Java Script

Программы на JavaScript могут быть вставлены в любое место HTML-документа с помощью тега <script>.F.e <script src="/js/script1.js"></script>

**alert** выводит на экран окно с сообщением и приостанавливает выполнение скрипта, пока пользователь не нажмёт «ОК» F.e alert('Мир'); alert('Мир');

**Однострочные комментарии начинаются с двойной косой черты //.**

Часть строки после // считается комментарием. Такой комментарий может как занимать строку целиком, так и находиться после инструкции. (Ctrl+/)

**Многострочные комментарии начинаются косой чертой со звёздочкой /\* и заканчиваются звёздочкой с косой чертой \*/.**

**Строгий режим — "use strict"**

[Переменная](https://learn.javascript.ru/variables" \l "peremennaya)-let F.e

let user = 'John';

let age = 25;

let message = 'Hello';

let message;

message = 'Hello!';

alert(message); // показывает содержимое переменной

1. 
2. Имя переменной должно содержать только буквы, цифры или символы $ и \_.
3. Первый символ не должен быть цифрой.
4. Существует [список зарезервированных слов](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Lexical_grammar#%D0%9A%D0%BB%D1%8E%D1%87%D0%B5%D0%B2%D1%8B%D0%B5_%D1%81%D0%BB%D0%BE%D0%B2%D0%B0), которые нельзя использовать в качестве имён переменных, потому что они используются самим языком.
5. Например: let, class, return и function зарезервированы.
6. Приведённый ниже код даёт синтаксическую ошибку

[Константы](https://learn.javascript.ru/variables#konstanty)

неизменяемую переменную, используйте const F.e const myBirthday = '18.04.1982';

* 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. Их можно использовать только в том случае, если из контекста кода очевидно, какие данные хранит переменная.
* 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.

## [Число](https://learn.javascript.ru/types" \l "chislo)

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.

NaN означает вычислительную ошибку. Это результат неправильной или неопределённой математической операции, например: alert( "не число" / 2 ); // NaN, такое деление является ошибкой

**[BigInt](https://learn.javascript.ru/types" \l "bigint)** В JavaScript тип «number» не может содержать числа больше, чем 253 (или меньше, чем -253 для отрицательных).

## [Строка](https://learn.javascript.ru/types" \l "stroka)---Строка (string) в JavaScript должна быть заключена в кавычки.

В JavaScript существует три типа кавычек.

1. Двойные кавычки: "Привет".
2. Одинарные кавычки: 'Привет'.
3. Обратные кавычки: `Привет`.
4. Двойные или одинарные кавычки являются «простыми», между ними нет разницы в JavaScript.
5. Обратные кавычки же имеют «расширенную функциональность». Они позволяют нам встраивать выражения в строку, заключая их в ${…}. Напримth let name = "Иван";

// Вставим переменную

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

// Вставим выражение

alert( `результат: ${1 + 2}` ); // результат: 3

## [Булевый (логический) тип](https://learn.javascript.ru/types" \l "bulevyy-logicheskiy-tip)

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

1. Булевые значения также могут быть результатом сравнений:

## [Значение «null»](https://learn.javascript.ru/types" \l "znachenie-null)- Это просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно».

## [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: typeof(x).
3. typeof undefined // "undefined"
4. typeof 0 // "number"
5. typeof 10n // "bigint"
6. typeof true // "boolean"
7. typeof "foo" // "string"
8. typeof Symbol("id") // "symbol"
9. typeof Math // "object" (1)
10. typeof null // "object" (2)
11. typeof alert // "function" (3)

**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

**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

**Операторы-**Многие операторы знакомы нам ещё со школы: сложение +, умножение \*, вычитание - и так далее.

* 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: let x = 1;

x = -x;

alert( x ); // -1, применили унарный минус

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

* let x = 1, y = 3;
* alert( y - x ); // 2, бинарный минус

Формально мы говорим о двух разных операторах: унарное отрицание (один операнд: меняет знак) и бинарное вычитание (два операнда: вычитает).

## [бинарный +](https://learn.javascript.ru/operators" \l "slozhenie-strok-binarnyy)

let s = "моя" + "строка";

alert(s); // моястрока

если хотя бы один из операндов является строкой, то второй будет также преобразован к строке.

 Другие арифметические операторы работают только с числами и всегда преобразуют операнды в числа. alert( 2 - '1' ); // 1

alert( '6' / '2' ); // 3

## [унарный плюс +](https://learn.javascript.ru/operators" \l "preobrazovanie-k-chislu-unarnyy-plyus)

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

let x = 1;

alert( +x ); // 1

let y = -2;

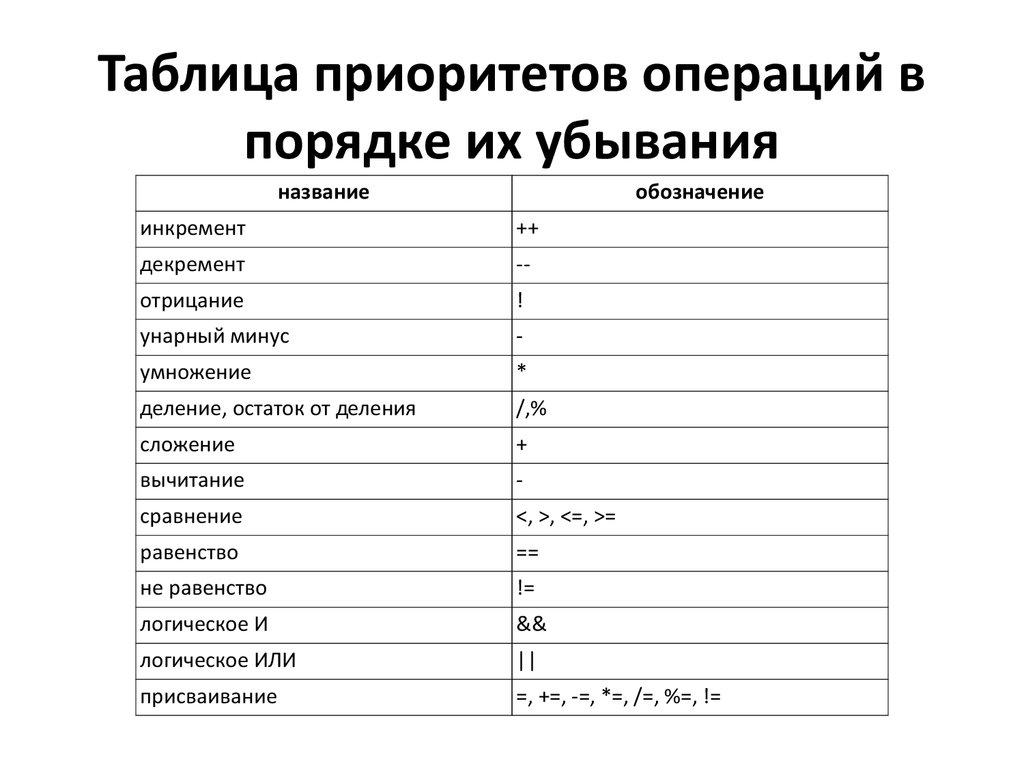
alert( +y ); // -2

// Преобразует нечисла в числа

alert( +true ); // 1

alert( +"" ); // 0

[таблицы приоритетов](https://developer.mozilla.org/ru/JavaScript/Reference/operators/operator_precedence)



## [Остаток от деления %](https://learn.javascript.ru/operators" \l "ostatok-ot-deleniya)

Оператор взятия остатка %, несмотря на обозначение, никакого отношения к процентам не имеет.

Его результат a % b – это остаток от деления a на b.

alert( 5 % 2 ); // 1, остаток от деления 5 на 2

alert( 8 % 3 ); // 2, остаток от деления 8 на 3

alert( 6 % 3 ); // 0, остаток от деления 6 на 3

## [Возведение в степень \*\*](https://learn.javascript.ru/operators" \l "vozvedenie-v-stepen)

Оператор возведения в степень \*\* недавно добавили в язык.

Для натурального числа b результат a \*\* b равен a, умноженному на само себя b раз.

Например:

alert( 2 \*\* 2 ); // 4 (2 \* 2)

alert( 2 \*\* 3 ); // 8 (2 \* 2 \* 2)

alert( 2 \*\* 4 ); // 16 (2 \* 2 \* 2 \* 2)

## [Инкремент/декремент](https://learn.javascript.ru/operators" \l "inkrement-dekrement)

Одной из наиболее частых операций в JavaScript, как и во многих других языках программирования, является увеличение или уменьшение переменной на единицу.

Для этого существуют даже специальные операторы:

* **Инкремент** ++ увеличивает на 1:
* let counter = 2;
* counter++; // работает как counter = counter + 1, просто запись короче

alert( counter ); // 3

* **Декремент** -- уменьшает на 1:
* let counter = 2;
* counter--; // работает как counter = counter - 1, просто запись короче

alert( counter ); // 1

let counter = 1;

alert( 2 \* ++counter ); // 4

Сравните с:

let counter = 1;

alert( 2 \* counter++ ); // 2, потому что counter++ возвращает "старое" значение

## [Побитовые операторы](https://learn.javascript.ru/operators" \l "pobitovye-operatory)

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) ( >>> )

## [Сокращённая арифметика с присваиванием](https://learn.javascript.ru/operators" \l "sokraschyonnaya-arifmetika-s-prisvaivaniem)

Часто нужно применить оператор к переменной и сохранить результат в ней же.

Например:

let n = 2;

n = n + 5;

n = n \* 2;

Эту запись можно укоротить при помощи совмещённых операторов += и \*=:

let n = 2;

n += 5; // теперь n=7 (работает как n = n + 5)

n \*= 2; // теперь n=14 (работает как n = n \* 2)

alert( n ); // 14

## [Оператор запятая](https://learn.javascript.ru/operators" \l "operator-zapyataya)

Оператор «запятая» , редко используется 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 3 + 4)

**Операторы сравнения**

Многие операторы сравнения известны нам из математики:

* 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 ≠. В JavaScript записывается как знак равенства с предшествующим ему восклицательным знаком: a != b.

## [Результат сравнения имеет логический тип](https://learn.javascript.ru/comparison" \l "rezultat-sravneniya-imeet-logicheskiy-tip)

Операторы сравнения, как и другие операторы, возвращают значение. Это значение имеет логический тип.

* true – означает «да», «верно», «истина».
* false – означает «нет», «неверно», «ложь».

**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.

Давайте проверим:

alert( 0 === false ); // false, так как сравниваются разные типы

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

## [prompt](https://learn.javascript.ru/alert-prompt-confirm" \l "prompt)

Функция prompt принимает два аргумента:

result = prompt(title, [default]);

Этот код отобразит модальное окно с текстом, полем для ввода текста и кнопками OK/Отмена.

**title**

Текст для отображения в окне.

**default**

Необязательный второй параметр, который устанавливает начальное значение в поле для текста в окне.

Пользователь может напечатать что-либо в поле ввода и нажать OK. Он также может отменить ввод нажатием на кнопку «Отмена» или нажав на клавишу Esc.

Вызов prompt вернёт текст, указанный в поле для ввода, или null, если ввод отменён пользователем.

Например:

let age = prompt('Сколько тебе лет?', 100);

alert(`Тебе ${age} лет!`); // Тебе 100 лет!

## [confirm](https://learn.javascript.ru/alert-prompt-confirm" \l "confirm)

Синтаксис:

result = confirm(question);

Функция confirm отображает модальное окно с текстом вопроса question и двумя кнопками: OK и Отмена.

Результат true, если нажата кнопка OK. В других случаях – false.

Например:

let isBoss = confirm("Ты здесь главный?");

alert( isBoss ); // true, если нажата OK

## [Оператор «if»](https://learn.javascript.ru/ifelse" \l "operator-if)

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

## [Блок «else»](https://learn.javascript.ru/ifelse" \l "blok-else)

Оператор if может содержать необязательный блок «else» («иначе»). Выполняется, когда условие ложно.

Например:

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

if (year == 2015) {

alert( 'Да вы знаток!' );

} else {

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

}

## [Несколько условий: «else if»](https://learn.javascript.ru/ifelse" \l "neskolko-usloviy-else-if)

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

Например:

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

if (year < 2015) {

alert( 'Это слишком рано...' );

} else if (year > 2015) {

alert( 'Это поздновато' );

} else {

alert( 'Верно!' );

}

Так называемый «условный» оператор «вопросительный знак» позволяет нам сделать это более коротким и простым способом

let accessAllowed;

let age = prompt('Сколько вам лет?', '');

if (age > 18) {

accessAllowed = true;

} else {

accessAllowed = false;

}

alert(accessAllowed);

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

# Логические операторы

## [|| (ИЛИ)](https://learn.javascript.ru/logical-operators" \l "ili)

let hour = 9;

if (hour < 10 || hour > 18) {

alert( 'Офис закрыт.' );

}

let hour = 12;

let isWeekend = true;

if (hour < 10 || hour > 18 || isWeekend) {

alert( 'Офис закрыт.' ); // это выходной

}

## [&& (И)](https://learn.javascript.ru/logical-operators" \l "i)

# Вышеуказанные правила схожи с поведением ИЛИ. Разница в том, что И возвращает первое ложное значение, а ИЛИ –  первое истинное

## [! (НЕ)](https://learn.javascript.ru/logical-operators" \l "ne)

# . Циклы while и for

# Для многократного повторения одного участка кода предусмотрены циклы.

while (condition) {

// код

// также называемый "телом цикла"

}

let i = 0;

while (i < 3) { // выводит 0, затем 1, затем 2

alert( i );

i++;

}

let i = 0;

while (i < 3) { // выводит 0, затем 1, затем 2

alert( i );

i++;

}

let i = 3;

while (i) alert(i--)

## [Цикл «do…while»](https://learn.javascript.ru/while-for" \l "tsikl-dowhile)

do {

// тело цикла

} while (condition);

let i = 0;

do {

alert( i );

i++;

} while (i < 3);

let i = 0;

## [Цикл «for»](https://learn.javascript.ru/while-for" \l "tsikl-for)

for (начало; условие; шаг) {

// ... тело цикла ...

}

for (let i = 0; i < 3; i++) { // выведет 0, затем 1, затем 2

alert(i);

}

let i = 0; // мы уже имеем объявленную i с присвоенным значением

for (; i < 3; i++) { // нет необходимости в "начале"

alert( i ); // 0, 1, 2

}

## [Прерывание цикла: «break»](https://learn.javascript.ru/while-for" \l "preryvanie-tsikla-break)

## [Переход к следующей итерации: continue](https://learn.javascript.ru/while-for" \l "continue)

for (let i = 0; i < 10; i++) {

// если true, пропустить оставшуюся часть тела цикла

if (i % 2 == 0) continue;

alert(i); // 1, затем 3, 5, 7, 9

}

* while – Проверяет условие перед каждой итерацией.
* do..while – Проверяет условие после каждой итерации.
* for (;;) – Проверяет условие перед каждой итерацией, есть возможность задать дополнительные настройки.

# Конструкция "switch"

## [Синтаксис](https://learn.javascript.ru/switch" \l "sintaksis)

Конструкция switch имеет один или более блок case и необязательный блок default.

switch(x) {

case 'value1': // if (x === 'value1')

...

[break]

case 'value2': // if (x === 'value2')

...

[break]

default:

...

[break]

}

* Переменная x проверяется на строгое равенство первому значению value1, затем второму value2 и так далее.
* Если соответствие установлено – switch начинает выполняться от соответствующей директивы case и далее, до ближайшего break (или до конца switch).
* Если ни один case не совпал – выполняется (если есть) вариант default.
* let a = 2 + 2;
* switch (a) {
* case 3:
* alert( 'Маловато' );
* break;
* case 4:
* alert( 'В точку!' );
* break;
* case 5:
* alert( 'Перебор' );
* break;
* default:
* alert( "Нет таких значений" );
* }

# Функции

Зачастую нам надо повторять одно и то же действие во многих частях программы.

Например, необходимо красиво вывести сообщение при приветствии посетителя, при выходе посетителя с сайта, ещё где-нибудь.

Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «строительными блоками» программы.

Примеры встроенных функций вы уже видели – это alert(message), prompt(message, default) и confirm(question). Но можно создавать и свои.

function имя(параметры) {

...тело...

}

function showMessage() {

alert( 'Всем привет!' );

}

showMessage();

showMessage();

## [Внешние переменные](https://learn.javascript.ru/function-basics" \l "vneshnie-peremennye)

У функции есть доступ к внешним переменным, например:

let userName = 'Вася';

function showMessage() {

let message = 'Привет, ' + userName;

alert(message);

}

showMessage(); // Привет, Вася

Функция обладает полным доступом к внешним переменным и может изменять их значение.

Например:

let userName = 'Вася';

function showMessage() {

userName = "Петя"; // (1) изменяем значение внешней переменной

let message = 'Привет, ' + userName;

alert(message);

}

alert( userName ); // Вася перед вызовом функции

showMessage();

alert( userName );

## [Параметры](https://learn.javascript.ru/function-basics" \l "parametry)

Мы можем передать внутрь функции любую информацию, используя параметры (также называемые аргументы функции).

В нижеприведённом примере функции передаются два параметра: from и text.

function showMessage(from, text) { // аргументы: from, text

alert(from + ': ' + text);

}

showMessage('Аня', 'Привет!'); // Аня: Привет! (\*)

showMessage('Аня', "Как дела?"); // Аня: Как дела? (\*\*)

function sum

function sum

## V[ыбор имени функции](https://learn.javascript.ru/function-basics#function-naming)

Функция – это действие. Поэтому имя функции обычно является глаголом. Оно должно быть простым, точным и описывать действие функции, чтобы программист, который будет читать код, получил верное представление о том, что делает функция.

Как правило, используются глагольные префиксы, обозначающие общий характер действия, после которых следует уточнение. Обычно в командах разработчиков действуют соглашения, касающиеся значений этих префиксов.

Например, функции, начинающиеся с "show" обычно что-то показывают.

Функции, начинающиеся с…

* "get…" – возвращают значение,
* "calc…" – что-то вычисляют,
* "create…" – что-то создают,
* "check…" – что-то проверяют и возвращают логическое значение, и т
* showMessage(..) // показывает сообщение
* getAge(..) // возвращает возраст (в каком либо значении)
* calcSum(..) // вычисляет сумму и возвращает результат
* createForm(..) // создаёт форму (и обычно возвращает её)
* checkPermission(..) // проверяет доступ, возвращая true/false