**Типы данных**

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

let message = "hello";

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

### Число

let n = 123;

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](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%81%D0%BA%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C#%D0%92_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B5) ∞. Это особое значение, которое больше любого числа.

Мы можем получить его в результате деления на ноль:

alert( 1 / 0 ); // Infinity

Или задать его явно:

alert( Infinity ); // Infinity

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

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

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

alert( "не число" / 2 + 5 ); // NaN

* Если где-то в математическом выражении есть NaN, то результатом вычислений с его участием будет NaN.

Математические операции в JavaScript «безопасны». Мы можем делать что угодно: делить на ноль, обращаться со 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.

### BigInt

В JavaScript тип «number» не может содержать числа больше, чем 253 (или меньше, чем -253 для отрицательных). Это техническое ограничение вызвано их внутренним представлением. 253 – это достаточно большое число, состоящее из 16 цифр, поэтому чаще всего проблем не возникает. Но иногда нам нужны действительно гигантские числа, 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:

// символ "n" в конце означает, что это BigInt

const bigInt = 1234567890123456789012345678901234567890n;

В данный момент BigInt поддерживается только в браузерах Firefox и Chrome

### Строка

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

let str = "Привет";

let str2 = 'Одинарные кавычки тоже подойдут';

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

let name = "Иван";

// Вставим переменную

alert( `Привет, ${name}!` ); // Привет, Иван!

// Вставим выражение

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!

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

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 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:

let nameFieldChecked = true; // да, поле отмечено

let ageFieldChecked = false; // нет, поле не отмечено

Булевые значения также могут быть результатом сравнений:

let isGreater = 4 > 1;

alert( isGreater ); // true (результатом сравнения будет "да")

### Значение «null»

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

Оно формирует отдельный тип, который содержит только значение null:

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 переменная age неизвестна или не имеет значения по какой-то причине.

### [Значение «undefined»](https://learn.javascript.ru/types" \l "znachenie-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:

let x;

alert(x); // выведет "undefined"

Технически мы можем присвоить значение undefined любой переменной:

let x = 123;

x = undefined;

alert(x); // "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). Объекты же используются для хранения коллекций данных или более сложных объектов. Тип **symbol** (символ) используется для создания уникальных идентификаторов объектов.

### [Оператор typeof](https://learn.javascript.ru/types" \l "type-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 два синтаксиса:

1. Синтаксис оператора**: typeof x**.
2. Синтаксис функции:

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

Вызов typeof x возвращает строку с именем типа:

typeof undefined // "undefined"

typeof 0 // "number"

typeof 10n // "bigint"

typeof true // "boolean"

typeof "foo" // "string"

typeof Symbol("id") // "symbol"

typeof Math // "object" (1)

typeof null // "object" (2)

typeof alert // "function" (3)

Последние три строки нуждаются в пояснении:

1. Math – это встроенный объект, который предоставляет математические операции и константы. Результатом вызова typeof null является "object". Это неверно. Это официально признанная ошибка в typeof, сохранённая для совместимости. Конечно, null не является объектом. Это специальное значение с отдельным типом. Повторимся, это ошибка в языке.
2. 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". Формально это неверно, но очень удобно на практике.

**Преобразование типов**

Чаще всего операторы и функции автоматически приводят переданные им значения к нужному типу.

Например, alert автоматически преобразует любое значение к строке. Математические операторы преобразуют значения к числам.

Есть также случаи, когда нам нужно явно преобразовать значение в ожидаемый тип.

### Строковое преобразование

Строковое преобразование происходит, когда требуется представление чего-либо в виде строки.

Например, alert(value) преобразует значение к строке.

Также мы можем использовать функцию **String(value)**, чтобы преобразовать значение к строке:

let value = true;

alert(typeof value); // boolean

value = String(value); // теперь value это строка "true"

alert(typeof value); // string

Преобразование происходит очевидным образом. false становится "false", null становится "null" и т.п.

### Численное преобразование

Численное преобразование происходит в математических функциях и выражениях.

Например, когда операция деления / применяется не к числу:

alert( "6" / "2" ); // 3, Строки преобразуются в числа

Мы можем использовать функцию **Number(value)**, чтобы явно преобразовать value к числу:

let str = "123";

alert(typeof str); // string

let num = Number(str); // становится числом 123

alert(typeof num); // number

Явное преобразование часто применяется, когда мы ожидаем получить число из строкового контекста, например из текстовых полей форм.

Если строка не может быть явно приведена к числу, то результатом преобразования будет NaN. Например:

let age = Number("Любая строка вместо числа");

alert(age); // NaN, преобразование не удалось

Правила численного преобразования:

|  |  |
| --- | --- |
| **Значение** | **Преобразуется в…** |
| undefined | NaN |
| null | 0 |
| true / false | 1 / 0 |
| string | Пробельные символы по краям обрезаются. Далее, если остаётся пустая строка, то 0, иначе из непустой строки «считывается» число. При ошибке результат NaN. |

**Примеры:**

alert( Number(" 123 ") ); // 123

alert( Number("123z") ); // NaN (ошибка чтения числа в "z")

alert( Number(true) ); // 1

alert( Number(false) ); // 0

Учтите, что null и undefined ведут себя по-разному. Так, null становится нулём, тогда как undefined приводится к NaN.

Сложение „+“ объединяет строки

Почти все математические операторы выполняют численное преобразование. Исключение составляет +. Если одно из слагаемых является строкой, тогда и все остальные приводятся к строкам.

Тогда они конкатенируются (присоединяются) друг к другу:

alert( 1 + '2' ); // '12' (строка справа)

alert( '1' + 2 ); // '12' (строка слева)

Так происходит, только если хотя бы один из аргументов является строкой. Во всех остальных случаях значения складываются как числа.

### Логическое преобразование

Логическое преобразование самое простое.

Происходит в логических операторах (позже мы познакомимся с подобными конструкциями), но также может быть выполнено явно с помощью функции Boolean(value).

Правило преобразования:

* Значения, которые интуитивно «пустые», вроде 0, пустой строки, null, undefined и NaN, становятся false.
* Все остальные значения становятся true.

Например:

alert( Boolean(1) ); // true

alert( Boolean(0) ); // false

alert( Boolean("Привет!") ); // true

alert( Boolean("") ); // false

Заметим, что строчка с нулём "0" это true

Некоторые языки (к примеру, PHP) воспринимают строку "0" как false. Но в JavaScript, если строка не пустая, то она всегда true.

alert( Boolean("0") ); // true

alert( Boolean(" ") ); // пробел это тоже true (любая непустая строка это true)

### Итого

В JavaScript есть 8 основных типов.

* number для любых чисел: целочисленных или чисел с плавающей точкой, целочисленные значения ограничены диапазоном ±253.
* bigint для целых чисел произвольной длины.
* string для строк. Строка может содержать один 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.
* object для более сложных структур данных.
* symbol для уникальных идентификаторов.

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.

Существует 3 наиболее широко используемых преобразований: строковое, численное и логическое.

**Строковое** – Происходит, когда нам нужно что-то вывести. Может быть вызвано с помощью String(value). Для примитивных значений работает очевидным образом.

**Численное** – Происходит в математических операциях. Может быть вызвано с помощью Number(value).

Преобразование подчиняется правилам:

|  |  |
| --- | --- |
| **Значение** | **Становится…** |
| undefined | NaN |
| null | 0 |
| true / false | 1 / 0 |
| string | Пробельные символы по краям обрезаются. Далее, если остаётся пустая строка, то 0, иначе из непустой строки «считывается» число. При ошибке результат NaN. |

**Логическое** – Происходит в логических операторах. Может быть вызвано с помощью Boolean(value).

Подчиняется правилам:

|  |  |
| --- | --- |
| **Значение** | **Становится…** |
| 0, null, undefined, NaN, "" | false |
| любое другое значение | true |

Большую часть из этих правил легко понять и запомнить. Особые случаи, в которых часто допускаются ошибки:

* undefined при численном преобразовании становится NaN, не 0.
* "0" и строки из одних пробелов типа " " при логическом преобразовании всегда true.