* Data types
* 6 типов данных являющихся примитивами:
  + [Undefined](https://developer.mozilla.org/ru/docs/Glossary/undefined) (Неопределённый тип) : typeof instance === "undefined"
  + [Boolean](https://developer.mozilla.org/ru/docs/Glossary/Boolean) (Булев, Логический тип) : typeof instance === "boolean"
  + [Number](https://developer.mozilla.org/ru/docs/Glossary/Number) (Число) : typeof instance === "number"
  + [String](https://developer.mozilla.org/ru/docs/Glossary/String) (Строка) : typeof instance === "string"
  + [BigInt](https://developer.mozilla.org/ru/docs/Glossary/BigInt) : typeof instance === "bigint"
  + [Symbol](https://developer.mozilla.org/ru/docs/conflicting/Web/JavaScript/Reference/Global_Objects/Symbol) (в ECMAScript 6) : typeof instance === "symbol"
* [Null](https://developer.mozilla.org/ru/docs/Glossary/Null) (Null тип ) : typeof instance === "object". Специальный примитив, используемый не только для данных но и в качестве указателя на финальную точку в [Цепочке Прототипов](https://developer.mozilla.org/ru/docs/Web/JavaScript/Inheritance_and_the_prototype_chain);
* [Object](https://developer.mozilla.org/ru/docs/Glossary/Object) (Объект) : typeof instance === "object". Простая структура, используемая не только для хранения данных, но и для создания других структур, где любая структура создаётся с использованием ключевого слова [new](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/new): new [Object](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object), new [Array](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array), new [Map (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map), new [Set](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Set), new [WeakMap](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/WeakMap), new [WeakSet](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/WeakSet), new [Date](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Date) и множество других структур;
* Number methods
* [Number.isNaN()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Number/isNaN) Экспериментальная возможность
* Определяет, является ли переданное значение значением [NaN](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/NaN).
* [Number.isFinite()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Number/isFinite) Экспериментальная возможность
* Определяет, является ли переданное значение конечным числом.
* [Number.isInteger()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Number/isInteger) Экспериментальная возможность
* Определяет, является ли тип переданного значения «числом», а само число — целым значением.
* [Number.isSafeInteger()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Number/isSafeInteger) Экспериментальная возможность
* Определяет, является ли переданное значение безопасным целым числом (числом в диапазоне от -(253 - 1) до 253 - 1).
* Number.toInteger()
* И
* [Number.parseFloat()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Number/parseFloat) Экспериментальная возможность
* Представляет собой тот же самый метод, что и метод [parseFloat](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/parseFloat) глобального объекта.
* [Number.parseInt()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Number/parseInt) Экспериментальная возможность
* Представляет собой тот же самый метод, что и метод [parseInt](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/parseInt) глобального объекта.
* Методы, унаследованные из [Function](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Function):
* [apply](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Function/apply), [call](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Function/call), [toSource](https://developer.mozilla.org/ru/docs/conflicting/Web/JavaScript/Reference/Global_Objects/Function/toString), [toString](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Function/toString)
* String methods
* [String.fromCharCode()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/fromCharCode)
* Возвращает строку, созданную из указанной последовательности значений Юникода.
* [String.fromCodePoint()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/fromCodePoint) Экспериментальная возможность
* Возвращает строку, созданную из указанной последовательности кодовых точек Юникода.
* [String.raw()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/raw) Экспериментальная возможность
* Возвращает строку, созданную из сырой шаблонной строки.
* Методы:

'substring', 'toLowerCase', 'toUpperCase', 'charAt',

'charCodeAt', 'indexOf', 'lastIndexOf', 'startsWith', 'endsWith',

'trim', 'trimLeft', 'trimRight', 'toLocaleLowerCase',

'toLocaleUpperCase', 'localeCompare', 'match', 'search',

'replace', 'split', 'substr', 'concat', 'slice'

* Ternary operator
* **Условный (тернарный) оператор** - единственный оператор в JavaScript, принимающий три операнда: условие, за которым следует знак вопроса (?), затем выражение, которое выполняется, если условие истинно, сопровождается двоеточием (:), и, наконец, выражение, которое выполняется, если условие ложно. Он часто используется в качестве укороченного варианта условного оператора [if](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/if...else).
* [**Синтаксис**](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Conditional_Operator#%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81)
* условие ? выражение1 : выражение2
* [**Параметры**](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Conditional_Operator#%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D1%8B)
* условие
* Выражение, принимающее значение true или false.
* выражение1, выражение2
* Выражения, значения которых могут принадлежать любому типу.
* [**Описание**](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Conditional_Operator#%D0%BE%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D0%B5)
* Оператор возвращает значение выражения1, если условие верно, и значение выражения2 в противном случае. Например, чтобы вывести сообщение, текст которого зависит от значения переменной isMember, можно использовать такое выражение:
* "The fee is " + (isMember ? "$2.00" : "$10.00")
* Switch case - examples, where it can be useful
* **Инструкция switch** сравнивает выражение со случаями, перечисленными внутри неё, а затем выполняет соответствующие инструкции.
* switch (expression) {
* case value1:
* //Здесь выполняются инструкции, если результат выражения равен value1
* [break;]
* case value2:
* //Инструкции, соответствующие value2
* [break;]
* ...
* case valueN:
* //Инструкции, соответствующие значению valueN
* //statementsN
* [break;]
* default:
* //Здесь находятся инструкции, которые выполняются при отсутствии соответствующего значения
* //statements\_def
* [break;]
* }
* expression
* Выражение, значение которого сравнивается со всеми случаями.
* case valueN
* Случай, который проверяется на соответствие выражению (expression).
* statementsN
* Инструкции, которые выполняются, если expression соответствуют случаю.
* statements\_def
* Инструкции, выполняемые если expression не соответствует ни одному случаю.
* 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.
* Типы преобразований:
* 1.Строковое

## 2. [Численное преобразование](https://learn.javascript.ru/type-conversions#chislennoe-preobrazovanie)

## 3. Логическое преобразование

Существует 3 наиболее широко используемых преобразования: 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 получаем 0, иначе из непустой строки «считывается» число. При ошибке результат NaN. |

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

Подчиняется правилам:

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

* let var const – differences

## *Переменные var Можно Обновлять и Переопределять*

## *Область Видимости Переменной var – Функция.*

Это означает, что мы можем "изолировать" переменную var, если создадим ее **внутри** какой-либо функции function(){}.

В случае, если переменная var создается **за пределами** функции – она доступна в **глобальной** области видимости, что может создавать **проблемы**.

В отличие от var, область видимости let и const не функция, а **блок** {…}, то есть сюда попадают как функции, так и любые другие элементы с **фигурными** скобками.

Если в нашем примере объявить переменную kilograms с помощью let или const, то она **не будет** доступна за пределами блока if (){…}.

## Строгое сравнение

JavaScript предоставляет три оператора сравнения величин:

* равенство ("двойное равно") использует [== (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#equality),
* строгое равенство (или "тройное равно" или "идентично") использует [=== (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#identity),
* и [Object.is](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/is) (новшество из ECMAScript 6).

Выбор оператора зависит от типа сравнения, которое необходимо произвести.

В общих чертах, двойное равно перед сравнением величин производит приведение типов; тройное равно сравнивает величины без приведения (если величины разных типов, вернёт false, даже не сравнивая); ну и Object.is ведёт себя так же, как и тройное равно, но со специальной обработкой для NaN, -0 и +0, возвращая false при сравнении -0 и +0, и true для операции Object.is(NaN, NaN). (В то время как двойное или тройное равенство вернут false согласно стандарту IEEE 754.) Следует отметить, что все эти различия в сравнениях применимы лишь для примитивов. Для любых не примитивных объектов x и y, которые имеют одинаковые структуры, но представляют собой два отдельных объекта (переменные x и y не ссылаются на один и тот же объект), все операторы сравнения вернут false.

[**Сравнение с использованием ==**](https://developer.mozilla.org/ru/docs/Web/JavaScript/Equality_comparisons_and_sameness#%D1%81%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC)

Перед сравнением оператор равенства *приводит* обе величины к общему типу. После приведений (одного или обоих операндов), конечное сравнение выполняется также как и для ===. Операция сравнения *симметрична*: A == B возвращает то же значение, что и B == A для любых значений A и B.

## Object.is()

Метод Object.is() определяет, являются ли два значения [одинаковыми значениями (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness). Два значения являются одинаковыми в следующих случаях:

* оба равны [undefined](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/undefined)
* оба равны [null](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/null)
* оба равны true, либо оба равны false
* оба являются строками с одинаковой длиной и одинаковыми символами
* оба являются одним и тем же объектом
* оба являются числами и
  + оба равны +0
  + оба равны -0
  + оба равны [NaN](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/NaN)
  + либо оба не равны нулю или [NaN](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/NaN) и оба имеют одинаковое значение

Поведение этого метода *не* аналогично оператору [== (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators). Оператор [== (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators) использует приведение типов обоих операндов (если они имеют различный тип) перед проверкой на равенство (в результате получается, что проверка "" == false даёт true), а метод Object.is приведение типов не выполняет.

Поведение этого метода *не* аналогично оператору [=== (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators). Оператор [=== (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators) (также как и оператор [== (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators)) считает числовые значения -0 и +0 равными, а значение [Number.NaN](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Number/NaN) не равным самому себе.

## Полифилы

Полифил — это фрагмент кода (в сети — обычно JavaScript), который позволяет использовать современную функциональность в более старых браузерах, которые не поддерживают ее по умолчанию.

Например, полифил можно использовать, чтобы эмулировать функциональность [text-shadow](https://developer.mozilla.org/ru/docs/Web/CSS/text-shadow) в IE7 с помощью нативных фильтров браузера, или рем и медиавыражения, динамически меняя стилизацию в нужных случаях с помощью JavaScript, или что-либо еще, что вам потребуется.

Из-за меньшей производительности и ограниченной функциональности нельзя использовать исключительно полифилы. Нативная реализация API быстрее и с ней можно сделать больше, чем с помощью полифила. Например, [полифил Object.create](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/create#polyfill) может эмулировать только то, что доступно для ненативной реализации Object.create.

В других случаях полифилы нужны, чтобы разрешить ситуации, когда браузеры реализуют одни и те же возможности разными способами. Тогда полифил использует нестандартные возможности конкретного браузера, чтобы в результате определенная функциональность была совместима с действующими стандартами JavaScript. Хотя такое применение полифилов и редкость сейчас, во времена IE6 и Netscape, когда каждый браузер реализовывал JavaScript очень по-разному, оно было широко распространено. [Первая версия JQuery](https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.js) была ранним примером полифила. Она представляла собой компиляцию из обходных путей, специфических для определенных браузеров, которая предоставляла JavaScript-разработчикам единый API для всех браузеров. В то время одной из наибольших проблем было заставить сайт работать на всех устройствах: браузеры настолько существенно различались, что порой код приходилось писать совершенно по-разному и разрабатывать разные пользовательские интерфейсы, исходя из используемого пользователем браузера. Таким образом, у JavaScript-разработчиков был доступ только к очень лимитированному количеству JavaScript API, которые работали более или менее одинаково во всех браузерах. Сейчас использование полифилов для взаимодействия со специфичными для браузера реализациями возможностей менее распространено, так как современные браузеры в большинстве своем имеют большой набор стандартизированных API.

## Всплытие

Принцип всплытия очень простой.

**Когда на элементе происходит событие, обработчики сначала срабатывают на нём, потом на его родителе, затем выше и так далее, вверх по цепочке предков.**

***Почти* все события всплывают.**

Ключевое слово в этой фразе – «почти».

Например, событие focus не всплывает. В дальнейшем мы увидим и другие примеры. Однако, стоит понимать, что это скорее исключение, чем правило, всё-таки большинство событий всплывают.

# Шаблонные строки

Шаблонными литералами называются строковые литералы, допускающие использование выражений внутри. С ними вы можете использовать многострочные литералы и строковую интерполяцию. В спецификациях до ES2015 они назывались "шаблонными строками".

Шаблонные литералы заключены в обратные кавычки (` `) вместо двойных или одинарных. Они могут содержать подстановки, обозначаемые знаком доллара и фигурными скобками (${выражение}). Выражения в подстановках и текст между ними передаются в функцию. По умолчанию функция просто объединяет все части в строку. Если перед строкой есть выражение (здесь это tag), то шаблонная строка называется "теговым шаблоном". В этом случае, теговое выражение (обычно функция) вызывается с обработанным шаблонным литералом, который вы можете изменить перед выводом. Для экранирования обратной кавычки в шаблонных литералах указывается обратный слеш **\**.

* arrow func/ func expression/ func declaration

[**Объявление функций**](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Functions#%D0%BE%D0%B1%D1%8A%D1%8F%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B9)

[**Функции вида "function declaration statement"**](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Functions#%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8_%D0%B2%D0%B8%D0%B4%D0%B0_function_declaration_statement)

Объявление функции (*function definition*, или *function declaration*, или *function statement*) состоит из ключевого слова [function](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/function) и следующих частей:

* Имя функции.
* Список параметров (принимаемых функцией) заключённых в круглые скобки () и разделённых запятыми.
* Инструкции, которые будут выполнены после вызова функции, заключают в фигурные скобки { }.

### [Функции вида "function definition expression"](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Functions#%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8_%D0%B2%D0%B8%D0%B4%D0%B0_function_definition_expression)

Функция вида "function declaration statement" по синтаксису является инструкцией (statement), ещё функция может быть вида "function definition expression". Такая функция может быть **анонимной** (она не имеет имени). Например, функция square может быть вызвана так:

var square = function(number) { return number \* number; };

var x = square(4); // x получает значение 16

Copy to Clipboard

Однако, имя может быть и присвоено для вызова самой себя внутри самой функции и для отладчика (debugger) для идентифицированные функции в стек-треках (stack traces; "trace" — "след" / "отпечаток").

[Стрелочные функции](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Functions/Arrow_functions) — функции вида "arrow function expression" (неверно **fat arrow function**) — имеют укороченный синтаксис по сравнению с function expression и лексически связывает значение this. Стрелочные функции всегда анонимны. Посмотрите также пост блога hacks.mozilla.org "[ES6 In Depth: Arrow functions](https://hacks.mozilla.org/2015/06/es6-in-depth-arrow-functions/)".

На введение стрелочных функций повлияли два фактора: более короткие функции и лексика this.

В arrow function значением this является окружающий его контекст

# Object.keys, values

## [Object.keys, values, entries](https://learn.javascript.ru/keys-values-entries" \l "object-keys-values-entries)

Для простых объектов доступны следующие методы:

* [Object.keys(obj)](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/keys) – возвращает массив ключей.
* [Object.values(obj)](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/values) – возвращает массив значений.
* [Object.entries(obj)](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/entries) – возвращает массив пар [ключ, значение].

Обратите внимание на различия (по сравнению с map, например):

|  | **Map** | **Object** |
| --- | --- | --- |
| Синтаксис вызова | map.keys() | Object.keys(obj), не obj.keys() |
| Возвращает | перебираемый объект | «реальный» массив |

Первое отличие в том, что мы должны вызвать Object.keys(obj), а не obj.keys().

Методы объектов js

## Object.create()-Метод [Object.create()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create) позволяет создавать новые объекты и соединять их с прототипами существующих объектов

## Object.keys()-Метод [Object.keys()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys) создает массив ключей объекта.

## Object.values()-Метод [Object.values()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/values) создает массив значений объекта.

## Object.entries()-Метод [Object.entries()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/entries) создает вложенный массив пар «ключ-значение» объекта.

## Object.assign()-Метод [Object.assign()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign) копирует значения из одного объекта в другой.