JAVASCRIPT

BASICS

EXTERNAL TRAINING .NET/WEB

EPAM SARATOV · SUMMER 2020

ПЛАН ЗАНЯТИЯ

- Общая информация о JavaScript
- Способы исполнения скриптов
- Переменные и области видимости
- Типы данных

ИСТОРИЯ JAVASCRIPT

- “JS был обязан «выглядеть как Java», только поменьше, быть эдаким младшим братом-тупицей для Java. Кроме того, он должен был быть написан за 10 дней, а иначе мы бы имели что-то похуже JS.”

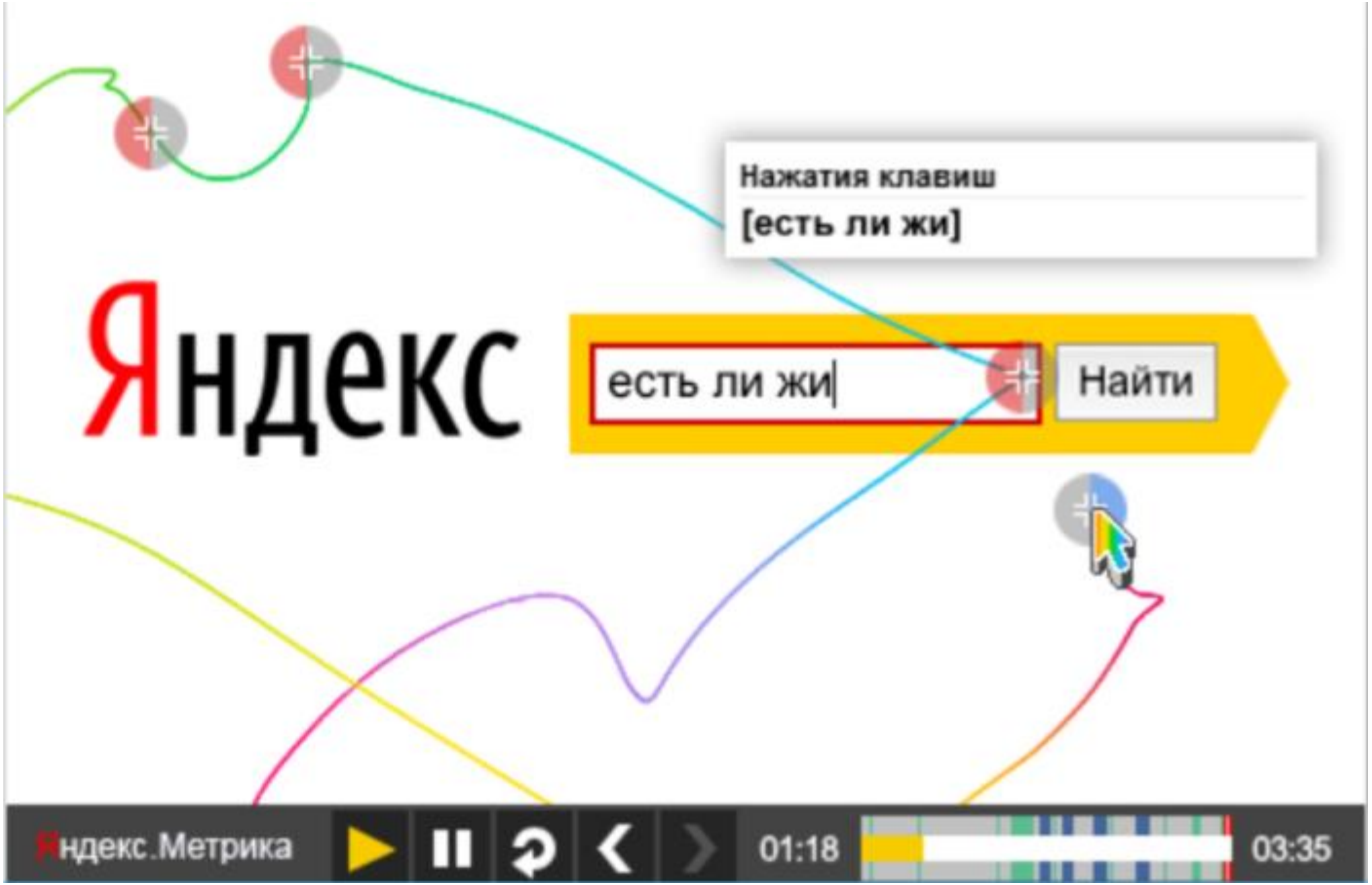
“I'll do better in the next life.”

Brendan Eich

Брендан Айк (создатель JavaScript)

JAVASCRIPT...
JAVASCRIPT EVERYWHERE...

1. 
   - Google docs, calendars...
2. 
   - Dodo, dominos
3. 
   - Наряди ёлочку


Нажатия клавиш
[есть ли жи]
Яндекс
есть ли жи
Найти
Яндекс.Метрика
01:18
03:35

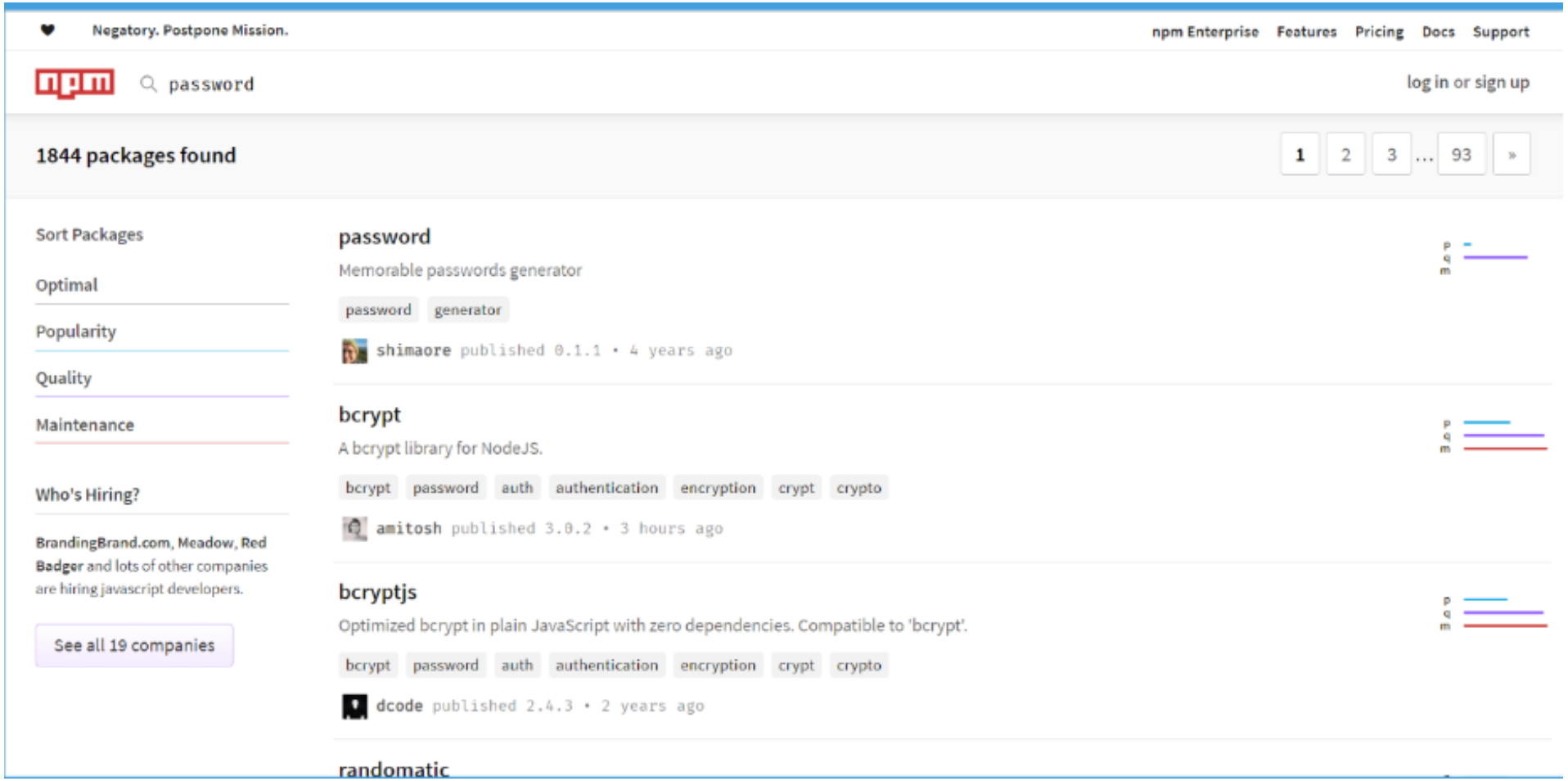
BACKEND?

MOBILE?


Create Native iOS and Android Apps With JavaScript
Open source framework for building truly native mobile apps with Angular, Vue.js, TypeScript, or JavaScript.
Get Started
Why NativeScript?
NativeScript
iOS Runtime
JavaScriptCore
Call Native APIs
NativeScript
Android Runtime
V8
Call Native APIs
ANDROID.WIDGET.BUTTON

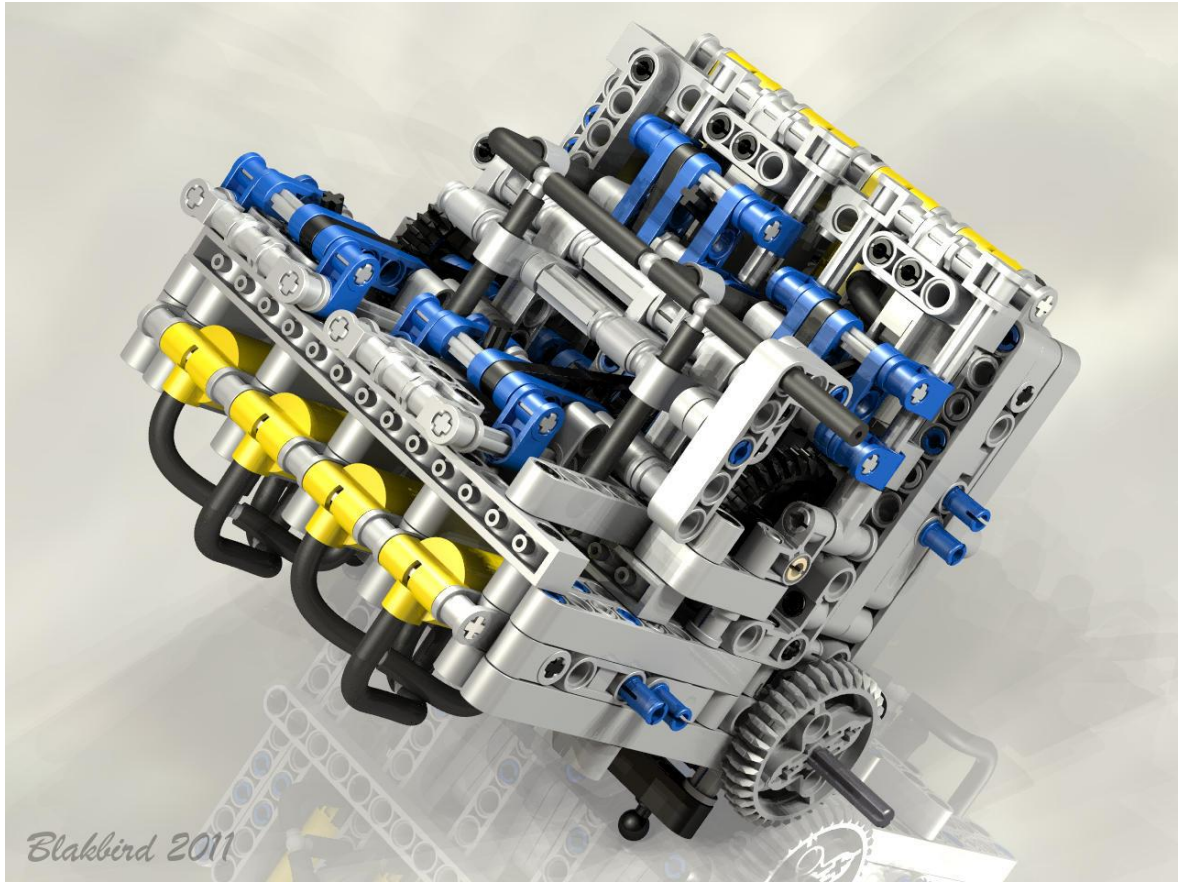
WHAT?

NPM

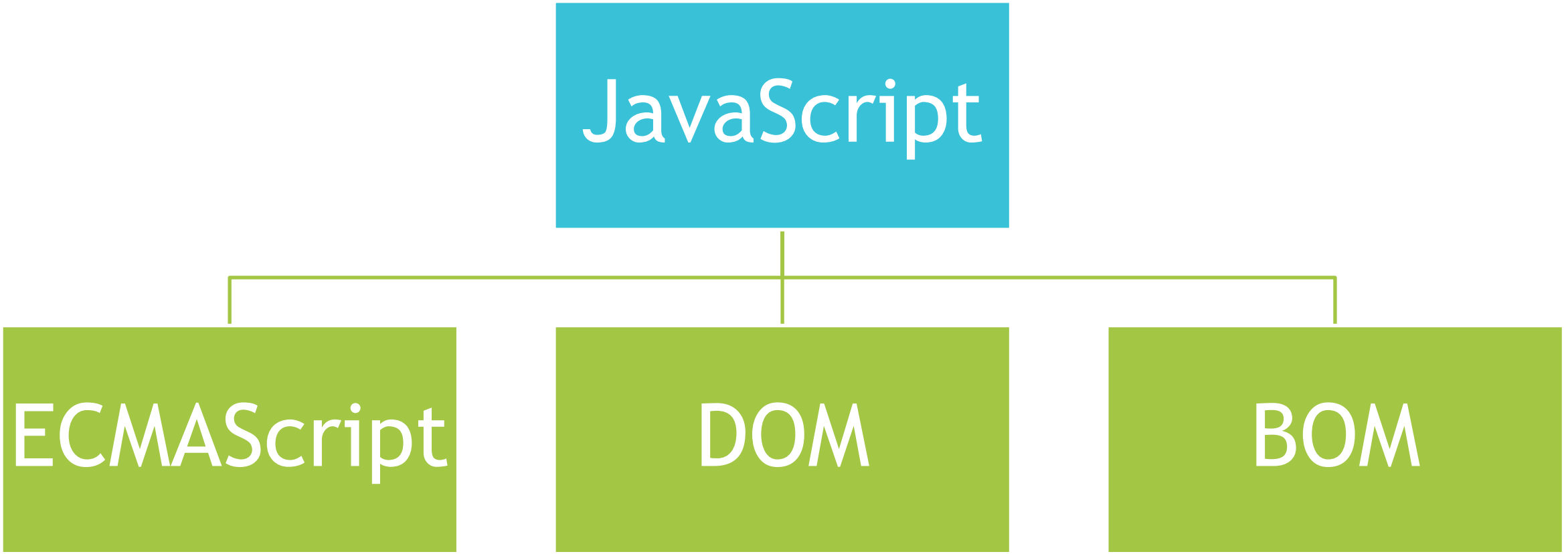

Negatory. Postpone Mission.
npm Enterprise
Features
Pricing
Docs
Support
npm
password
log in or sign up
1844 packages found
1
2
3
...
93
»
Sort Packages
Optimal
Popularity
Quality
Maintenance
Who's Hiring?
BrandingBrand.com, Meadow, Red Badger and lots of other companies are hiring javascript developers.
See all 19 companies
password
Memorable passwords generator
password
generator
shimaore published 0.1.1 • 4 years ago
bcrypt
A bcrypt library for NodeJS.
bcrypt
password
auth
authentication
encryption
crypt
crypto
amitosh published 3.0.2 • 3 hours ago
bcryptjs
Optimized bcrypt in plain JavaScript with zero dependencies. Compatible to 'bcrypt'.
bcrypt
password
auth
authentication
encryption
crypt
crypto
dcode published 2.4.3 • 2 years ago
randomatic

ДВИЖКИ JAVASCRIPT

- SpiderMonkey (Firefox)
- Rhino (by Mozilla)
- V8 (Google Chrome)
- JavaScriptCore (by Apple for Safari)
- KJS (Konqueror)
- Chakra (IE, Edge — different versions)
- …



СТРУКТУРА JAVASCRIPT



СПОСОБЫ ИСПОЛНЕНИЯ JAVASCRIPT

- Встраивание в разметку
- Тег <scripts>
- Внешний файл

- Node JS and others…

JAVASCRIPT, ВСТРОЕННЫЙ В РАЗМЕТКУ

- Браузер позволяет встраивать и исполнять JS код непосредственно в разметке;
- Достоинства:
  - Описываем сразу при описании элемента;
- Недостатки:
  - Смешение разметки и логики работы;
  - Сложность поддержки;
  - Необходимо описывать для каждой страницы.

```
<body>
    <a href="javascript:alert('hello world!')">Click!</a>
    <button onclick="alert('hello world!')">Click!</button>
</body>
```

ИСПОЛЬЗОВАНИЕ ТЕГА <SCRIPT>

- Используя тег **<script>**, мы можем записать в его тело код на JS;
- Достоинства:
  - Описание для конкретной страницы;
- Недостатки:
  - Смешение разметки и логики работы;
  - Необходимо описывать для каждой страницы.

```
<body>
    <script>
        alert('hello world!');
    </script>
</body>
```

ПОДКЛЮЧЕНИЕ ВНЕШНЕГО ФАЙЛА

- Подключение внешнего JavaScript файла производится так же при использовании тега **<script>**;
- Достоинства:
  - Независимость логики от разметки;
  - Кеширование файла;
- Недостатки:
  - Ради использования одной функции необходимо загружать весь файл.

```
<body>
    <!--
        Some code
    -->
    <script src="Scripts/main.js"></script>
</body>
```

JAVASCRIPT В HTML

- Использовать тег **<script>** для описания скриптов можно в любом месте страницы, но правильно это делать в конце **<body>**;
- Причины:
  - Ускорение загрузки сайта
  - Возможность работы с элементами после загрузки страницы.

```
<!DOCTYPE html>
<html>
<head>
    <title>My page</title>
    <meta charset="utf-8" />
    <link href="Styles/style.css" rel="stylesheet" />
    <script src="Scripts/main.js"></script>
</head>
    <body>
        <!--
            Some code
        -->
        <script src="Scripts/main.js"></script>
    </body>
</html>
```

ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ

- Для объявления переменных используется ключевое слово **var**;
- JavaScript является языком с динамической типизацией, поэтому при объявлении переменной тип данных не указывается: он может измениться в любое время.

```
var message;
var message = "строка";
message = 10;
```

ОБЛАСТЬ ВИДИМОСТИ

- Область видимости — это часть кода, в пределах которого доступна данная переменная;
- В JavaScript переменные по области видимости делятся на локальные и глобальные.

ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ

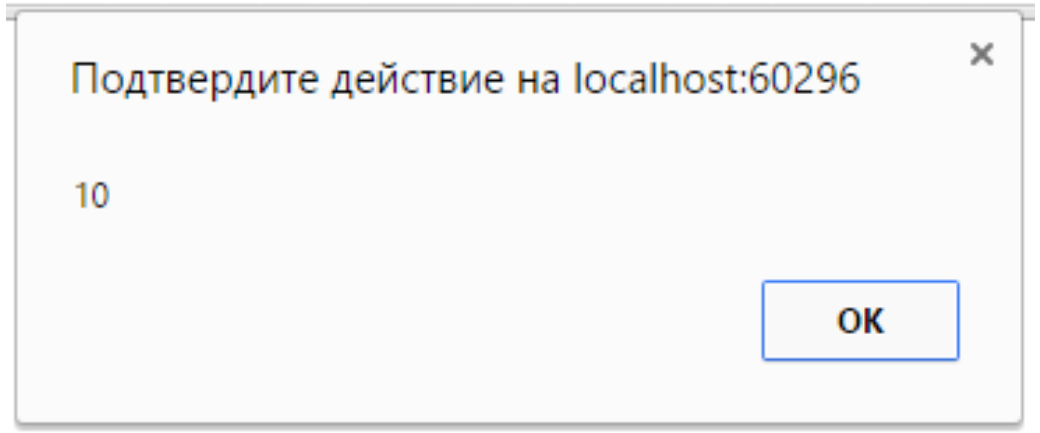
- В JavaScript все глобальные переменные и функции являются свойствами специального «глобального объекта» (global object);
- В браузере этот объект явно доступен под именем **window.**

```
var a = 10;
//объявление var создает свойство window.a
```

ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ

```
var a = 10;
alert(window.a);
alert(“a” in window)//true

window.a = 10;
alert(a);
```



ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ

- Областью видимости локальных переменных является функция;
- Все переменные внутри функции — это свойства специального внутреннего объекта **LexicalEnvironment**, который создаётся при её запуске.

```
function test() {
        var message = "hi";     //локальная
переменная
}
```

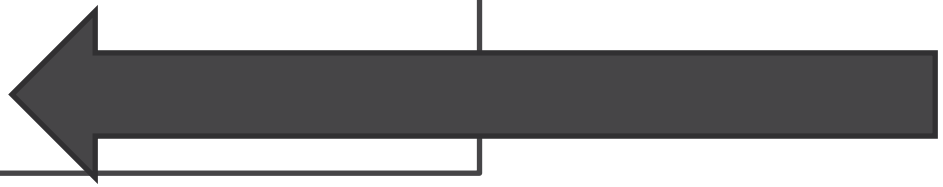
ОБЛАСТЬ ВИДИМОСТИ

```
var a = 10;
function test(){
   var message = “hi”; //локальная переменная
}
test();
alert(a);//10
alert(message); //ошибка
```

ОБЛАСТЬ ВИДИМОСТИ - ФУНКЦИЯ

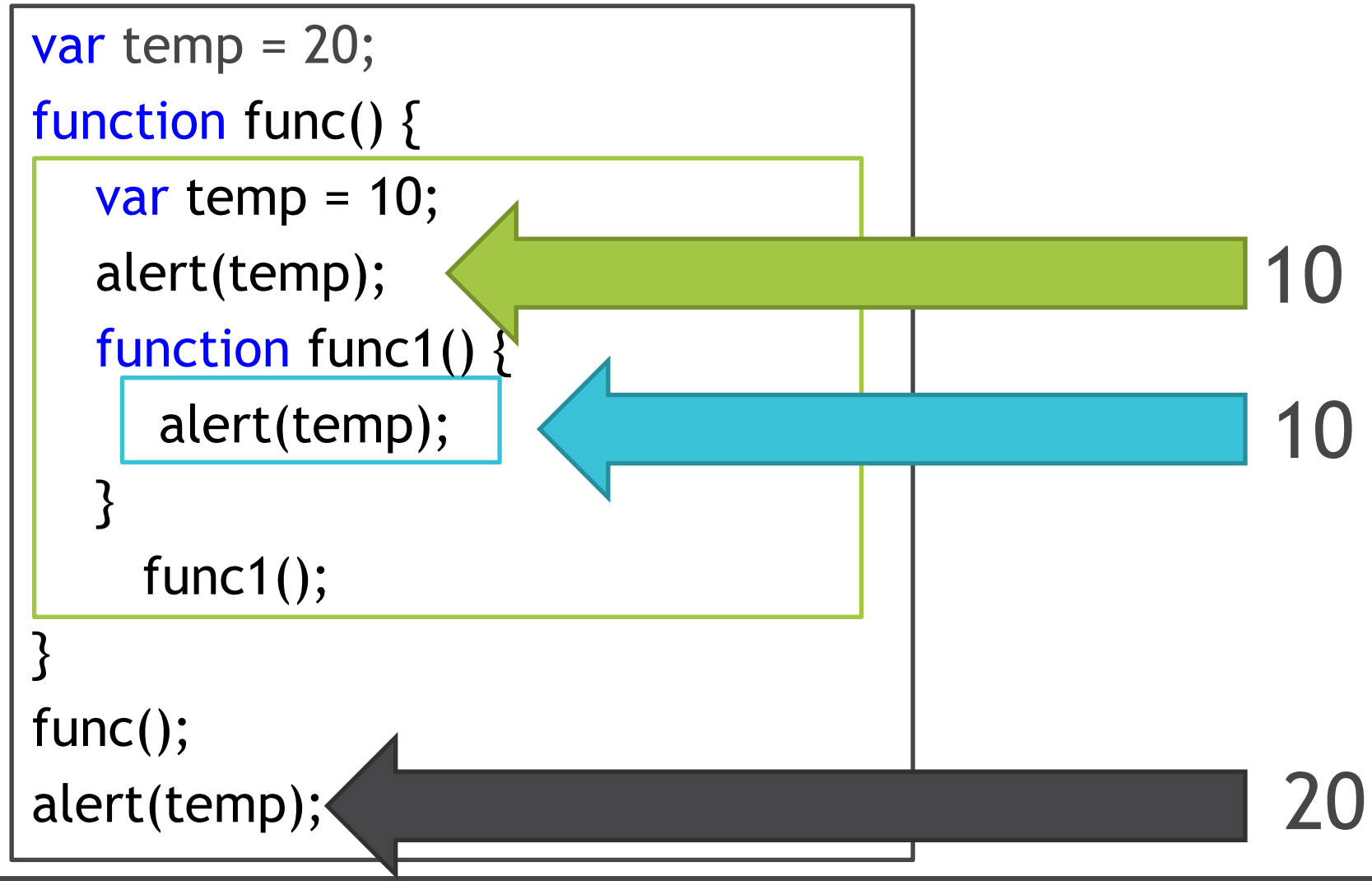
```
var temp = 20;
function func() {
    var temp = 10;
    alert(temp);
}
func();
alert(temp);
```

10



20

ОБЛАСТЬ ВИДИМОСТИ - ФУНКЦИЯ



ПОДЪЕМ ПЕРЕМЕННЫХ

- Движки для исполнения JavaScript исполняют код в 2 этапа:
  - Анализ + оптимизация;
  - Выполнение;
- На этапе анализа и оптимизации происходит выделение переменных, подъём описания переменных в самое начало функции или файла.

ПОДЪЁМ ПЕРЕМЕННЫХ

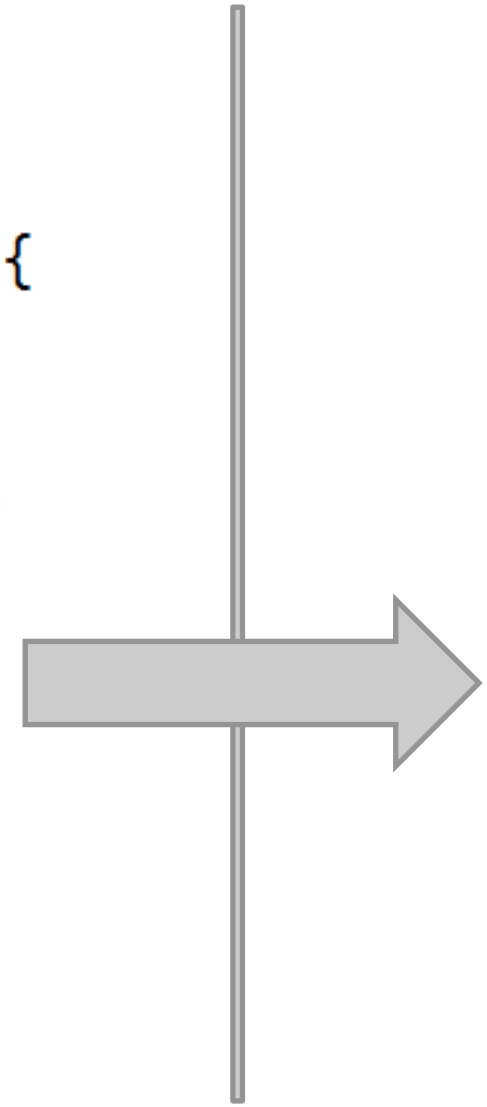
```
var a = 1;

function foo() {
    a = 3;

    //some code

    var a = 2;
}
foo();

console.log(a);
```



```
var a = 1;

function foo() {
    var a;

    a = 3;

    //some code

    a = 2;
}
foo();

console.log(a);
```

1

ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ

- При запуске функция создает объект LexicalEnvironment, записывает туда аргументы, функции и переменные.
- В отличие от window, объект LexicalEnvironment является внутренним, он скрыт от прямого доступа.

```
function sayHi(name) {
  // LexicalEnvironment = { name: 'Вася', phrase: undefined }
        var phrase = "Привет, " + name;
        alert( phrase );
}
```

ОБЛАСТЬ ВИДИМОСТИ

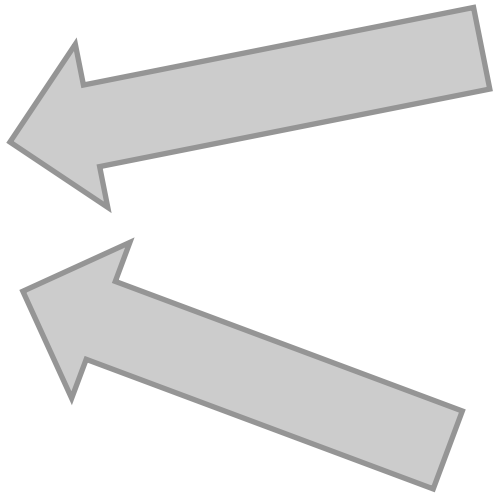
- Если мы используем необъявленную переменную, то она автоматически становится глобальной.

```
function func() {
    a = 10;
}
func();
alert(a);
```

10

ПОРЯДОК ОБРАБОТКИ

```
var b = 10;
function func() {
    var a = b = 10;
}
func();
alert(a);
alert(b);
```

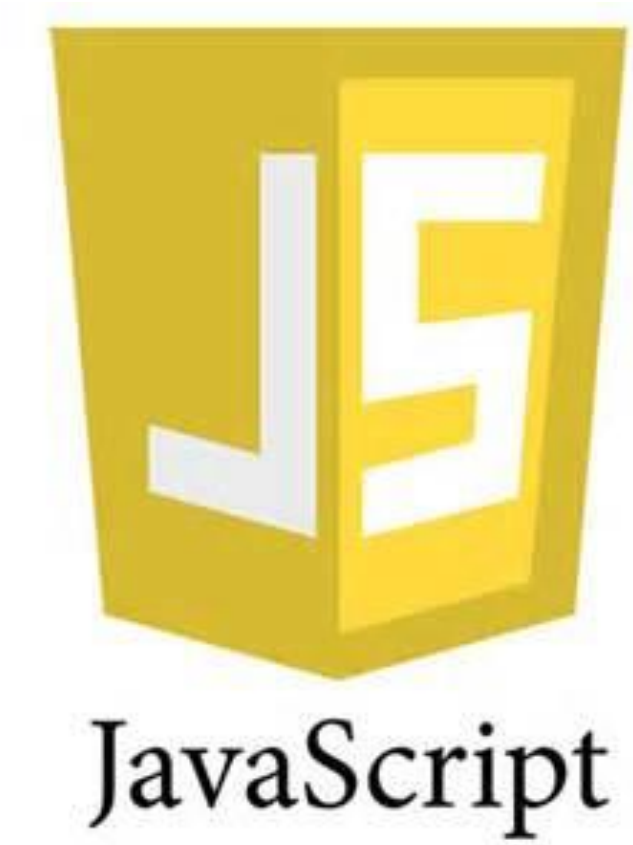


ошибка

10

ОСНОВНЫЕ ТИПЫ ДАННЫХ

- В JavaScript существует несколько основных типов данных:
  - Number
  - String
  - Null
  - Undefined
  - Boolean
  - Object



NULL

- Значение **null** не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения **null**;
- **null** — это просто специальное значение, которое имеет смысл “ничего” или “значение неизвестно”.

```
var person = null;
if (person != null){
    //что-нибудь
}
```

UNDEFINED

- Значение **undefined**, как и **null**, образует свой собственный тип, состоящий из одного этого значения. Оно имеет смысл “значение не присвоено”;
- Если переменная объявлена, но в неё ничего не записано, то её значение как раз и есть **undefined**.

```
var message; //undefined
alert(message); //”undefined”
```

STRING

- В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков;
- Внутренним форматом строк, вне зависимости от кодировки страницы, является Юникод (Unicode).

```
var text = "моя строка";

var anotherText = 'еще строка';

var str = "012345";
```

СПЕЦИАЛЬНЫЕ СИМВОЛЫ

- Строки могут содержать специальные символы. Самый часто используемый из таких символов — это «перевод строки» — \n

- alert('Привет\nМир');
  // выведет "Мир" на новой строке

Символ	Описание
\b	Удаление символа
\f	Разрыв страницы
\n	Перевод строки
\r	Возврат каретки
\t	Табуляция
\uNNNN	Символ в кодировке Юникод с шестнадцатеричным кодом NNNN. Например, \u00A9 — юникодное представление символа копирайт ©

STRING

- Если строка в одинарных кавычках, то внутренние одинарные кавычки внутри должны быть экранированы, то есть снабжены обратным слешем \‘;
- В двойных кавычках – экранируются внутренние двойные.

```
var message1 = "Hello",
    message2 = 'He said, \'hey.\'';
                    // экранирование

var text = "This is text";
alert(text.length); //12
```

STRING

- Содержимое строки в JavaScript нельзя изменять. Нельзя взять символ посередине и заменить его. Как только строка создана — она неизменна;
- Можно лишь создать новую строку и присвоить в переменную вместо старой.

```
var lang = "Java";

lang = lang + "Script"; //"JavaScript"
```

МЕТОДЫ STRING

split
charCodeAt
fromCharCode
charAt
concat
lastIndexOf
search
match
toLowerCase
toUpperCase
toLocaleLowerCase
toLocaleUpperCase
toString
valueOf
substring
slice
indexOf
substr
replace

ОПЕРАЦИИ СО СТРОКАМИ

```
var a = “string”;
```

Метод	Результат
a.charAt(3)	“i”
a[2]	“r”
a.concat(a,a)	“stringstringstring”
a.indexOf(“g”)	5
a.slice(0,2)	“st”
a.slice(1,-1)	“trin”
a.substr(2,2)	“ri”
a.substring(2,4)	“ri”

СРАВНЕНИЕ СТРОК

Сравнение строк работает лексикографически, иначе говоря, посимвольно.

Сравнение происходит в соответствии с порядком, по которому имена заносятся в орфографический словарь.

“100”<“19” - true

“cat”<“cd” - true

“cat”<“Cd” - false

```
var str1 = "cd",
    str2 = "ab";
var n =
 str1.localeCompare(str2);
```

n = 1

NUMBER

Все числа в JavaScript, как целые так и дробные, имеют тип Number и хранятся в 64-битном формате IEEE-754, также известном как «double precision».

```
var intNum = 55,
    floatNum = 1.1,
    floatNum2 = 0.1,
    num1 = 1.,
    num2 = 10.0;
```

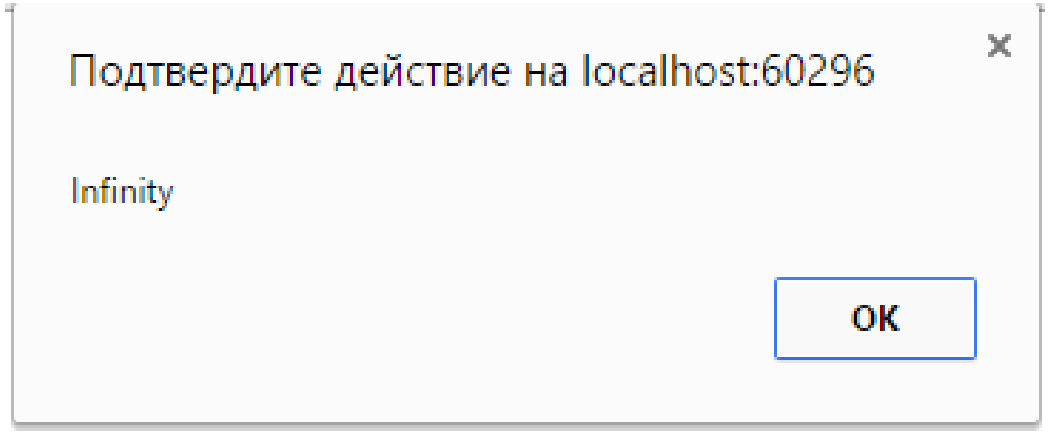
ДЕЛЕНИЕ НА НОЛЬ

Создатель JavaScript решил пойти математически правильным путем.

При делении на очень-очень маленькое число должно получиться очень большое.

Т.е. при делении на 0 мы получаем «бесконечность», которая обозначается символом ∞ или, в JavaScript: "Infinity".

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



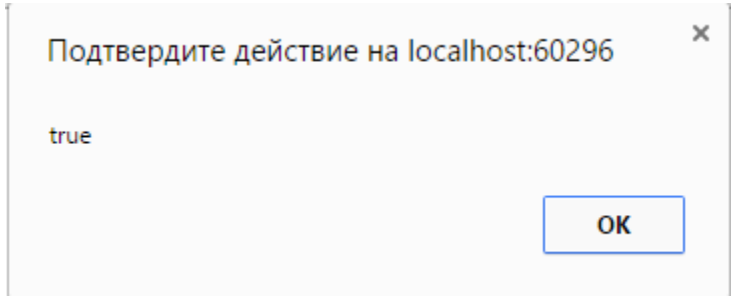
INFINITY

Infinity — особенное численное значение, которое ведет себя в точности как математическая бесконечность ∞.

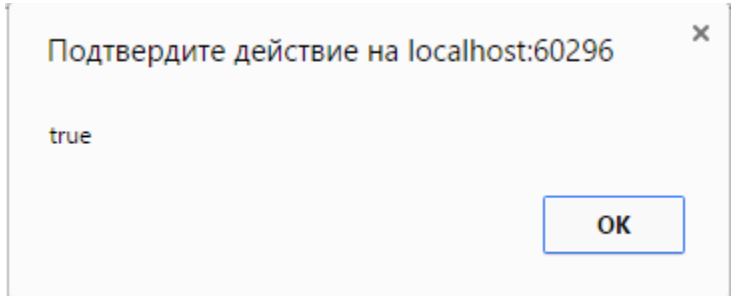
Infinity больше любого числа.

Добавление к бесконечности не меняет её.

```
alert( Infinity > 1234567890 ); // true
```




```
alert( Infinity + 5 == Infinity ); // true
```



NAN

- Если математическая операция не может быть совершена, то возвращается специальное значение NaN (Not-A-Number).
- Например, деление 0/0 в математическом смысле неопределенно, поэтому его результат NaN

```
alert( 0 / 0 ); // NaN
```

isNaN() – встроенная функция

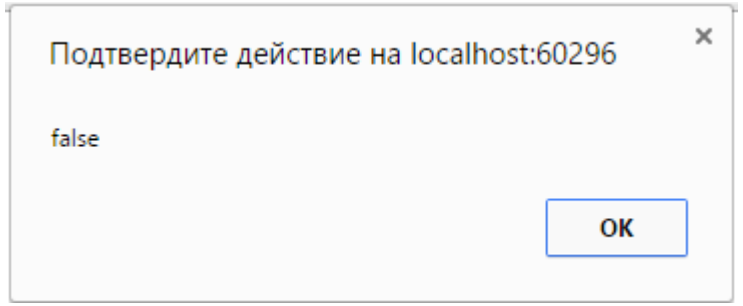
```
isNaN(NaN)); //true
isNaN(10)); //false
isNaN(“10”)); //false
isNaN(“blue”)); //true
isNaN(true)); //false
```

NUMBER

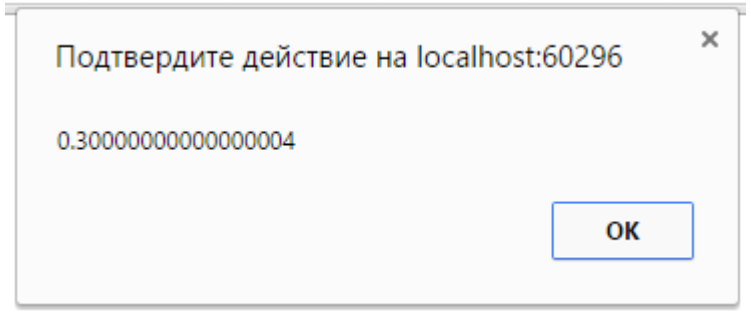
Двоичное значение бесконечных дробей хранится только до определенного знака, поэтому возникает неточность.

Тоже самое происходит в других языках, где используется формат IEEE-754, включая Java, C, PHP, Ruby, Perl.

```
alert(0.1 + 0.2 == 0.3);
```




```
alert(0.1 + 0.2);
```



МАТЕМАТИЧЕСКИЕ МЕТОДЫ

- JavaScript предоставляет базовые тригонометрические и некоторые другие функции для работы с числами.

Math.floor(x) - возвращает наибольшее целое, меньшее или равное аргументу

Math.ceil(x) - возвращает наименьшее целое, большее или равное аргументу

Math.round(x) – округляет до ближайшего целого

Math.sin(x), Math.cos(x) и т. д.

Math.sqrt(x), Math.log(x) и т.д.

NUMBER

Существует также специальный метод num.toFixed(precision), который округляет число num до точностиprecision и возвращает результат в виде строки

toFixed(n)

```
var num = 8.888;
num.toFixed(2) //“8.88”
```

ПРИВЕДЕНИЕ ТИПОВ

Если при сложении хоть один оператор строка, то результат тоже будет строкой.

Если присутствуют другие математические операции не определённые для строк, то стараемся привести результат к числу.

Любые операция с NaN возвращают NaN.

10 + 10 + "14"= "2014"

"10" + 10 + 10 - 1= 101009

"blue" – 2 = NaN

NUMBER

Функция parseInt и ее аналог parseFloat преобразуют строку символ за символом, пока это возможно.

При возникновении ошибки возвращается число, которое получилось:

- parseInt — читает целое число
- parseFloat — дробное.

parseInt()

parseFloat()

```
var num = parseInt(“1234blue”); //1234
parseInt("0xF", 16)
parseInt("F", 16)
parseInt("17", 8)
parseInt(021)=parseInt(21,8)=17
```

15

Если значение нельзя привести к числу, то получается специальное значение NaN

```
parseInt(“text”) = NaN
```

BOOLEAN

У него всего два значения: true (истина) и false (ложь).

Как правило, такой тип используется для хранения значения типа да/нет, например:

```
var a = true,
    b = false;
```

BOOLEAN

Логические преобразования которые приводят к false:

- Пустая строка
- 0
- NaN
- Null
- undefined

Всё остальное приводится к true

```
if (“text”){
    //some code
}
```

СРАВНЕНИЕ

Выражение	Значение
null == undefined	true
false == 0	true
true == 1	true
true == 2	false
null == 0	false
undefined == 0	false
“5” == 5	true

== и ===

Стандартные операторы равенства (== и !=) сравнивают два операнда без учета их типа, с приведенем.

Эквивалентность (=== и !==) производит сравнения операндов с учетом их типа.

```
“55” == 55 //true
“55” === 55 //false
true == 1 //true
true === 1 // false
```

TOSTRING()

Каждый объект обладает методом toString, который вызывается автоматически каждый раз, когда требуется строковое представление объекта.

Так же метод toString позволяет выполнять преобразование числа в строку в определенной системе счисления.

```
var a = 18;
alert(a);

a.toString() // “18”
a.toString(2) // “10010”
```

TYPEOF

Оператор typeof возвращает тип аргумента.

У него есть два варианта записи:

1. Синтаксис оператора: typeof x.
2. Синтаксис функции: typeof(x).

```
var message;
alert(typeof message); //”undefined”
alert(typeof age); //”undefined”
```

```
var text = “text”;
alert(typeof text); //“string”
var car = null;
alert(typeof car); //”object”
```


epam
THANKS FOR ATTENTION!
VIKENTII EKGART, SARATOV, RUSSIA