**[Подключение JS к странице и переменные](https://beonmax.com/courses/javascript/podklyuchenie-js-peremennye/)**

JS – высокоуровневый язык программирования. Отличие JS от других языков программирования – все, что мы пишем в коде, воспринимается браузером как есть, и не переводится в отличие от других языков программирования в двоичный код.

Варианты подключения:

* в html-файл: используется для очень маленьких скриптов. Мы пишем в head или перед закрывающим тегом body следующее:

<script>команда или код</script>

в качестве отдельного файла через src:

<script src=”путь к файлу”>команда или код</script>

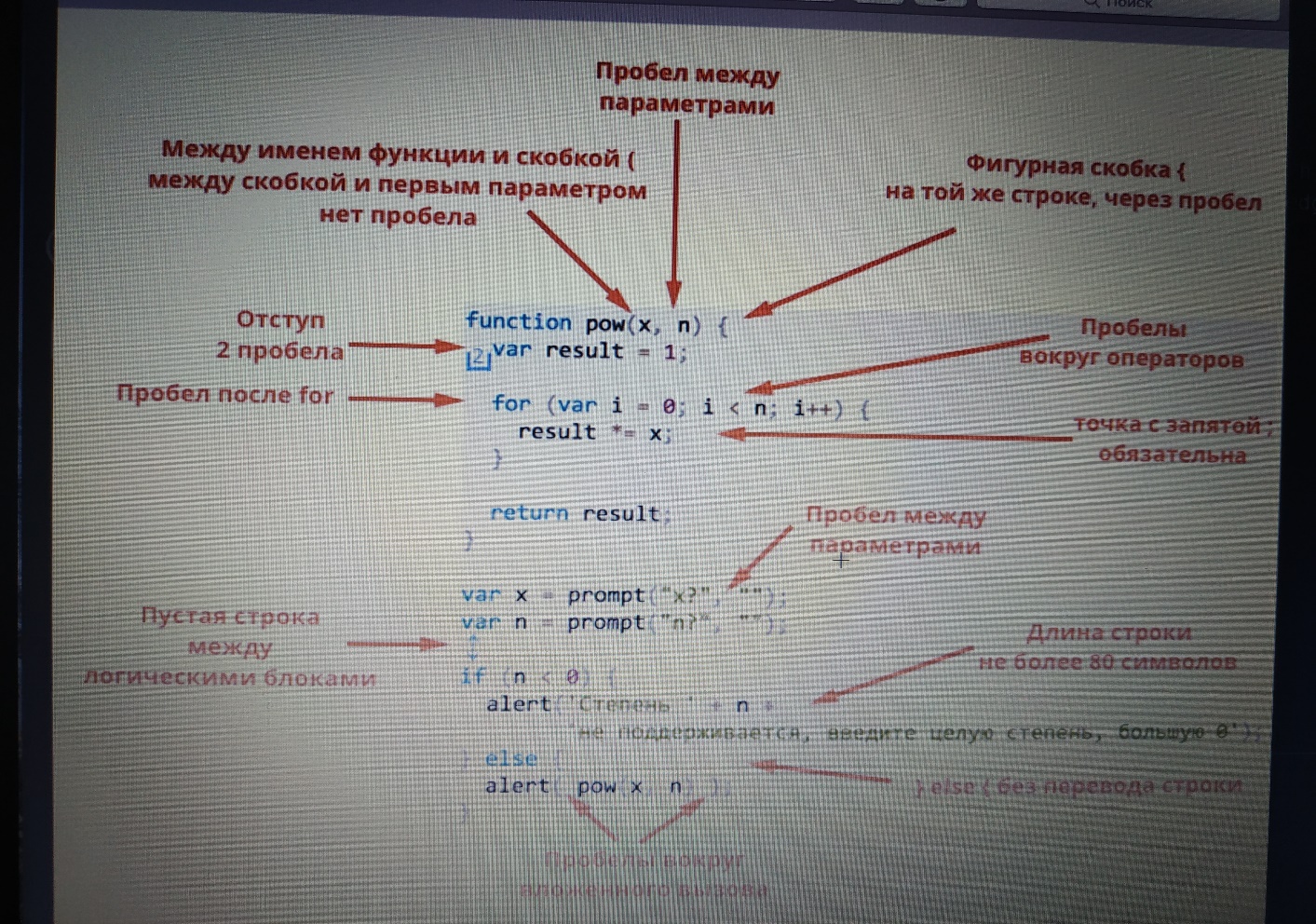
* можно встретить JS-событие с атрибутами в тегах

Есть файлы:

* локальные – файлы, которые лежат у нас на компьютере
* удаленные – подключаются через cdn и лежат где-то на серверах в интернете.

! <script src=”путь к файлу” type=”text/javascript”>команда или код</script> УЖЕ НЕАКТУАЛЬНО!!! Type – атрибут устаревший. Его нужно убирать из своих скриптов.

***Стилистические правила JS***



Также не забываем ставить точку с запятой после каждой логически завершенной операции.

**Консоль разработчика в гугл хроме**

elements – видим подробное описание верстки и некоторые обработчики события

console – сюда выводится информация о коде, в том числе все имеющиеся ошибки, предупреждения или даже сообщения от разрабочиков. Консоль подсказывает, где у нас была ошибка: в какой строчке и в каким файле и полное описание этой ошибки

sources – находим все приложения и материалы доступные к нашему сайту

network – помогает отслеживать запросы и тому подобные вещи

performance – тесты производительности нашей страницы

memory – объем занимаемой памяти

и пр.

Чтобы не запутаться, лучше писать к скрипту комментарии:

однострочные // комментарий

многострочные /\* комментарий \*/

**ПЕРЕМЕННЫЕ**

Переменные – это хранилище информации, своего рода подписанные коробки с чем-либо.

Способы объявления переменных:

- 1 -

*Переменная var.*

***var leftBorderWidth = 1;***

***var*** – мы говором JS, что сейчас будет переменная;

***leftBorderWidth*** – имя нашей переменной, мы пишем осмысленное имя переменной, которую хотим сохранить; Формат Camel Case ***leftBorderWidth –*** следующее слово идет с большой буквы и без пробелов. В таком ключе следует писать названия переменных в JS.

***=*** значит, что мы присваиваем переменной какое-то значение;

после знака равенства ставим значение, которое хотим присвоить.

Если ***var*** объявлена в коде, то она существует еще до написания скрипта. И она видна везде.

Ниже выводим переменную до того, как она объявлена, и смотрим в консоль браузера. Что мы увидим: undefined. Что это значит? Переменная создана, но она пока не имеет никакого значения.

console.log(leftBorderWidth);

var leftBorderWidth = 1;

Но если мы этот код запишем по-другому:

var leftBorderWidth = 1;

console.log(leftBorderWidth);

В данном случае мы увидим в консоле 1. Т.е. значение, которое мы присвоили нашей переменной.

- 2 -

*Переменная let.*

Если мы попробуем вывести в консоль переменную let до ее объявления также, как мы это делали с var, то получим ошибку.

console.log(second);

let second = 8;

Почему? Потому что код ее не может найти.

Если же укажем нижеприведенный вариант, то консоль выведет корректный вариант, а именно значение 8, которое мы присвоили нашей переменной.

let second = 8;

console.log(second);

let создается только тогда, когда код до нее доходит. Такое понятие называется хоистинг, или всплытие переменной. Это поведение позволяет нам избежать многих ошибок. С точки зрения оптимизации – это ускоряет нашу работу, т.к. код не должен хранить в себе все переменные. Целесообразнее создавать переменные в момент использования. Таким образом код работает быстрее, и экономит нашу память браузера. Кроме того, let видна только в части кода, ограниченного фигурными скобками {}.

Например,

{

let second = 8;

}

console.log(second);

В консоле мы увидим ошибку, потому что консоль не может найти нашу переменную, которая ограничена фигурными скобками.

- 3 -

*Переменная const*

const создает константу, которую нельзя поменять. Мы получим ошибку, если попытаемся это сделать, а в остальном – аналогично let.

При этом, следует заметить, что совсем констант в JS нет и const обходными путями менять можно, но об этом будет упомянуто в теме замыканий.

[**Классификация типов данных в JavaScript**](https://beonmax.com/courses/javascript/tipy-dannyh/)

Существует всего семь типов данных:

* число (целое или дробное); существуют также числовые значения, которые относятся к данному типу:

Infinity – можно получить при делении на нуль;

Not-a-number (NaN) – если мы проведем какую-то операцию, которая не подлежит математической логике: умножим или разделим число на строку.

* строка – ее обязательным атрибутом являются кавычки;
* символ (simbol);
* логическое значение true / false (либо да – соответствует, либо нет – не соответствует);
* null – когда чего-то в коде просто не существует. Например: *console.log(something);* Консоль выведет ошибку, потому что код не может найти ничего. Это и есть null. По сути – это ссылка на несуществующую вещь.
* undefined – когда объект уже существует, но значения еще не имеет. Например: *let* *something –* обозначим переменную, ничего ей не присваивая, и при попытке вывести ее в консоль, получим undefined;
* объект (по новым стандартам объект тоже является одним из типов данных; ранее их было всего шесть). Объект – это коллекция данных, это структура, которая используется для хранения любых данных. Этот тип данных называется комплексным, т.к. в нем могут храниться все типы данных, о которых мы говорили ранее. Синтаксис объекта создается при помощи фигурных скобок {} и может содержать в себе данные, которые будут называться свойством объекта или методом. Методы – это функции, о которых будем говорить позже. Создадим на примере:

let persone = {

name: “John”,

age: 25,

isMarried: false

}

где , name – это ключ, далее двоеточие, и в кавычках значение, которое мы в этот ключ ходим поместить (может быть и строка, и число, и логическое значение). После этого ставим запятую, чтобы объявить следующее свойство.

Если мы хотим добраться до значения определенного свойства, мы можем сделать это двумя способами:

1. через точку

Пример:

let persone = {

name: “John”,

age: 25,

isMarried: false

}

console.log(persone.name);

Таким образом, мы получили значение первого свойства.

1. через квадратные скобки

Этот способ используется реже, но тогда, когда нам нужно добраться до свойства, заданного любой строкой и которое при помощи точки мы получить не можем.

console.log(persone[“name”]);

значение, которое мы хотим получить прописывается обязательно в кавычках.

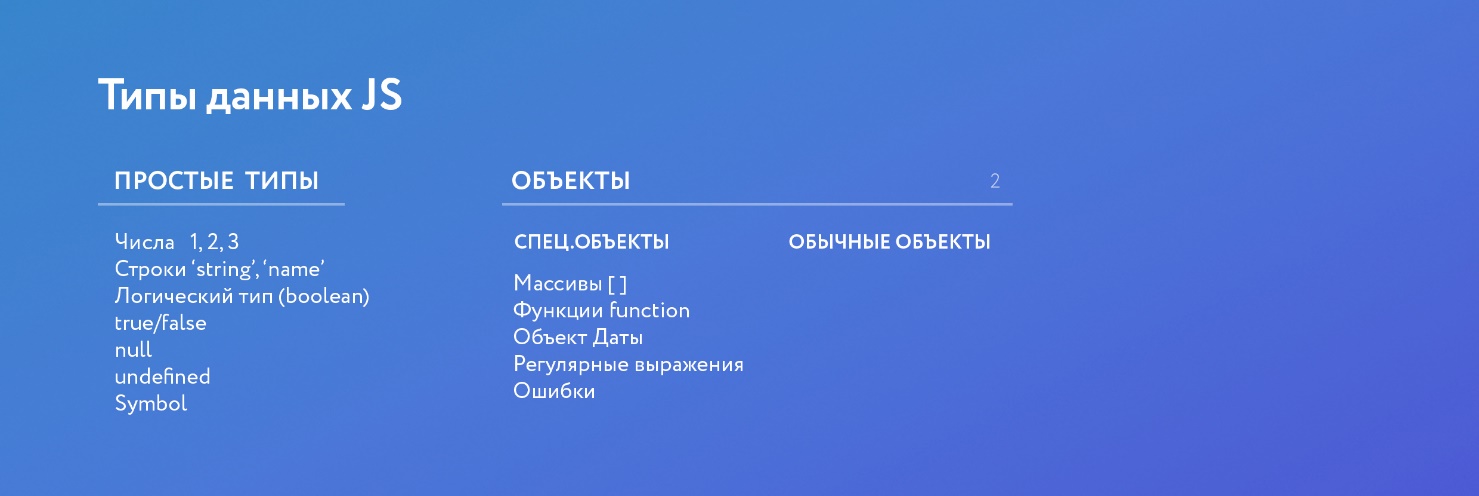
Также существуют массивы, которые предназначены для хранения данных которые идут по порядку. Массив заключается в квадратные скобки и построен так, что каждому значению в нем автоматически присваивается порядковый номер.

Создадим массив фотографий:

let arr = [‘plum.png’, ‘orange.jpg’, ‘apple.bmp’]; элементы массива перечисляются через запятую.

Массив служит для хранения любых типов данных.

Все типы данных делятся на такие группы:



[**Простое общение с пользователем**](https://beonmax.com/courses/javascript/obschenie-s-polzovatelem/)

Запустить код прямо в редакторе Studio Visual Code:

=> выделить код, который нужно запустить;

=> нажать на него правой кнопкой мыши;

=> выбрать «run code»

Чтобы очистить справа есть кнопочка, похожая на меню бургер с крестиком «Очистить выходные данные»

Остальные команды общения с пользователем работают только в браузере. К числу теких команд относится alert.

alert("Hello world!");

На странице появится окно, в котором мы должны будем подтвердить действие, пока пользователь не нажмет «ОК» окно не исчезнет.

Чтобы задать пользователю вопрос можем написать следующую команду:

confirm("Are you here?");

Чтобы узнать, что происходит в браузере, мы можем ввести следующую команду, после чего посмотреть в консоль браузера:

let answer = confirm("Are you here?");

console.log(answer);

Что мы увидим в консоле? – true при положительном ответе и false – при отрицательном

Команда prompt позволяет задать пользователю интересующий нас вопрос, после чего в окне браузера появляется поле для ввода ответа, а в браузере отображается ответ. Для отображения поля ввода ответа, после вопроса ставим запятую и оставляем пустые кавычки.

ВСЯ ИНФОРМАЦИЯ, КОТОРАЯ ПРИХОДИТ ОТ ПОЛЬЗОВАТЕЛЯ – ОТОБРАЖАЕТСЯ В ВИДЕ СТРОК.

let answer = prompt("Есть ли Вам 18?", “”);

console.log(answer);

Команда typeof: в консоле увидим ответ string, что также является строкой.

Команда typeof позволяет проверить информацию на тот тип данных, которым она является:

console.log(typeof(answer));

Если в typeof выводим arr, увидим в консоле object, т.к. массив – это частный случай.

Итак, мы можем получить информацию при помощи следующих команд:

console.log(); alert(); confirm(); prompt();

[**Операторы в JS**](https://beonmax.com/courses/javascript/operatory/)

Операторы сложения, вычитания, умножения и делений.

Особняком стоит знак плюс. Если к числу прибавить строку – получим строку. Если сложить две строки все равно получим строку. Это называется конкатинация.

console.log(“arr” + “- object”);

увидим в консоле: arr – object

если мы сложим console.log(“4” + “- object”);

увидим в консоле: 4 – object

То есть, все равно получим строку.

Если мы поставим + перед prompt, то увидим в консоле number, потому что знак +, который стоит перед prompt превращает строку в число:

let answer = +prompt("Есть ли Вам 18?", “”);

console.log(typeof(answer));

Такой же результат мы получим если: console.log(4 + + “- object”); ответ в консоле NaN, т.к. нами были проведены какие-то нематематические операции

У нас есть также такие операторы как инкримент и декремент. Они существуют для сокращения кода и для того, чтобы увеличить или уменьшить значение на единицу.

Чем отличается префиксная форма от постфиксной.

let incr = 10,

decr = 10;

incr++; // в консоле будет 11, т.к. значение увеличилось на единицу

decr--; // в консоле будет 9, т.к. значение уменьшилось на единицу

console.log(incr);

console.log(decr);

Префиксная форма возвращает уже измененное значение, а постфиксная – старое.

Просмотреть это можно на следующем примере. Значения incr++ и incr- - удалим из формулы и впишем в консоль. Это будет префиксная форма.

let incr = 10,

decr = 10;

console.log(++incr);

console.log(--decr); Префиксная форма возвращает уже измененное значение

Но если мы запишем и сохраним по-другому, мы получим 10 – то есть изначальное значение. Это постфиксная форма.

console.log(incr++);

console.log(decr--);

Оператор

= присваивание;

== сравнение;

=== строгое сравнение по типам данных

Пример:

«2» == 2; // true

«2» === 2; // false

и (&&)

let isChecked = true,

isClose = true;

console.log(isChecked && isClose); = true; // в консоле увидим true, но если одно из значений мы изменим на false – получим в консоле false

true возвращается, если все наши аргументы являются правдивыми

или (||)

возвращает нам верное значение, если хотя бы один из аргументов является правдивым

let isChecked = true, ПЕРЕЧИСЛЕНИЕ ЧЕРЕЗ ЗАПЯТУЮ

isClose = false;

console.log(isChecked || isClose); = true;

здесь мы можем получить false только если все значения будут false

Превратить правду в ложь, а ложь в правду мы можем при помощи оператора !, который можем ставить перед тем значением, которое нужно изменить

let isChecked = true,

isClose = true;

console.log(isChecked || !isClose); здесь isClose из true превращается в false

строгий режим директивы **‘use strict’** его мы ставим в самом начале документа в виде строки, и этим мы говорим, что пишем на стандарте ИС-6 и некоторые оибки прошлого работать не будут

например:

**‘use strict’;**

**а = 5;**

console.log(а); в консоле мы увидим ошибку: мы не указали, что а – это переменная, но если бы мы не написали фразу **‘use strict’;** - все сработало бы и ошибки не было б. Это ошибка старого стандарта, и в новом стандарте ее не будет

ПОЭТОМУ **‘use strict’;** стоит ставить в своих скриптах

**Приоритет операторов**

Данная таблица упорядочена с самого высокого приоритета (20) до самого низкого (1).

|  |  |  |  |
| --- | --- | --- | --- |
| **Приоритет** | **Тип оператора** | **Ассоциативность** | **Конкретные операторы** |
| 20 | Группировка | не определено | ( … ) |
| 19 | [Доступ к свойствам](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Property_Accessors#Dot_notation) | слева направо | … . … |
| [Доступ к свойствам с возможностью вычисления](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Property_Accessors#Bracket_notation) | слева направо | … [ … ] |
| [new](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/new) (со списком аргументов) | неопределено | new … ( … ) |
| [Вызов функции](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Functions) | слева направо | … ( … ) |
| 18 | [new](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/new) (без списка аргументов) | справа налево | new … |
| 17 | [Постфиксный инкремент](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Increment) | не определено | … ++ |
| [Постфиксный декремент](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Decrement) | … -- |
| 16 | Логическое отрицание | справа налево | ! … |
| [Побитовое отрицание](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_NOT) | ~ … |
| [Унарный плюс](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Unary_plus) | + … |
| [Унарный минус](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Unary_negation) | - … |
| [Префиксный инкремент](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Increment) | ++ … |
| [Префиксный декремент](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Decrement) | -- … |
| [typeof](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/typeof) | typeof … |
| [void](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/void) | void … |
| [delete](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/delete) | delete … |
| [await](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/await) | await … |
| 15 | [Возведение в степень](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Exponentiation) | справа налево | … \*\* … |
| 14 | [Умножение](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Multiplication) | слева направо | … \* … |
| [Деление](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Division) | … / … |
| [Остаток](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Remainder) | … % … |
| 13 | [Сложение](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Addition) | слева направо | … + … |
| [Вычитание](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Subtraction) | … - … |
| 12 | [Побитовый сдвиг влево](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators) | слева направо | … << … |
| [Побитовый сдвиг вправо](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators) | … >> … |
| [Сдвиг вправо с заполнением нулей](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators) | … >>> … |
| 11 | Меньше | слева направо | … < … |
| Меньше или равно | … <= … |
| Больше | … > … |
| Больше или равно | … >= … |
| [in](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/in) | … in … |
| [instanceof](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/instanceof) | … instanceof … |
| 10 | Равно | слева направо | … == … |
| Не равно | … != … |
| Строго равно | … === … |
| Строго не равно | … !== … |
| 9 | [Побитовое «И»](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_AND) | слева направо | … & … |
| 8 | [Побитовое исключающее «ИЛИ»](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_XOR) | слева направо | … ^ … |
| 7 | [Побитовое «ИЛИ»](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_OR) | слева направо | … | … |
| 6 | Логическое «И» | слева направо | … && … |
| 5 | Логическое «ИЛИ» | слева направо | … || … |
| 4 | Условный | справа налево | … ? … : … |
| 3 | [Присваивание](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Assignment_Operators) | справа налево | … = … |
| … += … |
| … -= … |
| … \*\*= … |
| … \*= … |
| … /= … |
| … %= … |
| … <<= … |
| … >>= … |
| … >>>= … |
| … &= … |
| … ^= … |
| … |= … |
| 2 | [yield](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/yield) | справа налево | yield … |
| [yield\*](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/yield*) | yield\* … |
| 1 | Запятая / Последовательность | слева направо | … , … |