**JavaScript**

**Введене:**

Движки браузеров, работающих с JavaScript

* [V8](https://ru.wikipedia.org/wiki/V8_(%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA_JavaScript)) – в Chrome и Opera.
* [SpiderMonkey](https://ru.wikipedia.org/wiki/SpiderMonkey) – в Firefox.
* Trident и Chakra - для IE
* ChakraCore - для Microsoft Edge
* Nitro и SquirrelFish - для Safari
* и т.д.

**Спецификации JavaScript:**

Спецификация ECMA-262 - <https://www.ecma-international.org/publications/standards/Ecma-262.htm>

<https://tc39.es/ecma262/>

<https://github.com/tc39/proposals>

**Справочники:**  
**MDN (Mozilla) JavaScript Reference -** <https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference>

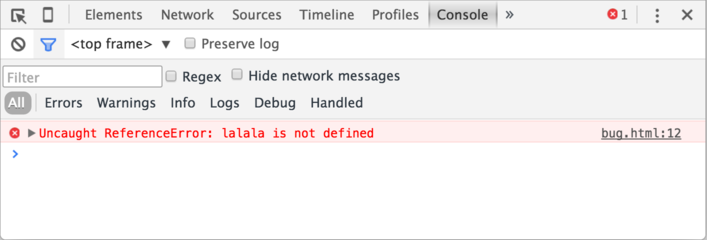
**MSDN -** <http://msdn.microsoft.com/>

**Таблицы совместимости**

* <http://caniuse.com> – таблицы с информацией о поддержке по каждой возможности языка. Например, чтобы узнать, какие движки поддерживают современные криптографические функции, посетите: <http://caniuse.com/#feat=cryptography>.
* <https://kangax.github.io/compat-table> – таблица с возможностями языка и движками, которые их поддерживают и не поддерживают.

**IDE**  
[WebStorm](http://www.jetbrains.com/webstorm/)  
  
**Devtools -** Chrome, Firefox, Edge и другие

Нажмите F12. По умолчанию в инструментах разработчика откроется вкладка Console (консоль). Она выглядит приблизительно следующим образом:



Точный внешний вид инструментов разработки зависит от используемой версии Chrome.

* В консоли мы можем увидеть сообщение об ошибке, отрисованное красным цветом. В нашем случае скрипт содержит неизвестную команду «lalala».
* Справа присутствует ссылка на исходный код bug.html:12 с номером строки кода, в которой эта ошибка и произошла.

Под сообщением об ошибке находится синий симв >. Он обозначает командную строку, в ней мы можем редактировать и запускать JavaScript-команды. Для их запуска нажмите Enter.

Многострочный ввод:

Обычно при нажатии Enter введённая строка кода сразу выполняется

Чтобы перенести строку, нажмите Shift+Enter.

Так можно вводить более длинный JS-код.

**Тег <script>**

Тег <script> содержит JavaScript-код, который автоматически выполнится, когда браузер его обработает.

**Внешние скрипты**

Если у вас много JavaScript-кода, вы можете поместить его в отдельный файл и вызывать его по абсолютному или относительному пути.

Если атрибут src установлен, содержимое тега script будет игнорироваться.  
<script src="/path/to/script.js"></script>

<script src=

"https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lodash.js">

</script>

<script src="/js/script1.js"></script>

<script src="/js/script2.js"></script>

Польза от отдельных файлов в том, что браузер загрузит скрипт отдельно и сможет хранить его в [кеше](https://en.wikipedia.org/wiki/Web_cache).

**Итого**

Для добавления кода JavaScript на страницу используется тег <script>

Атрибуты type и language необязательны.

Скрипт во внешнем файле можно вставить с помощью <script src="path/to/script.js"></script>.

**Структура кода**

Запись инструкций, структура кода, комментарии - Как у Джава!!!!

В JavaScript существует три типа кавычек.

Двойные кавычки: "Привет», Одинарные кавычки: 'Привет’, 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:

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.

**Строгий режим — "use strict"**

Для совместимости со старым кодом новые блоки инструкций записываются после команды -   
"use strict";

Нет директивы типа "no use strict", которая возвращала бы движок к старому поведению. Как только мы входим в строгий режим, отменить это невозможно.

**Консоль браузера**

В консоли **use strict** по умолчанию в ней выключен.

Перевод строки в консоли **Shift+Enter**.

'use strict'; <Shift+Enter для перехода на новую строку>

// ...ваш код...

<Enter для запуска>

Или для старых браузеров

(function() {

'use strict';

// ...ваш код...

})()

**Подытожим: пока очень желательно добавлять "use strict"; в начале ваших скриптов. Позже, когда весь ваш код будет состоять из классов и модулей, директиву можно будет опускать.**

**Переменные**

Для создания переменной в JavaScript используйте ключевое слово **let**.

Создание и инициализация переменных происходит как в джава.

Старые версии кода могут использовать let и устаревшее ключевое слово "var".

В JavaScript есть два ограничения, касающиеся имён переменных:

- имя переменной должно содержать только буквы, цифры или символы $ и \_.

- первый символ не должен быть цифрой.

- первый символ маленькая буква – дальше camelCase.

- $ и \_ могут быть именами переменных

- регистр имеет значение

- нелатинские буквы разрешены, но не рекомендуются

- нельзя использовать в качестве имён переменных - let, class, return и function

**Константы в верхнем регистре**

Чтобы объявить константную, то есть, неизменяемую переменную, используйте const вместо let.

Названия таких констант пишутся с использованием заглавных букв и подчёркивания.

Преимущества:

- COLOR\_ORANGE гораздо легче запомнить, чем "#FF7F00".

- Исключение ошибок при вводе "#FF7F00", чем при вводе COLOR\_ORANGE.

- При чтении кода COLOR\_ORANGE намного понятнее, чем #FF7F00.

Константа может вычисляться при первой инициализации, но потом не изменяется.

Несколько хороших правил:

- Используйте легко читаемые имена, такие как userName или shoppingCart.

- Избегайте использования аббревиатур или коротких имён, таких как a, b, c, за исключением тех случаев, когда вы точно знаете, что так нужно.

- Делайте имена максимально описательными и лаконичными. Примеры плохих имён: data и value. Такие имена ничего не говорят.

- Договоритесь с вашей командой об используемых терминах. Если посетитель сайта называется «user», тогда мы должны называть связанные с ним переменные currentUser или newUser, а не, к примеру, currentVisitor или newManInTown.

**Итого**

Мы можем объявить переменные для хранения данных с помощью ключевых слов var, let или const.

let – это современный способ объявления.

var – это устаревший способ объявления. Обычно мы вообще не используем его, но мы рассмотрим тонкие отличия от let в главе Устаревшее ключевое слово "var" на случай, если это всё-таки вам понадобится.

const – похоже на let, но значение переменной не может изменяться.

Переменные должны быть названы таким образом, чтобы мы могли легко понять, что у них внутри.

Типы данных

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

// Не будет ошибкой

let message = "hello";

message = 123456;

Языки программирования, в которых такое возможно, называются «динамически типизированными».

**Число**

let n = 123;

n = 12.345;

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

 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 – 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.

**[BigInt](https://learn.javascript.ru/types" \l "bigint)**

В JavaScript тип «number» не может содержать числа больше, чем (253-1) (т. е. 9007199254740991), или меньше, чем -(253-1) для отрицательных чисел.

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;

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

**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:

let age;

alert(age); // выведет "undefined"

Технически мы можем присвоить значение undefined любой переменной:

let age = 123; // изменяем значение на undefined

age = undefined;

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.

**Оператор typeof**

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:

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)*

**Итого**

В JavaScript есть 8 основных типов.

number для любых чисел: целочисленных или чисел с плавающей точкой; целочисленные значения ограничены диапазоном ±(253-1).

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.

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.

**Взаимодействие: alert, prompt, confirm**

Итого

alert - показывает сообщение.

prompt - показывает сообщение и запрашивает ввод текста от пользователя. Возвращает напечатанный в поле ввода текст или null, если была нажата кнопка «Отмена» или Esc с клавиатуры.

confirm - показывает сообщение и ждёт, пока пользователь нажмёт OK или Отмена. Возвращает true, если нажата OK, и false, если нажата кнопка «Отмена» или Esc с клавиатуры.

Все эти методы являются модальными: останавливают выполнение скриптов и не позволяют пользователю взаимодействовать с остальной частью страницы до тех пор, пока окно не будет закрыто.

На все указанные методы распространяются два ограничения:

* Расположение окон определяется браузером. Обычно окна находятся в центре.
* Визуальное отображение окон зависит от браузера, и мы не можем изменить их вид.

**Преобразование типов**

**Строковое преобразование**

Можно использовать функцию String(value), чтобы преобразовать значение к строке. 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";

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

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.

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 получаем 0, иначе из непустой строки «считывается» число. При ошибке результат NaN. |

**Логическое преобразование**

Логическое преобразование 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.

Заметим, что строчка с нулём "0" — это true

**Базовые операторы, математика**

**Термины: «унарный», «бинарный», «операнд»**

 *Операнд* – то, к чему применяется оператор. Например, в умножении 5 \* 2 есть два операнда: левый операнд равен 5, а правый операнд равен 2. Иногда их называют «аргументами» вместо «операндов».

 *Унарным* называется оператор, который применяется к одному операнду. Например, оператор унарный минус "-" меняет знак числа на противоположный:

 *Бинарным* называется оператор, который применяется к двум операндам. Тот же минус существует и в бинарной форме:

**Математика**

Поддерживаются следующие математические операторы:

* Сложение +,
* Вычитание -,
* Умножение \*,
* Деление /,
* Взятие остатка от деления %,
* Возведение в степень \*\*.

Результат a % b – это [остаток](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81_%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%BE%D0%BC) от целочисленного деления a на b.

В выражении a \*\* b оператор возведения в степень умножает a на само себя b раз.

Математически, оператор работает и для нецелых чисел. Например, квадратный корень является возведением в степень ½.

**Сложение строк при помощи бинарного +**

Обычно при помощи плюса '+' складывают числа, но если бинарный оператор '+' применить к строкам, то он их объединяет в одну.

lert(2 + 2 + '1' ); // будет "41", а не "221"

Здесь операторы работают один за другим. Первый + складывает два числа и возвращает 4, затем следующий + объединяет результат со строкой, производя действие 4 + '1' = 41.

Сложение и преобразование строк — это особенность бинарного плюса +. Другие арифметические операторы работают только с числами и всегда преобразуют операнды в числа.

**Приведение к числу, унарный +**

Унарный, то есть применённый к одному значению, плюс + ничего не делает с числами. Но если операнд не число, унарный плюс преобразует его в число.

// Не влияет на числа

let x = 1;

alert( +x ); // 1

let y = -2;

alert( +y ); // -2

// Преобразует не числа в числа

alert( +true ); // 1

alert( +"" ); // 0

На самом деле это то же самое, что и Number(...), только короче.

let apples = "2";

let oranges = "3";

*// оба операнда предварительно преобразованы в числа*

*alert( +apples + +oranges ); // 5*

// более длинный вариант

// alert( Number(apples) + Number(oranges) ); // 5

**Приоритет операторов**

Так как «унарный плюс» имеет приоритет 17, который выше, чем 13 у «сложения» (бинарный плюс), то в выражении "+apples + +oranges" сначала выполнятся унарные плюсы, а затем сложение.

|  |  |  |
| --- | --- | --- |
| Приоритет | Название | Обозначение |
| 17 | Унарный плюс | + |
| 17 | Унарный минус | - |
| 16 | Возведение в степень | \*\* |
| 15 | Умножение | \* |
| 15 | Деление | / |
| 13 | Сложение | + |
| 13 | Вычитание | - |
| 3 | Присваивание | = |

**Присваивание**

Присваивание = возвращает значение. Вызов x = value записывает value в x и возвращает его.

Присваивание по цепочке *a = b = c = 2 + 2;*

**Сокращённая арифметика с присваиванием**

Запись можно укоротить при помощи совмещённых операторов += и \*=  
Краткие формы записи существуют для всех арифметических и побитовых операторов: /= , -=

**Инкремент/декремент**

**Инкремент** ++ увеличивает переменную на 1

**Декремент** -- уменьшает переменную на 1

Инкремент/декремент можно применить только к переменной. Попытка использовать его на значении, типа 5++, приведёт к ошибке.

Операторы ++ и -- могут быть расположены не только после, но и до переменной.

* Когда оператор идёт после переменной — это «постфиксная форма»: counter++.
* «Префиксная форма» — это когда оператор идёт перед переменной: ++counter.

Если результат оператора не используется, а нужно только увеличить/уменьшить переменную, тогда без разницы, какую форму использовать:

let counter = 0;

counter++;

++counter;

alert( counter ); // 2, обе строки сделали одно и то же

Если хочется тут же использовать результат, то нужна префиксная форма:

let counter = 0;

alert( ++counter ); // 1

Если нужно увеличить и при этом получить значение переменной *до увеличения* – нужна постфиксная форма:

let counter = 0;

alert( counter++ ); // 0

**Побитовые операторы**

Поддерживаются следующие побитовые операторы:

AND(и) ( & )

OR(или) ( | )

XOR(побитовое исключающее или) ( ^ )

NOT(не) ( ~ )

LEFT SHIFT(левый сдвиг) ( << )

RIGHT SHIFT(правый сдвиг) ( >> )

ZERO-FILL RIGHT SHIFT(правый сдвиг с заполнением нулями) ( >>> )

**Оператор «запятая»**

Оператор «запятая» (,) редко применяется и является одним из самых необычных. Иногда он используется для написания более короткого кода, поэтому нам нужно знать его, чтобы понимать, что при этом происходит.

Оператор «запятая» предоставляет нам возможность вычислять несколько выражений, разделяя их запятой ,. Каждое выражение выполняется, но возвращается результат только последнего.

Например:

let a = (1 + 2, 3 + 4);

alert( a ); // 7 (результат вычисления 3 + 4)

Первое выражение 1 + 2 выполняется, а результат отбрасывается. Затем идёт 3 + 4, выражение выполняется и возвращается результат.

// три операции в одной строке

for (*a = 1, b = 3, c = a \* b*; a < 10; a++) {

...

}

**Операторы сравнения**

* Больше/меньше: a > b, a < b.
* Больше/меньше или равно: a >= b, a <= b.
* Равно: a == b. Обратите внимание, для сравнения используется двойной знак равенства ==. Один знак равенства a = b означал бы присваивание.
* Не равно. В математике обозначается символом ≠, но в JavaScript записывается как a != b.

**Результат сравнения имеет логический тип**

Все операторы сравнения возвращают значение логического типа:

true – означает «да», «верно», «истина».

false – означает «нет», «неверно», «ложь».

**Сравнение строк**

JavaScript использует «алфавитный» или «лексикографический» порядок.

Другими словами, строки сравниваются посимвольно.

Например:

alert( 'Я' > 'А' ); // true

alert( 'Коты' > 'Кода' ); // true

alert( 'Сонный' > 'Сон' ); // true

Алгоритм сравнения двух строк довольно прост:

1. Сначала сравниваются первые символы строк.
2. Если первый символ первой строки больше (меньше), чем первый символ второй, то первая строка больше (меньше) второй. Сравнение завершено.
3. Если первые символы равны, то таким же образом сравниваются уже вторые символы строк.
4. Сравнение продолжается, пока не закончится одна из строк.
5. Если обе строки заканчиваются одновременно, то они равны. Иначе, большей считается более длинная строка.

В примерах выше сравнение 'Я' > 'А' завершится на первом шаге, тогда как строки 'Коты' и 'Кода' будут сравниваться посимвольно:

1. К равна К.
2. о равна о.
3. т больше, чем д.
4. На этом сравнение заканчивается.
5. Первая строка больше.

Используется кодировка Unicode, а не настоящий алфавит

Приведённый выше алгоритм сравнения похож на алгоритм, используемый в словарях и телефонных книгах, но между ними есть и различия.

Например, в JavaScript имеет значение регистр символов. Заглавная буква "A" не равна строчной "a". Cтрочные буквы имеют больший код во внутренней таблице кодирования, которую использует JavaScript (Unicode).

**Сравнение разных типов**

При сравнении значений разных типов JavaScript приводит каждое из них к числу.

Например:

alert( '2' > 1 ); // true, строка '2' становится числом 2

alert( '01' == 1 ); // true, строка '01' становится числом 1

Логическое значение true становится 1, а false – 0.

Например:

alert( true == 1 ); // true

alert( false == 0 ); // true

Забавное следствие

**Возможна следующая ситуация:**

Два значения равны.

Одно из них true как логическое значение, другое – false.

Например:

let a = 0;

alert( Boolean(a) ); // false

let b = "0";

alert( Boolean(b) ); // true

alert(a == b); // true!

С точки зрения JavaScript, результат ожидаем. Равенство преобразует значения, используя числовое преобразование, поэтому "0" становится 0. В то время как явное преобразование с помощью Boolean использует другой набор правил.

**Строгое сравнение**

**Оператор строгого равенства === проверяет равенство без приведения типов.**

Другими словами, если a и b имеют разные типы, то проверка a === b немедленно возвращает false без попытки их преобразования.

Ещё есть оператор строгого неравенства !==, аналогичный !=.

Оператор строгого равенства дольше писать, но он делает код более очевидным и оставляет меньше места для ошибок.

**Сравнение с null и undefined**

Поведение null и undefined при сравнении с другими значениями — особое:

**При строгом равенстве ===**

Эти значения различны, так как различны их типы.

alert( null === undefined ); // false

**При нестрогом равенстве ==**

Эти значения равны друг другу и не равны никаким другим значениям. Это специальное правило языка.

alert( null == undefined ); // true

**При использовании математических операторов и других операторов сравнения < > <= >=**

Значения null/undefined преобразуются к числам: null становится 0, а undefined – NaN.

**Странный результат сравнения null и 0**

Сравним null с нулём:

alert( null > 0 ); // (1) false

alert( null == 0 ); // (2) false

alert( null >= 0 ); // (3) true

С точки зрения математики это странно. Результат последнего сравнения говорит о том, что "null больше или равно нулю", тогда результат одного из сравнений выше должен быть true, но они оба ложны.

Причина в том, что нестрогое равенство и сравнения > < >= <= работают по-разному. Сравнения преобразуют null в число, рассматривая его как 0. Поэтому выражение (3) null >= 0 истинно, а null > 0 ложно.

С другой стороны, для нестрогого равенства == значений undefined и null действует особое правило: эти значения ни к чему не приводятся, они равны друг другу и не равны ничему другому. Поэтому (2) null == 0 ложно.

**Несравненное значение undefined**

Значение undefined несравнимо с другими значениями:

alert( undefined > 0 ); // false (1)

alert( undefined < 0 ); // false (2)

alert( undefined == 0 ); // false (3)

Почему же сравнение undefined с нулём всегда ложно?

На это есть следующие причины:

* Сравнения (1) и (2) возвращают false, потому что undefined преобразуется в NaN, а NaN – это специальное числовое значение, которое возвращает false при любых сравнениях.
* Нестрогое равенство (3) возвращает false, потому что undefined равно только null, undefined и ничему больше.

**Как избежать проблем**

* Относитесь очень осторожно к любому сравнению с undefined/null, кроме случаев строгого равенства ===.
* Не используйте сравнения >= > < <= с переменными, которые могут принимать значения null/undefined, разве что вы полностью уверены в том, что делаете. Если переменная может принимать эти значения, то добавьте для них отдельные проверки.

**Итого**

* Операторы сравнения возвращают значения логического типа.
* Строки сравниваются посимвольно в лексикографическом порядке.
* Значения разных типов при сравнении приводятся к числу. Исключением является сравнение с помощью операторов строгого равенства/неравенства.
* Значения null и undefined равны == друг другу и не равны любому другому значению.
* Будьте осторожны при использовании операторов сравнений вроде > и < с переменными, которые могут принимать значения null/undefined. Хорошей идеей будет сделать отдельную проверку на null/undefined.

**Условное ветвление: if, '?'**

Для этого мы можем использовать инструкцию if и условный оператор ?, который также называют оператором «вопросительный знак».

**Инструкция «if»**

Инструкция if(...) вычисляет условие в скобках и, если результат true, то выполняет блок кода.

Например:

let year = prompt('В каком году была опубликована спецификация ECMAScript-2015?', '');

if (year == 2015) alert( 'Вы правы!' );

В примере выше, условие – это простая проверка на равенство (year == 2015), но оно может быть и гораздо более сложным.

Если мы хотим выполнить более одной инструкции, то нужно заключить блок кода в фигурные скобки:

if (year == 2015) {

alert( "Правильно!" );

alert( "Вы такой умный!" );

}

Мы рекомендуем использовать фигурные скобки {} всегда, когда вы используете инструкцию if, даже если выполняется только одна команда. Это улучшает читабельность кода.

**Преобразование к логическому типу**

Инструкция if (…) вычисляет выражение в скобках и преобразует результат к логическому типу.

Давайте вспомним правила преобразования типов из главы Преобразование типов:

* Число 0, пустая строка "", null, undefined и NaN становятся false. Из-за этого их называют «ложными» («falsy») значениями.
* Остальные значения становятся true, поэтому их называют «правдивыми» («truthy»).

**Блок «else»**

Инструкция if может содержать необязательный блок «else» («иначе»). Он выполняется, когда условие ложно.

if (year == 2015) {

alert( 'Да вы знаток!' );

} else {

alert( 'А вот и неправильно!' ); // любое значение, кроме 2015

}

**Несколько условий: «else if»**

Иногда, нужно проверить несколько вариантов условия. Для этого используется блок else if.

if (year < 2015) {

alert( 'Это слишком рано...' );

} else if (year > 2015) {

alert( 'Это поздновато' );

} else {

alert( 'Верно!' );

}

**Блоков else if может быть и больше. Присутствие блока else не является обязательным.**

**Тернарный оператор ?**

let result = условие ? значение1 : значение2;

Сначала вычисляется условие: если оно истинно, тогда возвращается значение1, в противном случае – значение2.

let accessAllowed = (age > 18) ? true : false;

Технически, мы можем опустить круглые скобки вокруг age > 18. Оператор вопросительного знака имеет низкий приоритет, поэтому он выполняется после сравнения >.

**На заметку:**

В примере выше вы можете избежать использования оператора вопросительного знака ?, т.к. сравнение само по себе уже возвращает true/false:

// то же самое

let accessAllowed = age > 18;

**Несколько операторов „?“**

Последовательность операторов вопросительного знака ? позволяет вернуть значение, которое зависит от более чем одного условия.

let age = prompt('Возраст?', 18);

let message = (age < 3) ? 'Здравствуй, малыш!' :

(age < 18) ? 'Привет!' :

(age < 100) ? 'Здравствуйте!' :

'Какой необычный возраст!';

alert( message );

Вот как это выглядит при использовании if..else:

if (age < 3) {

message = 'Здравствуй, малыш!';

} else if (age < 18) {

message = 'Привет!';

} else if (age < 100) {

message = 'Здравствуйте!';

} else {

message = 'Какой необычный возраст!';

}

**Логические операторы**

В JavaScript есть три логических оператора: || (ИЛИ), && (И) и ! (НЕ).

**|| (ИЛИ)**

ИЛИ предназначено только для манипулирования булевыми значениями: в случае, если какой-либо из аргументов true, он вернёт true, в противоположной ситуации возвращается false.

Существует всего четыре возможные логические комбинации:

alert( true || true ); // true

alert( false || true ); // true

alert( true || false ); // true

alert( false || false ); // false

В JavaScript, как мы увидим далее, этот оператор работает несколько иным образом.

Если значение не логического типа, то оно к нему приводится в целях вычислений.

Например, число 1 будет воспринято как true, а 0 – как false:

if (1 || 0) { // работает как if( true || false )

alert( 'truthy!' );

}

**ИЛИ «||» находит первое истинное значение**

Расширенный алгоритм работает следующим образом.

При выполнении ИЛИ || с несколькими значениями:

result = value1 || value2 || value3;

Оператор || выполняет следующие действия:

* Вычисляет операнды слева направо.
* Каждый операнд конвертирует в логическое значение. Если результат true, останавливается и возвращает исходное значение этого операнда.
* Если все операнды являются ложными (false), возвращает последний из них.

Значение возвращается в исходном виде, без преобразования.

Другими словами, цепочка ИЛИ "||" возвращает первое истинное значение или последнее, если такое значение не найдено.

**Получение первого истинного значения из списка переменных или выражений.**

Представим, что у нас имеется ряд переменных, которые могут содержать данные или быть null/undefined. Как мы можем найти первую переменную с данными?

С помощью ||:

let currentUser = null;

let defaultUser = "John";

let name = currentUser || defaultUser || "unnamed";

alert( name ); // выбирается "John" – первое истинное значение

Если бы и currentUser, и defaultUser были ложными, в качестве результата мы бы наблюдали "unnamed".