<https://codepen.io/pen> - песочница

**Основы JavaScript.**

* Для добавления кода JavaScript на страницу используется тег <script>
* Скрипт во внешнем файле можно вставить с помощью <script src="path/to/script.js"></script>.

Как правило, только простейшие скрипты помещаются в HTML. Более сложные выделяются в отдельные файлы.

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

Другие страницы, которые подключают тот же скрипт, смогут брать его из кеша вместо повторной загрузки из сети. И таким образом файл будет загружаться с сервера только один раз.

Это сокращает расход трафика и ускоряет загрузку страниц.

**«use strict»**

1. Директива "use strict" переключает движок в «современный» режим, изменяя поведение некоторых встроенных функций. Позже в учебнике мы увидим подробности.
2. Строгий режим включается путём размещения "use strict" в начале скрипта или функции. Некоторые функции языка, такие как «классы» и «модули», автоматически включают строгий режим.
3. Строгий режим поддерживается всеми современными браузерами.
4. Мы рекомендуем всегда начинать скрипты с "use strict". Все примеры в этом руководстве предполагают строгий режим, если (очень редко) не указано иное.

**Переменные**

В JavaScript есть два ограничения, касающиеся имён переменных:

1. Имя переменной должно содержать только буквы, цифры или символы $ и \_.
2. Первый символ не должен быть цифрой

**Зарезервированные имена**

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.

Например: let, class, return и function зарезервированы.

Приведённый ниже код даёт синтаксическую ошибку:

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"](https://learn.javascript.ru/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.**

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.

В JavaScript тип «number» не может содержать числа больше, чем 253

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 JavaScript существует три типа кавычек.

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

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

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.

**Преобразования типов данных.**

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.

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.

"" + 1 + 0 = "10" // (1)

"" - 1 + 0 = -1 // (2)

true + false = 1

6 / "3" = 2

"2" \* "3" = 6

4 + 5 + "px" = "9px"

"$" + 4 + 5 = "$45"

"4" - 2 = 2

"4px" - 2 = NaN

7 / 0 = Infinity

" -9 " + 5 = " -9 5" // (3)

" -9 " - 5 = -14 // (4)

null + 1 = 1 // (5)

undefined + 1 = NaN // (6)

" \t \n" - 2 = -2 // (7)

1. Сложение со строкой "" + 1 преобразует 1 к строке: "" + 1 = "1", и в следующем случае "1" + 0, работает то же самое правило.
2. Вычитание - (как и большинство математических операторов) работает только с числами, пустая строка "" приводится к 0.
3. Сложение со строкой превращает число 5 в строку и добавляет к строке.
4. Вычитание всегда преобразует к числу, значит строка " -9 " становится числом -9 (пробелы по краям обрезаются).
5. null становится 0 после численного преобразования.
6. undefined становится NaN после численного преобразования.
7. Пробельные символы, такие как \t и \n по краям строки игнорируются при преобразовании в число, так что строка \t \n, аналогично пустой строке, становится 0 после численного преобразования.

**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).

Тем не менее, помните, что операции выполняются слева направо. Если перед строкой идут два числа, то числа будут сложены перед преобразованием в строку

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

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

Плюс + существует в двух формах: бинарной, которую мы использовали выше, и унарной.

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

Например:

// Не влияет на числа

let x = 1;

alert( +x ); // 1

let y = -2;

alert( +y ); // -2

// Преобразует нечисла в числа

alert( +true ); // 1

alert( +"" ); // 0

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

Необходимость преобразовывать строки в числа возникает очень часто. Например, обычно значения полей HTML-формы – это строки. А что, если их нужно, к примеру, сложить?

Бинарный плюс сложит их как строки:

let apples = "2";

let oranges = "3";

alert( apples + oranges ); // "23", так как бинарный плюс складывает строки

Поэтому используем унарный плюс, чтобы преобразовать к числу:

let apples = "2";

let oranges = "3";

// оба операнда предварительно преобразованы в числа

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

// более длинный вариант

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

## [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

* Если результат оператора не используется, а нужно только увеличить/уменьшить переменную – 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* – постфиксная форма:
* let counter = 0;

alert( counter++ ); // 0

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

let n = 2;

n \*= 3 + 5;

alert( n ); // 16 (сначала выполнится правая часть, превратив выражение в n \*= 8)

## [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**

* 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.
* 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 > и < с переменными, которые могут принимать значения null/undefined. Хорошей идеей будет сделать отдельную проверку на null/undefined.
* **Оператор строгого равенства === проверяет равенство без приведения типов.**

5 > 4 → true

"ананас" > "яблоко" → false

"2" > "12" → true

undefined == null → true

undefined === null → false

null == "\n0\n" → false

null === +"\n0\n" → false

**Функции Взаимодействия в браузере**

**alert**

показывает сообщение.

**prompt**

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

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.

**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.

**Задача.** Создайте страницу, которая спрашивает имя у пользователя и выводит его.

let name = prompt("Ваше имя?", "");

alert(name);

# Условные операторы: if, '?'

# Пример сложного if

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!' );

}

Пример ?

let result = условие ? значение1 : значение2;

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

let age = prompt('Возраст?', 18);

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

(age < 18) ? 'Привет!' :

(age < 100) ? 'Здравствуйте!' :

'Какой необычный возраст!';

alert( message );

Смысл оператора вопросительный знак ? – вернуть то или иное значение, в зависимости от условия. Пожалуйста, используйте его именно для этого. Когда вам нужно выполнить разные ветви кода – используйте if.

## [Задачи](https://learn.javascript.ru/ifelse#tasks)

### [if (строка с нулём)](https://learn.javascript.ru/ifelse" \l "if-stroka-s-nulyom)

важность: 5

Выведется ли alert?

if ("0") {

alert( 'Привет' );

}

решение

**Да, выведется.**

Любая строка, кроме пустой (а строка "0" – не пустая), в логическом контексте становится true.

2.Используя конструкцию if..else, напишите код, который будет спрашивать: „Какое «официальное» название JavaScript?“

Если пользователь вводит «ECMAScript», то показать: «Верно!», в противном случае – отобразить: «Не знаете? ECMAScript!»

let name = prompt('Какое «официальное» название JavaScript?', '');

    if (name == 'ECMAScript') {

      alert('Верно!');

    }

    else {

      alert('Не знаете? ECMAScript!');

    }

### 3. [Перепишите 'if..else' в '?'](https://learn.javascript.ru/ifelse#perepishite-if-else-v)

важность: 5

Перепишите if..else с использованием нескольких операторов '?'.

Для читаемости рекомендуется разбить код на несколько строк.

let message;

if (login == 'Сотрудник') {

message = 'Привет';

} else if (login == 'Директор') {

message = 'Здравствуйте';

} else if (login == '') {

message = 'Нет логина';

} else {

message = '';

}

Решение: let login = prompt('Должность?', '');

   let message = ( login == 'Сотрудник') ? 'Привет' : ( login == 'Директор') ? 'Здравствуйте' : ( login == "") ? 'Нет логина' : '';

   alert (message);

# Логические операторы

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

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.

Например:

alert( 1 || 0 ); // 1

alert( true || 'no matter what' ); // true

alert( null || 1 ); // 1 (первое истинное значение)

alert( null || 0 || 1 ); // 1 (первое истинное значение)

alert( undefined || null || 0 ); // 0 (поскольку все ложно, возвращается последнее значение)

Это делает возможным более интересное применение оператора по сравнению с «чистым, традиционным, только булевым ИЛИ».

1. **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

**Сокращённое вычисление.**

Операндами могут быть как отдельные значения, так и произвольные выражения. ИЛИ вычисляет их слева направо. Вычисление останавливается при достижении первого истинного значения. Этот процесс называется «сокращённым вычислением», поскольку второй операнд вычисляется только в том случае, если первого недостаточно для вычисления всего выражения.

Это хорошо заметно, когда выражение, указанное в качестве второго аргумента, имеет побочный эффект, например, изменение переменной.

В приведённом ниже примере x не изменяется:

let x;

true || (x = 1);

alert(x); // undefined, потому что (x = 1) не вычисляется

Оператор && выполняет следующие действия:

* Вычисляет операнды слева направо.
* Каждый операнд преобразует в логическое значение. Если результат false, останавливается и возвращает исходное значение этого операнда.
* Если все операнды были истинными, возвращается последний.

Другими словами, И возвращает первое ложное значение. Или последнее, если ничего не найдено.

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

// Если первый операнд истинный,

// И возвращает второй:

alert( 1 && 0 ); // 0

alert( 1 && 5 ); // 5

// Если первый операнд ложный,

// И возвращает его. Второй операнд игнорируется

alert( null && 5 ); // null

alert( 0 && "no matter what" ); // 0

**Приоритет оператора && больше, чем у ||**

Приоритет оператора И && больше, чем ИЛИ ||, так что он выполняется раньше.

Таким образом, код a && b || c && d по существу такой же, как если бы выражения && были в круглых скобках: (a && b) || (c && d).

Приоритет НЕ ! является наивысшим из всех логических операторов, поэтому он всегда выполняется первым, перед && или ||.

**Задача.**

Что выведет код ниже?

alert( null || 2 && 3 || 4 );

решение

Ответ: 3.

alert( null || 2 && 3 || 4 );

Приоритет оператора && выше, чем ||, поэтому он выполнится первым.

Результат 2 && 3 = 3, поэтому выражение приобретает вид:

null || 3 || 4

Теперь результатом является первое истинное значение: 3.