

 → [Язык JavaScript](#) → [Основы JavaScript](#)

 27 августа 2022 г.

Типы данных

Значение в JavaScript всегда относится к данным определённого типа. Например, это может быть строка или число.

Есть восемь основных типов данных в JavaScript. В этой главе мы рассмотрим их в общем, а в следующих главах поговорим подробнее о каждом.

Переменная в JavaScript может содержать любые данные. В один момент там может быть строка, а в другой – число:

```

1 // Не будет ошибкой
2 let message = "hello";
3 message = 123456;
```

Языки программирования, в которых такое возможно, называются «динамически типизированными». Это значит, что типы данных есть, но переменные не привязаны ни к одному из них.

Число

```

1 let n = 123;
2 n = 12.345;
```

Числовой тип данных (**number**) представляет как целочисленные значения, так и числа с плавающей точкой.

Существует множество операций для чисел, например, умножение `*`, деление `/`, сложение `+`, вычитание `-` и так далее.

Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: `Infinity`, `-Infinity` и `NaN`.

- `Infinity` представляет собой математическую **бесконечность** ∞ . Это особое значение, которое больше любого числа.

Мы можем получить его в результате деления на ноль:

```
1 alert( 1 / 0 ); // Infinity
```



Или задать его явно:

```
1 alert( Infinity ); // Infinity
```



- NaN означает вычислительную ошибку. Это результат неправильной или неопределённой математической операции, например:

```
1 alert( "не число" / 2 ); // NaN, такое деление является ошибкой
```



Значение NaN «прилипчиво». Любая математическая операция с NaN возвращает NaN :

```
1 alert( NaN + 1 ); // NaN
2 alert( 3 * NaN ); // NaN
3 alert( "не число" / 2 - 1 ); // NaN
```



Если где-то в математическом выражении есть NaN, то оно распространяется на весь результат (есть только одно исключение: NaN ** 0 равно 1).

Математические операции – безопасны

Математические операции в JavaScript «безопасны». Мы можем делать что угодно: делить на ноль, обращаться с нечисловыми строками как с числами и т.д.

Скрипт никогда не остановится с фатальной ошибкой (не «умрёт»). В худшем случае мы получим NaN как результат выполнения.

Специальные числовые значения относятся к типу «число». Конечно, это не числа в привычном значении этого слова.

Подробнее о работе с числами мы поговорим в главе [Числа](#).

BigInt

В JavaScript тип `number` не может безопасно работать с числами, большими, чем $(2^{53}-1)$ (т. е. 9007199254740991) или меньшими, чем $-(2^{53}-1)$ для отрицательных чисел. Технически, тип `number` может хранить и гораздо большие значения (вплоть до $1.7976931348623157 \cdot 10^{308}$), однако за пределами безопасного диапазона $\pm(2^{53}-1)$ многие из чисел не могут быть представлены с помощью этого типа данных из-за ограничений, вызванных внутренним представлением чисел в двоичной форме. Например, нечётные числа, большие, чем $(2^{53}-1)$, невозможно хранить при помощи типа `number`, они с разной точностью будут автоматически округляться до чётных значений. В то же время некоторые чётные числа, большие, чем $(2^{53}-1)$, при помощи типа `number` хранить технически возможно (однако не стоит этого делать во избежание дальнейших ошибок).

Для большинства случаев достаточно безопасного диапазона чисел от $-(2^{53}-1)$ до $(2^{53}-1)$. Но иногда нам нужен диапазон действительно гигантских целых чисел без каких-либо ограничений или пропущенных значений внутри него. Например, в криптографии или при использовании метки времени («timestamp») с микросекундами.

Тип `BigInt` был добавлен в JavaScript, чтобы дать возможность работать с целыми числами произвольной длины.

Чтобы создать значение типа `BigInt`, необходимо добавить `n` в конец числового литерала:

```
1 // символ "n" в конце означает, что это BigInt
2 const bigInt = 1234567890123456789012345678901234567890n;
```

Так как `BigInt` -числа нужны достаточно редко, мы рассмотрим их в отдельной главе [BigInt](#). Ознакомьтесь с ней, когда вам понадобятся настолько большие числа.

Поддержка

В данный момент `BigInt` поддерживается только в браузерах Firefox, Chrome, Edge и Safari, но не поддерживается в IE.

Строка

Строка (`string`) в JavaScript должна быть заключена в кавычки.

```
1 let str = "Привет";
2 let str2 = 'Одинарные кавычки тоже подойдут';
3 let phrase = `Обратные кавычки позволяют встраивать переменные ${str}`;
```

В JavaScript существует три типа кавычек.

1. Двойные кавычки: `"Привет"` .
2. Одинарные кавычки: `'Привет'` .
3. Обратные кавычки: ``Привет`` .

Двойные или одинарные кавычки являются «простыми», между ними нет разницы в JavaScript.

Обратные же кавычки имеют расширенную функциональность. Они позволяют нам встраивать выражения в строку, заключая их в `${...}` . Например:

```
1 let name = "Иван";
2
3 // Вставим переменную
4 alert( `Привет, ${name}!` ); // Привет, Иван!
5
6 // Вставим выражение
7 alert( `результат: ${1 + 2}` ); // результат: 3
```



Выражение внутри `${...}` вычисляется, и его результат становится частью строки. Мы можем положить туда всё, что угодно: переменную `name` , или выражение `1 + 2` , или что-то более сложное.

Обратите внимание, что это можно делать только в обратных кавычках. Другие кавычки не имеют такой функциональности встраивания!

```
1 alert( "результат: ${1 + 2}" ); // результат: ${1 + 2} (двойные кавычки)
```



ничёгс

Мы рассмотрим строки более подробно в главе [Строки](#).

i Нет отдельного типа данных для одного символа.

В некоторых языках, например C и Java, для хранения одного символа, например "a" или "%", существует отдельный тип. В языках C и Java это `char`.

В JavaScript подобного типа нет, есть только тип `string`. Строка может содержать ноль символов (быть пустой), один символ или множество.

Булевый (логический) тип

Булевый тип (`boolean`) может принимать только два значения: `true` (истина) и `false` (ложь).

Такой тип, как правило, используется для хранения значений да/нет: `true` значит «да, правильно», а `false` значит «нет, не правильно».

Например:

```
1 let nameFieldChecked = true; // да, поле отмечено
2 let ageFieldChecked = false; // нет, поле не отмечено
```

Булевы значения также могут быть результатом сравнений:

```
1 let isGreater = 4 > 1;
2
3 alert( isGreater ); // true (результатом сравнения будет "да")
```



Мы рассмотрим булевы значения более подробно в главе [Логические операторы](#).

Значение «null»

Специальное значение `null` не относится ни к одному из типов, описанных выше.

Оно формирует отдельный тип, который содержит только значение `null`:

```
1 let age = null;
```

В JavaScript `null` не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках.

Это просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно».

В приведённом выше коде указано, что значение переменной `age` неизвестно.

Значение «undefined»

Специальное значение `undefined` также стоит особняком. Оно формирует тип из самого себя так же, как и `null`.

Оно означает, что «значение не было присвоено».

Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет `undefined`:



```
1 let age;  
2  
3 alert(age); // выведет "undefined"
```

Технически мы можем присвоить значение `undefined` любой переменной:



```
1 let age = 123;  
2  
3 // изменяем значение на undefined  
4 age = undefined;  
5  
6 alert(age); // "undefined"
```

...Но так делать не рекомендуется. Обычно `null` используется для присвоения переменной «пустого» или «неизвестного» значения, а `undefined` – для проверок, была ли переменная назначена.

Объекты и символы

Тип `object` (объект) – особенный.

Все остальные типы называются «примитивными», потому что их значениями могут быть только простые значения (будь то строка, или число, или что-то ещё). В объектах же хранят коллекции данных или более сложные структуры.

Объекты занимают важное место в языке и требуют особого внимания. Мы разберёмся с ними в главе [Объекты](#) после того, как узнаем больше о примитивах.

Тип `symbol` (символ) используется для создания уникальных идентификаторов в объектах. Мы упоминаем здесь о нём для полноты картины, изучим этот тип после объектов.

Оператор `typeof`

Оператор `typeof` возвращает тип аргумента. Это полезно, когда мы хотим обрабатывать значения различных типов по-разному или просто хотим сделать проверку.

У него есть две синтаксические формы:

1. Синтаксис оператора: `typeof x`.
2. Синтаксис функции: `typeof(x)`.

Другими словами, он работает со скобками или без скобок. Результат одинаковый.

Вызов `typeof x` возвращает строку с именем типа:

```
1 typeof undefined // "undefined"  
2  
3 typeof 0 // "number"  
4  
5 typeof 10n // "bigint"  
6  
7 typeof true // "boolean"  
8  
9 typeof "foo" // "string"
```

```
10
11 typeof Symbol("id") // "symbol"
12
13 typeof Math // "object" (1)
14
15 typeof null // "object" (2)
16
17 typeof alert // "function" (3)
```

Последние три строки нуждаются в пояснении:

1. **Math** — это встроенный объект, который предоставляет математические операции и константы. Мы рассмотрим его подробнее в главе [Числа](#). Здесь он служит лишь примером объекта.
2. Результатом вызова **typeof null** является **"object"**. Это официально признанная ошибка в **typeof**, ведущая начало с времён создания JavaScript и сохранённая для совместимости. Конечно, **null** не является объектом. Это специальное значение с отдельным типом.
3. Вызов **typeof alert** возвращает **"function"**, потому что **alert** является функцией. Мы изучим функции в следующих главах, где заодно увидим, что в JavaScript нет специального типа «функция». Функции относятся к объектному типу. Но **typeof** обрабатывает их особым образом, возвращая **"function"**. Так тоже повелось от создания JavaScript. Формально это неверно, но может быть удобным на практике.

Итого

В JavaScript есть 8 основных типов данных.

- Семь из них называют «примитивными» типами данных:
 - **number** для любых чисел: целочисленных или чисел с плавающей точкой; целочисленные значения ограничены диапазоном $\pm(2^{53}-1)$.
 - **bigint** для целых чисел произвольной длины.
 - **string** для строк. Строка может содержать ноль или больше символов, нет отдельного символьного типа.
 - **boolean** для **true** / **false**.
 - **null** для неизвестных значений — отдельный тип, имеющий одно значение **null**.
 - **undefined** для неприсвоенных значений — отдельный тип, имеющий одно значение **undefined**.
 - **symbol** для уникальных идентификаторов.
- И один не является «примитивным» и стоит особняком:
 - **object** для более сложных структур данных.

Оператор **typeof** позволяет нам увидеть, какой тип данных сохранён в переменной.

- Имеет две формы: **typeof x** или **typeof(x)**.
- Возвращает строку с именем типа. Например, **"string"**.
- Для **null** возвращается **"object"** — это ошибка в языке, на самом деле это не объект.

В следующих главах мы сконцентрируемся на примитивных значениях, а когда познакомимся с ними, перейдём к объектам.

✓ Задачи

важность: 5

Что выведет этот скрипт?

```
1 let name = "Ilya";
2
3 alert( `hello ${1}` ); // ?
4
5 alert( `hello ${"name"}` ); // ?
6
7 alert( `hello ${name}` ); // ?
```

решение



Предыдущий урок

Следующий урок



Поделиться



Карта учебника

Проводим курсы по JavaScript и фреймворкам.



Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите [на GitHub](#).
- Для одной строки кода используйте тег `<code>` , для нескольких строк кода — тег `<pre>` , если больше 10 строк — ссылку на песочницу ([plnkr](#), [JSBin](#), [codepen...](#))
- Если что-то непонятно в статье — пишите, что именно и с какого места.