JavaScript

Intro

О языке JavaScript:

Изначально JavaScript был создан, чтобы «сделать веб-страницы живыми».

Программы на этом языке называются ***скриптами***. Они могут встраиваться в **HTML** и выполняться *автоматически при загрузке веб-страницы*.

Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска.

«LiveScript» (первое имя языка JavaScript) создавался во время “бума” языка **Java**. Поэтому было решено, что позиционирование JavaScript как «младшего брата» Java будет полезно.

Со временем JavaScript стал полностью независимым языком со своей собственной спецификацией, называющейся **ECMAScript**, и сейчас не имеет никакого отношения к Java.

Возможности JavaScript:

В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером.

*Например, в браузере JavaScript **может**:*

- Добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили.
- Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.
- Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы (технологии AJAX и COMET).
- Получать и устанавливать куки, задавать вопросы посетителю, показывать сообщения.
- Запоминать данные на стороне клиента («local storage»).

Возможности JavaScript:

Возможности JavaScript в браузере ограничены ради безопасности пользователя. Цель заключается в предотвращении доступа недобросовестной веб-страницы к личной информации или нанесения ущерба данным пользователя

*Например, в браузере JavaScript **НЕ может**:*

- JavaScript на веб-странице не может читать/записывать произвольные файлы на жёстком диске, копировать их или запускать программы. Он не имеет прямого доступа к системным функциям ОС.

- Различные окна/вкладки не знают друг о друге. Иногда одно окно, используя JavaScript, открывает другое окно. Но даже в этом случае JavaScript с одной страницы не имеет доступа к другой, если они пришли с разных сайтов (с другого домена, протокола или порта).

- JavaScript может легко взаимодействовать с сервером, с которого пришла текущая страница. Но его способность получать данные с других сайтов/доменов ограничена. Хотя это возможно в принципе, для чего требуется явное согласие (выраженное в заголовках HTTP) с удалённой стороной. Опять же, это ограничение безопасности.

Возможности JavaScript:

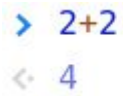
JavaScript изначально создавался только для браузера, но сейчас используется на многих других платформах.

Сегодня JavaScript занимает уникальную позицию в качестве самого распространённого языка для браузера, **обладающего полной интеграцией с HTML/CSS**.

Многие языки могут быть «транспилированы» в JavaScript для предоставления дополнительных функций. Рекомендуется хотя бы кратко рассмотреть их после освоения JavaScript.

Использование JavaScript-кода:

- В консоли разработчика (F12)



- Тег «script»

```
<script>
    alert( 'Привет, мир!' );
</script>
```

- Внешние скрипты

```
<script src="/path/to/script.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/.../lodash.js "></script>

<script src="/js/script1.js"></script>
<script src="/js/script2.js"></script>
…
```

Если атрибут **src** установлен, содержимое тега script будет игнорироваться.

Инструкции:

Инструкции – это синтаксические конструкции и команды, которые выполняют действия.

```
alert('Привет'); alert('Мир');
```

```
alert('Привет');
alert('Мир');
```

Так тоже будет работать:

```
alert('Привет')
alert('Мир')
```

В большинстве случаев новая строка подразумевает точку с запятой. Но «в большинстве случаев» не значит «всегда»!

```
alert(3 +
1
+ 2);
```

Код выведет 6, потому что JavaScript не вставляет здесь точку с запятой. Интуитивно очевидно, что, если строка заканчивается знаком "+", значит, это «незавершённое выражение», поэтому точка с запятой не требуется. И в этом случае всё работает, как задумано.

Но есть ситуации, где JavaScript «забывает» вставить точку с запятой там, где она нужна.

Переменные:

Переменная – это «именованное хранилище» для данных.

let - объявление **переменной**

var - объявление переменной (устаревший вариант), выходящей за пределы блока

const - объявление константы (неизменяемое “хранилище”)

Приведенная ниже инструкция создаёт (другими словами: объявляет или определяет) переменную с именем «message», а затем инструкция присвоения строчного значения:

```
let message;
message = 'Hello'; // сохранить строку
```

Строка сохраняется в области памяти, связанной с переменной. Мы можем получить к ней доступ, используя имя переменной:

```
alert(message); // показывает содержимое переменной
```

Или:

```
let message = 'Hello!'; // определяем переменную и присваиваем ей значение
alert(message); // Hello!
```

Переменные:

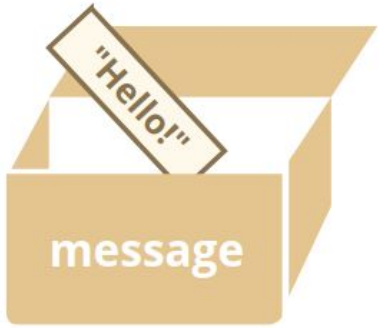
Также можно объявить несколько переменных в одной строке:

```
let user = 'John', age = 25, message = 'Hello';
```

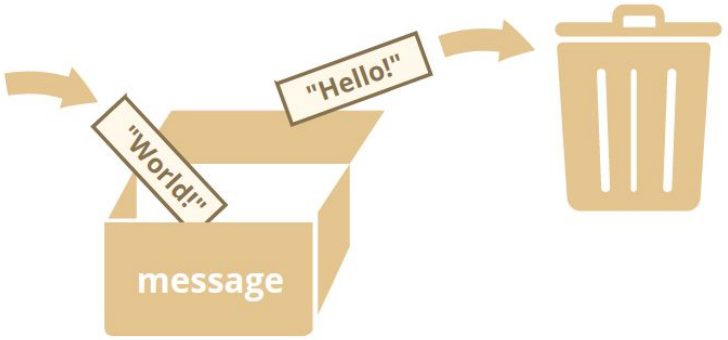
Многострочный вариант немного длиннее, но легче для чтения:

```
let user = 'John';
let age = 25;
let message = 'Hello';
```

Переменные:




```
let message;
message = 'Hello!';
message = 'World!'; // значение изменено
alert(message);
```




```
let hello = 'Hello world!';
let message;

// копируем значение 'Hello world' из переменной hello в переменную message
message = hello;

// теперь две переменные содержат одинаковые данные
alert(hello); // Hello world!
alert(message); // Hello world!
```

Названия переменных:

Ограничения на названия переменных:

1. Имя переменной должно содержать только буквы, цифры или символы **$** и **_**.
2. Первый символ не должен быть цифрой.
3. Запрещено использование зарезервированных слов в чистом виде

Зарезервированные ключевые слова в ECMAScript 2015:

break	**default**	**for**	**new**	**typeof**
case	**delete**	**function**	**return**	**var**
class	**do**	**if**	**super**	**void**
catch	**else**	**import**	**switch**	**while**
const	**export**	**in**	**this**	**with**
continue	**extends**	**instanceof**	**throw**	**yield**
debugger	**finally**	**let**	**try**	

Константы:

Переменные, объявленные с помощью const, называются «константами». Их нельзя изменить. Попытка сделать это приведёт к ошибке:

```
const myBirthday = '18.04.1982';
myBirthday = '01.01.2001'; // ошибка, константу нельзя перезаписать!
```

Широко распространена практика использования констант в качестве псевдонимов для трудно запоминаемых значений, которые известны до начала исполнения скрипта.

Названия таких констант пишутся с использованием заглавных букв и подчёркивания.

```
const COLOR_RED = "#F00";
const COLOR_ORANGE = "#FF7F00";

// ...когда нам нужно выбрать цвет
let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

Типы данных:

JavaScript - не строго типизированный язык (*динамически типизированный*), поэтому переменная может содержать любые данные. В один момент там может быть строка, а в другой – число:

```
let message = "hello";
message = 123456;
```

Числовой тип данных (**number**): целочисленные значения / с плавающей точкой.

```
let n = 123;
n = 12.345;
```

Специальные числовые значения:

Infinity / -Infinity	-	бесконечность / минус бесконечность
NaN	-	(Not a Number) результат неправильной или неопределённой математической операции

```
alert( 1 / 0 ); // Infinity
```

```
alert( "не число" / 2 + 5 ); // NaN
```

Типы данных:

Строка (**string**) в JavaScript должна быть заключена в кавычки.

```
let str = "Привет";
let str2 = 'Одинарные кавычки тоже подойдут' ;
let phrase = `Обратные кавычки позволяют встраивать переменные ${str}`;
```

Булевый тип (**boolean**) может принимать только два значения: ***true*** (*истина*) и ***false*** (*ложь*).

```
let nameFieldChecked = true; // да, поле отмечено
let ageFieldChecked = false; // нет, поле не отмечено
```

Булевые значения также могут быть результатом сравнений:

```
let isGreater = 4 > 1;
alert( isGreater ); // true (результатом сравнения будет "да")
```

Типы данных:

Специальное значение **null** формирует отдельный тип, который содержит только значение ***null*** и не относится ни к одному из типов, описанных ранее.

```
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»***.

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 x;
alert(x); // выведет "undefined"
```

Оператор 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 (результат одинаковый)*:*

- Синтаксис оператора: typeof x.
- Синтаксис функции: typeof(x).

Вызов ***typeof*** x возвращает **строку** *с именем типа*:

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof true // "boolean"
typeof "foo" // "string"
typeof Symbol("id") // "symbol"
typeof Math // "object"  (1)
typeof null // "object"  (2)
typeof alert // "function"  (3)
```

Math – это встроенный объект, который предоставляет математические операции и константы.

Результатом вызова **typeof null** является "***object***". Это неверно. Это официально признанная ошибка в typeof, сохранённая для совместимости.

Преобразование типов:

Строковое преобразование:

Строковое преобразование происходит, когда требуется представление чего-либо в виде строки.

```
let value = true;
alert(typeof value); // boolean

value = String(value); // теперь value это строка "true"
alert(typeof value); // string
```

alert(*value*) преобразует значение к строке

Преобразование типов:

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";
alert(typeof str); // string
let num = Number(str); // становится числом 123
alert(typeof num); // number
```

Если строка не может быть явно приведена к числу, то результатом преобразования будет NaN:

```
let age = Number("Любая строка вместо числа");
alert(age); // NaN, преобразование не удалось
```

Преобразование типов:

Правила численного преобразования:

```
alert( Number("   123   ") ); // 123
alert( Number("123z") );      // NaN (ошибка чтения числа в "z")
alert( Number(true) );        // 1
alert( Number(false) );       // 0
```

При преобразовании строки (*string*) в число пробельные символы по краям обрезаются. Далее, если остаётся пустая строка, то 0, иначе из непустой строки «считывается» число. При ошибке результат **NaN**.

Учтите, что **null** и **undefined** ведут себя по-разному. Так, **null** становится ***нулём***, тогда как **undefined** приводится к ***NaN***.

Преобразование типов:

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.

Преобразование типов:

Логическое преобразование:

Происходит в ***логических операторах***, но также 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***.

```
alert( Boolean(1) ); // true
alert( Boolean(0) ); // false
alert( Boolean("Привет!") ); // true
alert( Boolean("") ); // false

alert( Boolean("0") ); // true
alert( Boolean(" ") ); // пробел это тоже true (любая непустая строка это true)
```