модифицировать ваш код по своему усмотрению. К тому же вам не предоставляется отдельное доменное имя (подобное `www.mywebsite.com`).

Сторонние компании, предоставляющие услуги хостинга, обычно требуют ежемесячную плату за размещение вашего сайта в Интернете, хотя у них и бывают бесплатные предложения с весьма строгими ограничениями на публикуемые данные. Далее мы расскажем о том, как получить временный хостинг на сайте x10Hosting (`www.x10hosting.com`).

К тому моменту, когда вы будете читать книгу, компания x10Hosting может изменить свою политику и начать взимать плату за временный хостинг сайтов. Не переживайте, процедура заказа хостинга у других провайдеров мало чем отличается от приведенной ниже. Просто проведите поиск в Интернете по фразе `веб-хостинг`, чтобы найти выгодное для вас предложение.

Хостинг на сайте x10Hosting

Чтобы создать собственную учетную запись и веб-сайт на площадке x10Hosting, выполните следующие действия.

1. Запустите браузер и перейдите по адресу `www.x10hosting.com`.

   Начальная страница сайта выглядит так, как показано на рис. 18.4.

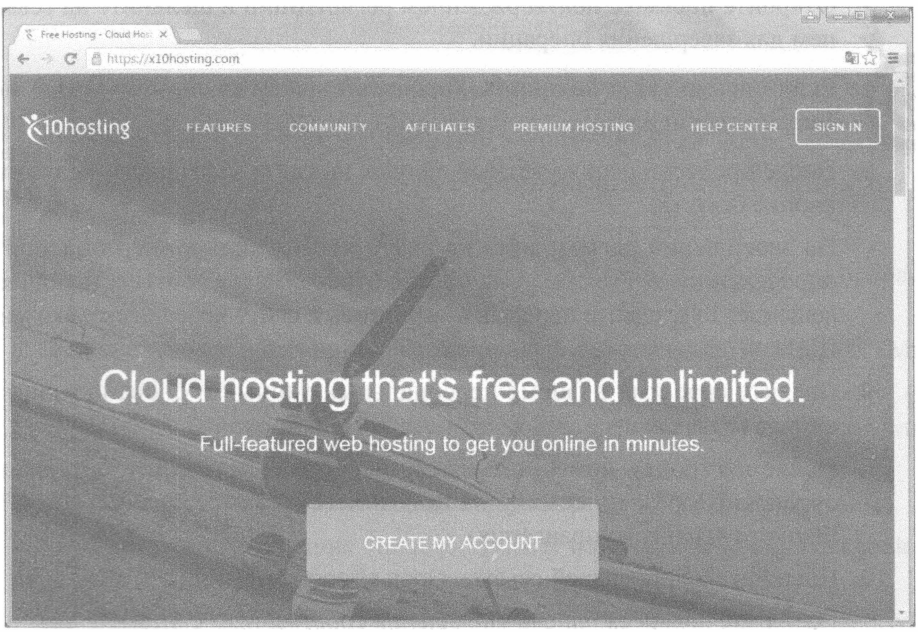


Рис. 18.4. Домашняя страница компании x10Hosting

2. Щелкните на кнопке Create My Account (Создать учетную запись).

   На экране появится форма, в которой вводятся настройки вашей учетной записи.

3. Введите имя своей учетной записи, как показано на рис. 18.5, и щелкните на кнопке Continue (Продолжить).

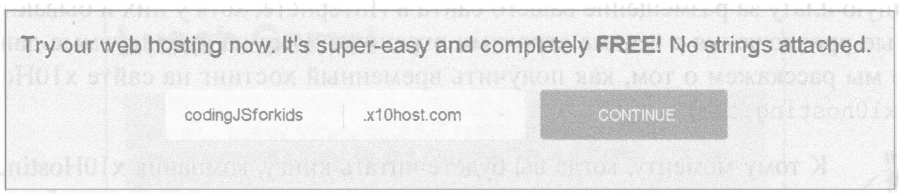


Рис.18.5. Создание имени для собственной учетной записи

4. На следующей странице введите адрес своего почтового ящика и снова щелкните на кнопке Continue.
5. Придумайте и введите пароль для учетной записи. Щелкните на кнопке Continue.
6. Примите условия соглашения и щелкните на кнопке Submit (Подтвердить), чтобы завершить регистрацию.

   На вашу электронную почту будет отправлено письмо с подтверждением регистрации учетной записи.

7. Откройте письмо с подтверждением регистрации и щелкните на ссылке в нем для завершения операции.

   Если письмо так и не пришло, проверьте папку со спамом — оно могло затеряться в ней.

8. После проверки своих учетных данных на сайте x10Hosting щелкните на кнопке Continue.

   На завершение регистрации на сайте x10Hosting может понадобиться определенное время. Если на экране отображается соответствующее извещение, то сделайте перерыв и вернитесь к сайту через несколько минут. Щелкните на кнопке Continue, как только она станет доступной.

9. Введите свое имя в форме персональных данных и снова щелкните на кнопке Continue.

   После настройки доменного имени вы увидите страницу, содержащую справочный раздел, ссылку на ваш домен, а также ссылку Open cPanel (Запустить панель управления).

10. Щелкните на последней ссылке, Open cPanel.

    На экране появится панель управления собственным сайтом.

11. Щелкните на ссылке Add Website (Добавить веб-сайт).

12. Присвойте новому сайту имя, оставьте домен без изменений, а поле Address Path оставьте незаполненным, как показано на рис. 18.6.

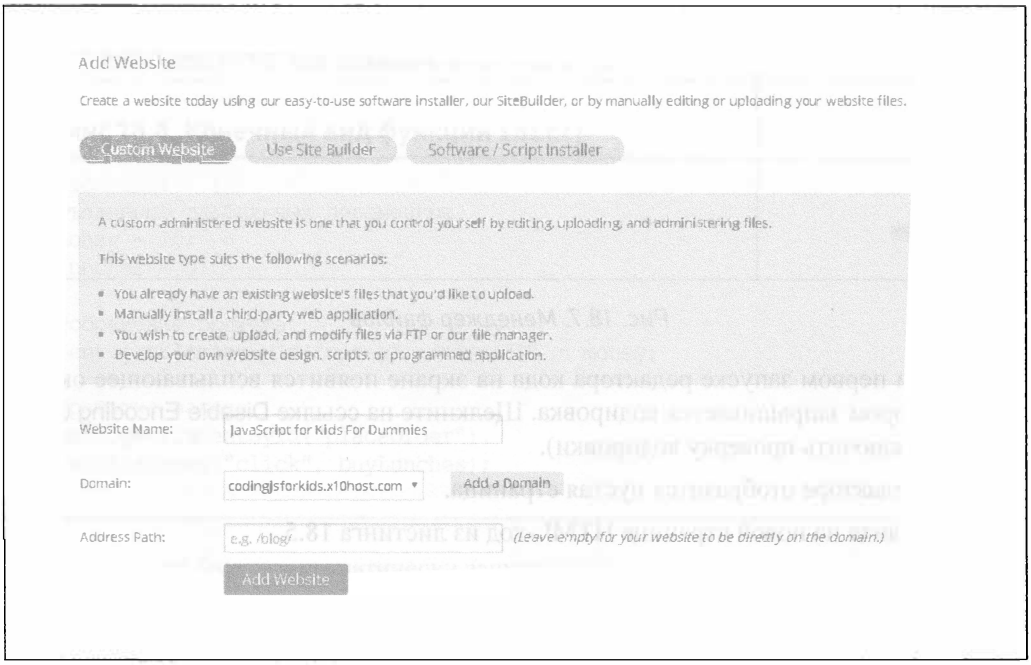


Рис. 18.6. Создание веб-сайта

13. В нижней части страницы щелкните на кнопке Add Website.

Вы только что создали новый сайт с уникальным именем.

Запомните или запишите адрес сайта. Вы будете использовать его в дальнейшем.

14. Щелкните на команде Continue to My Websites (перейти к управлению веб-сайтами).

15. На странице управления сайтами щелкните на ссылке File Manager (Менеджер файлов).

    На экране появится окно, отображающее файловую структуру хранилища для вашей учетной записи (рис. 18.7).

16. В верхней части окна щелкните на кнопке New File (Создать файл).

17. Дайте файлу название `lunchgame.html` и щелкните на кнопке Create New File (Создать файл).

18. Выделите только что созданный файл, щелкнув на нем, и воспользуйтесь кнопкой Code Editor (Редактор кода), расположенной в верхней части окна.

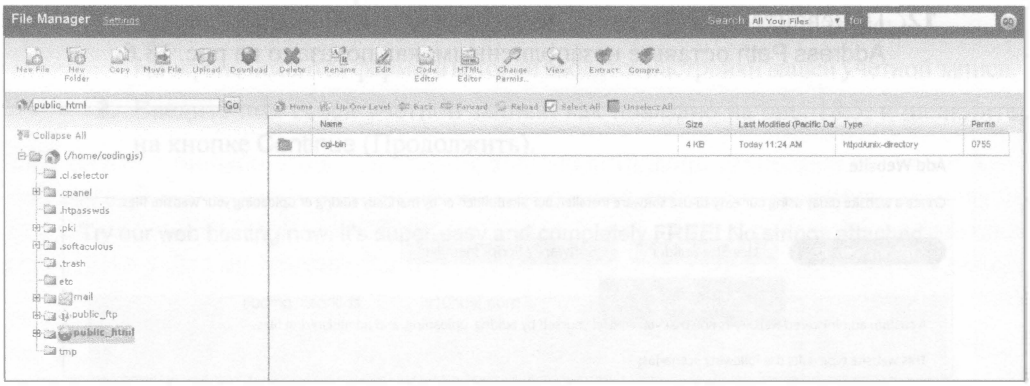


Рис. 18.7. Менеджер файлов

19. При первом запуске редактора кода на экране появится всплывающее окно, в котором запрашивается кодировка. Щелкните на ссылке **Disable Encoding Check** (Отключить проверку кодировки).

В редакторе отобразится пустая страница.

20. Введите на новой странице HTML-код из листинга 18.5.

Листинг 18.5. Стандартный шаблон HTML

```
<!doctype html>
<html>
<head>
   <title>Lunch Game</title>
<style>
</style>
<script>
function init() {

}
</script>

</head>
<body onload="init();">

</body>
</html>
```

21. На новой вкладке браузера откройте сайт JSFiddle.net и загрузите программу Lunch Game.

22. Скопируйте содержимое панели HTML и вставьте его на странице редактора кода (файл `lunchgame.html`) между открывающим и закрывающим тегами элемента `body`.

23. В приложении JSFiddle скопируйте содержимое панели CSS и вставьте его между тегами <style> и </style> файла lunchgame.html.
24. В приложении JSFiddle скопируйте первые строки кода JavaScript (до объявления функции buyLunches()) и вставьте его в тело функции init() файла lunchgame.html, как показано в листинге 18.6.

Листинг 18.6. Конечный вид функции init()

```
function init() {
// Объявление глобальных переменных
var money = 20;
var lunches = 0;

// Отображение бюджета
document.getElementById("money").innerHTML = money;

// Обработка заказа
document.getElementById("placeOrder").
addEventListener("click", buyLunches);
}
```

Функция init() автоматически запустится на этапе загрузки веб-страницы.

25. Вставьте остальное содержимое панели JavaScript (из приложения JSFiddle) непосредственно под кодом функции init(), но все еще между тегами <script> и </script>.
26. Щелкните в левом верхнем углу окна на кнопке **Save** (Сохранить).
27. Щелкните на кнопке **Close** (Закрыть), расположенной слева от кнопки **Save**.
28. Если на экране появляется сообщение, в котором упоминается кодировка (encoding), то щелкните в нем на кнопке **OK**, чтобы вернуться к менеджеру файлов.
29. На новой вкладке браузера перейдите к собственному сайту, введя его адрес.

    На странице отобразится список файлов сайта. В текущий момент времени в вашем распоряжении находятся всего одна папка (cgi-bin) и единственный файл (lunchgame.html).

    Если вам не нравится каждый раз наблюдать такой список при переходе к собственному сайту, то создайте новый файл index.html, и при следующем посещении он будет отображаться вместо списка.
30. Чтобы запустить игру Lunch Game, щелкните на файле lunchgame.html.
31. Игра Lunch Game запускается в окне браузера, как показано на рис. 18.8.

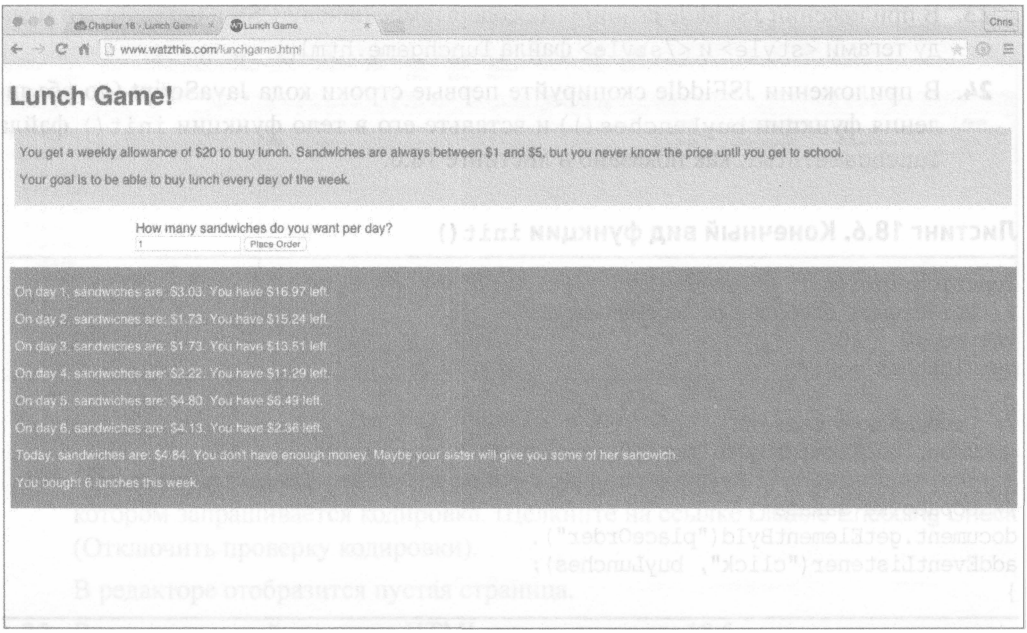


Рис. 18.8. Игра Lunch Game, запущенная с собственного веб-сайта

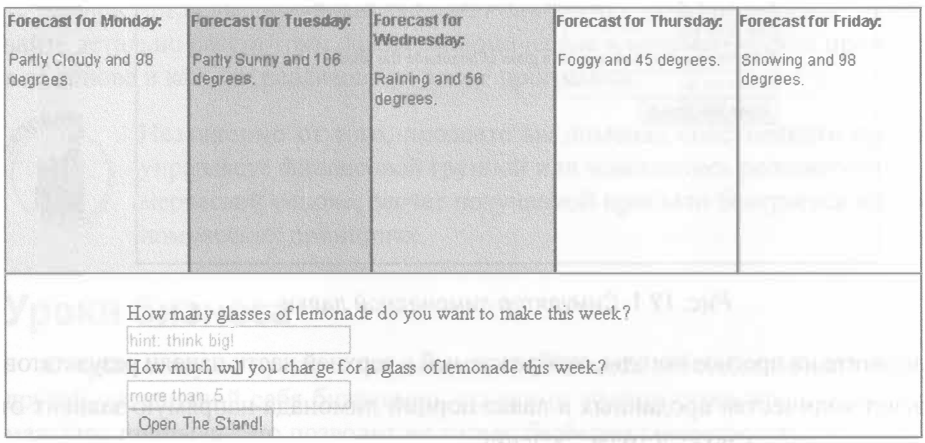
ГЛАВА 19

Лавка по продаже лимонада

Когда мы работали над приложениями прогноза погоды и игры Lunch Game, нас посетила безумная идея: совместить их в одной программе, предназначенной для обслуживания торговой точки по продаже лимонада.

Как и в случае прогноза погоды для вымышленного города, ваше предприятие должно выгодно отличаться от остальных лимонадных лавок в городе. Давайте проанализируем главную идею, лежащую в основе выбранной стратегии продвижения товара. Лучше всего лимонад продается в жаркую погоду. Соответственно, самую большую цену на лимонад нужно выставлять в дни с повышенной температурой воздуха. Удачно прогнозируя количество и цену лимонада в зависимости от погодных условий, вы увеличите прибыль и минимизируете потери в виде нереализованного лимонада.

В этой главе мы займемся созданием приложения, помогающего продавать лимонад всем жаждущим.



Продажа лимонада как игра

Уже традиционно перед тем, как приступить к изучению и написанию кода приложения, давайте посмотрим, как оно работает в готовом виде.

Чтобы сыграть в симулятор лимонадной лавки, выполите следующие действия.

1. Запустите браузер, загрузите приложение JSFiddle и перейдите в общедоступный кабинет.

   `http://jsfiddle.net/user/forkids/fiddles`

2. Найдите в списке программ название **Chapter 19 — Lemonade Stand** и щелкните на нем.

   Игра запустится и будет ожидать ваших действий, как показано на рис. 19.1.

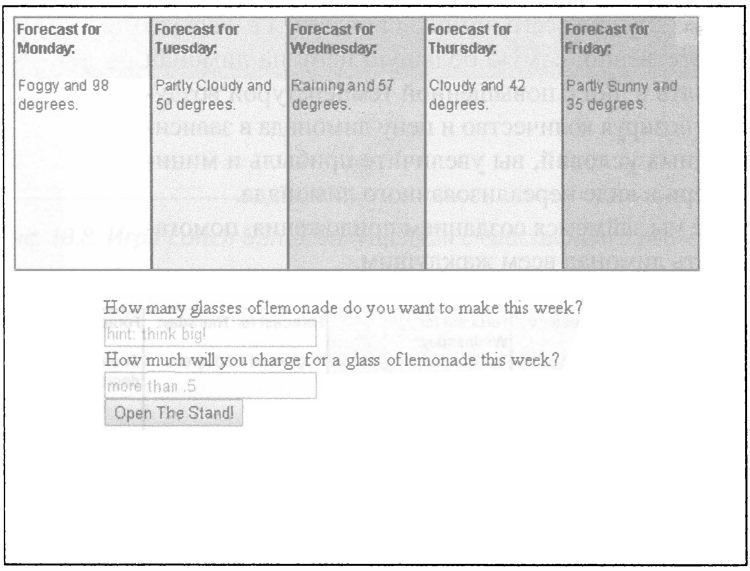


Рис. 19.1. Симулятор лимонадной лавки

3. Взгляните на прогноз погоды, отображаемый в верхней части панели результатов.

   Расчет количества проданных в лавке порций лимонада напрямую зависит от указанных в прогнозе погоды сведений.

4. Введите в поле **How many glasses of lemonade do you want to make this week?** (Сколько порций лимонада вы ходите продать за неделю?) некое число, указывающее количество порций лимонада, которое нужно реализовать за рабочую неделю.

   Помните, что вы торгуете целых пять дней, и указанного вами количества должно хватить до пятницы включительно. ***Совет:*** попробуйте ввести самое разное количество порций, включая несколько сотен.

5. Введите цену, которая запрашивается за порцию лимонада.

   Входящая цена (стоимость для вас) порции лимонада составляет полдоллара (`0.50`), поэтому указывайте число, заведомо большее этого значения.

6. Щелкните на кнопке **Open The Stand** (Начать торговлю).

   На экране появится отчет о дневных и недельных продажах лимонада в лавке. В последней строке отчета указывается полученная вами прибыль. Следите за тем, чтобы в ней выводилось значение, большее нуля. Отрицательные значения указывают на то, что вы работаете в убыток.

7. Измените цену порции лимонада, основываясь на полученных ранее результатах. Снова щелкните на кнопке **Open The Stand**.

   Постарайтесь отследить зависимость получаемой прибыли от цены на лимонад. Ваша задача — выбрать стратегию, при которой вы получаете максимальную прибыть и минимальный остаток лимонада на конец торговой недели.

8. В верхней части приложения JSFiddle щелкните на кнопке **Run**, чтобы перезапустить рабочую неделю с новыми погодными условиями.

9. Определите разницу в объемах продаж лимонада в дни с разной температурой на улице.

   Убедитесь в том, что между температурой на улице и количеством проданных порций существует прямая зависимость.

Теперь, когда вы знаете, как работает симулятор лавки по продаже лимонада, давайте детально рассмотрим, какие экономические и математические принципы лежат в ее основе и как они реализованы в коде программы.

Независимо от того, продаете вы лимонад собственного производства, управляете финансовой группой или занимаетесь репетиторством на коммерческой основе, расчет получаемой прибыли базируется на общих экономических принципах.

Уроки бизнеса

Открыв лавку по продаже лимонада, вы начинаете собственное дело. Как и любой другой уважающий себя бизнесмен, вы всеми силами стремитесь получить максимальную прибыль. Это позволит не только безбедно существовать, но и со временем увеличить объемы продаж.

Конечно, продавая лимонад, вы можете преследовать и другие цели, например обеспечить себя общением на всю оставшуюся жизнь, постоянно быть в курсе всех городских событий или узнать о лимонаде больше, чем известно производителю. Но если вы не будете получать прибыль, то ни одна из второстепенных целей точно не будет достигнута. Такой бизнес никому не нужен!

Для того чтобы увеличить прибыль, нужно понять, почему клиенты идут за лимонадом именно к вам. Так же, как вы имеете множество причин начать собственное дело, ваши покупатели чем-то руководствуются при выборе именно вашей торговой точки. На принятое ими решение влияет огромное количество факторов — погода, цена порции лимонада, количество денег в кошельке, расположение лавки и вкус напитка. Это только основные, но далеко не все критерии оценки. Как видите, такое простое дело, как продажа напитков, при детальном рассмотрении оборачивается большими трудностями.

Поскольку мы создаем скорее игру, а не бизнес-приложение, давайте сконцентрируемся в нашем проекте на нескольких факторах, влияющих на объем продаж лимонада.

Прибыль — основа всего

Прибыль — это то, что остается у вас после вычитания из полученной выручки (наторгованных за неделю денег) всех расходов (все, на что вы тратили деньги для обеспечения продаж товара).

Владелец лавки по продаже лимонада несет следующие затраты: покупка компонентов лимонада (сахар, лимон, лед, тара) и обслуживание торговой точки (коммунальные услуги, ремонт помещения и оборудования). Проведя нужные вычисления, мы определили, что в расчете на одну порцию лимонада затраты составляют полдоллара. Другими словами, чтобы получить прибыль с продаж лимонада в своей лавке, нужно выставлять цену порции не ниже 0,5 доллара.

Психология покупателя

Погода изменчива! Чем сильнее припекает на улице, тем длиннее очереди за лимонадом. Но если стоимость лимонада будет слишком высокой, то вы не продадите ни одной порции даже в самый знойный день.

Ваша задача как владельца торговой точки — определить, сколько нужно произвести лимонада и по какой цене его реализовать, чтобы получить максимальную прибыль.

Математические вычисления

Для подсчета количества проданных за день порций лимонада в нашем приложении используется следующая формула.

```
Продано порций = Температура на улице / Цена лимонада
```

Так, если температура на улице — 100 градусов (по Фаренгейту), а стоимость порции лимонада составляет 2 доллара, то согласно этой формуле мы получим следующее.

```
Продано порций = 100 / 2
```

Вы продадите целых 50 порций.

А вот если температура упадет до 50 градусов, то при той же цене лимонада расчетная формула будет выглядеть так.

```
Продано порций = 50 / 2
```

Несложно подсчитать, что вы продадите всего 25 порций лимонада.

Тем не менее, если сбросить цену до одного доллара, то формула примет следующий вид.

```
Продано порций = 50 / 1
```

Вы продадите все те же 50 порций, что и при жаркой погоде. Снижая цену, вы увеличите объем продаж до прежнего уровня.

Связь между ценой, температурой и объемом продаж

Чтобы осилить игру, вам нужно отследить связь между объемом продаж, температурой воздуха и ценой на товар. Для представления этой зависимости в виде графика выполните такие действия.

1. Запустите браузер и перейдите на сайт `www.wolframalpha.com`.

   Домашняя страница этого сайта показана на рис. 19.2.

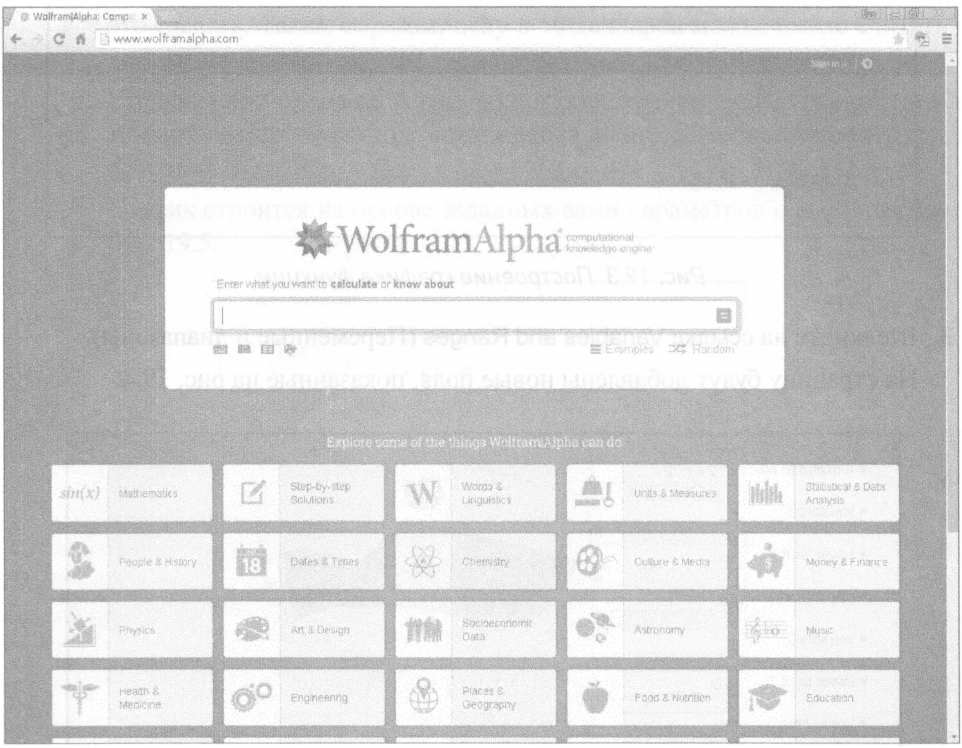


Рис. 19.2. Начальная страница сайта WolframAlpha

2. В верхнем поле формы введите фразу **3D plot** (Объемный график).

   Результат поиска функции построения объемного графика показан на рис. 19.3. Обратите внимание на поле **Function to plot** (График функции).

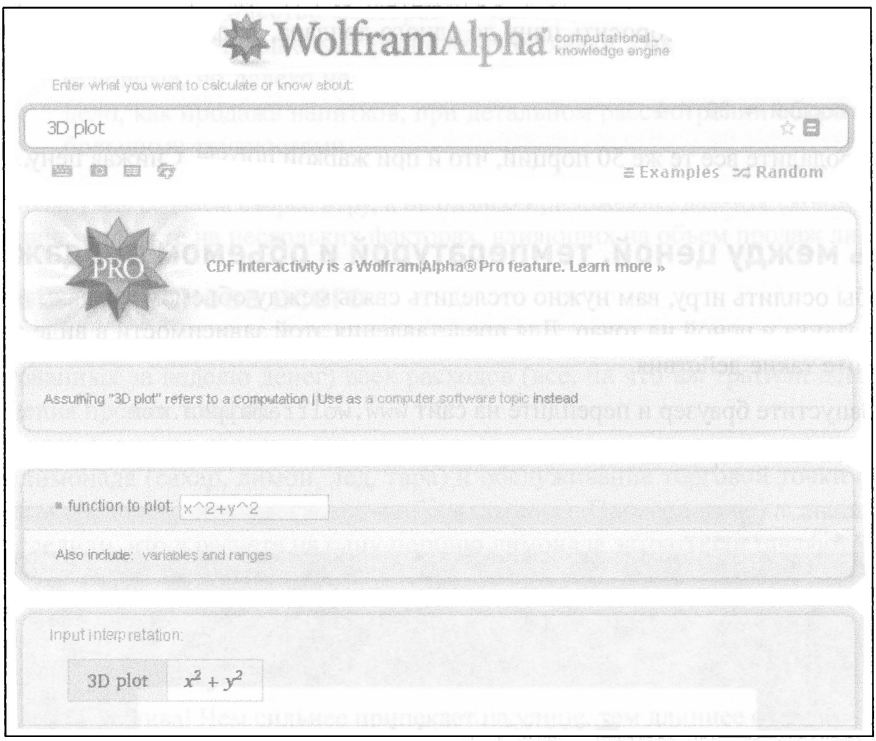


Рис. 19.3. Построение графика функции

3. Щелкните на ссылке **Variables and Ranges** (Переменные и диапазоны).

   На страницу будут добавлены новые поля, показанные на рис. 19.4.

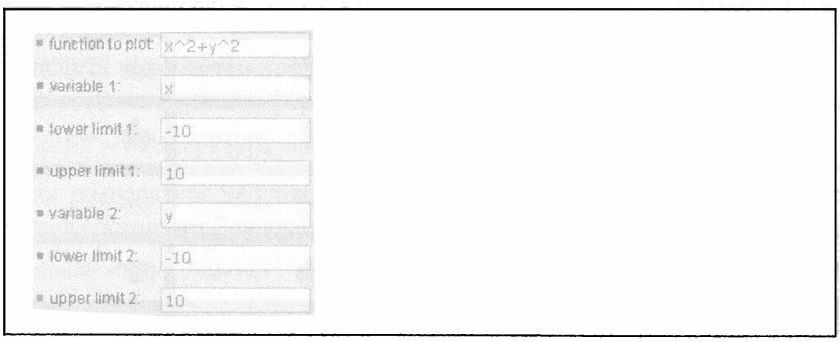


Рис. 19.4. Переменные функции и диапазоны отображаемых значений

4. В поле 3D plot введите формулу `z = x/y`.

   Здесь буква `z` представляет количество проданных порций лимонада, символ `x` соответствует температуре на улице, а `y` — цене одной порции.

5. В поле Lower Limit 1 (Нижний предел 1) введите значение `0`.

   В данном поле указывается наименьшее значение переменной `x`, отображаемое на графике. В нашем случае оно представляет минимальную температуру воздуха.

6. В поле Upper Limit 1 (Верхний предел 1) введите значение `100`.

   Указанное значение соответствует максимальной температуре воздуха на улице.

7. В поле Lower Limit 2 введите значение `0`.

   Это минимально возможная цена порции лимонада.

   При цене 0 долларов за порцию вы продадите сколько угодно лимонада. Но такая стратегия ценообразования совершенно не подходит для ведения бизнеса!

8. В поле Upper Limit 2 введите число `10`.

   Как бы ни хотелось продавать лимонад по такой цене, вам вряд ли удастся это сделать. Таким образом, цену в 10 долларов можно смело считать недостижимым максимумом.

9. Щелкните на оранжевой кнопке с изображением знака "равно", отображаемой справа от текущего поля ввода данных, чтобы построить график функции.

   График строится на основе заданных вами параметров и выглядит, как на рис. 19.5.

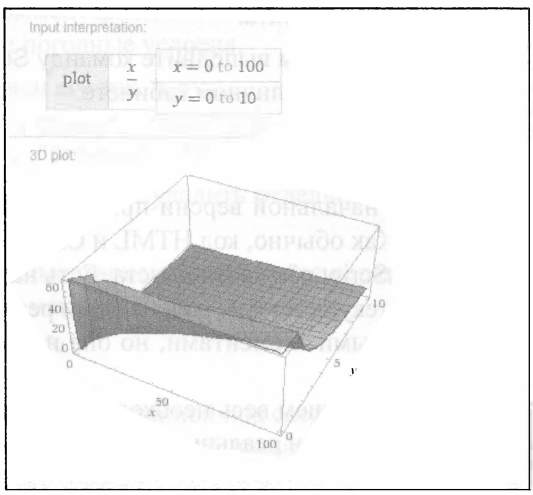


Рис. 19.5. График зависимости между объемом продаж, температурой на улице и ценой

Если проанализировать график, показанный на рис. 19.5, то становится очевидным, что максимальный объем продаж достигается при максимальной температуре воздуха и минимальной цене на единицу товара.

Сайт WolframAlpha имеет просто невероятные возможности. Обязательно поэкспериментируйте с другими функциями, рабочими диапазонами и типами диаграмм.

Разработка последней игры

Теперь, когда вы получили полное представление о математических функциях, которые лежат в основе расчетов, выполняемых в разрабатываемом приложении, можете смело приступать к написанию его кода.

Как и ранее, нами уже создана начальная версия приложения. Перед продолжением обязательно ознакомьтесь с ней и создайте отдельную копию в своем личном кабинете.

Копирование кода

Чтобы скопировать в свой личный кабинет начальную версию приложения, подсчитывающего прибыль с продаж лимонада, выполните следующие действия.

1. Войдите в приложении JSFiddle под собственной учетной записью и перейдите в общедоступный кабинет.
   `http://jsfiddle.net/user/forkids/fiddles`
2. Отыщите в списке программу **Chapter 19 — Lemonade Stand — Start** и щелкните на ее названии.
3. В верхней части окна щелкните на кнопке **Fork** для заимствования кода.
4. Раскройте меню **Fiddle Options**, расположенное на левой панели, и введите название приложения для нового проекта.
5. Щелкните на кнопке **Update**, а затем выполните команду **Set as Base**, чтобы сохранить полученную копию игры в личном кабинете.

Код JavaScript

Просмотрите панели с кодом начальной версии приложения, обеспечивающего управление лимонадной лавкой. Как обычно, код HTML и CSS введен полностью без вашего участия, а вот панель JavaScript абсолютно пуста. Есть над чем потрудиться!

Если запустить приложение в текущем виде, то на панели результатов отобразится веб-страница со всеми необходимыми элементами, но она не будет реагировать на действия пользователя.

Давайте в пошаговом режиме напишем весь необходимый код JavaScript, заменив комментарии командами, выполняющими реальные действия.

Объявление переменных

Первым делом объявим глобальные переменные, в которых будут храниться рабочие данные. Они перечислены ниже.

- ✓ Массив, элементы которого представляют дни рабочей недели.
- ✓ Массив для хранения погодных условий.
- ✓ Переменные, в которые заносятся возможные значения минимальной и максимальной температуры воздуха.
- ✓ Переменная, указывающая стоимость производства порции лимонада.
- ✓ Массив, элементы которого представляют температуру воздуха каждый рабочий день.

Перед написанием кода давайте введем на панели JavaScript комментарии, в которых описываются все переменные программы (листинг 19.1).

Листинг 19.1. Комментарии по созданию переменных

```
// Создание массива дней недели
// Определение типов погодных условий
// Установка минимальной и максимальной температур
// Расчет себестоимости порции лимонада
// Создание массива, представляющего температуру на каждый день
```

Дальнейшие действия мы будем выполнять, сверяясь с намеченным планом, представленным в комментариях.

1. Под строкой первого комментария введите следующий код.

   ```
   var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
   ```

2. Под вторым комментарием объявите массив, элементы которого представляют все возможные погодные условия.

   Мы представляем его так.

   ```
   var weather = ["Sunny", "Partly Sunny", "Partly Cloudy", "Cloudy",
      ↳"Raining", "Snowing", "Thunderstorm", "Foggy"];
   ```

   Вы вольны добавить или удалить отдельные элементы этого массива на свое усмотрение.

3. Под третьим комментарием создайте две переменные, отвечающие за хранение минимальной и максимальной температур.

   ```
   var maxTemp = 100;
   var minTemp = 0;
   ```

4. В новой строке, расположенной под следующим комментарием, объявите переменную `lemonadeCost`, которой назначается себестоимость производства порции лимонада, выраженная в долларах.

   ```
   var lemonadeCost = 0.5;
   ```

5. Создайте новый массив и назовите его `dailyTemp`. В нем будут храниться температуры, до которых прогревается воздух каждый день рабочей недели.

```
var dailyTemp = [];
```

6. Чтобы сохранить введенный выше код JavaScript, щелкните на кнопке **Update**.
7. На данном этапе код JavaScript нашего приложения должен выглядеть, как в листинге 19.2.

Листинг 19.2. Глобальные переменные для программы управления лавкой по продаже лимонада

```
// Создание массива дней недели
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];

// Определение типов погодных условий
var weather = ["Sunny", "Partly Sunny", "Partly Cloudy", "Cloudy",
   ↳"Raining", "Snowing", "Thunderstorm", "Foggy"];

// Установка минимальной и максимальной температур
var maxTemp = 100;
var minTemp = 0;

// Расчет себестоимости порции лимонада
var lemonadeCost = 0.5;

// Создание массива, хранящего температуру на каждый день
var dailyTemp = [];
```

Определение погодных условий

Следующий этап написания программы заключается в учете погоды. К счастью, в нашем распоряжении уже есть функция генерации произвольных погодных условий (см. главу 17).

В настоящем проекте мы внесем в функцию `generateWeather()` всего одно изменение: теперь все возможные погодные условия будут представляться массивом `dailyTemp`.

Выполните следующие действия, чтобы изменить функцию `generateWeather()`, известную по главе 17, в которой рассматривалось приложение прогноза погоды.

1. Напишите комментарий, описывающий назначение функции.

```
/**
Генерация погодных условий для рабочей недели
**/
```

2. Создайте заголовок функции.

```
function generateWeather() {
```

3. Объявите две локальные переменные, хранящие сведения о текущей погоде и температуре на улице.

```
var weatherToday;
var tempToday;
```

4. Начните цикл, в котором обрабатываются все дни недели.

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

5. Добавьте код извлечения произвольных погодных условий из соответствующего массива. Назначьте полученное значение переменной `weatherToday`.

6. Создайте инструкцию получения произвольной температуры в диапазоне от `minTemp` до `maxTemp`.

```
tempToday = Math.floor(Math.random() * (maxTemp - minTemp) + minTemp);
```

7. Сохраните температуру в массиве `dailyTemp`.

```
dailyTemp[i] = tempToday;
```

8. Сгенерируйте сообщение, содержащее сведения о текущей погоде.

```
document.getElementById("5DayWeather").innerHTML += "<div id='" +
   ↳days[i] + "' class='" + weatherToday + "'><b>Forecast for " +
   ↳days[i] + ":</b><br><br>" + weatherToday + " and " + tempToday
   ↳+ " degrees.</div>";
```

9. Завершите цикл и функцию.

```
   }
}
```

10. Для вызова функции во время загрузки программы введите сразу после объявления глобальных переменных следующую строку кода.

```
generateWeather();
```

11. Чтобы сохранить внесенные в код JavaScript изменения, щелкните на кнопке **Update**.

На этом создание функции, отвечающей за генерацию произвольных погодных условий, завершено. Если вы умудрились не наделать ошибок, то расписание погоды на рабочую неделю, отображаемое над формой ввода данных панели **Results**, будет иметь вид, подобный показанному на рис. 19.6.

Сравните только что введенный код с показанным в листинге 19.3 и убедитесь в том, что они совпадают.

Листинг 19.3. Объявление переменных и функция `generateWeather()`

```
// Создание массива дней недели
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];

// Определение типов погодных условий
var weather = ["Sunny", "Partly Sunny", "Partly Cloudy", "Cloudy",
   ↳"Raining", "Snowing", "Thunderstorm", "Foggy"];

// Установка минимальной и максимальной температур
var maxTemp = 100;
var minTemp = 0;
```

```
// Расчет себестоимости порции лимонада
var lemonadeCost = 0.5;

// Создание массива, представляющего температуру на каждый день
var dailyTemp = [];

// Получение погодных условий
generateWeather();

/**
Генерация погодных условий для рабочей недели
**/

function generateWeather() {
   var weatherToday;
   var tempToday;
   for (var i = 0; i < days.length; i++) {
      weatherToday = weather[Math.floor(Math.random() * weather.length)];
      tempToday = Math.floor(Math.random() * (maxTemp - minTemp) +
         ↳minTemp);
      dailyTemp[i] = tempToday;
      document.getElementById("5DayWeather").innerHTML += "<div
      ↳id='" + days[i] + "' class='" + weatherToday + "'><b>
      ↳Forecast for " + days[i] + ":</b><br><br>" + weatherToday +
      ↳" and " + tempToday + " degrees.</div>";
   }
}
```

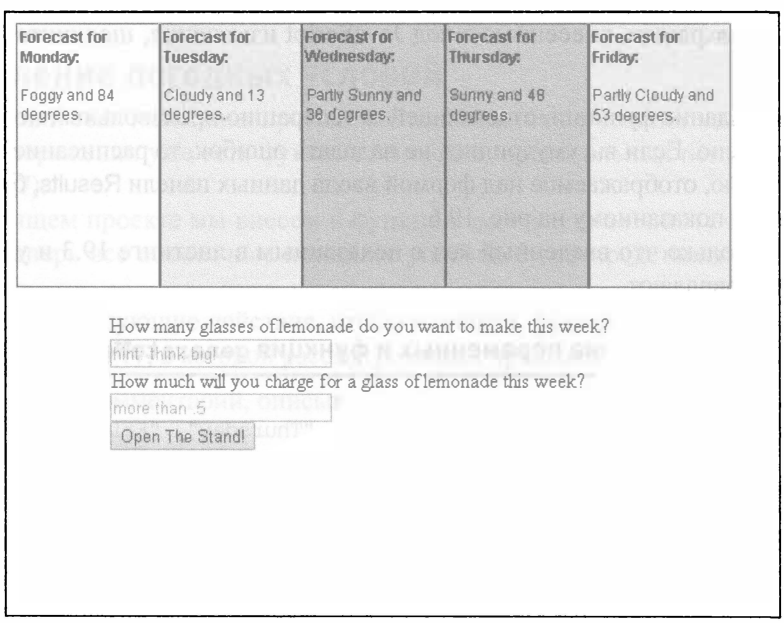


Рис. 19.6. Панель результатов, содержащая прогноз погоды на рабочую неделю

Открытие лавки

Следующая функция, которую мы будем создавать, вычисляет количество порций лимонада, проданных за рабочую неделю.

Чтобы создать функцию `openTheStand()`, следуйте приведенным далее инструкциям.

1. Напишите комментарий, в котором описывается назначение функции. В новой строке создайте заголовок функции.

```
/**
Вычисление объемов продаж
**/

function openTheStand() {
```

2. Создайте три новые переменные: первая предназначается для хранения количества ежедневно проданных порций лимонада, вторая указывает количество лимонада, проданного за неделю, а третья определяет непроданный остаток. Исходно всем трем переменным устанавливается нулевое значение.

```
var glassesSold = 0; // ежедневно
var totalGlasses = 0; // за неделю
var glassesLeft = 0; // остаток
```

3. Добавьте выражение вызова функции `resetForm()`, которая сбрасывает область результатов, предотвращая необходимость перезапуска игры.

```
// Очистка предыдущих результатов
resetForm();
```

Код функции `resetForm()` мы напишем после создания функции `openTheStand()`.

4. Получите данные полей формы.

```
// Получение данных
var numGlasses = Number(document.getElementById("numGlasses").
value);
var glassPrice = Number(document.getElementById("glassPrice").
value);
```

5. Создайте новый цикл для просмотра данных всех дней рабочей недели.

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

6. Вычислите количество проданных порций лимонада.

```
// Объем продаж в зависимости от температуры и цены
glassesSold = Math.floor(dailyTemp[i] / glassPrice);
```

7. Вычислите остаток лимонада.

```
// Количество порций непроданного лимонада
glassesLeft = numGlasses - totalGlasses;
```

8. Проверьте наличие лимонада на складе в условной конструкции `if... else`.

```
// Нельзя продать больше, чем имеется на складе
if (glassesSold > glassesLeft) {
glassesSold = glassesLeft;
}
```

Если значение переменной `glassesSold` превышает остаток, то ей присваивается значение `glassesLeft` или то, что осталось.

9. Прибавьте проданный за день лимонад к недельному объему.

```
// Прибавление проданного за день к недельному объему
totalGlasses = glassesSold + totalGlasses;
```

10. Выведите объем продаж за рабочую неделю.

```
// Отображение объема продаж за неделю
document.getElementById("result").innerHTML += "<p>" + days[i] +
   ↳", you sold " + glassesSold + " glasses of lemonade.</p>";
```

11. Завершите цикл закрывающей фигурной скобкой.

```
}
```

12. Создайте команду вызова функции, которая выводит недельный объем продаж, принимая сразу три аргумента: `numGlasses`, `glassPrice` и `totalGlasses`.

```
displayResults(numGlasses,glassPrice,totalGlasses);
```

13. Завершите функцию, введя в ее конце закрывающую фигурную скобку.

```
}
```

14. Сохраните код JavaScript, щелкнув на кнопке Update.

Если вы все сделали правильно, то код вашей функции `openTheStand()` будет совпадать с приведенным в листинге 19.4.

Листинг 19.4. Код функции `openTheStand()`

```
/**
Вычисление объемов продаж
**/
function openTheStand() {
   var glassesSold = 0; // ежедневно
   var totalGlasses = 0; // за неделю
   var glassesLeft = 0; // остаток

   // Очистка предыдущих результатов
   resetForm();

   // Получение данных
   var numGlasses = Number(document.getElementById("numGlasses").
      ↳value);
   var glassPrice = Number(document.getElementById("glassPrice").
      ↳value);

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

```
      // Объем продаж в зависимости от температуры и цены
      glassesSold = Math.floor(dailyTemp[i] / glassPrice);

      // Количество порций непроданного лимонада
      glassesLeft = numGlasses - totalGlasses;

      // Нельзя продать больше, чем имеется на складе
      if (glassesSold > glassesLeft) {
         glassesSold = glassesLeft;
      }

      // Прибавление проданного за день к недельному объему
      totalGlasses = glassesSold + totalGlasses;

      // Отображение объема продаж за неделю
      document.getElementById("result").innerHTML += "<p>" +
      ↳days[i] + ", you sold " + glassesSold + " glasses of
      ↳lemonade.</p>";

   }

   displayResults(numGlasses, glassPrice, totalGlasses);

}
```

Сброс программы

В первой строке функции `openTheStand()` вызывается функция `resetForm()`. Она очень проста. Ее главная задача — очищать содержимое области результатов, что позволяет запускать приложение повторно, избегая вывода следующих результатов непосредственно под предыдущими.

Полный код функции `resetForm()` приведен в листинге 19.5. Введите его на панели JavaScript сразу под кодом функции `openTheStand()` (на данный момент — в конец программы).

Листинг 19.5. Код функции `resetForm()`

```
/**
Сброс игры для обеспечения ее повторного запуска
**/
function resetForm() {
   document.getElementById("result").innerHTML = "";
}
```

Добавив в программу функцию `resetForm()`, не забудьте щелкнуть на кнопке Update, расположенной в верхней части окна приложения JSFiddle.

Вывод результатов

Нам осталось рассмотреть последнюю функцию в симуляторе лавки по продаже лимонада: `displayResults()`. Она отвечает за вычисление недельных объемов продаж, для чего принимает в качестве аргумента результат, возвращаемый функцией `openTheStand()`. Более того, эта функция выводит результат вашей торговой деятельности на экран.

Чтобы написать функцию `displayResults()`, выполните следующие действия.

1. Введите в конце панели JavaScript комментарий, указывающий назначение функции `displayResults()`, а также ее заголовок, в котором объявляется использование трех параметров: `weeklyInventory`, `glassPrice` и `weeklySales`.

```
/**
Вычисление и вывод результата
**/
function displayResults(weeklyInventory, glassPrice, weeklySales) {
```

2. Рассчитайте общий доход как произведение количества проданных порций лимонада (за неделю) и цены одной порции.

```
var revenue = weeklySales * glassPrice;
```

3. Определите понесенные затраты, умножив себестоимость порции лимонада на объем продаж за неделю.

```
var expense = weeklyInventory * lemonadeCost;
```

4. Вычислите количество лимонада, оставшегося у вас после вычитания проданного за неделю объема.

```
var leftOver = weeklyInventory - weeklySales;
```

5. Вычислите прибыль, вычтя из дохода расходную часть.

```
var profit = revenue - expense;
```

6. Напишите код выводимого пользователю отчета, который представляется сразу четырьмя строками.

```
// Вывод недельного отчета
document.getElementById("result").innerHTML += "<p>You sold a
   ↳total of " + weeklySales + " glasses of lemonade this week.</p>";
document.getElementById("result").innerHTML += "<p>Total revenue:
   ↳$" + revenue + ".</p>";
document.getElementById("result").innerHTML += "<p>You have " +
   ↳leftOver + " glasses of lemonade left over.</p>";
document.getElementById("result").innerHTML += "<p>Each glass
   ↳costs you $" + lemonadeCost + ". Your profit was $" + profit + ".";
```

7. Завершите функцию, введя закрывающую фигурную скобку.

```
}
```

8. Чтобы сохранить код JavaScript, щелкните на кнопке **Update**.

Полный код функции `displayResults()` приведен в листинге 19.6.

Листинг 19.6. Код функции `displayResults()`

```
/**
Вычисление и вывод результата
**/
function displayResults(weeklyInventory, glassPrice,
weeklySales) {
   // Вычисление результата
   var revenue = weeklySales * glassPrice;
   var expense = weeklyInventory * lemonadeCost;
   var leftOver = weeklyInventory - weeklySales;
  var profit = revenue - expense;

   // Вывод недельного отчета
   document.getElementById("result").innerHTML += "<p>You sold
      ↳a total of " + weeklySales + " glasses of lemonade this
      ↳week.</p>";
   document.getElementById("result").innerHTML += "<p>Total
      ↳revenue: $" + revenue + ".</p>";
   document.getElementById("result").innerHTML += "<p>You have " +
      ↳leftOver + " glasses of lemonade left over.</p>";
   document.getElementById("result").innerHTML += "<p>Each glass
      ↳costs you $" + lemonadeCost + ". Your profit was $" +
      ↳profit + ".";
}
```

Завершение и тестирование программы

Если вы протестируете программу в текущем виде, то обнаружите, что она умеет всего лишь выводить прогноз погоды на рабочую неделю.

Что же мы забыли сделать? Как исправить обнаруженную неполадку?

Имея достаточный опыт в написании приложений, вы наверняка уже догадались: в нашей программе недостает обработчика события `click` для кнопки. В его отсутствие процесс обработки всех выполняемых в приложении расчетов не запускается.

Для завершения программы и последующего ее тестирования следуйте таким инструкциям.

1. На панели JavaScript введите следующий код, разместив его под кодом объявления функций.

   ```
   // обработчик заказов
   document.getElementById("OpenTheStand").addEventListener("click",
      ↳openTheStand);
   ```

2. Для сохранения изменений в приложении щелкните на кнопке **Update**, а затем выполните команду **Set as Base**.

   На завершающем этапе код на панели JavaScript нашего приложения должен выглядеть, как в листинге 19.7.

Листинг 19.7. Программа управления продажами лимонадной лавки

```
// Создание массива дней недели
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];

// Определение типов погодных условий
var weather = ["Sunny", "Partly Sunny", "Partly Cloudy", "Cloudy",
   ↳"Raining", "Snowing", "Thunderstorm", "Foggy"];

// Установка минимальной и максимальной температур
var maxTemp = 100;
var minTemp = 0;

// Расчет себестоимости порции лимонада
var lemonadeCost = 0.5;

// Создание массива, представляющего температуру на каждый день
var dailyTemp = [];

// Обработчик заказов
document.getElementById("OpenTheStand").addEventListener("click",
   ↳openTheStand);

// Получение погодных условий
generateWeather();

/**
Генерация погодных условий на рабочую неделю
**/
function generateWeather() {
   var weatherToday;
   var tempToday;
   for (var i = 0; i < days.length; i++) {
      weatherToday = weather[Math.floor(Math.random() * weather.length)];
      tempToday = Math.floor(Math.random() * (maxTemp - minTemp) +
         ↳minTemp);
      dailyTemp[i] = tempToday;
      document.getElementById("5DayWeather").innerHTML += "<div
      ↳id='" + days[i] + "' class='" + weatherToday + "'><b>
      ↳Forecast for " + days[i] + ":</b><br><br>" + weatherToday +
      ↳" and " + tempToday + " degrees.</div>";
   }
}

/**
Вычисление объемов продаж
**/
function openTheStand() {
   var glassesSold = 0; // ежедневно
   var totalGlasses = 0; // за неделю
   var glassesLeft = 0; // остаток
```

```
   // Очистка предыдущих результатов
   resetForm();

   // Получение данных
   var numGlasses = Number(document.getElementById("numGlasses").
      ↳value);
   var glassPrice = Number(document.getElementById("glassPrice").
      ↳value);

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

      // Объем продаж в зависимости от температуры и цены
      glassesSold = Math.floor(dailyTemp[i] / glassPrice);

      // Количество порций непроданного лимонада
      glassesLeft = numGlasses - totalGlasses;

      // Нельзя продать больше, чем имеется на складе
      if (glassesSold > glassesLeft) {
         glassesSold = glassesLeft;
      }

      // Прибавление проданного за день к недельному объему
      totalGlasses = glassesSold + totalGlasses;

      // Отображение объема продаж за неделю
      document.getElementById("result").innerHTML += "<p>" +
      ↳days[i] + ", you sold " + glassesSold + " glasses of
      ↳lemonade.</p>";

   }

   displayResults(numGlasses, glassPrice, totalGlasses);

}

/**
Вычисление и вывод результата
**/
function displayResults(weeklyInventory, glassPrice, weeklySales) {
   // Вычисление результата
   var revenue = weeklySales * glassPrice;
   var expense = weeklyInventory * lemonadeCost;
   var leftOver = weeklyInventory - weeklySales;
  var profit = revenue - expense;

   // Вывод недельного отчета
   document.getElementById("result").innerHTML += "<p>You sold
      ↳a total of " + weeklySales + " glasses of lemonade this
      ↳week.</p>";
   document.getElementById("result").innerHTML += "<p>Total
      ↳revenue: $" + revenue + ".</p>";
```

```
    document.getElementById("result").innerHTML += "<p>You have " +
        ↳leftOver + " glasses of lemonade left over.</p>";
    document.getElementById("result").innerHTML += "<p>Each glass
        ↳costs you $" + lemonadeCost + ". Your profit was $" +
        ↳profit + ".";
}

/**
Сброс игры для обеспечения ее повторного запуска
**/
function resetForm() {
    document.getElementById("result").innerHTML = "";
}
```

3. В поле **How many glasses of lemonade do you want to make for the week?** (Сколько порций лимонада вы ходите продать за неделю?) введите количество порций лимонада, которые планируется продать за неделю.
4. В поле **How much will you charge for a glass of lemonade this week?** (Сколько вы планируете брать за порцию лимонада на этой неделе?) укажите цену, запрашиваемую за порцию лимонада.
5. Щелкните на кнопке **Open The Stand** (Начать торговлю).

   В области результатов будет указано, сколько порций лимонада вы продали за каждый рабочий день. В конце формы выводятся недельный объем продаж и полученная прибыль (рис. 19.7).

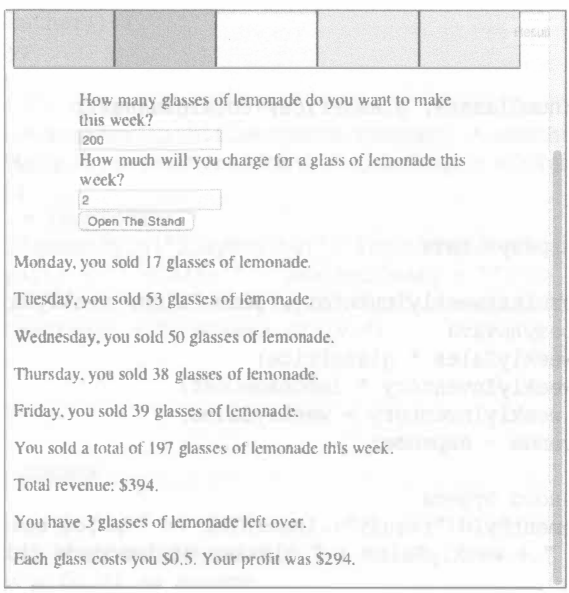


Рис. 19.7. Симулятор продажи лимонада

Как вам первые шаги на новом поприще? Удалось не прогореть и остаться с прибылью в кармане? Как максимизировать прибыль, и какие значения для этого нужно вводить в полях формы? Какое из входных значений сильнее влияет на получаемый результат? Что произойдет, если количество продаваемых порций или цену сделать слишком большой? И какую прибыль можно получить, если ввести в оба поля очень маленькие числа?

Ответив на все приведенные выше вопросы, попробуйте выяснить, как можно улучшить приложение. Обязательно прислушайтесь к приведенным в следующем разделе рекомендациям.

Улучшение симулятора лимонадной лавки

Описанная выше игра не только интересна, но и демонстрирует несколько важных методик программирования на языке JavaScript. Вдоволь поэкспериментировав, у вас точно появятся идеи, как сделать игру интереснее, сбалансированнее и реалистичнее.

Если вы дочитали до последней главы, то гарантировано овладели основными навыками программирования на JavaScript. Их вполне достаточно, чтобы начать работу над собственными проектами. Просто поверьте в себя!

Ниже приведено несколько идей относительно того, как лучше всего модифицировать программу управления лимонадной лавкой.

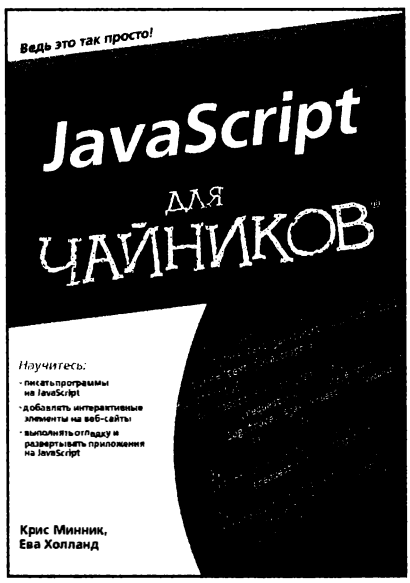
- ✓ Позвольте пользователю изменять цену и объем продаж ежедневно, а не раз в неделю.
- ✓ Учитывайте при расчете объемов продаж не только температуру на улице, но и другие погодные факторы: дождь, снег, извержение вулкана и т.п.
- ✓ Сделайте ежедневную цену на порцию лимонада случайной.
- ✓ Воспользуйтесь средствами HTML и CSS для улучшения внешнего вида игры.
- ✓ Добавьте на форму кнопку, отвечающую за сброс и обновление погодных условий, чтобы избавить пользователей от необходимости перезапускать игру.
- ✓ Создайте таблицу рекордов, в которой сохраняйте только лучший результат из всех предпринятых попыток.
- ✓ Вычислите, сколько сахара, лимонов и остальных ингредиентов потребуется для производства лимонада. Какова себестоимость одной порции?
- ✓ Создайте в игре случайные события, например град или атаку пришельцев, заведомо мешающих нормальной торговле.

Здесь приведены только некоторые из сотен идей по улучшению имеющегося приложения. Пофантазируйте, и вы обязательно придумаете новые функции, которые было бы неплохо включить в симулятор лимонадной лавки. Нет предела совершенству!

Предметный указатель

JAVASCRIPT ДЛЯ ЧАЙНИКОВ

Крис Минник,
Ева Холланд



www.dialektika.com

JavaScript — ключевой инструмент создания современных сайтов, и благодаря данному руководству, ориентированному на новичков, вы сможете изучить язык в короткие сроки и с минимумом усилий. Узнайте, какова структура языка, как правильно записывать его инструкции, как применять CSS, работать с онлайн-графикой и подключать программные интерфейсы HTML5. Все темы можно закрепить практическими упражнениями, доступными для выполнения на сайте Codecademy.com.

Основные темы книги:

- как настроить среду разработки;
- для чего нужны массивы;
- применение циклов;
- использование библиотеки jQuery;
- создание анимации в JavaScript;
- работа с CSS и графикой;
- AJAX и JSON;
- как избежать распространенных ошибок.

ISBN 978-5-8459-2036-2 **в продаже**

HTML5 И CSS3 ДЛЯ ЧАЙНИКОВ

Эд Титтел,
Крис Минник

www.dialektika.com

Полагаете, будто создавать веб-сайты очень сложно? Ошибаетесь! С появлением HTML5, новейшей версии стандарта HTML, создавать и изменять дизайн веб-страниц стало проще, чем когда-либо. С помощью этой замечательной книги, написанной простым и понятным языком, вы освоите искусство веб-дизайна, изучите основы HTML5 и CSS3 и научитесь создавать собственные сайты.

Основные темы книги:

- создание веб-страниц;
- форматирование веб-страниц с помощью (X)HTML;
- просмотр и публикация веб-страниц в Интернете;
- применение метаданных поисковыми системами;
- управление текстовыми блоками, списками и таблицами;
- создание ссылок на документы и другие веб-сайты;
- настройка стилевых правил CSS;
- что можно, а чего нельзя делать с помощью HTML.

ISBN 978-5-8459-2035-5 **в продаже**

JavaScript для профессионалов

Второе издание

Джон Резиг,
Расс Фергюсон,
Джон Пакстон

www.williamspublishing.com

Эта книга является незаменимым пособием для профессиональных разработчиков современных веб-приложений на JavaScript. Читатель найдет в ней все, что требуется знать о современном состоянии языка JavaScript, его достоинствах и недостатках, новых языковых средствах, внедренных в последних версиях стандарта ECMAScript, передовых приемах отладки и тестирования кода, а также инструментальных средствах разработки. Книга изобилует многочисленными практическими и подробно разбираемыми примерами кода, повторно используемых функций и классов, экономящих время разработчиков. Она помогает им овладеть практическими навыками написания динамических веб-приложений на высоком профессиональном уровне, а также повысить свою квалификацию.

Книга рассчитана на тех, кто интересуется разработкой веб-приложений и имеет опыт программирования на JavaScript.

ISBN 978-5-8459-2054-6 **в продаже**

BESTSELLING
BOOK
SERIES

Программирование на JavaScript® для чайников

СЕРИЯ
КОМПЬЮТЕРНЫХ
КНИГ ОТ
ДИАЛЕКТИКИ

Шпаргалка

Соответствия имен свойств CSS ключевым словам JavaScript

Свойство CSS	Ключевое слово JavaScript
background	background
background-attachment	backgroundAttachment
background-color	backgroundColor
background-image	backgroundImage
background-position	backgroundPosition
background-repeat	backgroundRepeat
border	border
border-bottom	borderBottom
border-bottom-color	borderBottomColor
border-bottom-style	borderBottomStyle
border-bottom-width	borderBottomWidth
border-color	borderColor
border-left	borderLeft
border-left-color	borderLeftColor
border-left-style	borderLeftStyle
border-left-width	borderLeftWidth
border-right	borderRight
border-right-color	borderRightColor
border-right-style	borderRightStyle
border-right-width	borderRightWidth
border-style	borderStyle
border-top	borderTop
border-top-color	borderTopColor
border-top-style	borderTopStyle
border-top-width	borderTopWidth
border-width	borderWidth
clear	clear
clip	clip
color	color
cursor	cursor
display	display
filter	filter
float	cssFloat
font	font
font-family	fontFamily
font-size	fontSize
font-variant	fontVariant

Свойство CSS	Ключевое слово JavaScript
font-weight	fontWeight
height	height
left	left
letter-spacing	letter-spacing
line-height	lineHeight
list-style	listStyle
list-style-image	listStyleImage
list-style	backgroundImage
list-style-image	listStyleImage
list-style-position	listStylePosition
list-style-type	listStyleType
margin	margin
margin-bottom	marginBottom
margin-left	marginLeft
margin-right	marginRight
margin-top	marginTop
overflow	overflow
padding	padding
padding-bottom	paddingBottom
padding-left	paddingLeft
padding-right	paddingRight
padding-top	paddingTop
page-break-after	pageBreakAfter
page-break-before	pageBreakBefore
position	position
text-align	textAlign
text-decoration	textDecoration
text-decoration: blink	textDecorationBlink
text-decoration: line-through	textDecorationLineThrough
text-decoration: none	textDecorationNone
text-decoration: overline	textDecorationOverline
text-decoration: underline	textDecorationUnderline
text-indent	textIndent
text-transform	textTransform
top	top
vertical-align	verticalAlign
visibility	visibility
width	width
z-index	zIndex

BESTSELLING
BOOK
SERIES

Программирование на JavaScript® для чайников

СЕРИЯ
КОМПЬЮТЕРНЫХ
КНИГ ОТ
ДИАЛЕКТИКИ

Шпаргалка

События браузера, обрабатываемые в JavaScript

Событие	Описание
abort	Загрузка файла прервана
blur	Элемент потерял фокус ввода. Другими словами, это событие возникает, когда пользователь фокусируется на другом элементе
change	Происходит по окончании изменения значения элемента, когда это изменение зафиксировано
click	Щелчок мышью на элементе
dbclick	Двойной щелчок мышью на элементе
error	Ошибка при загрузке файла
focus	Элемент получает фокус ввода. Другими словами, пользователь переходит к использованию этого элемента
input	Изменяется значение элемента `input` или `textarea`
keydown	Нажатие клавиши
keyup	Клавиша отпускается после нажатия
load	Страница и все вложенные файлы успешно загружены
mousedown	Кнопка мыши нажимается при наведенном на элемент указателе
mouseenter	Указатель мыши наводится на элемент
mouseleave	Указатель мыши убирается с элемента
mousemove	Указатель мыши перемещается по элементу
mouseout	Указатель мыши убирается с элемента или с одного из его потомков
mouseover	Указатель мыши наводится на элемент или один из его потомков
mouseup	Кнопка мыши отпускается при наведенном на элемент указателе
mousewheel	Вращается колесико мыши
reset	Форма сбрасывается
resize	Документ изменяет свой размер
scroll	Документ или элемент прокручивается
select	Выделяется текст
submit	Данные формы отправляются

BESTSELLING
BOOK
SERIES

Программирование на JavaScript® для чайников

СЕРИЯ
КОМПЬЮТЕРНЫХ
КНИГ ОТ
ДИАЛЕКТИКИ

Список зарезервированных слов JavaScript

abstract	final	public
boolean	finally	return
break	float	short
byte	for	static
case	function	super
catch	goto	switch
char	if	synchronized
class	implements	this
const	import	throw
continue	in	throws
debugger	instanceof	transient
default	int	true
delete	interface	try
do	long	typeof
double	native	var
else	new	void
enum	null	volatile
export	package	while
extends	private	with
false	protected	

Научитесь программировать и создавать интересные приложения!

Язык программирования JavaScript оживляет веб-страницы и позволяет запускать на них интерактивные приложения, эффективно взаимодействующие с пользователями. В этой книге вы узнаете, как с помощью JavaScript написать рассказ, разработать веб-страницу, создать игру в слова, построить калькулятор, организовать лавку по продаже лимонада и выполнить другие не менее захватывающие проекты. Удивите своих друзей!

В книге...

- Использование консоли JavaScript своего браузера в самых первых проектах
- Создание полезных приложений и увлекательных игр
- Проявление творческого подхода к написанию приложений и разработке веб-сайтов

Крис Минник и **Ева Холланд** — опытные веб-разработчики и преподаватели, авторы ряда книг по программированию, включая *JavaScript для чайников*. Совместно основали компанию WatzThis?, которая разрабатывает курсы по программированию и созданию веб-сайтов.

Изображение на обложке:
©Depositphotos.com/40422091
Автор: stori

ДИАЛЕКТИКА
www.dialektika.com

Программирование/JavaScript

ISBN 978-5-907144-39-2

для ЧАЙНИКОВ®