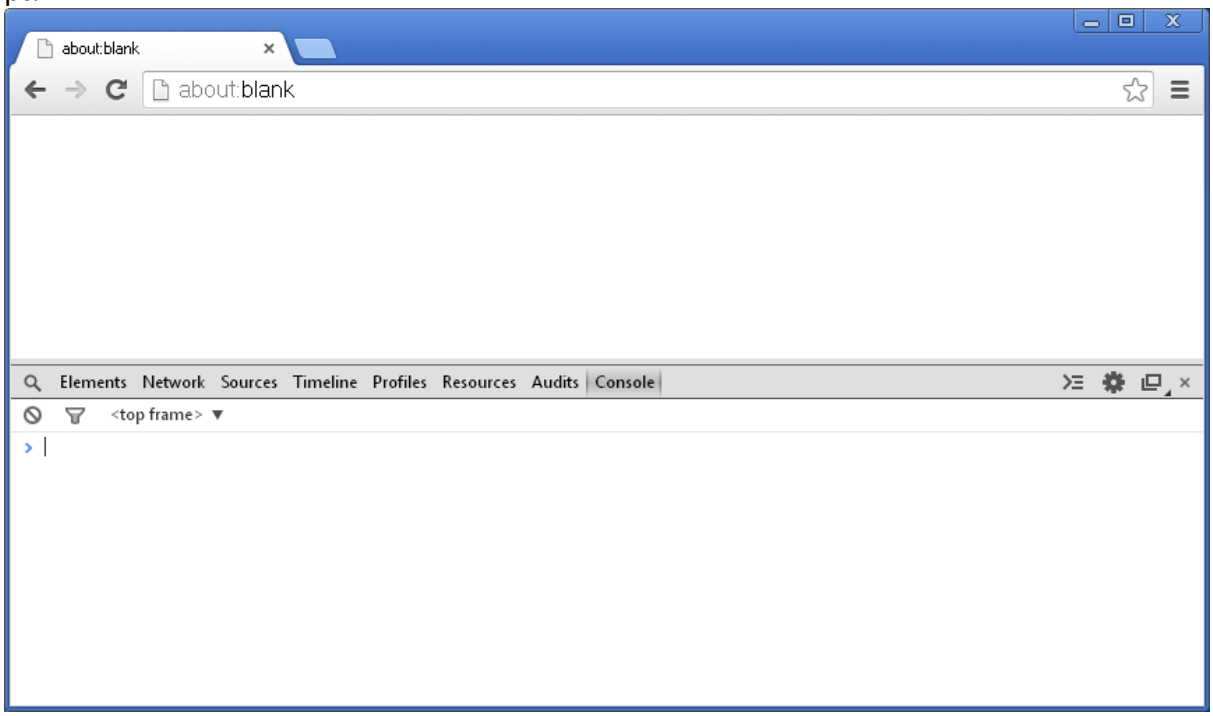
Пробуем JavaScript

Давайте напишем простую JavaScript-программку с помощью браузера Microsoft Internet Explorer или Google Chrome. Запустите, например, Chrome и введите слова about:blank в адресной строке. Теперь нажмите ENTER — откроется пустая страничка.

Начнем с программирования в JavaScript-консоли Chrome (это секретный инструмент для тестирования коротких программ на JavaScript). Нажмите и не отпускайте клавиши CTRL и SHIFT, а затем нажмите J(в Internet Explorer нажмите клавишу F12) .

Если вы все сделали правильно, то увидите пустую веб-страницу, под которой стоит значок угловой скобки (>), а после него мигает курсор (|). Здесь нам и предстоит писать код на языке JavaScript!



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

```
3 + 4;
```

Теперь нажмите ENTER. JavaScript должен напечатать результат сложения (7) на следующей строке:

```
3 + 4;
7
```

Как видите, ничего сложного. Но JavaScript — это нечто определенно большее, чем просто затейливый калькулятор. Давайте попробуем кое-что еще.

Строение JavaScript-программы

Давайте позабавимся — напишем JavaScript-программу, которая печатает японские смайлики каомодзи в виде кошачьей мордочки:

```
=^.^=
```

В отличие от простого сложения, с которого мы начали, эта программа занимает несколько строк. Чтобы ввести ее в консоли, нужно будет в конце каждой строки переходить на новую строку нажатием SHIFT-ENTER. Если нажать просто ENTER, Chrome попытается выполнить те команды, которые вы уже ввели, и программа не будет работать правильно. Введите в консоли браузера:

```
// Рисуем столько котиков, сколько захотим!
var drawCats = function (howManyTimes)
{
    for(var i = 0; i < howManyTimes; i++)
    {
       console.log(i + " =^.^=");
    }
};
```

drawCats(10); // Вместо 10 тут может быть другое число

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

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

Если при вводе программы вы где-то ошиблись, результат может оказаться другим — возможно, вы даже получите сообщение об ошибке.

Не будем сейчас вдаваться в подробности, объясняя, как работает этот код, однако давайте рассмотрим некоторые особенности этой программы, да и JavaScript-программ в целом.

Синтаксис

В нашей программе встречается много символов, таких как скобки (), точки с запятой фигурные скобки {}, знаки плюс +, а также некоторые таинственные на первый взгляд слова (например, var и console.log). Все это является частью синтаксиса JavaScript — то есть правил, указывающих, как объединять символы и слова, чтобы составить работающую программу.

Одна из главных сложностей при освоении нового языка программирования — запомнить правила написания команд. Поначалу легко пропустить какие-нибудь скобки или запутаться в очередности записи значений.

Комментарии

В первой строке нашей программы написано:

// Рисуем столько котиков, сколько захотим!

Это называется комментарий. Программисты пишут комментарии, чтобы другим программистам было легче читать и понимать их код. Компьютер же комментарии игнорирует. В JavaScript комментарии начинаются с двух символов наклонной черты (//). Все, что идет следом за ними (в той же строке), интерпретатор JavaScript пропускает, поэтому комментарии не оказывают влияния на выполнение программы — это всего лишь пояснение.

В конце нашей программы-примера есть еще один комментарий. Напоминаю: все, что записано после символов //, компьютер игнорирует!

drawCats(10); // Вместо 10 тут может быть другое число

Комментарии могут занимать отдельную строку или следовать сразу после кода. Но если вы поставите // перед кодом, вот так:

// drawCats(10);

...то не произойдет вообще ничего! Chrome решит, что вся эта строка — комментарий, хоть там и записаны инструкции на языке JavaScript.

Иногда вам будут попадаться комментарии, которые выглядят иначе:

/*
Рисуем столько котиков,
сколько захотим.
*/

Это другая разновидность комментариев; их обычно используют, когда текст примечания не помещается на одной строке. Однако принцип здесь тот же: текст, записанный между /* и */, — это комментарий, и выполнять его компьютер не будет.

Типы данных и переменные

Программирование — это работа с данными, но что такое данные? Данные — это информация, которая хранится в наших компьютерных программах. Например, ваше имя — это

элемент данных, и ваш возраст тоже. Цвет волос, количество братьев и сестер, ваш адрес и пол — все это данные.

В JavaScript есть три основных типа данных: числа, строки и булевы значения. Числа — они и есть числа, тут все понятно. Например, числом можно выразить возраст или рост.

В JavaScript числа записываются так:

```
5;
```

Любые текстовые данные записываются в строки. В JavaScript ваше имя можно выразить строкой (так же как и адрес вашей электронной почты).

Строки выглядят так:

```
"Привет, я строка!";
```

Булевы значения могут хранить одну из двух величин — либо это true («истина»), либо false («ложь»). Например, таким способом можно показать, любите ли вы поесть или имеете ли вы зависимость от смартфона.

Пример булева значения:

```
true;
```

С данными разных типов и обращаться следует по-разному. Например, перемножить два числа можно, а перемножить две строки — нет. Зато, имея строку, можно выделить пять ее первых символов. Взяв два булевых значения, можно проверить, являются ли они оба «истиной» (true). Вот все эти действия на примере:

```
99*123;
12177
"Вот длинная строка".slice(0, 3);
"Вот"
true && false;
false
```

Любые данные в JavaScript — не более чем сочетание этих основных типов. Далее мы по очереди рассмотрим каждый тип данных и изучим различные способы работы с ними.

Кстати, надеюсь, вы заметили, что все эти команды оканчиваются на точку с запятой (;). Этим символом обозначают конец каждой отдельной команды или инструкции языка JavaScript — примерно так же, как точка отмечает конец предложения.

Числа и операторы

JavaScript позволяет выполнять основные математические операции, такие как сложение, вычитание, умножение и деление. Для их записи используются символы +, -, *, и /, которые называют операторами.

Консоль JavaScript можно использовать как калькулятор. Один из примеров — сложение 3 и 4 — нам уже знаком. Давайте вычислим что-нибудь посложнее: сколько будет 12345 плюс 56789?

```
12345 + 56789;
69134
```

Можно сложить несколько чисел с помощью нескольких знаков «плюс»:

```
22 + 33 + 44;
99
```

Также JavaScript умеет вычитать...

```
1000 - 17;
983
```

умножать (с помощью символа «звездочка»)...

```
123 * 456;
56088
```

и делить (с помощью косой черты — слэша)...

```
12345 / 250;
49.38
```

Кроме того, можно объединять эти простые операции, составляя более сложные выражения, вроде такого:

```
1234 + 57 * 3 - 31 / 4;
```

1397.25

Есть один нюанс — результат вычислений зависит от порядка, в котором JavaScript выполняет отдельные операции. В математике существует правило, по которому умножение и деление выполняются прежде, чем сложение и вычитание, и JavaScript ему следует.

Переменные

Значениям в JavaScript можно давать имена, используя переменные. Переменная похожа на ящичек, в который помещается лишь один предмет. Чтобы положить туда что-то еще, прежнее содержимое придется заменить.

Чтобы создать новую переменную, используйте ключевое слово var, после которого укажите имя переменной. Ключевое слово — это слово, обладающее для JavaScript особым значением. В данном случае, когда JavaScript встречает слово var, он понимает, что следом указано имя новой переменной. Например, вот как создать переменную с именем nick:

```
var nick;
undefined
```

Мы создали новую переменную под названием nick. В ответ консоль выдала undefined — «значение не определено». Однако это не ошибка! JavaScript всегда так делает, если команда не возвращает какого-либо значения. Вы спросите, а что такое «возвращать значение»? Вот пример: когда вы ввели 12345 + 56789;, консоль вернула значение 69134. Однако в JavaScript команда создания переменной никакого значения не возвращает, поэтому интерпретатор печатает undefined.

Итак, чтобы задать переменной значение, используйте знак «равно»:

```
var age = 18;
undefined
```

Задание значения переменной называют присваиванием (здесь мы присваиваем значение 18 переменной age). И опять в консоли появляется undefined, поскольку мы только что создали новую переменную. (В дальнейших примерах я буду пропускать это undefined.)

Теперь в интерпретаторе есть переменная age, которой присвоено значение 18. И если ввести в консоли имя age, интерпретатор выдаст значение этой переменной:

```
age;
18
```

При этом значение переменной не высечено в камне (переменные потому так и зовутся, что могут менять значения), и, если вам вздумается его обновить, просто используйте знак «равно» еще раз.

```
age = 19;
19
```

На этот раз мы не использовали ключевое слово var, поскольку переменная age уже существует. Писать var нужно только при создании переменной, а не при ее использовании. И обратите внимание: поскольку мы не создавали новой переменной, команда присваивания вернула значение 19, которое и было напечатано в следующей строке.

Имена переменных

Вводя имена переменных, будьте внимательны и не допускайте опечаток. Даже если вы перепутаете строчные и заглавные буквы, интерпретатор JavaScript не поймет, чего вы от него хотите! Например, если вы случайно введете имя age с заглавной буквой A, возникнет ошибка:

```
Age / 2;
ReferenceError: Age is not defined
```

Увы, JavaScript следует вашим указаниям буквально. Если вы неправильно ввели имя переменной, JavaScript не поймет, что вы имели в виду, и выдаст сообщение об ошибке.

Еще один нюанс именования переменных в JavaScript — в именах не должно быть пробелов, из-за чего они могут оказаться сложными для чтения. Если бы мы назвали переменную numberofstudents, без заглавных букв, читать программу стало бы труднее, поскольку неясно, где в этом имени заканчиваются отдельные слова.

Один из обычных способов решения этой проблемы — писать каждое слово с заглавной буквы: NumberOfStdudents. Такую манеру именования называют верблюжьей записью, поскольку выпирающие заглавные буквы напоминают верблюжьи горбы.

Имена переменных принято начинать со строчной буквы, поэтому с заглавной буквы обычно пишут все слова имени, кроме самого первого: numberOfStudents. Я также буду использовать эту форму верблюжьей записи; впрочем, вы можете называть свои переменные как вам угодно.

Создание новых переменных на основе вычислений

Можно создавать новые переменные, выполняя математические действия с переменными, созданными ранее. Давайте с помощью переменных выясним, сколько секунд в году и каков ваш возраст в секундах! Но для начала разберемся, сколько секунд в одном часе.

Сколько секунд в часе

Сначала создадим две новые переменные — secondsInAMinute (количество секунд в минуте) и minutesInAHour (количество минут в часе) — и присвоим им обеим значение 60 (поскольку, как мы знаем, в минуте 60 секунд, а в часе 60 минут). Теперь создадим переменную secondsInAHour (количество секунд в часе), и пусть ее значение равняется secondsInAMinute умножить на minutesInAHour. И наконец введем secondsInAHour, что означает «покажи мне содержимое переменной secondsInAHour», и JavaScript тут же выдаст ответ: 3600.

```
var secondsInAMinute = 60;
var minutesInAHour = 60;
var secondsInAHour = secondsInAMinute * minutesInAHour;
secondsInAHour;
3600
```

Сколько секунд в сутках

Теперь создадим переменную hoursInADay (количество часов в сутках) и присвоим ей значение 24- Затем создадим переменную secondsInADay (количество секунд в сутках), и пусть она равняется secondsInAHour умножить на hoursInADay. Запросив в строке значение secondsInADay, получим 86400 — именно столько секунд в сутках.

```
var hoursInADay = 24;
var secondsInADay = secondsInAHour * hoursInADay;
secondsInADay;
86400
```

Сколько секунд в году

И наконец, создадим переменные daysInAYear (количество дней в году) и secondsInAYear (количество секунд в году): daysInAYear присвоим значение 365, а secondsInAYear пусть равняется secondsInADay умножить на daysInAYear. Запрашиваем значение secondsInAYear и видим, что это число 31536000 (более 31 миллиона секунд!).

```
var daysInAYear = 365;
var secondsInAYear = secondsInADay * daysInAYear;
secondsInAYear;
31536000
```

Возраст в секундах

Теперь, зная, сколько секунд в году, вы можете запросто узнать свой возраст в секундах (с точностью до последнего дня рождения). К примеру, вам 18 лет:

```
var age = 18;
age * secondsInAYear;
567648000
```

Смотрите-ка, вам исполнилось больше 567 миллионов секунд!

Инкремент и декремент

Вам как программисту понадобится увеличивать или уменьшать значения числовых переменных на единицу. Например, у вас в программе может быть переменная для подсчета, сколько раз за день вам сказали «Привет!». И при каждом новом приветствии эту переменную надо будет увеличить на 1.

Увеличение на 1 называют инкрементом, а уменьшение на 1 — декрементом. Выполняются инкремент и декремент с помощью операторов ++ и --

```
var hello = 0;
++hello;
1
++hello;
2
--hello;
1
```

После выполнения оператора ++ значение hello (количество приветствий) увеличится на 1, а после выполнения оператора -- уменьшится на 1. Также эти операторы можно писать после имени переменной — эффект будет прежним, однако после выполнения такой команды JavaScript вернет первоначальное значение переменной, каким оно было до инкремента или декремента.

```
hello = 0;
hello++;
0
hello++;
1
hello;
2
```

В этом примере мы сначала обнулили значение hello. Команда hello++ увеличивает переменную на 1, но число, которое печатает после этого JavaScript, является значением до инкремента. Однако, запрашивая значение highFives в самом конце (после двух инкрементов), мы получаем 2.

+= (плюс-равно) и -= (минус-равно)

Чтобы увеличить значение переменной на заданное число, можно написать такой код:

```
var x = 10;
x = x + 5;
x;
15
```

Сначала мы создаем переменную x и даем ей значение 10. Затем присваиваем x значение x + 5 — то есть используем старое значение x, чтобы получить новое значение. Таким образом, выражение x = x + 5 по сути означает «увеличить x на 5».

В арсенале JavaScript есть более простой способ увеличения или уменьшения переменной на заданную величину: это операторы += и -=. Пусть у нас есть переменная x, тогда команда x += 5 означает то же самое, что и x = x + 5. Оператор -= работает аналогично, то есть x -= 9 соответствует x = x - 9 (уменьшить x на 9). С помощью этих операторов можно, например, управлять подсчетом очков в игре:

```
var score = 10;
score += 7;
17
score -= 3;
14
```

В этом примере мы сначала присваиваем переменной score (счет игры) начальное количество очков (10). Потом, победив монстра, мы увеличиваем счет на 7 очков с помощью оператора += (score += 7 соответствует score = score + 7). Поскольку изначально в score было число 10, а 10 + 7 = 17, этой командой мы установили счет в 17 очков.

После победы над монстром мы столкнулись с метеоритом, и счет уменьшился на 3 очка. Опять же, score -= 3 — это то же самое, что и score = score - 3. Поскольку перед этим в score было 17, score - 3 равняется 14; это число и будет новым значением score.

Есть и другие операторы, похожие на += и -=. Например, *= и /=. Как вы думаете, для чего они?

Строки

До сих пор мы имели дело только с числами. Пора познакомиться с еще одним типом данных — со строками. В JavaScript (как и в большинстве других языков программирования) строка является набором символов — букв, цифр, знаков пунктуации и пробелов. Чтобы JavaScript знал, где начинается и заканчивается строка, ее берут в кавычки. Вот классический пример с фразой «Привет, мир!»:

"Привет, мир!";

"Привет, мир!"

Чтобы создать строку, поставьте знак двойной кавычки ("), затем введите какой-нибудь текст и закройте строку еще одной двойной кавычкой. Можно пользоваться и одинарными кавычками ('), однако, чтобы не путаться, все строки в наших примерах будут в двойных кавычках.

Строки можно хранить в переменных, так же как числа:

var myAwesomeString = "Что-то ОЧЕНЬ крутое!!!";

Также ничто не мешает присвоить строковое значение переменной, где раньше хранилось число:

var myThing = 5;

myThing = "это строка";

"это строка"

А что если записать в кавычках число? Строка это будет или число? В JavaScript строка остается строкой, даже если там хранятся цифровые символы. Например:

var numberNine = 9;

var stringNine = "9";

В переменной numberNine (число девять) хранится число, а в переменной stringNine (строка девять) — строка. Чтобы выяснить, в чем их различие, посмотрим, как они реагируют на сложение:

numberNine + numberNine;

18

stringNine + stringNine;

"99"

Объединение строк

Как мы только что убедились, оператор + можно использовать и со строками, однако действует он при этом совсем иначе, чем с числами. С помощью оператора + строки можно объединять: результатом будет новая строка, состоящая из первой строки, к концу которой присоединена вторая:

var greeting = "Привет";

var myName = "Ксюша";

greeting + myName;

"ПриветКсюша"

Здесь мы создали две переменные (greeting и myName) и присвоили каждой из них строковое значение ("Привет" и "Ксюша" соответственно). При сложении этих переменных строки объединяются, образуя новую строку — "ПриветКсюша".

Впрочем, не все тут идеально — между "Привет" и "Ксюша" должен стоять пробел и запятая. JavaScript не ставит пробелов по собственной инициативе, зато его можно попросить его об этом, добавив пробел к одной из первоначальных строк:

var greeting = "Привет, ";

var myName = "Ксюша";

greeting + myName;

"Привет, Ксюша"

Дополнительный пробел перед закрывающей кавычкой дает пробел в середине результирующей строки.

Помимо их объединения, со строками можно выполнять множество разных действий. Вот несколько примеров.

Как узнать длину строки

Чтобы узнать длину строки, достаточно добавить к ее концу . length:

```
"Супермегадлиннаястрока".length;
22
```

Можно добавлять . length к концу как самой строки, так и переменной, содержащей строку:

```
var java - "Java"; java-length;
4
var script = "Script"; script.length;
var javascript - java + script; j avascript. lengt;'.;
10
```

Здесь мы присвоили строковое значение "Java" переменной java, а значение "Script" — переменной script. Затем мы добавили .length к концу каждой из переменных, узнав таким образом длины отдельных строк, а также длину составленной из них новой строки.

Обратите внимание: я говорил «можно добавлять .length к концу

```
var java = "Java";
java.length;
4
var script = "Script";
script.length;
6
var javascript = java + script;
javascript.length;
10
```

Здесь мы присвоили строковое значение "Java" переменной java, а значение "Script" — переменной script. Затем мы добавили .length к концу каждой из переменных, узнав таким образом длины отдельных строк, а также длину составленной из них новой строки.

Обратите внимание: я говорил «можно добавлять .length к концу как самой строки, так и переменной, содержащей строку». Это касается очень важного свойства переменных: в любом месте программы, где допустимо использовать число или строку, можно также использовать переменную, в которой хранится число или строка.

Получение отдельного символа строки

Иногда требуется получить из строки одиночный символ. Например, вы можете зашифровать в наборе слов тайное послание, состоящее из вторых символов каждого слова. Тогда, чтобы узнать это послание, нужно получить все вторые символы и объединить их в новую строку.

Чтобы получить символ, стоящий в определенной позиции строки, используйте квадратные скобки — []. Возьмите строку (или переменную, в которой хранится строка) и поставьте сразу после нее квадратные скобки, в которых указана позиция нужного символа. Например, чтобы получить первый символ строковой переменной myName, используйте запись myName[0]:

```
var myName = "Ксюша";
myName[0];
"К"
myName[1];
"с"
myName[2];
"ю"
```

Обратите внимание — чтобы получить первый символ, мы указали в скобках позицию 0, а не 1. Дело в том, что JavaScript (как и многие другие языки программирования) ведет отсчет символов с нуля. Таким образом, для получения первого символа строки указывайте позицию 0, второго — 1 и т. д.

Попробуем разгадать наш тайный шифр, где во вторых буквах некоторого набора слов скрыто послание. Вот как это сделать:

```
var codeWord1 = "обернись";
var codeWord2 = "неужели";
var codeWord3 = "огурцы";
var codeWord4 = "липкие";
```

var codeWord5 = "?!";

codeWord1[1] + codeWord2[1] + codeWord3[1] + codeWord4[1] + codeWord5[1];

"беги!"

И снова обращаю внимание — второй символ каждой строки мы получаем, указав позицию 1.

Получение среза строки

Чтобы получить часть, или «срез», строки, используйте slice. Например, представьте, что вам нужен отрывок из длинного описания фильма для анонса на вашем сайте. Чтобы воспользоваться slice, поставьте в конце строки (или переменной, содержащей строку) точку, а после нее слово slice и круглые скобки. В скобках укажите позицию первого символа той части строки, которую вы хотите получить, затем запятую, а затем позицию последнего символа.

Например:

var longString = "Эта длинная строка такая длинная";

longString.slice(4, 18);

"длинная строка"

Первое число в скобках — позиция символа, с которого начинается срез, а второе число — позиция символа, который *следует за* последним символом среза.

По сути, мы попросили JavaScript: «Вырежи из этой длинной строки часть, которая начинается с символа в позиции 4 и продолжается до позиции 18».

Если указать в скобках после slice только одно число, мы получим строку-срез, которая начинается сданной позиции и длится до конца строки:

var longString = "Эта длинная строка такая длинная";

longString.slice(4);

"длинная строка такая длинная"

Перевод строки в заглавный или строчный регистр

Если нужно вывести какой-нибудь текст заглавными буквами, воспользуйтесь toUpperCase.

"Эй, как дела?".toUpperCase();

"ЭЙ, КАК ДЕЛА?"

.toUpperCase() возвращает новую строку, все буквы в которой — заглавные.

Можно произвести и обратную операцию, использовав toLowerCase:

"Эй, как дела?".toLowerCase();

"эй, как дела?"

.toLowerCase() делает все символы строчными.

Булевы значения

Теперь поговорим о булевых значениях. В сущности, есть лишь два варианта таких значений — это либо true (истина), либо false (ложь). Например, вот простое выражение с булевым значением:

var javascriptIsCool = true;

javascriptIsCool;

true

Здесь мы создали новую переменную с именем javascriptIsCool и присвоили ей булево значение true. Следующей строкой мы запросили содержимое javascriptIsCool и, разумеется, получили true.