Владимир Дронов

JavaScript

20 уроков для начинающих

Санкт-Петербург
«БХВ-Петербург»
2021

УДК 004.438JavaScript
ББК 32.973.26-018.1
Д75

Дронов В. А.

Д75 JavaScript: 20 уроков для начинающих. — СПб.: БХВ-Петербург, 2021. — 352 с.: ил. — (Для начинающих)

ISBN 978-5-9775-6589-9

В книге 20 иллюстрированных уроков, 40 практических упражнений на тему программирования веб-сценариев и более 70 заданий для самостоятельной работы. Изложены основы JavaScript: данные и операторы, выражения и управляющие конструкции, функции, классы, объекты и массивы, средства отладки. Раскрыты механизмы управления веб-страницами: события и их обработка, управление элементами, графика и мультимедиа, веб-формы и элементы управления, регулярные выражения, навигация и управление окнами. Рассмотрена работа с HTML API и компонентное программирование: асинхронное программирование, работа с внешними данными, программная графика, объявление своих классов, создание компонентов. Освещены технологии взаимодействия с сервером: AJAX, PHP, разработка фронтендов и бэкендов, серверные сообщения.

Электронный архив на сайте издательства содержит коды всех примеров и результаты выполнения упражнений.

Для начинающих веб-разработчиков

УДК 004.438JavaScript
ББК 32.973.26-018.1

Группа подготовки издания:

Руководитель проекта	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Сависте*
Компьютерная верстка	*Ольги Сергиенко*
Дизайн обложки	*Карины Соловьевой*

Подписано в печать 07.07.20.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 28,38.
Тираж 1000 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-5-9775-6589-9

Оглавление

Введение

Здравствуйте, уважаемый читатель!

Если вы держите в руках эту книгу, значит, желаете научиться языку программирования JavaScript. Это очень правильное желание. А это очень правильная книга!

JavaScript — один из популярнейших в настоящее время языков для написания программ, а в деле веб-программирования у него вообще нет конкурентов. Знание JavaScript требуется от каждого, кто намеревается серьезно заняться написанием сайтов. Любой работодатель, узнавший, что претендент на должность веб-разработчика не владеет этим языком, тут же даст ему от ворот поворот.

Почему JavaScript?

Прежде всего, он — один из трех «китов», на которых зиждется величественный дворец веб-разработки. Первый «кит» — язык HTML — описывает саму веб-страницу, ее содержание, второй — CSS — ее оформление, а JavaScript — поведение.

JavaScript может «оживить» страницу, заставляя ее реагировать на действия пользователя. Он может превратить набор обычных картинок в красивую фотогалерею. Он может загрузить с сервера какие-либо данные и вывести их прямо на странице в виде абзаца, списка, таблицы или графика. Он даже может превратить страницу в настоящую программу, обрабатывающую занесенные пользователем данные и выводящую результаты их обработки.

JavaScript — единственный язык программирования, позволяющий «влезть» внутрь веб-страницы, исправить или дополнить ее содержание. JavaScript — единственный язык программирования, непосредственно поддерживаемый веб-обозревателями.

Ранее говорилось, что у этого языка в своей области нет конкурентов. И в ближайшем будущем они не появятся.

Что вы найдете в этой книге?

Философия программирования — это часть программирования, со всех сторон окруженная водой.

- Двадцать коротких, емких, наглядных, иллюстрированных уроков, дающих необходимые теоретические знания.
- Более сорока практических упражнений, выполняемых под руководством автора. Выполняя их, вы закрепите полученные знания и приобретете программистский опыт, что безусловно оценит ваш будущий работодатель.
- Восемнадцать упражнений, рассчитанных на самостоятельное выполнение. Чтобы справиться с ними, необходимо лишь внимательно изучать приведенный в уроках теоретический материал.
- Описание самых востребованных в настоящее время приемов программирования: объявления своих классов, написания компонентов, программирования фронтендов и бэкендов, использования фоновых процессов и серверных сообщений.
- Вводный курс популярной программной платформы PHP, знание которой очень пригодится веб-программисту.
- Полное отсутствие описаний устаревших и малополезных инструментов, отвлеченных рассуждений, философствований на тему программирования и прочей «воды».

> *Чтобы по этой книге овладеть JavaScript-программированием, вам хватит и месяца.*

Книгу сопровождает электронный архив (см. *приложение 3*), содержащий результаты выполнения всех упражнений, равно как и необходимые исходные файлы. Архив выложен на FTP-сервер издательства «БХВ-Петербург» по адресу **ftp://ftp.bhv.ru/9785977565899.zip**. Ссылка доступна и со страницы книги на сайте издательства **www.bhv.ru**.

Что вам понадобится?

Для изучения этой книги вам, уважаемый читатель, нужны следующие программы:

- *веб-обозреватель*, разумеется. Автор тестировал подготовленные им примеры в Google Chrome и Mozilla Firefox (остальные веб-обозреватели либо аналогичны Chrome, либо малопопулярны);
- *текстовый редактор* — можно использовать Блокнот, но автор рекомендует какой-либо из чисто «программистских» редакторов: Visual Studio Code (**https://code.visualstudio.com/**), Atom (**https://atom.io/**), Notepad++ (**https://notepad-plus-plus.org/**), Sublime Text (**https://www.sublimetext.com/**), Brackets (**http://brackets.io/**) и т. п.;

- *пакет веб-хостинга XAMPP* (**https://www.apachefriends.org/ru/index.html**). Автор применял версию 7.3.5, включающую веб-сервер Apache HTTP Server 2.4.39 и PHP 7.3.5. Описание пакета веб-хостинга XAMPP и инструкция по его установке на компьютер приведены в *приложении 2*.

Еще от вас требуется:

- владеть языками HTML и CSS, на которых пишутся сами веб-страницы и оформление для них;
- иметь минимальные навыки веб-верстки;
- знать в общих чертах как работает веб-сервер (это понадобится нам в *части IV* книги).

Типографские соглашения

В книге будут часто приводиться различные языковые конструкции, применяемые в JavaScript. Для наглядности при их написании использованы следующие типографские соглашения (в реальном коде они недействительны):

- HTML-, CSS-, JavaScript- и PHP-код набран моноширинным шрифтом:

```
<script type="text/javascript">
    const ins = window.prompt('Величина в дюймах', 0);
    const cents = ins * 2.54;
</script>
```

- в угловые скобки <> заключаются наименования различных значений, которые дополнительно выделяются *курсивом*. В реальный код, разумеется, должны быть подставлены реальные значения. Например:

```
return <возвращаемое значение>
```

  Здесь вместо подстроки *возвращаемое значение* должно быть подставлено реальное возвращаемое значение;

- в квадратные скобки [] заключаются необязательные фрагменты кода. Например:

```
toString([<основание>])
```

  Здесь *основание* может указываться, а может и не указываться;

- слишком длинные, не помещающиеся на одной строке книги фрагменты, автор разрывал на несколько строк и в местах разрывов ставил знаки ↳. Например:

```
active = document.querySelector↳
('nav a.active');
```

  Приведенный код здесь разбит на две строки, но должен быть набран в одну. Символ ↳ при этом нужно удалить;

- троеточием . . . помечены фрагменты кода, пропущенные ради сокращения объема текста книги:

```
function circleLength(d) {
    d = d || 2;
    . . .
}
```

Здесь весь код между второй и последней строками пропущен.

Обычно такое можно встретить в исправленных впоследствии фрагментах кода — приведены лишь собственно исправленные строки, а оставшиеся неизмененными пропущены. Также троеточие используется, чтобы показать, в какое место должен быть вставлен вновь написанный код: в начало исходного фрагмента, в его конец или в середину, между уже присутствующими в нем строками.

> ***Внимание!***
>
> Все приведенные здесь типографские соглашения имеют смысл только в примерах написания языковых конструкций JavaScript.

ЧАСТЬ I

Начала JavaScript

- Данные, которыми манипулирует JavaScript.
- Переменные.
- Операторы.
- Выражения.
- Управляющие конструкции.
- Функции.
- Объекты и классы.
- Веб-сценарии: внутренние и внешние.
- Средства отладки.
- Исключения и обработка ошибок.

Урок 1
Типы данных, операторы и переменные

Консоль веб-обозревателя
Типы данных
Числа
Строки
Логические значения
Операторы
Переменные
Преобразование типов

1.1. Упражнение. Используем консоль веб-обозревателя

Изучение основ языка JavaScript удобно начать, набирая код в консоли.

> *Консоль* — интерактивная исполняющая среда, встроенная в веб-обозреватель. В консоли можно набрать какое-либо выражение и сразу увидеть результат его выполнения.

Вызовем консоль у установленного у нас веб-обозревателя и выполним в ней несколько выражений.

1. Запустим веб-обозреватель (автор книги использовал Google Chrome).
2. Нажмем клавишу <F12>, чтобы вызвать встроенные в веб-обозреватель отладочные инструменты, в число которых входит и консоль. По умолчанию панель с отладочными инструментами появляется у нижней стороны окна веб-обозревателя (рис. 1.1).
3. В верхней части панели находятся корешки вкладок, на которых располагаются различные отладочные инструменты. Переключимся на вкладку **Console**, щелкнув на ней мышью (рис. 1.2).

   На этой вкладке и находится консоль веб-обозревателя (рис. 1.3).

   В белой области редактирования, занимающей бóльшую часть вкладки, виден мигающий текстовый курсор. Здесь вводятся выражения JavaScript, которые нужно выполнить.

- Для начала сложим два числа, для чего наберем в области редактирования следующий простой код:

```
27 + 14
```

и нажмем клавишу <Enter>.

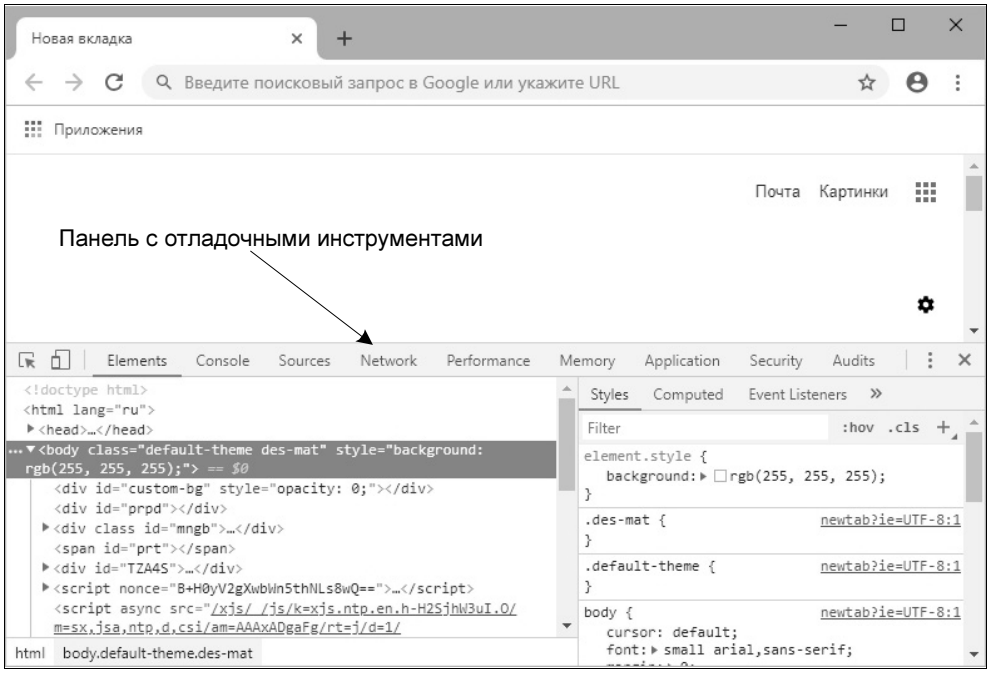

Рис. 1.1. Панель с отладочными инструментами веб-обозревателя

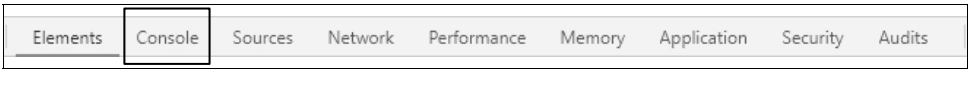

Рис. 1.2. Корешок вкладки **Console** на панели с отладочными инструментами веб-обозревателя

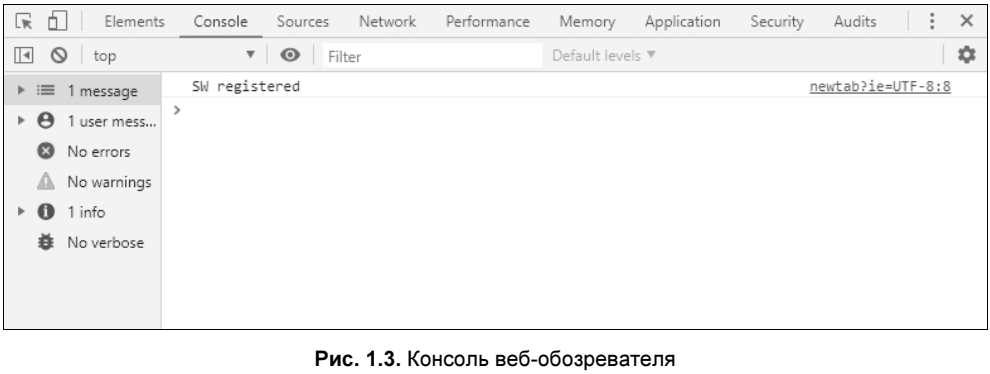

Рис. 1.3. Консоль веб-обозревателя

Ниже набранного кода мы увидим результат сложения чисел 27 и 14 (рис. 1.4).

Символом «больше» >, расположенным в консоли слева, помечается строка, в которую в настоящий момент будет вводиться код JavaScript.

Рис. 1.4. Результат сложения чисел 27 и 14 в консоли веб-обозревателя

Символом «меньше» < помечается строка, в которой выводится результат исполнения набранного кода после нажатия клавиши <Enter>.

- Мы можем обрабатывать и строки: давайте объединим строки `Java` и `Script` в одну:

```
> 'Java' + 'Script'
< "JavaScript"
```

Здесь символ + обозначает операцию объединения строк.

- Еще мы можем сравнивать значения: проверим, действительно ли число 10 меньше 20:

```
> 10 < 20
< true
```

Результатом будет логическое значение `true` — «истина». Так веб-обозреватель сообщает нам, что записанное сравнение выполняется (оно истинно).

- Напоследок для эксперимента выполним заведомо не выполняющееся, ложное сравнение:

```
> 15 > 25
< false
```

Результатом станет логическое значение `false` — «ложь».

4. Очистим консоль, щелкнув на кнопке **Clear console**, что находится в окне консоли слева, под набором корешков вкладок (рис. 1.5). Для очистки консоли можно также нажать комбинацию клавиш <Ctrl>+<L>.

Рис. 1.5. Кнопка очистки консоли **Clear console**

1.2. Типы данных и операторы

Любое значение, обрабатываемое JavaScript, относится к определенному типу данных.

> *Тип данных* — определяет характер значения и набор выполняемых над ним операций.

В *разд. 1.1* мы познакомились с тремя типами данных (табл. 1.1).

Таблица 1.1. *Три типа данных JavaScript*

Числовой	Строковый	Логический
Обычные числа	Строки из любых символов	Логические значения
`14`, `27`, `10000`	`'Java'`, `'Script'`, `'HTML и CSS'`	`true` — «истина» `false` — «ложь»

Там же, в *разд. 1.1*, для обработки значений мы использовали операторы.

> *Оператор* — выполняет элементарное действие над заданными значениями (*операндами*): сложение, объединение строк, сравнение и пр. — и производит выдачу (*возврат*) результата.

Примеры:

```
> 27 + 14
< 41
```
Оператор сложения `+` складывает операнды `27` и `14`
После чего возвращает результат — сумму `41`

```
> 10 < 20
< true
```
Оператор «меньше» `<` сравнивает операнды `10` и `20`
И возвращает результат — логическое значение `true`

Некоторые операторы JavaScript не принимают операндов и (или) не возвращают результат.

1.3. Числа, арифметические операторы и приоритет операторов

Числа в языке JavaScript записываются по обычным (школьным) правилам. Но в дробных числах (их еще называют *вещественными* и *числами с плавающей точкой*) в качестве десятичного разделителя применяется точка (не запятая!).

Примеры:

Целые числа: `10`, `20`, `1234567` Вещественные числа: `1.5`, `23.67`, `0.053`

У отрицательных чисел в начале ставится знак «минус»: `-`. Для записи чисел в экспоненциальной форме: <*мантисса*> × $10^{<порядок>}$ — применяется формат:

<*мантисса*>e<*порядок*>

Примеры:

Отрицательные числа:	Числа в экспоненциальной форме:
`-10`, `-20`, `-0.32`, `-0.053`	`1e6` (1000000), `1.3e-5` (0,000013), `-1e3` (–1000)

Помимо десятичных, можно записывать восьмеричные и шестнадцатеричные числа, предварив их, соответственно, нулем или символами `0x`.

Примеры:

Восьмеричные числа:	Шестнадцатеричные числа:
`010` (8), `0123` (83), `-045` (–37)	`0x10` (16), `0x53` (83), `-0x321` (–801)

Для выполнения арифметических действий применяются *арифметические операторы*.

> *Бинарные* (принимающие два операнда) операторы записываются в формате:
> *<операнд 1> <оператор> <операнд 2>*

Пояснение

Операторы бывают *унарные* (принимающие один операнд), *бинарные* (принимающие два операнда) и *тернарные* (принимающие три операнда). Обо всех их подробно рассказывается по мере их упоминания.

Поддерживаемые JavaScript арифметические операторы приведены в табл. 1.2.

Таблица 1.2. *Поддерживаемые JavaScript арифметические операторы*

Оператор	Описание	Пример	Результат
+ (плюс)	Сложение	`> 27 + 14`	`< 41`
– (минус)	Вычитание	`> 40 - 100`	`< -60`
* (звездочка)	Умножение	`> 12 * 32`	`< 384`
/ (слеш)	Деление	`> 3 / 4`	`< 0.75`
% (процент)	Остаток от деления	`> 4 % 3`	`< 1`
**	Возведение в степень	`> 2 ** 10`	`< 1024`

> Очередность выполнения операторов задается их *приоритетом*: операторы с бóльшим приоритетом выполняются раньше операторов с меньшим приоритетом.

Из рассмотренных ранее операторов наибольший приоритет имеет возведение в степень: `**`, операторы умножения: `*`, деления: `/` и остатка от деления: `%` — меньший, а сложения: `+` и вычитания: `-` — еще меньший (таблицу с приоритетами разных операторов можно найти в *приложении 1*).

- В следующем примере сначала будет выполнено деление `82` на `2`, а потом полученное частное сложится с `37`:

```
> 37 + 82 / 2
< 78
```

- Операторы с одинаковым приоритетом выполняются в порядке слева направо, поэтому в следующем примере сначала будет выполнено сложение `10` и `34`, а потом из суммы вычтется `26`:

```
> 10 + 34 - 26,
< 18
```

- Порядок выполнения операторов можно менять с помощью обычных круглых скобок. Оператор в круглых скобках будет выполнен раньше, независимо от его приоритета. Поэтому здесь сначала будет выполнено сложение `37` и `82`, а потом полученная сумма разделится на `2`:

```
> (37 + 82) / 2
< 59.5
```

К числам также относятся три специальных значения:

- `NaN` (Not a Number, не число) — значение, получаемое при попытке, например, умножить число на строку (о строках будет рассказано позже):

```
> 20 * 'JavaScript'
< NaN
```

- `Infinity` — математическая «бесконечность»: ∞. Это значение можно получить делением положительного числа на `0`:

```
> 10 / 0
< Infinity
```

- `-Infinity` — математическая «минус бесконечность»: –∞. Может быть получена делением отрицательного числа на `0`.

1.4. Строки. Конкатенация строк

> *Строки* заключаются в одинарные или двойные прямые кавычки (автор книги предпочитает одинарные).

Примеры строк: `'Java'`, `"Script"`, `'Языки HTML и CSS применяются в веб-разработке'`

Если строка заключена в одинарные кавычки, в ней не допускаются символы одинарной кавычки. Если строка заключена в двойные кавычки, в ней не допускаются символы двойной кавычки. В любом случае в строках не допускаются символы обратного слеша.

Примеры:

Правильно:	Неправильно:
`"Шеймус O'Брайен"`	`'Шеймус O'Брайен'`
`'Суши-бар "Йокогама"'`	`"Суши-бар "Йокогама""`

Указанные запреты можно обойти, применив *строковые литералы* (специальные последовательности символов): `\'`, `\"` — для кавычек и `\\` — для обратного слеша.

Примеры использования литералов:

```
'Шеймус О\'Брайен'
"Суши-бар \"Йокогама""
```

Последовательность литералов `\r\n` вставляет в строковое значение разрыв строки.

Оператор `+`, будучи примененным к операндам-строкам, выполняет объединение (*конкатенацию*) строк в одну.

Объединяем строки `'Java'` и `'Script'`:

```
> 'Java' + 'Script'
< "JavaScript"
```

1.5. Условия и операторы сравнения

Для того чтобы сравнить какие-либо значения, применяется условие.

> *Условие* — языковая конструкция, сравнивающая заданные значения. Записывается с помощью оператора сравнения.

> *Оператор сравнения* — сравнивает два операнда и возвращает результат в виде логического значения `true` («истина») или `false` («ложь»).

Операторы сравнения являются *бинарными* и записываются в формате:
<операнд 1> <оператор> <операнд 2>

Все поддерживаемые JavaScript операторы сравнения приведены в табл. 1.3.

Таблица 1.3. *Поддерживаемые JavaScript операторы сравнения*

Оператор	Описание	Пример	Результат
`==`	Равно[1]	`> 1 == 1.000`	`< true`
`!=`	Не равно	`> 1 != 1.001`	`< true`
`<`	Меньше	`> 1000 < 100`	`< false`
`<=`	Меньше или равно	`> 100 <= 1000`	`< true`
`>`	Больше	`> 10 > 9`	`< true`
`>=`	Больше или равно	`> 1e6 >= 1e9`	`< false`
`===`	Строго равно		
`!==`	Строго не равно		

Два последних оператора мы рассмотрим в конце урока, когда будем разбираться с преобразованием типов.

[1] Два знака «равно». Один знак «равно» — это совсем другой оператор, который будет рассмотрен позже.

1.6. Логические операторы

> *Логический оператор* — принимает логические операнды и возвращает результат также в виде логического значения.

В JavaScript имеются три логических оператора:

- `!` (*логическое отрицание*, или *логическое НЕ*) — *унарный*, возвращает `false`, если *операнд* имеет значение `true`, и `true` в противном случае. Записывается этот оператор в формате: `!<операнд>` без пробела. Пример:

```
> !(10 == 100)
< true
```

  В выражениях с оператором `!` операторы сравнения всегда следует брать в круглые скобки. Приоритет оператора `!` выше, чем у операторов сравнения, следовательно, он будет выполнен первым, и результат без скобок окажется неверным;

- `&&` (*логическое умножение*, или *логическое И*) — возвращает `true`, если оба *операнда* равны `true`, и `false` в противном случае;

- `||` (*логическое сложение*, или *логическое ИЛИ*) — возвращает `true`, если хотя бы один *операнд* равен `true`, и `false` в противном случае.

  Два последних оператора записываются в формате:

```
<операнд 1> <оператор && или ||> <операнд 2>
```

  Примеры:

```
> 10 <= 100 && 0 != 1
< true

> 10 >= 100 && 0 != 1
< false

> 10 <= 100 && 0 == 1
< false

> 10 >= 100 && 0 == 1
< false
```

```
> 10 <= 100 || 0 != 1
< true

> 10 >= 100 || 0 != 1
< true

> 10 <= 100 || 0 == 1
< true

> 10 >= 100 || 0 == 1
< false
```

1.7. Переменные

> *Переменная* — ячейка памяти, предназначенная для временного хранения какого-либо значения. Должна иметь уникальное имя, по которому выполняется обращение к этой переменной.

> *Имя переменной* может содержать буквы латиницы, цифры и символы подчеркивания _, причем оно **не** должно начинаться с цифры. Пробелы в имени переменной не допускаются.

Примеры:

Правильные имена:	Неправильные имена:
a, b, someValue, _private	число, some value, 234value

Перед первым использованием переменной ее следует объявить.

> *Объявление переменной* — указание создать переменную с заданным именем. Должно предшествовать первому использованию переменной.

Сразу при объявлении переменной она получает значение `undefined`, обозначающее отсутствие какой-либо значащей величины.

Оператор объявления переменных `let` объявляет переменные с указанными *именами*:

```
let <список имен объявляемых переменных через запятую>
```

В качестве результата оператор `let` всегда возвращает значение `undefined`.

Для объявления переменных можно использовать еще два оператора, которые мы рассмотрим в конце этого раздела.

Объявляем переменную с именем `sum` (и получим результат `undefined`):

```
> let sum
< undefined
```

> ***Примечание***
>
> В дальнейшем результат `undefined`, возвращаемый оператором `let`, ради компактности указываться не будет.

Объявляем сразу три переменные:

```
> let x, y, x
```

Объявив переменную, мы можем присвоить ей сохраняемое значение.

> *Присваивание* — занесение значения в переменную. Выполняется *оператором присваивания* =[1], имеющим формат:
> *<переменная> = <значение>*

Присваиваем только что объявленной переменной `sum` значение `10`:

```
> sum = 10
< 10
```

В качестве результата оператор = возвращает присвоенное переменной значение.

Можно совместить объявление переменной с присваиванием ей значения — так мы сократим код. Для этого в операторе `let` вместо имени переменной нужно указать оператор присваивания.

Объявляем переменную `a` и сразу же присваиваем ей число `1`:

```
> let a = 1
```

Сохраненное в переменной значение можно использовать в вычислениях, выполнив обращение к этой переменной.

---

[1] Оператор присваивания обозначается одним знаком «равно» (=), а оператор сравнения «равно» — двумя (==). Не путайте эти два оператора!

> *Обращение к переменной* — указание извлечь хранящееся в переменной значение и использовать его в качестве операнда какого-либо оператора или иным образом (например, для вывода на экран).

Обращение к переменной выполняется простым указанием имени нужной переменной в качестве операнда.

Присваиваем переменной `a` произведение значения переменной `sum` (в текущий момент: `10`) и числа `20`:

```
> a = sum * 20
< 200
```

> ***Внимание!***
> При присваивании переменной нового значения старое значение теряется.

Прибавляем к значению переменной `sum` (в текущий момент: `10`) число `20` и заносим получившуюся сумму в ту же переменную:

```
> sum = sum + 20
< 30
```

> *Оператор комбинированного присваивания* (табл. 1.4) извлекает из указанной *переменной* значение, выполняет над ним и заданным *операндом* какое-либо действие и присваивает результат той же *переменной*. Формат этого оператора:
> *<переменная> <оператор> <операнд>*

Таблица 1.4. *Операторы комбинированного присваивания*

Оператор комбинированного присваивания	Его аналог
`a += b`	`a = a + b`
`a -= b`	`a = a - b`
`a *= b`	`a = a * b`
`a /= b`	`a = a / b`
`a %= b`	`a = a % b`

Прибавим к переменной `sum` (сейчас там `30`) число `70`:

```
> sum += 70
< 100
```

Операторы комбинированного присваивания выполняются быстрее аналогичных комбинаций арифметического оператора и обычного оператора присваивания.

1.7.1. Три оператора объявления переменных

Язык JavaScript поддерживает три *оператора объявления переменных*: уже знакомый нам `let`, а также `const` и `var`. Они имеют один и тот же формат записи, но разные особенности:

- `let` — объявляет обычную переменную, которой можно присваивать значения сколько угодно раз. Повторное объявление переменной этим оператором не допускается — оно вызовет ошибку:

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

  Здесь в последних двух строках выведено сообщение об ошибке;

- `const` — объявляет *константу* — переменную, значение которой можно присвоить лишь один раз и лишь непосредственно при ее объявлении. Попытка присвоить константе новое значение вызовет ошибку. Пример:

  ```
  > const y = 2
  ```

  Если значение переменной не меняется в дальнейшем, рекомендуется объявлять ее как константу, поскольку константы обрабатываются веб-обозревателем быстрее;

- `var` — полностью аналогичен оператору `let`, однако позволяет объявить переменную повторно (в этом случае он ничего не делает). Есть еще один нюанс, который мы рассмотрим в *уроке 2*.

  Оператор `var` остался от старых версий JavaScript. В настоящее время его применение не рекомендовано, однако он часто встречается в старом JavaScript-коде.

1.7.2. Еще об объявлении переменных

Объявлять переменные перед их использованием необязательно. Можно просто присвоить переменной какое-либо значение — и веб-обозреватель сам неявно объявит эту переменную:

```
sum = 10
```

Однако все же лучше объявлять переменные явно. Во-первых, это хороший стиль программирования, а во-вторых, при работе в строгом режиме (о котором будет рассказано в *уроке 2*) веб-обозреватель будет требовать явного объявления переменных.