Посвящается Чи – самому важному человеку в моей жизни

Содержание

Предисловие 12
Об авторе 16
От издательства 17


Глава 1. Значения и переменные 18
1.1. Запуск JavaScript 18
1.2. Типы и оператор typeof 20
1.3. Комментарии 21
1.4. Объявления переменных 22
1.5. Идентификаторы 23
1.6. Числа 24
1.7. Арифметические операторы 25
1.8. Булевы значения 27
1.9. null и undefined 27
1.10. Строковые литералы 28
1.11. Шаблонные литералы 30
1.12. Объекты 31

1.13. Синтаксис объектного литерала 32
1.14. Массивы 33
1.15. JSON 34

1.16. Деструктуризация 35

1.17. Еще о деструктуризации 37
1.17.1. Дополнительные сведения о деструктуризации объектов 37
1.17.2. Объявление прочих 38
1.17.3. Значения по умолчанию 38
Упражнения 39


Глава 2. Управляющие конструкции 40
2.1. Выражения и предложения 40
2.2. Вставка точки с запятой 41
2.3. Ветвления 44

2.4. Булевость 46
2.5. Сравнение 46

2.6. Смешанное сравнение 48
2.7. Логические операторы 49

2.8. Предложение switch 51

2.9. Циклы while и do 51
2.10. Циклы for 52
2.10.1. Классический цикл for 52
2.10.2. Цикл for of 53
2.10.3. Цикл for in 54

2.11. Break и continue 55
2.12. Перехват исключений 57
Упражнения 58


Глава 3. Функции и функциональное программирование 60
3.1. Объявление функций 60
3.2. Функции высшего порядка 61
3.3. Функциональные литералы 62
3.4. Стрелочные функции 63
3.5. Функциональная обработка массива 64
3.6. Замыкания 65

3.7. Крепкие объекты 67
3.8. Строгий режим 69
3.9. Проверка типов аргументов 70
3.10. Передача большего или меньшего числа аргументов 71
3.11. Аргументы по умолчанию 72
3.12. Прочие параметры и оператор расширения 73

3.13. Имитация именованных аргументов с помощью деструктуризации 74

3.14. Поднятие 75
3.15. Возбуждение исключений 77

3.16. Перехват исключений 78

3.17. Ветвь finally 79
Упражнения 80


Глава 4. Объектно-ориентированное программирование 83
4.1. Методы 83
4.2. Прототипы 84
4.3. Конструкторы 87
4.4. Синтаксис классов 88

4.5. Акцессоры чтения и записи 89

4.6. Поля экземпляра и закрытые методы 90

4.7. Статические методы и поля 91
4.8. Подклассы 92
4.9. Переопределение методов 94
4.10. Конструирование подкласса 95

4.11. Классовые выражения 95

 4.12. Ссылка this 96
Упражнения 99

 **Глава 5. Числа и даты** 102
5.1. Числовые литералы 102
5.2. Форматирование чисел 103
5.3. Разбор чисел 103
5.4. Функции и константы в классе Number 104
 5.5. Математические функции и константы 105
 5.6. Большие целые 106
 5.7. Конструирование дат 107
 5.8. Функции и методы класса Date 110
 5.9. Форматирование дат 111
Упражнения 111

 **Глава 6. Строки и регулярные выражения** 114
6.1. Преобразование между строками и последовательностями кодовых точек 114
6.2. Подстроки 115
6.3. Прочие методы класса String 116
 6.4. Тегированные шаблонные литералы 119
 6.5. Простые шаблонные литералы 120
 6.6. Регулярные выражения 121
 6.7. Литеральные регулярные выражения 124
 6.8. Флаги 125
 6.9. Регулярные выражения и Юникод 126
 6.10. Методы класса RegExp 127
 6.11. Группы 128
 6.12. Методы класса String для работы с регулярными выражениями ... 130
 6.13. Еще о методе `replace` 132
 6.14. Экзотические возможности 133
Упражнения 134

 **Глава 7. Массивы и коллекции** 137
7.1. Конструирование массива 137
7.2. Свойство length и индексные свойства 138
7.3. Удаление и добавление элементов 139

7.4. Прочие методы изменения массива 141
7.5. Порождение элементов 143
7.6. Поиск элементов 144
7.7. Перебор всех элементов 145

7.8. Разреженные массивы 147

7.9. Редукция 148
7.10. Отображения 151
7.11. Множества 153

7.12. Слабые отображения и множества 154

7.13. Типизированные массивы 155

7.14. Буферные массивы 157
Упражнения 158


Глава 8. Интернационализация 161
8.1. Понятие локали 161
8.2. Задание локали 162
8.3. Форматирование чисел 164
8.4. Локализация даты и времени 166
8.4.1. Форматирование объектов Date 166
8.4.2. Диапазоны 167
8.4.3. Относительное время 167
8.4.4. Форматирование с точностью до отдельных частей 168
8.5. Порядок следования 168
8.6. Другие методы класса String, чувствительные к локали 170

8.7. Правила образования множественного числа и списков 171

8.8. Различные средства, относящиеся к локалям 173
Упражнения 174


Глава 9. Асинхронное программирование 176
9.1. Конкурентные задачи в JavaScript 176
9.2. Создание обещаний 179
9.3. Немедленно улаживаемые обещания 181
9.4. Получение результата обещания 182
9.5. Сцепление обещаний 182
9.6. Обработка отвергнутых обещаний 184
9.7. Выполнение нескольких обещаний 185
9.8. Гонка нескольких обещаний 186
9.9. Асинхронные функции 187
9.10. Асинхронно возвращаемые значения 189
9.11. Конкурентное ожидание 191
9.12. Исключения в асинхронных функциях 191
Упражнения 192


Глава 10. Модули 196
10.1. Понятие модуля 196
10.2. Модули в ECMAScript 197
10.3. Импорт по умолчанию 197
10.4. Именованный импорт 198

10.5. Динамический импорт 199
10.6. Экспорт 200
10.6.1. Именованный экспорт 200
10.6.2. Экспорт по умолчанию 201
10.6.3. Экспортируемые средства – это переменные 202
10.6.4. Реэкспорт 202
10.7. Упаковка модулей 203
Упражнения 204


Глава 11. Метапрограммирование 207
11.1. Символы 207
11.2. Настройка с помощью символьных свойств 208
11.2.1. Настройка метода toString 209
11.2.2. Управление преобразованием типов 210
11.2.3. Символ Species 210
11.3. Атрибуты свойств 211
11.4. Перечисление свойств 213
11.5. Проверка наличия свойства 215
11.6. Защита объектов 215
11.7. Создание и обновление объектов 216
11.8. Доступ к прототипу и его обновление 216
11.9. Клонирование объектов 217
11.10. Свойства-функции 220
11.11. Привязка аргументов и вызов методов 221
11.12. Прокси 222
11.13. Класс Reflect 224
11.14. Инварианты прокси 226
Упражнения 228


Глава 12. Итераторы и генераторы 232
12.1. Итерируемые значения 232
12.2. Реализация итерируемого объекта 233
12.3. Закрываемые итераторы 235
12.4. Генераторы 236
12.5. Вложенное yield 238
12.6. Генераторы как потребители 240
12.7. Генераторы и асинхронная обработка 241
12.8. Асинхронные генераторы и итераторы 243
Упражнения 246

 **Глава 13. Введение в TypeScript** 249
13.1. Аннотации типов 250
13.2. Запуск TypeScript 251
13.3. Терминология, относящаяся к типам 252
13.4. Примитивные типы 253
13.5. Составные типы 254
13.6. Выведение типа 256
13.7. Подтипы 259
13.7.1. Правило подстановки 259
13.7.2. Факультативные и лишние свойства 261
13.7.3. Вариантность типов массива и объекта 262
13.8. Классы 263
13.8.1. Объявление классов 263
13.8.2. Тип экземпляра класса 264
13.8.3. Статический тип класса 265
13.9. Структурная типизация 266
13.10. Интерфейсы 267
 13.11. Индексные свойства 268
 13.12. Более сложные параметры функций 269
13.12.1. Факультативные, подразумеваемые по умолчанию и прочие параметры 269
13.12.2. Деструктуризация параметров 270
13.12.3. Вариантность типа функции 271
13.12.4. Перегрузка 273
 13.13. Обобщенное программирование 275
13.13.1. Обобщенные классы и типы 275
13.13.2. Обобщенные функции 276
13.13.3. Ограничения на типы 277
13.13.4. Стирание 278
13.13.5. Вариантность обобщенных типов 279
13.13.6. Условные типы 280
13.13.7. Отображаемые типы 281
Упражнения 282

Предметный указатель 285

Предисловие

Опытные программисты, знакомые с такими языками, как Java, C#, C или C++, нередко оказываются в ситуации, когда необходимо поработать с JavaScript. Пользовательские интерфейсы все чаще размещаются в вебе, а JavaScript – язык, поддерживаемый всеми браузерами. Каркас Electron распространяет эту возможность на обогащенные клиентские приложения, и существует несколько решений для создания мобильных JavaScript-приложений. К тому же JavaScript все активнее проникает и в серверное программирование.

Много лет назад JavaScript задумывался как язык для «простенького программирования», набор включенных в него средств приводил в замешательство и мог спровоцировать ошибки в больших программах. Однако предпринятые усилия по стандартизации и созданный инструментарий далеко превзошли первоначальные скромные задумки.

К сожалению, довольно трудно изучить современный JavaScript, не увязнув в трясине старых версий. Целью большинства книг, курсов и статей в блогах является переход от прежних версий JavaScript на современную, что не слишком полезно пришельцам с других языков.

Именно эту проблему призвана решить данная книга. Я предполагаю, что читатель – знающий программист, который понимает, что такое ветвления, циклы, функции, структуры данных, и знаком с основами объектно-ориентированного программирования. Я объясню, что значит быть продуктивным программистом на современном JavaScript, лишь в скобках упоминая об ушедших в прошлое средствах. Вы узнаете, как поставить себе на службу современный JavaScript, избежав древних ловчих ям.

JavaScript, быть может, и не идеален, но, как показывает практика, хорошо приспособлен для программирования пользовательских интерфейсов и многих серверных задач. Как прозорливо заметил Джефф Этвуд, «любое приложение, которое *можно* написать на JavaScript, в конце концов *будет* написано на JavaScript».

Проработав эту книгу, вы сумеете написать следующую версию своего приложения на современном JavaScript!

Пять золотых правил

Держась подальше от немногих «классических» средств JavaScript, вы сможете заметно упростить себе освоение и использование языка. Возможно, прямо сейчас эти правила покажутся вам бессмысленными, но я все же приведу их, чтобы ссылаться в дальнейшем. И не пугайтесь – их совсем немного.

1. При объявлении переменных употребляйте ключевые слова `let` или `const`, а не `var`.
2. Пользуйтесь строгим режимом.

3. Обращайте внимание на типы и избегайте автоматического преобразования типов.
4. Разберитесь, что такое прототипы, но для работы с классами, конструкторами и методами применяйте современный синтаксис.
5. Не используйте ключевое слово `this` вне конструкторов и методов.

И еще одно метаправило: *избегайте «что это?!»* – фрагментов странного JavaScript-кода, сопровождаемых саркастическим «Что это?!». Некоторым доставляет удовольствие демонстрировать якобы ужасы JavaScript, анатомируя запутанный код. Я ни разу не почерпнул ничего полезного, спускаясь в эту кроличью нору. Зачем, к примеру, знать, что `2 * ['21']` равно 42, а `2 + ['40']` не равно, если золотое правило 3 призывает избегать преобразований типов? В общем случае, оказываясь в сбивающей с толку ситуации, я задаю себе вопрос, как избежать ее, а не как объяснить ее таинственные, но бесполезные детали.

Пути к познанию

Работая над книгой, я старался помещать информацию туда, где вы сможете найти ее, когда понадобится. Но это необязательно самое подходящее место при первом прочтении книги. Чтобы помочь вам проложить собственный путь к познанию, я пометил каждую главу значком, обозначающим ее базовый уровень сложности. Разделы же, более сложные, чем глава в целом, помечены собственными значками. Вы можете без опаски пропускать такие разделы и возвращаться к ним, когда будете готовы воспринять материал.

Вот эти значки.

Нетерпеливый кролик означает тему **начального** уровня, пропускать которую не должен даже самый нетерпеливый читатель.

Алиса обозначает тему среднего уровня, с которой стоило бы познакомиться большинству программистов, но, возможно, не при первом чтении.

Чеширский кот обозначает тему **повышенного** уровня, от которой расплывется в улыбке лицо разработчика каркасов. Большинство прикладных программистов могут спокойно пропустить эти разделы.

Наконец, значок Безумного шляпника сопровождает **сложную**, способную свести с ума тему, предназначенную только для тех, кто одержим нездоровым любопытством.

Краткое содержание книги

В главе 1 рассказывается об основных понятиях JavaScript: значениях и их типах, переменных и, самое важное, объектных литералах. В главе 2 описы-

вается поток управления. Если вы знакомы с Java, C# или C++, можете пролистать ее по диагонали. В главе 3 вы узнаете о функциях и функциональном программировании – вещи, крайне важной в JavaScript. Объектная модель в JavaScript сильно отличается от языков программирования, основанных на классах. Глава 4 посвящена деталям с упором на современный синтаксис. В главах 5 и 6 описаны библиотечные классы, которые чаще всего используются при работе с числами, датами, строками и регулярными выражениями. В основном это материал начального уровня, но встречаются разделы повышенного типа.

Следующие четыре главы посвящены темам промежуточного уровня. В главе 7 вы научитесь работать с массивами и другими коллекциями, имеющимися в стандартной библиотеке JavaScript. Если ваша программа рассчитана на пользователей со всего мира, то обратите особое внимание на вопросы интернационализации, которые освещаются в главе 8. Глава 9 об асинхронном программировании чрезвычайно важна для всех программистов. Асинхронное программирование на JavaScript когда-то считалось весьма сложным предметом, но после включения в язык обещаний и ключевых слов `async` и `await` значительно упростилось. Теперь в JavaScript имеется стандартная система модулей, которая описывается в главе 10. Вы узнаете, как использовать модули, написанные другими программистами, и как создать свой собственный.

В главе 11 рассматривается метапрограммирование на повышенном уровне. Читать ее имеет смысл, если вы собираетесь создать инструмент для анализа и преобразования произвольных JavaScript-объектов. В главе 12 описание JavaScript завершается рассмотрением еще одной продвинутой темы: итераторов и генераторов – мощных механизмов, предназначенных для организации обхода коллекций и порождения произвольных последовательностей значений.

Наконец, имеется дополнительная глава 13, посвященная TypeScript. TypeScript – это надмножество JavaScript, добавляющее проверку типов на этапе компиляции. Не будучи частью стандартного JavaScript, эта надстройка очень популярна. Прочитайте эту главу и сами решите, к чему склоняетесь: к обычному JavaScript или к системе типов на этапе компиляции.

Цель данной книги – заложить прочные основы для уверенного использования самого *языка* JavaScript. За информацией о постоянно изменяющихся инструментах и каркасах придется обратиться в другое место.

Почему я написал эту книгу

JavaScript – один из самых широко распространенных языков программирования на планете. Как и многие программисты, я был знаком с *ломаным* JavaScript, однако настал день, когда нужно было спешно научиться писать на JavaScript по-серьезному. Но как?

Есть немало учебников по основам JavaScript для непрофессиональных веб-разработчиков, но на таком уровне я его и так знал. *Книга с носорогом*

Флэнагана[1] была чудом в 1996 году, но теперь нагружает читателей слишком большим наследием прошлого. Книга Крокфорда «JavaScript: The Good Parts»[2] стала сигналом к действию в 2008-м, но многие содержащиеся в ней призывы уже вошли в последующие версии языка. Существует множество книг, помогающих программистам на JavaScript старой школы вступить в мир новых стандартов, но в них предполагается хорошее знакомство с «классическим» JavaScript, чем я похвастаться не мог.

Разумеется, веб кишит блогами на тему JavaScript разного качества – одни содержат точную и систематическую информацию, другие – просто случайный набор фактов. На мой взгляд, просеивать веб-блоги и оценивать степень их достоверности – занятие не слишком эффективное. Как ни странно, я не смог найти ни одной книги для миллионов программистов, которые знают Java или другой подобный язык и хотят изучить JavaScript в его современном виде, не обремененном историческим багажом.

Поэтому мне пришлось написать ее самому.

Благодарности

Я хотел бы еще раз поблагодарить своего редактора Грега Денча (Greg Doench) за поддержку этого проекта, а также Дмитрия и Алину Кирсановых за корректуру и верстку книги. Отдельное спасибо рецензентам Гэйлу Андерсону (Gail Anderson), Тому Остину (Tom Austin), Скотту Дэвису (Scott Davis), Скотту Гуду (Scott Good), Кито Манну (Kito Mann), Бобу Николсону (Bob Nicholson), Рону Маку (Ron Mak) и Генри Тремблею (Henri Tremblay), которые исправно указали на ошибки и внесли продуманные предложения по улучшению книги.

Кэй Хорстманн
Берлин
Март 2020

[1] *David Flanagan.* JavaScript: The Definitive Guide. Sixth Edition (O'Reilly Media, 2011).

[2] Вышла в издательстве O'Reilly Media в 2008 году.

Об авторе

Кэй С. Хорстманн – главный автор книг «Core Java™», т. I и II, 11-е изд. (Pearson, 2018), «Scala for the Impatient», 2-е изд. (Addison-Wesley, 2016) и «Core Java SE 9 for the Impatient» (Addison-Wesley, 2017). Кэй – заслуженный профессор информатики в университете Сан-Хосе, пропагандист Java, часто выступает на конференциях по компьютерной тематике.

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Springer очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Глава 1

Значения и переменные

В этой главе вы узнаете о типах данных в JavaScript-программе: числах, строках и других примитивных типах, а также об объектах и массивах. Вы увидите, как сохранять значения в переменных, как преобразовывать значения из одного типа в другой и как применять к значениям операторы для получения новых значений.

Даже самые отчаянные программисты на JavaScript согласятся, что некоторые языковые конструкции – задуманные для того, чтобы сократить размер программ, – могут давать интуитивно неочевидные результаты, поэтому их лучше избегать. В этой и следующих главах я буду обращать внимание на такие проблемы и сформулирую простые правила безопасного программирования.

1.1. Запуск JavaScript

Выполнять встречающиеся в этой книге программы можно несколькими способами.

Изначально задумывалось, что JavaScript будет выполняться внутри браузера. Можно встроить JavaScript-код в HTML-файл и, вызвав метод `window.alert`, отобразить значения. Вот пример такого файла:

```
<html>
  <head>
    <title>My First JavaScript Program</title>
    <script type="text/javascript">
      let a = 6
      let b = 7
      window.alert(a * b)
    </script>
  </head>
  <body>
  </body>
</html>
```

Просто откройте этот файл в своем любимом браузере – и в диалоговом окне увидите результат (см. рис. 1.1).

Рис. 1.1 ❖ Выполнение JavaScript-кода в браузере

Можно набрать короткую последовательность команд на консоли, которая является частью комплекта средства разработки, входящего в состав браузера. Чтобы открыть средства разработки, нажмите соответствующую клавишу или выберите пункт из меню (во многих браузерах это клавиша **F12** или комбинация **Ctrl**+**Alt**+**I**, а в Mac – **Cmd**+**Alt**+**I**). Затем перейдите на вкладку **Console** и введите свой JavaScript-код (рис. 1.2).

Рис. 1.2 ❖ Выполнение JavaScript-кода на консоли разработчика

Третий способ – установить Node.js с сайта http://nodejs.org. Затем откройте терминал и запустите программу `node`, которая входит в цикл «читать-выполнять-печатать» (цикл REPL). Вводите команды и смотрите на их результаты (рис. 1.3).

Рис. 1.3 ❖ Выполнение JavaScript-кода на консоли разработчика

Если последовательность команд длиннее, поместите ее в файл и вызовите метод `console.log`, чтобы увидеть результат. Например, поместите следующие команды в файл `first.js`:

```
let a = 6
let b = 7
console.log(a * b)
```

Затем выполните команду

```
node first.js
```

Результат команды `console.log` будет выведен в окно терминала.

Можно также воспользоваться средой разработки, например Visual Studio Code, Eclipse, Komodo или WebStorm. Все они позволяют редактировать и выполнять JavaScript-код, как показано на рис. 1.4.

1.2. Типы и оператор typeof

Значения в JavaScript могут иметь следующие типы:

- число;
- булево значение `false` или `true`;
- специальные значения `null` и `undefined`;
- строка;
- символ;
- объект.

Все типы, кроме объекта, собирательно называются *примитивными*.

Подробнее об этих типах рассказано в следующих разделах, за исключением символов, о которых речь пойдет в главе 11.

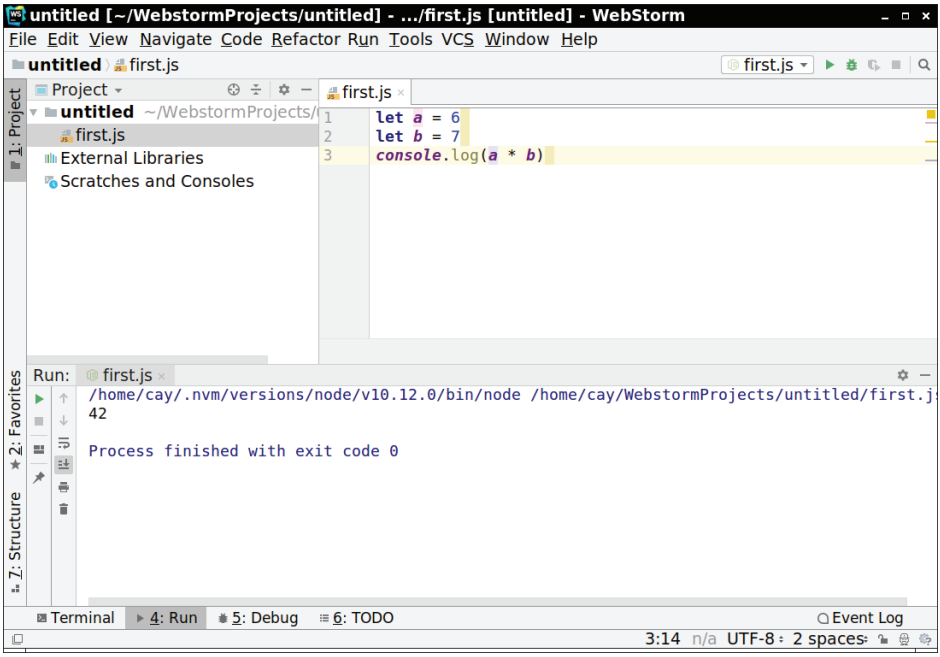

Рис. 1.4 ❖ Выполнение JavaScript-кода в среде разработки

Чтобы узнать тип значения, следует воспользоваться оператором `typeof`, который возвращает одну из строк `'number'`, `'boolean'`, `'undefined'`, `'object'`, `'string'`, `'symbol'` и еще нескольких. Например, вызов `typeof 42` возвращает строку `'number'`.

> **Примечание.** Хотя тип `null` отличается от типа `object`, значением `typeof null` является строка `'object'`. Так сложилось исторически.

> **Предостережение.** Как и в Java, можно сконструировать объекты, обертывающие числа, булевы значения и строки. Например, `typeof new Number(42)` и `typeof new String('Hello')` возвращают `'object'`. Однако в JavaScript нет причин конструировать такие обертки. Поскольку подобные действия могут приводить к недоразумениям, стандарты кодирования нередко их явно запрещают.

1.3. Комментарии

В JavaScript есть два вида комментариев. Однострочный комментарий начинается двумя литерами // и продолжается до конца строки, например:

```
// как-то так
```

Комментарии, заключенные между парами литер /* и */, могут занимать несколько строк, например:

```
/*
  как-то
  так
*/
```

В этой книге комментарии набраны моноширинным шрифтом для простоты восприятия. Понятно, что в текстовом редакторе они, скорее всего, будут выделены цветом.

> **Примечание.** В отличие от Java, в JavaScript нет специальных комментариев для оформления документации. Однако существуют сторонние инструменты, например JSDoc (http://usejsdoc.org), предлагающие аналогичную функциональность.

1.4. Объявления переменных

Для сохранения значения в переменной служит предложение `let`:

```
let counter = 0
```

В JavaScript у переменных нет типа. В любой переменной можно сохранить значение любого типа. Например, допустимо заменить содержимое `counter` строкой:

```
counter = 'zero'
```

Почти никогда так делать не стоит. Но бывают ситуации, когда наличие нетипизированных переменных упрощает написание обобщенного кода, работающего с разными типами.

Если переменная явно не инициализирована, то она принимает специальное значение `undefined`:

```
let x // объявляет x и присваивает ей значение undefined
```

> **Примечание.** Вы, наверное, обратили внимание, что предложения не завершаются точкой с запятой. В JavaScript, как и в Python, точка с запятой в конце строки необязательна. В Python даже считается неподобающим добавлять ненужные точки с запятой. Но программисты на JavaScript по этому вопросу разделились на два лагеря. Мы обсудим аргументы за и против в главе 2. Вообще-то, я стараюсь не занимать ничью сторону в пустопорожних спорах, но в этой книге мне пришлось выбрать что-то одно. Я остановился на стиле «без точки с запятой» по одной простой причине: чтобы код не был похож на Java или C++. Глядя на фрагмент кода, можно сразу сказать, что он написан на JavaScript.

Если значение переменной не планируется изменять, то следует объявить ее в предложении `const`:

```
const PI = 3.141592653589793
```

Попытка модифицировать так объявленное значение приведет к ошибке во время выполнения.

В одном предложении const или let можно объявить несколько переменных:

```
const FREEZING = 0, BOILING = 100
let x, y
```

Но многие программисты предпочитают объявлять каждую переменную в отдельной строке.

> **Предостережение.** Избегайте двух устаревших форм объявления переменных: с помощью ключевого слова var и вообще без ключевого слова:
>
> ```
> var counter = 0 // устаревшая форма
> coutner = 1 // обратите внимание на опечатку – будет создана новая переменная!
> ```
>
> У объявления с помощью var есть несколько серьезных недостатков, о них будет сказано в главе 3. Создание при первом присваивании, очевидно, опасно. Если сделать опечатку в имени переменной, то будет создана новая переменная. По этой причине такое поведение считается ошибкой в *строгом режиме*, запрещающем устаревшие конструкции. В главе 3 я расскажу, как включить строгий режим.
>
> **Совет.** В предисловии я перечислил пять золотых правил, следование которым позволит устранить большую часть недоразумений, вызванных «классическими» средствами JavaScript. Первые два из них гласят:
>
> 1. При объявлении переменных употребляйте ключевые слова let или const, а не var.
> 2. Пользуйтесь строгим режимом.

1.5. Идентификаторы

Имя переменной должно быть выбрано с соблюдением общего синтаксиса *идентификаторов*. Идентификатор может включать буквы Юникода, цифры и литеры _ и $. Первая литера не должна быть цифрой. Имена, включающие литеру $, иногда используются в библиотеках и инструментальных средствах. Некоторые программисты применяют идентификаторы, начинающиеся или заканчивающиеся знаком подчеркивания, чтобы показать, что речь идет о «закрытых» членах. В своих именах лучше избегать использования $, а также _ в начале и в конце. Подчерки внутри имени не вызывают никаких нареканий, но многие JavaScript-программисты предпочитают «верблюжью нотацию» camelCase, когда границы слов обозначаются сменой регистра.

Следующие ключевые слова не разрешается использовать в качестве идентификаторов:

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

В строгом режиме запрещены также такие ключевые слова:

```
implements interface let package protected private public static
```

Следующие ключевые слова добавлены в язык недавно; их можно использовать в качестве идентификаторов ради обратной совместимости, но лучше этого не делать:

```
await as async from get of set target yield
```

> **Примечание.** В идентификаторах разрешено использовать любые буквы и цифры Юникода, например:
>
> `const π = 3.141592653589793`
>
> Однако это не принято, потому что у многих программистов нет метода ввода таких литер.

1.6. Числа

В JavaScript нет явного типа целого числа. Все числа с плавающей точкой двойной точности. Конечно, можно использовать и целые значения, просто не обращайте внимания на разницу между `1` и `1.0` (к примеру). А как насчет округления? Целые числа в диапазоне от `Number.MIN_SAFE_INTEGER` ($-2^{53} + 1$, или `-9 007 199 254 740 991`) и `Number.MAX_SAFE_INTEGER` ($2^{53} - 1$, или `9 007 199 254 740 991`) представляются точно. Диапазон целых чисел шире, чем в Java. Коль скоро результат остается в этом же диапазоне, арифметические операции над целыми также точны. Но при выходе за границы диапазона возникают ошибки округления. Например, вычисление `Number.MAX_SAFE_INTEGER * 10` дает `90071992547409900`.

> **Примечание.** Если диапазона целых чисел недостаточно, можно воспользоваться «большими целыми», число цифр в которых не ограничено. Большие целые описываются в главе 5.

Как и в любом языке программирования, избежать ошибок округления при операциях над числами с плавающей точкой невозможно. Например, `0.1 + 0.2` дает `0.30000000000000004` – точно так же, как в Java, C++ или Python. Это неизбежно, поскольку десятичные числа вроде 0.1, 0.2 или 0.3 не имеют точного двоичного представления. Для вычислений с долларами и центами следует представлять все денежные суммы в центах.

Другие формы числовых литералов, в частности шестнадцатеричные числа, описаны в главе 5.

Для преобразования строки в число предназначены функции `parseFloat` и `parseInt`:

```
const notQuitePi = parseFloat('3.14') // число 3.14
const evenLessPi = parseInt('3') // целое число 3
```

Метод `toString` преобразует число обратно в строку:

```
const notQuitePiString = notQuitePi.toString() // строка '3.14'
const evenLessPiString = (3).toString() // строка '3'
```