**JAVASCRIPT BASICS**

**CPU – central processing unit**

**RAM – random access memory**

**ROM – read-only memory**

**IP address – internet protocol adrress**

**Compiler – a compiler converts a high level language code(like C#, Java) into machine code which is executed by CPU or VM** .

* C , C++, Go, Swift use complilers to produce native code for the CPU
* C# and Java use compilers to produce intermediate code(CIL in .NET) for VM(CLR in .NET, JVM for Jva)

Wiki - is a [computer program](https://en.wikipedia.org/wiki/Computer_program) that transforms [source code](https://en.wikipedia.org/wiki/Source_code) written in a [programming language](https://en.wikipedia.org/wiki/Programming_language) (the source language) into another computer language (the target language), with the latter often having a binary form known as [object code](https://en.wikipedia.org/wiki/Object_code). The most common reason for converting a source code is to create an [executable](https://en.wikipedia.org/wiki/Executable) program.

**Interpreter** - is a [computer program](https://en.wikipedia.org/wiki/Computer_program) that directly [executes](https://en.wikipedia.org/wiki/Execution_(computers)) instructions written in a [programming](https://en.wikipedia.org/wiki/Programming_language) or [scripting language](https://en.wikipedia.org/wiki/Scripting_language), without previously [compiling](https://en.wikipedia.org/wiki/Compiler) them into a [machine language](https://en.wikipedia.org/wiki/Machine_language) program.

**JIT(just-in time compilers)** – compile the code at runtime, during the execution, on demand

**Virtual machine** – a virtual computer inside the computer, runs intermediate code

The **Common Language Runtime** (**CLR**), the [virtual machine](https://en.wikipedia.org/wiki/Virtual_machine) component of [Microsoft's](https://en.wikipedia.org/wiki/Microsoft) [.NET framework](https://en.wikipedia.org/wiki/.NET_framework), manages the execution of .NET programs. A process known as [just-in-time compilation](https://en.wikipedia.org/wiki/Just-in-time_compilation) converts compiled code into machine instructions which the computer's [CPU](https://en.wikipedia.org/wiki/CPU) then executes. The CLR provides additional services including [memory management](https://en.wikipedia.org/wiki/Memory_management), [type safety](https://en.wikipedia.org/wiki/Type_safety), [exception handling](https://en.wikipedia.org/wiki/Exception_handling),[garbage collection](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)), security and [thread management](https://en.wikipedia.org/wiki/Thread_management). All programs written for the .NET framework, regardless of [programming language](https://en.wikipedia.org/wiki/Programming_language), are executed by the CLR. All versions of the .NET framework include CLR.

**Node.js** - Event-driven I/O server-side JavaScript environment based on V8.

-мулти-платформена среда за изпълнение на сървърни и мрежови приложения с отворен код. Приложенията се пишат на JavaScript и могат да се изпълняват в Node.js среда под OS X, Microsoft Windows, Linux и IBM.

Node.js предоставя задвижвана от събития архитектура и неблокираща входно-изходна система за програмиране на приложенията (API), която оптимизира производителността и мащaбируемостта на дадена програма. Тези технологии са често използвани за мрежови приложения, работещи в реално време.

Node.js използва JavaScript двигателя V8 на Google, за да изпълнява код и голяма част от основните модули са написани на JavaScript. Node.js съдържа вградени библиотеки, които позволяват приложенията да работят като сървъри, без софтуер като Apache HTTP Server или IIS.

The **V8 JavaScript Engine**  [is an open source](https://en.wikipedia.org/wiki/Open_source) [JavaScript engine](https://en.wikipedia.org/wiki/JavaScript_engine) developed by The Chromium Project for the [Google Chrome](https://en.wikipedia.org/wiki/Google_Chrome) web browser. It has since seen use in many other projects, such as [Node.js](https://en.wikipedia.org/wiki/Node.js) and [MongoDB](https://en.wikipedia.org/wiki/MongoDB) that are used server side.

V8 [compiles](https://en.wikipedia.org/wiki/Just-in-time_compilation) [JavaScript](https://en.wikipedia.org/wiki/JavaScript) to native [machine code](https://en.wikipedia.org/wiki/Machine_code) instead of more traditional techniques such as [interpreting](https://en.wikipedia.org/wiki/Interpreter_(computing)) bytecode or compiling the whole program to machine code and executing it from a filesystem. The compiled code is additionally optimized (and re-optimized) dynamically at runtime, based on heuristics of the code's execution profile. Optimization techniques used include [inlining](https://en.wikipedia.org/wiki/Inlining),[elision](https://en.wikipedia.org/wiki/Copy_elision) of expensive runtime properties, and [inline caching](https://en.wikipedia.org/wiki/Inline_caching), among many others.

**DHTML**  - collection of technologies used together to create interactive web sites – web pages to react and change in response to the user’s actions

DHTML is a combination oh HTML + CSS + JavaScript + DOM

**JavaScript**  is a scripting programming language developed by Netscape for dynamic Web content. Embedded in HTML page, interpreted by web browsers.

- Client-side and server-side technology

- Powerfull to manipulate the DOM

**JavaScript/ECMAScript engines:**

An **ECMAScript engine** is a program/**VM** that interprets / **executes** source code written in a version of the [ECMAScript](https://en.wikipedia.org/wiki/ECMAScript) language standard, for example  [**JavaScript**](https://en.wikipedia.org/wiki/JavaScript).

These are new generation ECMAScript engines for web browsers, all implementing [just-in-time compilation](https://en.wikipedia.org/wiki/Just-in-time_compilation) (JIT) or variations of that idea.

[**Carakan**](https://en.wikipedia.org/wiki/Presto_(layout_engine)#ECMAScript_engines)**:**  included in the [10.50](https://en.wikipedia.org/wiki/Opera_10#10.50) release of the [**Opera**](https://en.wikipedia.org/wiki/Opera_(web_browser)) web browser, until switching to [V8](https://en.wikipedia.org/wiki/V8_(JavaScript_engine)) with **Opera 15** (released in 2013).

[**Chakra** (JScript9)](https://en.wikipedia.org/wiki/Chakra_(JScript_engine)): [**Internet Explorer**](https://en.wikipedia.org/wiki/Internet_Explorer).

[**Chakra**](https://en.wikipedia.org/wiki/Chakra_(JScript_engine)): [**Microsoft Edge**](https://en.wikipedia.org/wiki/Microsoft_Edge).

[**SpiderMonkey**](https://en.wikipedia.org/wiki/SpiderMonkey_(JavaScript_engine))**:** Mozilla [Gecko](https://en.wikipedia.org/wiki/Gecko_(layout_engine)) applications, including [**Firefox**](https://en.wikipedia.org/wiki/Mozilla_Firefox).

[**JavaScriptCore**](https://en.wikipedia.org/wiki/WebKit#JavaScriptCore)**:**  [WebKit](https://en.wikipedia.org/wiki/WebKit) project and applications such as [**Safari**](https://en.wikipedia.org/wiki/Safari_(web_browser))**.** Also known as Nitro, SquirrelFish and SquirrelFish Extreme.

[**Tamarin**](https://en.wikipedia.org/wiki/Tamarin_(JavaScript_engine)): An [ActionScript](https://en.wikipedia.org/wiki/ActionScript) and ECMAScript engine used in [**Adobe Flash**](https://en.wikipedia.org/wiki/Adobe_Flash).

[**V8**](https://en.wikipedia.org/wiki/V8_(JavaScript_engine)): A JavaScript engine used in [Google **Chrome**](https://en.wikipedia.org/wiki/Google_Chrome), **Opera**, [**Node.js**](https://en.wikipedia.org/wiki/Node.js), and [**V8.NET**](http://v8dotnet.codeplex.com/).

[**Nashorn**](https://en.wikipedia.org/wiki/Nashorn_(JavaScript_engine))**:** A JavaScript engine used in [**Oracle**](https://en.wikipedia.org/wiki/Oracle_Corporation) [Java Development Kit](https://en.wikipedia.org/wiki/Java_Development_Kit) (JDK) since version 8

**Web browsers(layout) engines:**

[**Blink**](https://en.wikipedia.org/wiki/Tamarin_(JavaScript_engine)): Chrome, Opera from 15.0

**Gecko**: Mozilla Firefox

[**WebKit**](https://en.wikipedia.org/wiki/Tamarin_(JavaScript_engine)): Apple Safari

[**Trident**](https://en.wikipedia.org/wiki/Tamarin_(JavaScript_engine)): IE

[**E**](https://en.wikipedia.org/wiki/Tamarin_(JavaScript_engine))**dgeHTML**: Edge

**AJAX** - Update a web page without reloading the page. AJAX is about updating parts of a web page, without reloading the whole page. AJAX is an important front-end web technology that lets JavaScript communicate with a web server. It lets you load new content without leaving the current page. Изпълнява се паралелно с другия код, без да прекъсва изпълнението на програмата.

**JavaScript Advantages**

* Dynamically load and change page content(through AJAX)
* Implementing advanced form validation
* Respond to user action, e.g.handle events
* Implementing browser-based interactive games
* Implementing SPA

**JavaScript Syntax**

<**script src="script.js" type="text/javascript"** ></**script**>

**async** and **defer**

<**script src="script.js" async**></**script**> - executes asynchronously as soon as the script is downloaded withou blocking the browser in the meantime

<**script src="script.js" defer**></**script**> - executes in after the entire document has been loaded

By default JavaScript code is executed during the page loading or when the browser fires an event.

**Standart Popup Boxes**

**alert, confirm, prompt**

**JavaScript Console Object** – used to write log messages at runtime

**debug, info, log, warn, error**

**JavaScript Types** – **number, object, undefined, boolean, string, function**

**undefined – the variable is defined, but its value – not.**

**Array, Date, null -> object**

**Infinity, -Infinity, NaN -> number**

**Number.MAX\_VALUE, Number.MIN\_VALUE**

**Math.floor(надолу), Math.round(както си е действ.), Math.ceil(нагоре)**

**Cast string -> Number**

**Number(str)**

**parseInt(str) - only to integer**

**parseFloat(str)**

**str \* 1**

**+str**

**str | 0 - only to integer**

**!!** function *isInt*(n) { return n % 1 === 0; }

**Native functions**

**‘pesho’.charCodeAt(2)** //115

**‘pesho’.charAt(2)** //’s’

Shift + Esc – отваря TaskManager-a на Chrome

**Just JS**

NaN + 1 //NaN

null + 1 //1

null + null //0

undefined + 1 //NaN

NaN == NaN //false

[] + [] = “”

{} + [] = 0

[] + {} = “[object Object]”

{} + {} = NaN

**var a = Number("pesho");**

**alert(a) // NaN**

**alert(a == NaN) //false**

**JavaScript Variables**

**unresolvable, undefined, null, local, global**

**False-like Conditions**

0 == **false** 1 == **true  
"0"** == **false "1"** == **true  
""** == **false !0** == **true**[] == **false** ![] == **false**

**!1** == **false**

**Truthy values in conditions - if()**

true **{}**

**[]**string **except ""**

**number except 0 and NaN**

**new Date()**

**object except null**

**typeof undefined**

**Falsy values in conditions - if()**

**""**

**0**

**NaN**

**null**

**undefined**

**Declare and Initialize Array in JS**

var arr = new Array(1, 2, 3, 4, 5);

var arr = new Array(10); **//undefined \* 10**

var arr = new Array();

var arr = [1, 2, 3, 4, 5];

**var arr = [];**

**arr[10] = 5; //arr.length = 11;0-9-> undefined**

**for**(**var *index* in *arr***){  
 ***console.log(arr[index]);* //for-in loop skip undefined**

}

**Loops**

while, do while, for, for in, **for of**(EcmaScript 6)

**Array manipulations**

**(1) PUSH**

var arr = [1, 2, 3];

arr.**push**(10); **//[1, 2, 3, 10]**

**(2) POP**

var last = arr.**pop**(10); **//[1, 2, 3], last = 10**

**(3) UNSHIFT**

arr.**unshift**(5); **//[5, 1, 2, 3]**

**(4) SHIFT**

var first = arr.**shift**(10); **//[1, 2, 3], first = 5**

**(5) JOIN**

var arrToStr = arr.**join**(**"|"**); **//1|2|3 – convert array to string**

**(6) FILTER**

**var *newArr*** = ***arr***.**filter**(**function**(el) { **//[2, 3]**  
 **return** el > 1; **// must return bool value**  
 });

**var *arr***=[1,2,3,4,0,**''**]; **//[1, 2, 3, 4]**  
 **var** newArr = ***arr***.filter(**function**(el) {  
 **return** el;  
 });

**var *newArr*** = ***arr***.filter(**function**(el) {**//[1, 2, 3, 4, 0, '']**  
 **return** !isNaN(el);  
 });

//

**var *arr*** = [1, 2, 3, 4, 5, 6];  
 **function** *inRange*(from, to) {  
 **return function**(number) {  
 **return** number >= from && number <= to;  
 }  
 }  
 **var *newArr*** = ***arr***.filter(*inRange*(2, 5));  
 **console**.log(***newArr***);*//[2, 3, 4, 5]*

**(7) MAP мапа не работи с undefined-и**

**var *newArr*** = ***arr***.**map**(**function**(el) { **//[2, 4, 8]**  
 **return** el \* 2;  
 });

**Мапа не работи с undefined-и, ама зависи от undefined-ите.Look below:**

**var** arr = [**undefined**, **undefined**, 5];  
**var *arr1*** = **new** Array(5);  
**console**.log(arr); *//[ undefined, undefined, 5 ]***console**.log(***arr1***); *//[ , , , , ]***var *newArr*** = arr.map(**function** (el) {  
 **return 'Item: '** + el;  
});  
**var *newArr1*** = ***arr1***.map(**function** (el) {  
 **return 'Item: '** + el;  
})  
**console**.log(***newArr***);*//[ 'Item: undefined', 'Item: undefined', 'Item: 5' ]***console**.log(***newArr1***);*//[ , , , , ]*

**(8) SORT**

**var *arr***=[101, 22, 11];  
 ***arr***.**sort**();**//[101, 11, 22]**

***arr***.**sort**(callback);

***arr***.**sort**(**function**(x, y) { **//[11, 22, 101]**  
 **return** x > y; **//or return** x > y ? (x == y ? 0 : 1) : -1;

**// За float, date use return x – y или дългото**

**// За string use return str1.localeCompare(str2);**

**!!!Иначе не бачка кат‘ хората.**

});

**(9) REVERSE**

***arr***.**reverse**();

920

**(10) EVERY**

***arr***.**every**(**function**(el, index, arr){});//returns Boolean

**var *arr*** = [1, 2, 3];  
 **var *isArrElementsArePositiveNumbers*** = ***arr***.**every**(**function**(el) {  
 **return** el > 0;  
 });  
 **console**.log(***isArrElementsArePositiveNumbers***);*//true*

**(11) SOME**

**var *arr*** = [-1, -2, 3, -4, -5];  
 **var *isExistsPositiveNumber*** = ***arr***.**some**(**function**(el) {  
 **console**.log(el);  
 **return** el > 0;  
 });  
  
 **console**.log(***isExistsPositiveNumber***);*//true*

**(12) SLICE**

**var *newArr*** = ***arr***.**slice**(startIndex, endIndex(optional)); without EndIndex!!

**(13) CONCAT**

**var *newArr*** = ***arr***.**concat**(***array or numbers***);

**var arr** = [1, 2, 3, 4, 5, 6, 7];

**(14) SPLICE**

**var *newArr*** = **arr**.splice(1, 3); *// from index 1, 3 elements* **console**.log(**arr**); *//[1, 5, 6, 7]* **console**.log(***newArr*** ); *//[2, 3, 4]* **var *newAr*** = **arr**.splice(1, 3, **'pesho'**, **'gosho'**);*//[1, 'pesho', 'gosho', 5, 6, 7]* **console**.log(**arr**);*//[1, 'pesho', 'gosho', 5, 6, 7]* **console**.log(***newAr***);*//[2, 3, 4]*

**(15) IndexOF, LastIndexOF**

***arr***.**indexOf**(element) – **returns index of first appearance of element** || -1

***arr***.**lastIndexOf**(element) – **returns index of first appearance of element**

**(16) REDUCE**

**var *arr*** = [1, 2, 3, 4, 5];  
 **var *sum*** = ***arr***.reduce(**function** (sum, number) {  
 **return** sum + number;  
 }, 0);  
 **console**.log(***sum***);*//15*

*//sum only even numbers*

**var *arr*** = [1, 2, 3, 4, 5];  
 **var *sum*** = ***arr***.reduce(**function** (sum, number) {  
 **return** sum + (!(number % 2) ? number : 0) ;  
 }, 0);  
 **console**.log(***sum***);*//6 (sum all even numbers)*

*//flat array*

**var *arr*** = [1, [2, 3], 4, 5, [6, 7, 8]];  
 **var *flattenArr*** = ***arr***.reduce(**function** (newArr, el) {  
 **if**(Array.isArray(el)) {  
 newArr = newArr.concat(el);  
 } **else** {  
 newArr.push(el);  
 }  
 **return** newArr;  
 }, []);  
 **console**.log(***flattenArr***);*//[ 1, 2, 3, 4, 5, 6, 7, 8 ]*

*//flat array с вложени масиви(рекурсивно, моя творба)*

**var *arr*** = [1, [[2, 3], 4, 5], [6, 7, 8], 9, [[[10,11], 12], 13, 14], 15, 16];  
**function** *flat*(newArr, el) {  
 **if**(Array.isArray(el)) {  
 el.***forEach***(**function**(innerEl) {  
 **if**(Array.isArray(innerEl)) {  
 *flat*(newArr, innerEl);  
 } **else** {  
 newArr.push(innerEl);  
 }  
 });  
 } **else** {  
 newArr.push(el);  
 }  
 **return** newArr;  
}  
**var *flattenArr*** = ***arr***.reduce(*flat*, []);  
**console**.log(***flattenArr***);*//[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ]*

**(17) FIND - returns leftmost element new in EcmaScript6**

**var *arr*** = [1, 2, 3, 4, 5, 6, 7, **'pesho'**];  
**var *foundEl*** = ***arr***.***find***(**function** (el) {  
 **return** el.**length** == 5;  
});  
**console**.log(***foundEl***);*//pesho*

//polyfill . Ako браузъра не поддържа find, правим feature detection, демек добавяме следния код:

**if**(!Array.**prototype**.find) {  
 Array.**prototype**.find = **function**(callback) {  
 **var** i, len;  
 **for** (i = 0, len = **this**.**length**; i < len; i+=1) {  
 **if**(callback(**this**[i], i, **this**)) {  
 **return this**[i];  
 };  
 }  
 **return undefined**;  
 }  
}

**(18) FindINDEX**

**if**(!Array.**prototype**.findIndex) {  
 Array.**prototype**.findIndex = **function**(callback) {  
 **var** i, len;  
 **for** (i = 0, len = **this**.**length**; i < len; i+=1) {  
 **if**(callback(**this**[i], i, **this**)) {  
 **return** i;  
 };  
 }  
 **return** -1;  
 }  
}  
**var *arr*** = [1, 2, 3, 4, 5, 6, 7, **'pesho'**];  
**var *foundEl*** = ***arr***.findIndex(**function** (el) {  
 **return** el == 3;  
});  
**console**.log(***foundEl***);*//2*

**(19) FILL – change the array**

**var *arr*** = [1, 2, 3, 4, 5];   
**var *newArr*** = ***arr***.fill(**'\*'**, 0, 2);  
**console**.log(***newArr***); *//[ '\*', '\*', 3, 4, 5 ]*

**var *arr*** = **new** Array(5);  
**var *newArr*** = ***arr***.fill(**'\*'**);  
**console**.log(***newArr***); *//[ '\*', '\*', '\*', '\*', '\*' ]*

**var *arr*** = [];  
**var *count*** = 5;  
***arr***[***count*** - 1] = **undefined**; //or arr.length = 15;  
**var *newArr*** = ***arr***.fill(**'\*'**);  
**console**.log(***newArr***); *//[ '\*', '\*', '\*', '\*', '\*' ]*

//polyfill

**if**(!Array.**prototype**.fill) {  
 Array.**prototype**.fill = **function**(value, from ,to){  
 **var** from = from || 0,  
 to = to || **this**.**length** - 1,  
 i;  
 **for** (i = from; i <= to; i++) {  
 **this**[i] = value;  
 }  
 **return this**;  
 };  
}

**Array.isArray**(arr) -> check if the object is Array

**var arr = new Array(5).fill('\*') //['\*', '\*', '\*', '\*', '\*']**

**var arr = new Array(6).join('\*') // \*\*\*\*\***

//

**var *arr*** = [1];  
***arr***[**'a'**] = 5;  
***arr***[-1] = 56;  
**console**.log(***arr***);// [ 1, a: 5, '-1': 56 ]

//

**var *arr*** = [1, 2, 3];  
***arr***.**length** = 2;  
**console**.log(***arr***);*//[1, 2]****arr***.**length** = 5;  
**console**.log(***arr***);*//[1, 2, undefined,undefined, undefined]*

*//*

**var *arr*** = **new** Array(16).join(**'lol'** - 2) + **' batman!'**;  
**console**.log(***arr***);*//NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN batman! ☺*

//

**var *a*** = parseInt(1/0, 19); *//the same like parseInt(Infinity, 19)*  
**console**.log(***a***); *//18*

*Explanation: parseInt treats its first argument as a string which means first of all Infinity.toString() is called, producing the string “Infinity”. So it works the same as if you asked it to convert "Infinity" in base 19 to decimal.*

*I in base 19-> is 18 in base 10, n doesn’t exists in base 19.*

*//*

**parseInt**(numberAsString, **numeral system**)  
parseInt(**'101'**, 2);*//5*parseInt(**'101'**, 10);*//101*

parseInt(**'1012s'**, 2);*//5 – взема от 0позиция докъдето стигне*

parseInt(**'2101'**, 2);*//NaN*

parseInt(**'Infinity'**, 19);*//18*

**Date**

**new** Date(2015, 0, 1)//

date = new Date(1900, 0, 1); (year, month, day) !!month->[0, 11]

date.toDateString(); // Mon Jan 01 1900

**Value and Reference**

***-value***

**var *a*** = 5;  
**function** *changeValue*(x) {  
 x++;  
}  
*changeValue*(***a***);  
**console**.log(***a***); *// 5*

***-reference***

**var *arr*** = [1, 2, 3];  
**function** *changeArr*(x) { *// подават се не стойностите, а адреса*

**for** (**var** j = 0; j < ***arr***.**length**; j++) {  
 ***arr***[j] = 5;  
 }  
}  
*changeArr*(***arr***);  
**console**.log(***arr***);*//[5, 5, 5]*

**String manipulations**

**str[index] – ако няма такъв index връща undefined**

**str.charAt[index] – ако няма такъв index връща string**

**var *str*** = **'Pesho'**;

**var *newStr*** = ***str***.**concat**(otherStr);

**var *newStr*** = ***str***.replace(**'Pe'**, **'Go'**); *//Gosho //replace-ва само първата поява*

**var *newStr*** = ***str***.**search**(/esh/); *//1 // search by Regex returns index or -1*

**var *index*** = ***str***.**indexOf**(**'Pe'**);

**var *index*** = ***str***.**lastIndexOf**(**'Pe'**);

**var *newStr*** = ***str***.**substr**(**1**,3); *//esh*

**var *newStr*** = ***str***.**substring**(1,3); *//es startIndex, endIndex without endIndex(като slice при масив)*

**var *str*** = **new** String(**'Pesho'**);  
**console**.log(**typeof *str***); *//object //[String: 'Pesho']***console**.log(**typeof *str***.**valueOf()**); *//string //Pesho->връща примитивен тип от Object String*

**var *newStr*** = ***str***.**trim()**;  
**var *newStr*** = ***str***.**trimLeft();**  
**var *newStr*** = ***str***.**trimRight();**

**var *newStr*** = **'\*'**.**repeat(5);** *//\*\*\*\*\**

**var *arr*** = ***str***.**split**(**','**);

**String Escaping**

String.**prototype**.htmlEscape = **function**() {  
 **var** escapedStr = String(**this**).replace(/&/g, **'&amp;'**);  
 escapedStr = escapedStr.replace(/</g, **'&lt;'**);  
 escapedStr = escapedStr.replace(/>/g, **'&gt;'**);  
 escapedStr = escapedStr.replace(/"/g, **'&quot;'**);  
 escapedStr = escapedStr.replace(/'/g, **'&#39'**);

**return** escapedStr;  
}

**JavaScript Functions**

Functions are objects!

Functions can be defined by **3 ways**:

*//by function* ***declaration, така декларирана функция няма значение къде ще е в кода*****function** *printHello*() {**console**.log(**'Hello'**);}  
  
*//by function* ***expression****(anonymous function)***var** *printHello* = **function**() {**console**.log(**'Hello'**);}  
  
*//by function* ***constructor*****var *printHello*** = **new Function**(**'console.log("Hello");'**);   
**var *sum*** = **new Function**(**'a'**, **'b'**, **'return a + b;'**);

**Variable Hoisting**

Когато сме във функция и имаме декларирани локални променливи където и да е по кода вътре, интерпретаторът минава и ги изнася най-отгоре. Т.е. променливите се декларират в началото, а присвояването на стойностите им става на конкретния ред.

**Functions**

**About ;**

**function** *returnPerson*()  
{  
 **return** {*//ако тази скоба е на долния ред - йок,защото слага ‘;’след return-a*   
 **name**: **'pesho'** }   
}

**function** *sum*(a, b) {  
 **console**.log(arguments); *//{ '0': 5, '1': 6 }*  
}

arguments -> Array-like object, има property length, връща аргументите, с които е извикана функцията, дори да са повече от параметрите.

Ако искаме да превърнем arguments в масив:

args = [].slice.apply(arguments);

args = Аrray.prototype.slice.apply(arguments);

**Ако една функция не връща нищо(void) , връща undefined**

**Function scope**

**В JavaScript само функциите създават scope, няма block-scope(като в C#) .**

**Ако променливата е декларирана без var, става глобална и се вижда извън функцията.**

**Function overloading**

**В JavaScript няма function overloading. Ако имаме 2 функции с еднакво име, дори да са с различен брой параметри, втората дефинирана ще презапише първата. Алтернативи:**

**function** *print*(a, b) {  
 **switch(arguments.length)** {  
 **case** 1: **console**.log(a);**break**;  
 **case** 2: **console**.log(a + **', '** + b);**break**;  
 }  
}

**function** *print*(a) {  
 **switch(typeof a)** {  
 **case 'number'**: **console**.log(**'a is number'**);**break**;  
 **case 'string'**: **console**.log(**'a is string'**);**break**;  
 }  
}

**В JavaScript (before ecmascript 6) няма default parameters.**

**function** *substring*(str, start, end) { *//(str, start = 0, end = str.length)*  
 **var** start = start || 0,  
 end = end || str.**length**;  
 *//other code*}

**Function context**

**Depends on how the function was invoked:**

**\*\* as function - this refers to the global object**

**\*\* as constructor - this refers to the new instance**

**\*\* as method - this refers to the owner object**

**\*\* as callback - this refers to the global object**

**var *a*** = 5;  
**function** *f*() {  
 **this**.**a** = 6;  
 **console**.log(**'inner '** + **this**.**a**);  
}

**as function**   
*f*();  
**console**.log(**'outer '** + **this**.***a***); //inner 6, outer 6 //закача се към window обекта

**as constructor**

**new** *f*();  
**console**.log(**'outer '** + **this**.***a***); //inner 6, outer 5 //закача се към обекта f

**Aко е в браузъра, защото this e глобалния обект(window**), през node.js е undefined, в node.js глобалния обект е **global**.

**as constructor**

**function** *f*() {  
 **this**.**x** = 8;  
 **console**.log(**this**);  
}  
**new** *f*(); //f{x:8}

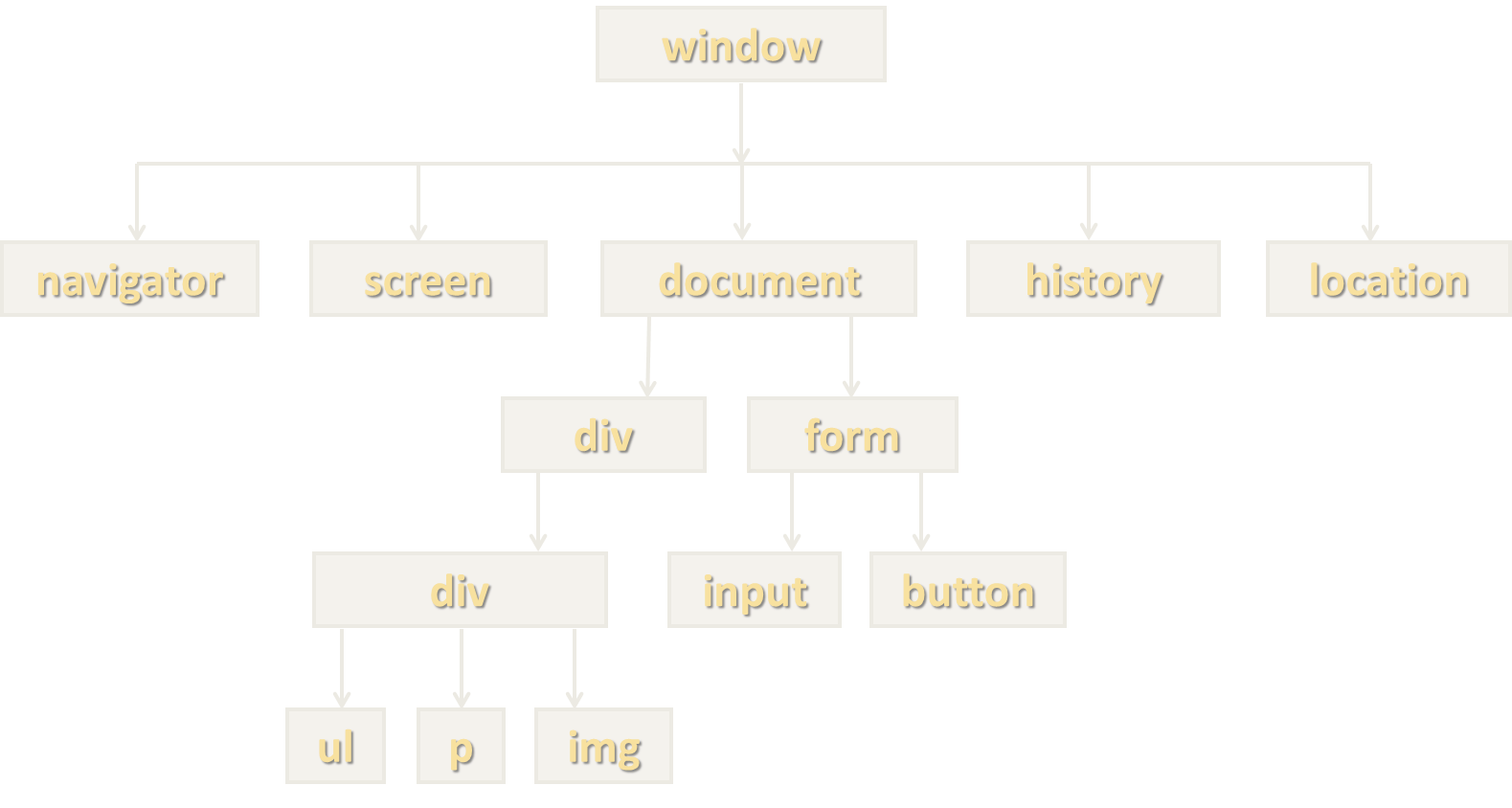
**as method**

**var *obj*** = {  
 **name**: **'Pesho'**,   
 getThis: **function**(){**return this**.name;}  
}  
**console**.log(***obj***.getThis()); *//Pesho,* ***this*** *е* ***текущия обект***

**as callback**

**var *obj*** = {  
 name: **'Pesho'**,   
 getThis: **function**(){**console**.log(**this**.name);}  
}  
  
setTimeout(***obj***.getThis, 100); *// undefined//context refers to the global object*

**In EcmaScript6 => FAT ARROW FUNCTIONS (like lambda expression in C#)**

**Built-in Browser Objects**

**Navigator – holds information about the browser**

**Screen – holds the user’s display properties**

**Document – holds information of the current loaded document**

**Window – holds information about the browser**

**API – application programming interface**

**Всеки браузър има API.Данчо: “Някаква библиотека, която се предоставя на готово и която може да използваме и която има някаква функционалност, която ни трябва”.**

**window.location == document.location – url на страницата, в която се намираме.**

**Разлика между window.location and document.location. Сега няма, преди document.location е връщало само url-то като стринг.Сегa за тази цел има document.URL**

**Hashtag # :**

<**a href="#bow"**>Върни са горе</**a**>  
<**a href="#bottom"**>Иди най-отдолу</**a**>

Отива до елемента със съответното id.Може да е на същата или на друга страница, ако е на друга страница, трябва да укажем пътя с повече материал:

<**a href="//site.com/#bottom"**>Иди най-отдолу на стraница с адрес site.com</**a**>

Може и така: **href="//site.com#bottom" – без слаша**

**DOM– Document Object Model**

**A concept of representing a HTML document as a “DOM tree”, consist of elements that have child elements. Elements have properties(attribute + value) and events**

**DOM provides an API for traversing/modifying the DOM tree. Enables developers to modify the HTML content and the visual presentation of the currently loaded HTML document.**

**Web browsers provide a DOM API. Consists of objects and methods to interact with the HTML page, can modify, add, remove HTML elements, attributes, can apply CSS styles.**

**HTML elements are mapped to JS objects**

**document**.**documentElement** *// <html> element***document**.**body** *// <body>*

**Each HTML element has a corresponding DOM object type.**

**HTMLLIElement represents <li>**

**Each of these objects have its specific properties**

**HTMLAnchorElement has href property**

**The document object is a special object.**

**It represents t**he **entry point for the DOM API (the DOM tree root)**

**var *div*** = **document**.getElementById(**'koza'**);  
**console**.log(***div* instanceof HTMLDivElement**); *//true*

**Selecting DOM Elements**

*//select a single element***var *header*** = **document**.getElementById(**'header'**);  
**var *main*** = **document**.querySelector(**'#main'**);  
  
*//select a collection***var *lis*** = **document**.getElementsByTagName(**'li'**); *//returns* ***live*** *NodeList*  
**var *header*** = **document**.querySelectorAll(**'#main li'**); *//returns* ***static*** *NodeList*  
  
*//predefined collections***var *anchors*** = **document**.**anchors**;  
**var *forms*** = **document**.**forms**;  
**var *links*** = **document**.**links**;

***Live*** *NodeList needs to cache its length for better performance, it’s slower than a regular array.*

**Difference between static and live node list example:**

**var *divs*** = **document**.getElementsByTagName(**'div'**),  
 ***queryDivs*** = **document**.querySelectorAll(**'div'**);  
  
**function** *addDiv*() {  
 **var** newDiv = **document**.createElement(**'div'**);  
 newDiv.**innerText** = **'Another Div'**;  
 **document**.**body**.appendChild(newDiv);  
   
 **console**.log(**'Live node list length:'** + ***divs***.**length**);  
 **console**.log(**'Static node list length:'** + ***queryDivs***.**length**);  
}

**Traversing the DOM**

**element.parentNode returns the direct parent og the element(null for the document)**

**element.childNodes – returns a NodeList of all the child nodes(inlcuding the text nodes)**

**element.firstChild/lastChild**

**element.nextSibling/nextElementSibling – текст(node type 3)/елемент(nodeType 1)**

**element.previousSibling/element.previousElementSibling**

**Node Types**

Node type constants

| **Constant** | **Value** | **Description** |
| --- | --- | --- |
| Node.ELEMENT\_NODE | 1 | An [Element](https://developer.mozilla.org/en-US/docs/Web/API/Element) node such as [<p>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p) or [<div>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div). |
| Node.TEXT\_NODE | 3 | The actual [Text](https://developer.mozilla.org/en-US/docs/Web/API/Text) of [Element](https://developer.mozilla.org/en-US/docs/Web/API/Element) or [Attr](https://developer.mozilla.org/en-US/docs/Web/API/Attr). |
| Node.PROCESSING\_INSTRUCTION\_NODE | 7 | A [ProcessingInstruction](https://developer.mozilla.org/en-US/docs/Web/API/ProcessingInstruction) of an XML document such as <?xml-stylesheet ... ?> declaration. |
| Node.COMMENT\_NODE | 8 | A [Comment](https://developer.mozilla.org/en-US/docs/Web/API/Comment) node. |
| Node.DOCUMENT\_NODE | 9 | A [Document](https://developer.mozilla.org/en-US/docs/Web/API/Document) node. |
| Node.DOCUMENT\_TYPE\_NODE | 10 | A [DocumentType](https://developer.mozilla.org/en-US/docs/Web/API/DocumentType) node e.g. <!DOCTYPE html>for HTML5 documents. |
| Node.DOCUMENT\_FRAGMENT\_NODE | 11 | A [DocumentFragment](https://developer.mozilla.org/en-US/docs/Web/API/DocumentFragment) node. |

**var *divs*** = **document**.getElementsByTagName(**'div'**)[0].**childNodes**;  
**for**(**var *index* in *divs***) {  
 **console**.log(***index*** + **': '** + ***divs***[***index***].**nodeType**);//  
}

**Document Fragment**

**var *div*** = **document**.createElement(**'div'**),  
 ***fragment*** = **document**.createDocumentFragment();  
***fragment***.appendChild(***div***);  
  
**document**.**body**.appendChild(***fragment***); //for optimization

**Events**

**Publish subscribe design pattern** *//(Данчо обяснение то със селото и глашатая)*

**Common Event Types**

**Mouse events – click, hover, mouseup, mousedown, mouseout**

**Keyboard events - keydown, keypressed, keyup**

**DOM/UI events- load, abort, select, resize, change**

**Focus events - focuse, blur, focusin, focusout**

**Touch events - touchstart, touchend, touchcancel, touchleave, touchmove**

***1).domElement***.addEventListener(eventType, eventHandler, isCaptureEvent)

***2).domElement***.onclick = eventHadler

***3).<button onclick = eventHandler>***

***element.onclick = function(e) { //спира дефолтното поведение на браузъра***

***e.preventDefault();***

***}***

***HTML-а не предава евента by default(3-ия начин).Трябва да го подадем като аргумент:***

***onclick = “onmouseover(event)”***

***В javascripta може да не се подава като аргумент, може да е като (e) или каквото и да друго име.***

**document**.addEventListener(**'DOMContentLoaded'**, *printAreas*(), **false**); - нещо, което да се изпълни, когато DOM дървото е заредено.

**Capturing and Bubbling Events**

element**.addEventListener (** [string] eventType, [function] listener, [boolean] useCapture **)**

Третият аргумент на addEventListener() e useCapture. By default е false. В примера е false, false, true(default, default, true). Alert-ва последователно div, btn, p

<**div**>  
 <**p**>  
 <**button**>Click</**button**>  
 </**p**>  
</**div**>

***btn***.addEventListener(**'click'**, **function**(){  
 **console**.log(**this** === ***btn***); *//true* alert (**'Button Clicked'**);  
});  
  
***p***.addEventListener(**'click'**, **function**(){  
 **console**.log(**this** === ***p***);*//true* alert (**'Paragraph Clicked'**);  
});  
  
***div***.addEventListener(**'click'**, **function**(){  
 **console**.log(**this** === ***div***); *//true* alert (**'Div Clicked'**);  
}, **true**);

false/true, false, false: **btn, p, div**

false/true, false, true: **div, btn, p**

false/true, true, true: **div, p, btn**

false/true, true, false: **p, btn, div**

**XSS (**Cross-site scripting) пример:

<**img src="Invalid source" onload="**logToConsole()**" onerror="niakakvaMizeria()"**/>

**Switch/case**

**//In JS** може и така:  
**var *score*** = **'3.5'**;  
**switch** (**true**) {  
 **case**(isNaN(***score***)):  
 **console**.log(**'Not a number'**);  
 **break**;  
 **case**(***score*** < 4): {  
 **console**.log(**'Sreden'**);  
 **break**;  
 }  
}

**var *number*** = 0;

if(!number) {числото е 0}

if(!!number) {числото не е 0}

**Най-бързият начин да умножаваме и делим с 2**

2 << 1 *//4*2 << 2 *//8*2 << 3 *//16*2 >> 1 *//1*2 >> 2 *//0*2 >> 3 *//0*8 >> 2 *//0*8 >> 3 *//1*

**console**.log(2, **'pesho'**, 4);*//2 'pesho' 4*

**RegEx**

*//Regex declarations:* - **var *regex*** = /regEx/gi;

- **var *regex*** = **new** RegExp(**'regEx'**, **'gi'**);

**(1) match []**

**var *str*** = **'text'**;  
**var *matches*** = ***str***.**match**(/t/g);  
**console**.log(***matches***);*//[ 't', 't' ]*

**(2) exec [] or null**

**(3) test** boolean

**var *str*** = **'texts'**,  
***beginWithT*** = /^t/.test(***str***),  
***endWithS*** = /s$/.test(***str***);  
**console**.log(***beginWithT***); *//true***console**.log(***endWithS***);*//true*

**var *str*** = **'088-123'**;  
**var *isValid*** = /^0\d{2}-\d{3}$/.test(***str***);  
**console**.log(***isValid***);*//true*

**(4) replace** string

**var *str*** = **'0088-123'**;  
**var *replacedStr*** = ***str***.replace(/0/g,**'\*'**);  
**console**.log(***replacedStr***);*//\*\*88-123*

**(5) split []**

**var *str*** = **'Koi? Az, pesho i gosho! Dobre.'**;  
**var *replacedStr*** = ***str***.split(/[?!,.\s]/).filter(Boolean);  
**console**.log(***replacedStr***);*//[ 'Koi', 'Az', 'pesho', 'i', 'gosho', 'Dobre' ]*

**(6) search integer number(index or -1)**

**var *str*** = **'pesho e pich'**;  
**var *index*** = ***str***.search(/pich/);  
**console**.log(***index***);*//8*

**Special characters**

**^ - начало на стринга //^A**

**$ - край на стринга //a$**

**\* - 0 или повече**

**+ - 1 или повече**

**? – 0 или 1**

**. – всичко без нов ред**

**| - или**

**[abc] – character set (или а, или b, или c)**

**[^abc] – everything excepts a, b, c**

**[A-Z]**

**[a-z]**

**[A-z]**

**{2, 5} – от 2 до 5 пъти**

**{2,} – 2 или повече**

**{3} – 3 или повече**

**\s – single space \S – everything not space**

**\d, [0-9] – digit \D [^0-9] – non-digit**

**\w – letters, digits and \_ (underscore)**

**(?:Hi|Hello),\s\*(\w+)$ => Hi, Peter //define a non-*capturing group***

***(?<=#)\d{1,4} => Gladstone #354 //positive lookbehind (not* support in JScript!!)**

**(?<![0-9\-])\d+ => Gladstone St. #-2 -123 354 2 //negative lookbehind(not support)**

**.\*?(?=\!) => This is not a drill! //positive lookahead**

**(?!subexpression) // negative lookahead**

**Escaping**

**{ }става с \\, веднъж за ескейпване на Regex-aи още веднъж за ескейпване на \ в JS**

**Това когато го билдваме с new RegExp()**

**Общи:**

**Когато търсим min и max суми - +Infinity and –Infinity. (Number.MIN\_VALUE,Number.MAX\_VALUE - не)**

Genre.prototype.deleteMovie = **function** (movie) {  
 **var** index = **this**.**\_movies**.indexOf(movie);  
 **this**.**\_movies**.splice(index, 1);  
}  
  
Genre.prototype.deleteMovieById = **function** (movieId) {  
 **var** movie = **this**.**\_movies**.filter(**function** (movie) {  
 **return** movie.**\_id** === movieId;  
 });  
 **this**.deleteMovie(movie);  
}