JAVA Script

* 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>

**use strict – строгий режим JS.**

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.

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.

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.
* 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.

Мы рассмотрели 3 вида циклов:

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

Чтобы организовать бесконечный цикл, используют конструкцию while (true). При этом он, как и любой другой цикл, может быть прерван директивой break.

Если на данной итерации цикла делать больше ничего не надо, но полностью прекращать цикл не следует – используют директиву continue.

Обе этих директивы поддерживают *метки*, которые ставятся перед циклом. Метки – единственный способ для break/continue выйти за пределы текущего цикла, повлиять на выполнение внешнего.

Заметим, что метки не позволяют прыгнуть в произвольное место кода, в JavaScript нет такой возможности.

Массив – это особый тип объекта, предназначенный для работы с упорядоченным набором элементов.

* Объявление:
* // квадратные скобки (обычно)
* let arr = [item1, item2...];
* // new Array (очень редко)

let arr = new Array(item1, item2...);

Вызов new Array(number) создаёт массив с заданной длиной, но без элементов.

* Свойство length отражает длину массива или, если точнее, его последний цифровой индекс плюс один. Длина корректируется автоматически методами массива.
* Если мы уменьшаем length вручную, массив укорачивается.

Мы можем использовать массив как двустороннюю очередь, используя следующие операции:

* push(...items)добавляет items в конец массива.
* pop() удаляет элемент в конце массива и возвращает его.
* shift() удаляет элемент в начале массива и возвращает его.
* unshift(...items) добавляет items в начало массива.

Чтобы пройтись по элементам массива:

* for (let i=0; i<arr.length; i++) – работает быстрее всего, совместим со старыми браузерами.
* for (let item of arr) – современный синтаксис только для значений элементов (к индексам нет доступа).
* for (let i in arr) – никогда не используйте для массивов!

Мы вернёмся к массивам и изучим другие методы добавления, удаления, выделения элементов и сортировки массивов в главе: [Методы массивов](https://learn.javascript.ru/array-methods).