Васильев А.Н.

ПРОГРАММИРОВАНИЕ НА JavaScript

В ПРИМЕРАХ И ЗАДАЧАХ

- Основы веб-разработки с «нуля»
- Концепция и основные принципы ООП
- От сценариев и инструкций к элементам управления и событиям
- Наглядные примеры и их разбор с пояснениями автора
- Подходит для самостоятельного обучения

РОССИЙСКИЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

УДК 004.43
ББК 32.973.26-018.2
В19

Васильев, Алексей Николаевич.

В19 JavaScript в примерах и задачах / Алексей Васильев. – Москва : Издательство «Э», 2017. – 720 с. – (Российский компьютерный бестселлер).

ISBN 978-5-699-95459-9

Простой и интересный самоучитель по JavaScript, наиболее популярному сегодня языку программирования во всем мире. Полный спектр сведений о языке JavaScript с примерами и разбором задач от автора учебников-бестселлеров по языкам программирования Алексея Васильева. С помощью этой книги освоить язык JavaScript сможет каждый желающий – от новичка до специалиста.

УДК 004.43
ББК 32.973.26-018.2

ISBN 978-5-699-95459-9

ОГЛАВЛЕНИЕ

ЧАСТЬ II. JAVASCRIPT И ООП

ЧАСТЬ III. ИСПОЛЬЗОВАНИЕ JAVASCRIPT

Часть I

ОСНОВЫ JAVASCRIPT

Глава 1
ЗНАКОМСТВО С JAVASCRIPT

Так где мы можем обсудить аспекты, так сказать, нашего общего дела?

из к/ф «Клуб самоубийц, или Приключения титулованной особы»

Итак, мы начинаем знакомство с языком JavaScript. При этом нам придется постоянно иметь дело со сценариями. В основном наша стратегия будет заключаться в том, чтобы включать блок с программным кодом, написанным на языке JavaScript, в документ с гипертекстовой разметкой. Поскольку пока что нас язык JavaScript интересует сам по себе, вне контекста веб-документа, мы будем использовать определенный очень простой шаблон документа с HTML-кодом, в который «инкапсулирована» инструкция по включению в документ сценария на языке JavaScript.

ЯЗЫК ГИПЕРТЕКСТОВОЙ РАЗМЕТКИ HTML

Для вставки в HTML-документ блока сценария используются дескрипторы <script> и </script>. В открывающем дескрипторе <script> описывается атрибут type со значением "text/javascript". Поэтому начинаться блок сценария будет инструкцией <script type="text/javascript">, а заканчиваться — инструкцией </script>.

НА ЗАМЕТКУ

Прежде чем перейти непосредственно к рассмотрению основ языка JavaScript, некоторое внимание придется уделить способам «инкапсуляции» сценарного кода в документ.

Сценарий должен выполнять определенные действия (иначе какой в нем смысл?). Результат этих действий должен как-то проявляться. Самый простой способ «проявления» сценария в документе — вывод информации. Информацию будем выводить в рабочую область

документа. Как иллюстрацию для начала рассмотрим очень простой пример, в котором средствами языка JavaScript в рабочем документе отображается текст.

Отображение текста в рабочем документе

Вы получите то, что желали, согласно намеченным контурам.

из к/ф «Формула любви»

В самом первом примере мы решим очень простую задачу: сценарий при выполнении отобразит в рабочем документе текст. В принципе, как решается такая задача, иллюстрировалось во *вступлении*. Здесь мы немножко расширим наше представление о возможностях сценариев в плане отображения текста в документе. Обратимся к программному коду, представленному в листинге 1.1.

Листинг 1.1. Отображение сценарием текста в документе

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Листинг 1.1</title>
</head>
<body><h3>Листинг 1.1</h3><hr>

<!-- Начало сценария -->
<script type="text/javascript">
  document.write("Приступаем к изучению <b>JavaScript</b>")
</script>
<!-- Завершение сценария -->

</body>
</html>
```

Речь идет об HTML-коде, содержащем, кроме прочего, сценарий. На рис. 1.1 показано окно текстового редактора с данным кодом.

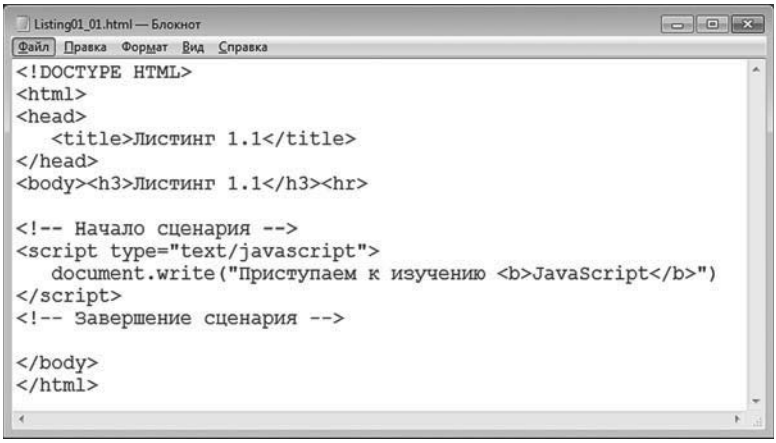

Рис. 1.1. Окно текстового редактора с кодом документа

Если этот же документ открыть в браузере, получим результат, как на рис. 1.2.

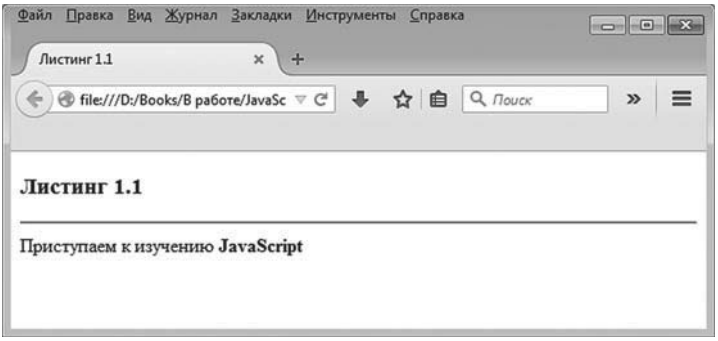

Рис. 1.2. Результат отображения веб-документа в окне браузера

Результат выполнения сценария — текст под горизонтальной линией в области документа. Все, что выше горизонтальной линии, включая саму линию, определяется HTML-кодом документа.

ЯЗЫК ГИПЕРТЕКСТОВОЙ РАЗМЕТКИ HTML

Код гипертекстовой разметки кратко обсуждался во *вступлении*. Здесь на всякий случай напомним назначение основных блоков HTML-кода.

Инструкция `<!DOCTYPE HTML>` является стандартным началом HTML-кода документа. Сам программный код заключается между дескрипто-

рами <html> и </html>. В <head>-блоке описываются основные свойства документа. В частности, в блоке между дескрипторами <title> и </title> указывается рабочее название документа (отображается в корешке вкладки окна документа в браузере). Основной код документа размещается в <body>-блоке. В данном конкретном примере этот блок содержит заголовок 3-го уровня (специально выделенный текст, отображается в рабочей области документа). Заголовок выделяется дескрипторами <h3> и </h3>. Инструкция <hr> используется для отображения горизонтальной линии в области рабочего документа. Все, что находится между инструкциями <!-- и -->, является комментарием. Сценарий, как отмечалось ранее, размещается в <script>-блоке.

Фактически код сценария состоит всего из одной команды:

```
document.write("Приступаем к изучению <b>JavaScript</b>")
```

Здесь мы имеем дело с вызовом метода write() из объекта document. Объект document — это объект рабочего документа (документ, в котором размещен код сценария). Метод write(), который вызывается из объекта документа document, отображает в данном документе текст, переданный аргументом методу. Причем отображаемый текст может содержать не только собственно текст, но и HTML-кодировку, как это имеет место в рассмотренном примере. В частности, в текстовом выражении "Приступаем к изучению JavaScript", которое передается аргументом методу write(), слово JavaScript выделено дескрипторами и . В кодировке HTML это означает выделение жирным шрифтом соответствующего текстового фрагмента при отображении его в окне браузера. Так, собственно, и происходит (см. рис. 1.2).

Таким образом, при отображении текста в рабочем документе с помощью метода write() речь идет не просто об отображении текста в окне, а об отображении HTML-кода. Такой код может содержать всевозможные HTML-дескрипторы (то есть дело не ограничивается дескрипторами и или подобными им).

НА ЗАМЕТКУ

Здесь открываются достаточно широкие возможности в плане динамического наполнения веб-документа. Другими словами, с помощью сценария можно управлять наполнением веб-документа в динамическом режиме. Технологии, подобные этой, рассматриваются в третьей части книги.

Способы реализации сценария

> Да, это от души. Замечательно. Достойно восхищения. Ложки у меня пациенты много раз глотали, не скрою. Но вот чтобы так, за обедом на десерт, и острый предмет — замечательно. За это вам наша искренняя сердечная благодарность.
>
> *из к/ф «Формула любви»*

Выше мы помещали код сценария непосредственно в HTML-код веб-документа. Для не очень больших примеров такой подход вполне удобен. Фактически речь идет об использовании определенного шаблонного кода. Если отталкиваться от предыдущего примера, то шаблон такой, как показано ниже:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Листинг 1.1</title>
</head>
<body><h3>Листинг 1.1</h3><hr>

<!-- Начало сценария -->
<script type="text/javascript">
...// Код сценария
</script>
<!-- Завершение сценария -->

</body>
</html>
```

Программный код сценария размещается в том месте, где размещен комментарий // Код сценария (для удобства восприятия соответствующее место в коде выделено жирным шрифтом).

Существует и другой способ «подключения» сценария к документу. В этом случае сценарий записывается в отдельный файл, а в HTML-коде документа добавляется ссылка на этот файл. Ссылка на файл со сценарием указывается значением атрибута src в дескрипторе <script>.

Например, если файл со сценарием называется Listing01_02.js и находится в той же папке, что и файл с HTML-кодом, то HTML-код может быть таким, как показано в листинге 1.2. Жирным шрифтом выделена инструкция с указанием загружаемого в документ файла со сценарием.

Листинг 1.2. Загрузка сценария из внешнего файла

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Листинг 1.2</title>
</head>
<body><h3>Листинг 1.2</h3><hr>

<!-- Начало сценария -->
<script type="text/javascript" src="Listing01_02.js">
</script>
<!-- Завершение сценария -->

</body>
</html>
```

Здесь, как и в предыдущем примере, мы используем дескриптор <script> для включения в веб-документ блока сценария. Вместе с тем сам <script>-блок пустой, а в открывающем дескрипторе <script> появилась инструкция src="Listing01_02.js", которой для атрибута src задается значение "Listing01_02.js" — имя файла со сценарием.

НА ЗАМЕТКУ

Если файл со сценарием находится в папке, отличной от той, где находится файл веб-документа, значением атрибута src указывается полный путь к файлу сценария.

В файл Listing01_02.js со сценарием помещаем такой код:

```
document.write("Сценарий загружается из файла")
```

На рис. 1.3 показан веб-документ с кодом из листинга 1.2, открытый в текстовом редакторе.

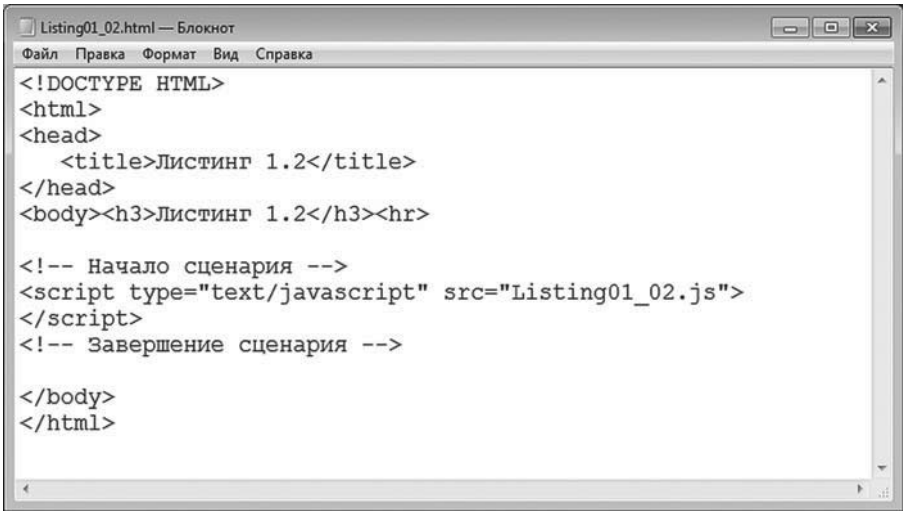

Рис. 1.3. Веб-документ открыт в текстовом редакторе

На рис. 1.4 показано окно текстового редактора с открытым в нем файлом со сценарием.

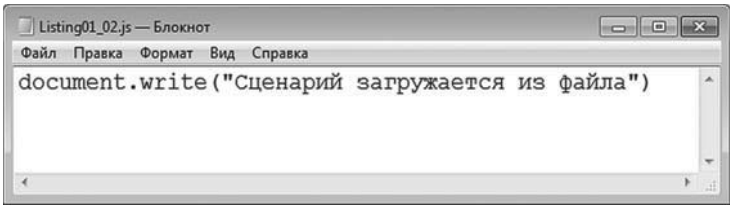

Рис. 1.4. Документ с кодом сценария открыт в текстовом редакторе

Наконец, веб-документ с кодом из листинга 1.2, который открыт в браузере, показан на рис. 1.5.

С точки зрения конечного результата загрузка сценария из внешнего файла не отличается от ситуации, когда код сценария включался непосредственно в код веб-документа.

НА ЗАМЕТКУ

Различия, конечно, есть: например, в части скорости загрузки сценария (да и надежности — файл со сценарием при неправильной ссылке на файл сценария не будет найден вовсе). Но нас такие подробности пока не интересуют.

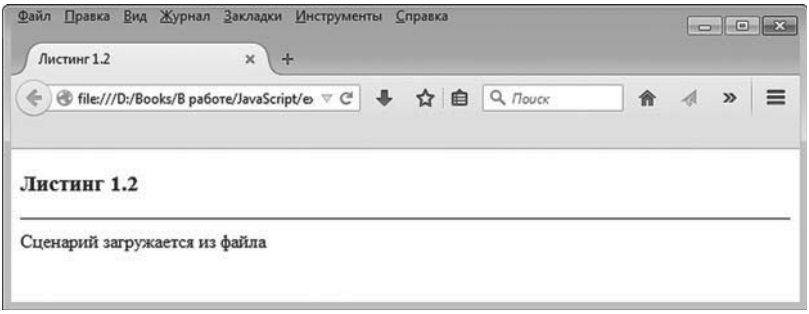

Рис. 1.5. Веб-документ открыт в браузере

На будущее, в зависимости от обстоятельств, мы будем или включать код сценария непосредственно в HTML-код документа, или записывать сценарный код в отдельный файл, а в код веб-документа добавлять инструкцию загрузки внешнего файла со сценарием.

Знакомство с переменными

> Видала я такую чепуху, по сравнению с которой эта чепуха — толковый словарь!
>
> *Л. Кэрролл «Алиса в Стране чудес»*

Вывод информации в окно рабочего документа — это хорошо. Но сценарий, состоящий из одной-единственной команды, выглядит уж слишком скромно.

Мы усложняем ситуацию и переходим на новый уровень в освоении премудростей JavaScript. Пришло время познакомиться с *переменными*.

Вообще переменная представляет собой именованную область памяти, к которой можно обращаться через имя для считывания значения и записи значения. Таким образом, у переменной есть имя (которое задается программистом). Еще у переменной есть *тип*.

НА ЗАМЕТКУ

Размер памяти, выделяемой для переменной, в принципе зависит от ее типа. Во многих языках программирования (но не в JavaScript) тип переменной указывается при ее объявлении и впоследствии не может быть изменен.

Значения, которыми оперируют в программном коде JavaScript, относятся к одному из следующих типов:

- текстовая строка;
- числовое значение;
- логическое значение (*истина* или *ложь*);
- объект;
- функция.

Объекты и функции мы пока трогать не будем. Здесь разговор будет отдельный. Пока что в зоне наших интересов текст и числа (логические значения рассмотрим при обсуждении операторов и управляющих инструкций).

ДЕТАЛИ

Если переменная принимает значение логического типа, то это означает, что она принимает одно из двух возможных значений: true (истина) или false (ложь). Обычно логические значения используются при проверке условий. Для работы с логическими значениями предназначена группа операторов, которые называются логическими.

А теперь очень важный момент: в языке JavaScript тип переменной *не фиксируется*. Буквально сказанное означает, что одна и та же переменная на разных этапах выполнения программы может принимать не просто разные значения, а значения *разных типов*. Другими словами, значением переменной сначала, например, может быть число, затем текст, затем что-то еще (в том числе объект или даже функцию — но об этом позже).

НА ЗАМЕТКУ

Для тех, кто знаком с такими языками программирования, как C++, C# или Java, означенный «демократизм» языка JavaScript в плане типизации переменных может вызвать некоторый шок. Тем не менее имеем то, что имеем.

Обычно переменные объявляются. Объявление переменной — некая декларация, цель которой в том, чтобы сообщить о намерении

использовать переменную в программном коде. Хотя такое понятие, как *тип данных,* в JavaScript существует, при объявлении переменной тип не указывается. В этом просто нет смысла, поскольку, как отмечалось выше, тип значения переменной может меняться. Более того, в JavaScript переменные можно вообще не объявлять, а сразу их использовать. Мы тем не менее будем придерживаться стиля, при котором используемые в сценарии переменные объявляются.

Сценарий с одной переменной

Как объявить переменную в сценарии? Достаточно просто. Используем ключевое слово var, после которого указываем название переменной. Если переменных несколько, их названия указываются через запятую. Как иллюстрация в листинге 1.3 приведен программный код сценария (сценарий вынесен в отдельный файл).

Листинг 1.3. Использование переменной (файл Listing01_03.js)

```
var txt
txt="Используем переменную"
document.write(txt)
```

Для тестирования кода создаем веб-документ с таким кодом:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Листинг 1.3</title>
</head>
<body><h3>Листинг 1.3</h3><hr>

<!-- Начало сценария -->
<script type="text/javascript" src="Listing01_03.js">
</script>
<!-- Завершение сценария -->

</body>
</html>
```

В результате выполнения сценария выводится следующее сообщение.

Результат выполнения сценария (из листинга 1.3)

```
Используем переменную
```

Сценарий содержит три команды. Командой var txt объявляется переменная с названием txt. Командой txt="Используем переменную" переменной txt присваивается текстовое значение "Используем переменную". Наконец, с помощью команды document.write(txt) значение переменной txt отображается в рабочей области документа.

ДЕТАЛИ

При желании для большей наглядности в конце команд можно ставить точку с запятой. Если каждая команда находится в новой строке, этого можно не делать. Если несколько команд находятся в одной строке, они разделяются точкой с запятой.

Текстовые литералы заключаются в двойные или одинарные кавычки. Поэтому вместо выражения txt="Используем переменную" можно было использовать команду txt='Используем переменную'.

Объявление переменной можно совмещать с присваиванием переменной значения. Например, вместо команд var txt и txt="Используем переменную" мы могли бы использовать одну команду var txt="Используем переменную".

На рис. 1.6 показан веб-документ с результатом выполнения сценария.

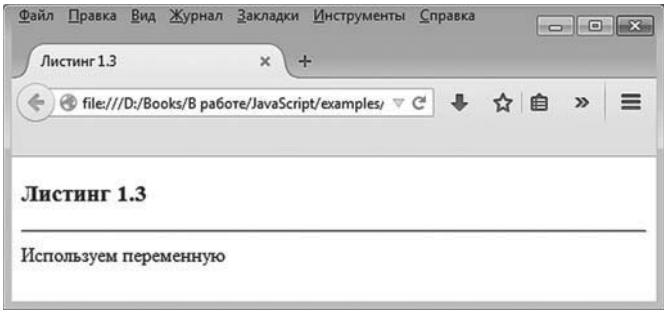

Рис. 1.6. Результат выполнения сценария, в котором использована переменная

Понятно, что переменных может быть больше, чем одна.

Сценарий с двумя переменными

Следующий пример иллюстрирует использование двух переменных. Код сценария приведен в листинге 1.4.

Листинг 1.4. Использование двух переменных (файл Listing01_04.js)

```
var txt,num
txt="Значение числа: "
num=123
document.write(txt+num)
```

Код сценария тестируем с помощью такого веб-документа:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Листинг 1.4</title>
</head>
<body><h3>Листинг 1.4</h3><hr>

<!-- Начало сценария -->
<script type="text/javascript" src="Listing01_04.js">
</script>
<!-- Завершение сценария -->

</body>
</html>
```

Результат выполнения сценария приведен ниже.

Результат выполнения сценария (из листинга 1.4)

```
Значение числа: 123
```

В окне браузера все выглядит так, как показано на рис. 1.7. Как и в предыдущем случае, здесь все достаточно просто. Командой `var txt,num` объявляются две переменные: одна называется `txt`, другая называется `num`. Командами `txt="Значение числа: "` и `num=123` переменным присваиваются

значения (текстовое переменной txt и числовое переменной num). После присваивания переменным значения командой document.write(txt+num) в рабочей области документа отображается текст. Здесь аргументом методу write() передано выражение txt+num, которым формально вычисляется сумма текстового значения и числового значения. Подобные операции обрабатываются так: число автоматически переводится в текстовый формат, и выполняется объединение (конкатенация) текстовых строк. Скажем, если значение переменной txt равно "Значение числа: ", а значение переменной num равно 123, то результатом выражения txt+num будет текст "Значение переменной: 123".

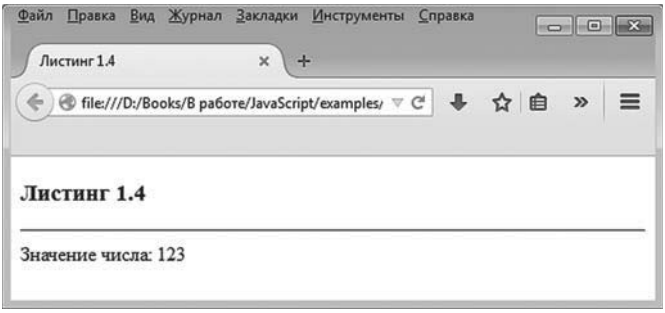

Рис. 1.7. Результат выполнения сценария с двумя переменными

Присваивание переменной значений разных типов

Практически такой же сценарий реализуем, с использованием не двух, а всего одной переменной. Сценарий приведен в листинге 1.5.

Листинг 1.5. Использование двух переменных (файл Listing01_05.js)

```
var x
x="Значение числа: "
document.write(x)
x=123
document.write(x)
```

В сценарии объявляется переменная x, которая сначала принимает текстовое значение, и это значение командой document.write(x) выводится в рабочее окно. Затем переменной x присваивается числовое значение, и снова в игру вступает команда document.write(x), благодаря которой текущее числовое значение переменной x также отображается в рабо-

чем окне (в той же строке, что и предыдущее сообщение). Чтобы проверить корректность работы данного сценария, файл со сценарием загружаем в веб-документ с помощью такого кода:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Листинг 1.5</title>
</head>
<body><h3>Листинг 1.5</h3><hr>

<!-- Начало сценария -->
<script type="text/javascript" src="Listing01_05.js">
</script>
<!-- Завершение сценария -->

</body>
</html>
```

Непосредственно результат выполнения сценария приведен ниже.

Результат выполнения сценария (из листинга 1.5)

```
Значение числа: 123
```

Соответствующий веб-документ в окне браузера выглядит так, как показано на рис. 1.8.

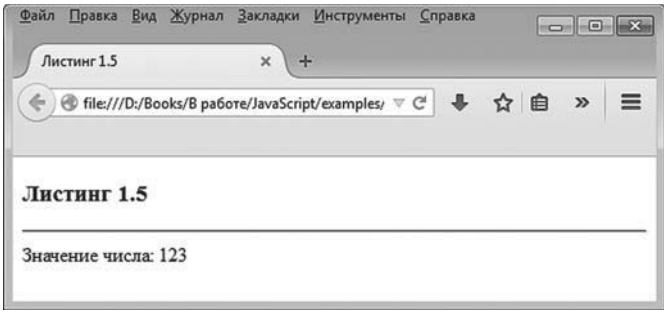

Рис. 1.8. Результат выполнения сценария с одной переменной, принимающей значения разных типов

В рассмотренном примере наиболее важный и показательный момент связан с возможностью присваивать переменной значения разных типов.

Вычисление выражений

Язык JavaScript очень гибкий в плане синтаксиса и структуры программного кода. В частности, в JavaScript есть очень полезная и эффективная функция eval().

Если аргументом функции передать текст, то результатом функции возвращается значение, которое получается при вычислении выражения, «спрятанного» в тексте. Например, выражение "3+(5*2+6)/4" является текстом. Но в этом тексте записано арифметическое выражение 3+(5*2+6)/4, которое имеет смысл: очевидно, речь идет о выражении $3+\frac{5\cdot 2+6}{4}$, которое равно 7.

Если воспользоваться командой eval("3+(5*2+6)/4"), то результат такой команды — значение выражения 3+(5*2+6)/4.

В листинге 1.6 приведен код сценария, в котором с помощью функции eval() вычисляется значение выражения, «упакованного» в текстовую строку.

Листинг 1.6. Вычисление выражения (файл Listing01_06.js)

```
var x="3 + (5*2 + 6) / 4"
document.write(x+" = ")
document.write(eval(x))
```

Здесь мы объявляем переменную x, а значением переменной присваивается текст "3 + (5*2 + 6) / 4" (для удобства восприятия между арифметическими операторами добавлены пробелы).

Командой document.write(x+" = ") данный текст (с добавленным знаком равенства) отображается в рабочем окне. А вот при выполнении команды document.write(eval(x)) отображается значение 7 (результат вычисления выражения в тексте).

Файл со сценарием загружаем в веб-документ, для чего используем следующий код:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Листинг 1.6</title>
</head>
<body><h3>Листинг 1.6</h3><hr>

<!-- Начало сценария -->
<script type="text/javascript" src="Listing01_06.js">
</script>
<!-- Завершение сценария -->

</body>
</html>
```

В итоге результат выполнения сценария такой.

Результат выполнения сценария (из листинга 1.6)

```
3 + (5*2 + 6) / 4 = 7
```

На рис. 1.9 показан веб-документ, открытый в браузере.

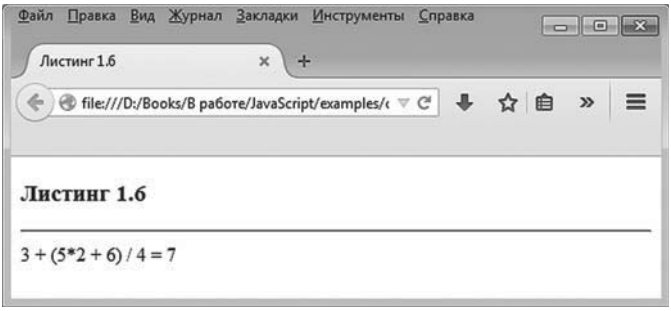

Рис. 1.9. Результат выполнения сценария с вычислением значения выражения

Понятно, что здесь проиллюстрирована довольно простая ситуация. Вместе с тем функция eval() находит самое широкое применение на практике. Особенно она полезна при реализации взаимодействия сценария с веб-документом.

Основные операторы

Живьём брать демонов!.. Живьём брать самозванцев!

из к/ф «Иван Васильевич меняет профессию»

Наряду с переменными в JavaScript используется достаточно много операторов. Обычно их разделяют на четыре группы:

- арифметические операторы;
- операторы сравнения;
- логические операторы;
- побитовые операторы.

Далее рассмотрим операторы каждой группы отдельно и еще кое-что.

Арифметические операторы

К *арифметическим* относятся операторы, предназначенные для выполнения арифметических действий. Арифметические операторы языка JavaScript описаны в табл. 1.1. Операнды у арифметических операторов, как правило, числовые (хотя могут быть и исключения — они обсуждаются отдельно). Все операторы, за исключением операторов *инкремента* и *декремента*, бинарные — у них по два операнда. Операторы инкремента и декремента унарные. Такие операторы используются с одним операндом.

НА ЗАМЕТКУ

У операторов инкремента и декремента есть постфиксная и префиксная формы. Результат вычисления выражений с такими операторами может зависеть от формы (префиксная или постфиксная) оператора.

В принципе многие из арифметических операторов достаточно точно соответствуют своим математическим аналогам, поэтому их назначение интуитивно понятно. Тем не менее имеются некоторые моменты, требующие пояснения.

Результат выражения `A%B` с оператором `%` вычисления остатка от деления рассчитывается как остаток от целочисленного деления зна-

чения операнда A на значение операнда B. Например, результатом выражения 13%5 будет 3. Объяснение такое: при делении 13 нацело на 5 получаем 2. Остаток от такого деления вычисляется как 13 – 5 · 2 = 3. Если операнды нецелые числа, принцип вычисления результата такой же. Скажем, результатом выражения 10.5%3.3 является значение 0.6, поскольку при целочисленном делении 10,5 на 3,3 получаем 3, а остаток от деления 10,5 – 3,3 · 3 = 10,5 – 9,9 = 0,6.

Таблица 1.1. Арифметические операторы JavaScript

Оператор	Описание
+	Оператор сложения. Результатом выражения A+B является сумма значений числовых операндов A и B
-	Оператор вычитания. Результатом выражения A-B является разность значений числовых операндов A и B
*	Оператор умножения. Результатом выражения A*B является произведение значений числовых операндов A и B
/	Оператор деления. Результатом выражения A*B является отношение значений числовых операндов A и B
%	Остаток от целочисленного деления. Результатом выражения A%B является остаток от деления нацело значения операнда A на значение операнда B. Операнды могут быть целыми или нецелыми числами
++	Оператор инкремента. В результате вычисления выражения A++ (постфиксная форма) или ++A (префиксная форма) операнд A увеличивает свое значение на 1. Таким образом, инструкция A++, равно как и инструкция ++A, эквивалентна команде A=A+1
--	Оператор декремента. При вычислении выражений --A (префиксная форма) и A-- (постфиксная форма) значение операнда A уменьшается на 1. Таким образом, каждая из инструкций --A или A-- эквивалентна команде A=A-1

НА ЗАМЕТКУ

Из-за ошибок округления при выполнении операций с числами с плавающей точкой реальный результат может несколько отличаться от «точного» значения.

Как отмечалось выше, у операторов инкремента ++ и декремента -- есть префиксная и постфиксные формы. В префиксной форме оператор указывается перед операндом (например, ++A или --A), а в постфиксной форме оператор указывается после операнда (например, A++ или A--). По отношению к значению операнда разницы в префиксной и постфиксной формах нет. Так, что в результате выполнения инс-