

# Tehnici Web

## CURSUL 5

Semestrul II, 2019-2020  
Carmen Chirita

<https://sites.google.com/site/fmitehnicweb/>

# JavaScript-istoric

- Inventat de Brendan Eich în 1995, la Netscape (denumit initial **Mocha** și **LiveScript**);
- Implementat de browserul Netscape Navigator sub numele de **JavaScript**;
- Adaptat de Microsoft în 1996 și denumit **Jscript**;
- Standardizat în 1997 de ECMA (European Computer Manufacturer's Association) sub numele de **ECMAScript**;

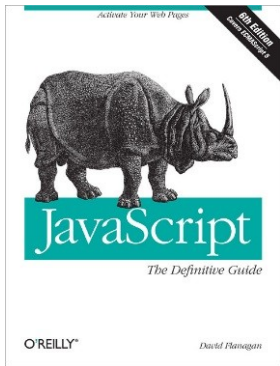
ultima versiune: **ECMAScript 2018**

# JavaScript-caracteristici

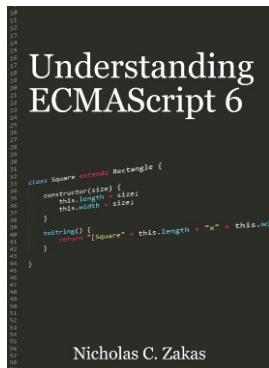
- Este un limbaj de scripting pentru pagini web (pe partea de client)
- Este bazat pe prototipuri
- Este un limbaj interpretat (scriptul este executat direct, fără compilare prealabila)
- Este „loosely typed”

# JavaScript

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>



JavaScript The Definitive Guide, David Flanagan



Understanding ECMAScript 6, Nicholas C. Zakas

<https://leanpub.com/understandings6/>

# JavaScript și HTML


- Orice pagina Web este reprezentată în memorie ca un arbore de obiecte  
(Document Object Model – DOM)
- JavaScript poate accesa elementele HTML prin intermediul DOM-ului.

element HTML      →      Obiect în JavaScript

Atribut al unui element HTML      →      Proprietate a obiectului în JavaScript

## Codul JavaScript poate fi plasat:

Oriunde in documentul HTML, folosind tagul **script**



```
<head>  
<script type="text/javascript">  
/* cod JavaScript */  
</script>  
</head>
```

Într-un **fișier extern** (NumeFisier.js) care este importat in documentul HTML (varianta recomandata)



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

Exemplu de cod JavaScript care calculează și afiseaza într-o fereastră alert suma nr. de la 1 la 9.

```
<head>

<script type="text/javascript" >

var s = 0;
for (var i=1; i< 10 ; i++) s=s+i;
alert(s);
</script>

<meta charset="utf-8">
<title> JavaScript </title>
</head>
<body>

</body>
```

Varianta folosind o functie

```
<head>

<script type="text/javascript" >

function suma(x) {
var s = 0;
for (var i=1; i< x ; i++) s=s+i;
return s;}

alert('Suma este ' +suma(10));
</script>

<meta charset="utf-8">
<title> JavaScript </title>
</head>
<body> </body>
```

JavaScript este **CASE SENSITIVE** și folosește setul de caractere **Unicode**

### Identificatorii:

- denumesc variabile, cuvinte cheie, funcții, etichete;
- formați din: cifre, litere, \_, \$;  
primul caracter: **litera, \_, \$**

; separator (ex. a = 3; b = 4;)

{ } bloc de instrucțiuni

```
/* comentariu  
   pe mai multe linii  
*/
```

```
// comentariu pe o singura linie
```



## Tipuri de date:

**primitive:** number, string, boolean, null, undefined

**tipul** object

**Obiecte predefinite:** Array, Function, String, Number, Boolean, Math, Date,...

## Instrucțiuni:

=, if/else, for, switch, while, return, { inst1; inst2;}

## Funcții:

function Nume(param1,param2,..) {corpul functiei}

## Tipuri de date în JavaScript:

**primitive:** number, string, boolean, null, undefined

**tipul** object

In Javascript **nu** este necesara precizarea tipului de date, ca în alte limbaje de programare (ex. C/C++, Java)

Browserul realizeaza singur identificarea tipului de date

## Variabile în JavaScript

- Pot fi declarate explicit folosind cuvântul cheie **var**; optional variabila poate fi initializata cu o valoare;

variabilele declarate astfel pot fi locale (declarate în interiorul functiilor) sau globale (declarate în afara functiilor).

```
var a = 1, b = 5; //variabila number  
var r = 2.5; //variabila number  
var mesaj= " acesta este un string"; //string  
var x; //undefined
```

- Atribuirea unei valori unei variabile nedeclarate înainte cu **var** creaza o variabila **globala** **indiferent de locul unde apare în program**

```
y = "sunt globala" // variabila globala
```

## Variabile în JavaScript

- Variabilele se mai pot declara folosind cuvântul cheie **let**; acesta declara o variabila locala vizibila **doar** în blocul, instrucțiunea sau expresia în care este folosită.

ex.1

```
var g = 0;
{
  let g = 1;
}
console.log (g); // 0
```

ex.2

```
function letTest() {
  let x = 1;
  if (true) {
    let x = 2;
    console.log(x); // 2
  }
  console.log(x); // 1
}
```

In ECMAScript 6 au fost introduse declaratii de constante

```
const Max =100;
```

## Variabilele au tipuri dinamice

### JavaScript este "loosely typed"

Tipul variabilei nu este specificat explicit, dar poate fi aflat cu `typeof(x)`

Tipul unei variabile nedeclarate este undefined.

```
typeof(null) // "object"
```

```
typeof(undefined) // "undefined"
```

```
> x=2
2
> typeof(x)
"number"
> x="Hello"
"Hello"
> typeof(x)
"string"
> typeof(y)
"undefined"
```

# Scopul variabilelor: zona din program în care sunt declarate variabilele; exista scop global și scop local

In JavaScript scopul este creat de functii;  
orice functie creaza un scop.

ex.1

```
var x = "globala"; //globala
function func1() {
  var x="locala";
  return x;
}
func1();    // => "locala"
alert(x);   // => "globala"
```

ex.2

```
x = "globala"; //globala
function func1() {
  x="locala";
  return x;
}
func1();    // => "locala"
alert(x);   // => "locala"
```

```
// scopul global
function fA (){
  //scopul A
  function fB () {
    // scopul B
  }
}
```

## Scop Lexical

Toate variabilele/obiectele/functiile declarate de o functie parinte sunt vizibile in descendenti ei

ex.3

```
var x = "globala";
function func1()
{
  var x="locala1";
  function func2()
  {var x="locala2";
   return x;
  }
  return func2();
}
func1(); // => "locala2"
```

# Hoisting in JavaScript

Domeniul de vizibilitate al unei variabile coincide cu functia in interiorul careia a fost definita;

Inainte de a fi executat, codul JavaScript este parsat si "rearanjat" a.i. toate declaratiile de variabile (nu si operatiile de atribuire) sunt mutate (ridicate) la inceputul zonei de vizibilitate (adica la inceputul functiei).

```
var x = 5; //globala
function host() {
  if (x == 5) {
    var x = 10;
  }
  alert(x);
}
host(); // va afisa undefined
alert(x); // va afisa 5
```

echivalent cu

```
var x = 5; //globala
function host() {
  var x; // variabila locala
  if (x == 5) { // x este undefined
    x = 10;
  }
  alert(x);
}
host(); // va afisa undefined
alert(x); // va afisa 5
```

# Tipul number (reprezentare binara pe 64 biti)

```
var a = 4;  
var r = 34.7;
```

## Operatorii aritmetici specifici:

+ - \* / % ++ --

## Conversia de tip automata

```
x = "2" * 7; // 14  
y = "2" + 7; // "27"  
z = parseInt("2") + 7; // 9  
t = "2" * "7"; // 14  
u = 2 + 3 + "4"; // "54"  
z = "2" + 3 + 4; // "234"
```

## Obiectul [Math](#)

Math.PI //=> 3.14

Math.pow(2,3) //=> 8

Math.round(4.7) //=> 5

Math.random() // intre 0 si 1

Math.sqrt(-1) // => NaN

conține proprietăți  
și metode

> x=5

5

> y=3.4

3.4

> typeof(x)

"number"

> typeof(y)

"number"

> parseInt("5hello")

5

> parseFloat("3.4Hello")

3.4

> parseInt("3.4Hello")

3



## Tipul string (sir de caractere scris intre ' ' sau " ")

```
var s = "Ana Popescu";  
var t = 'Ana Popescu';  
var pnume = s.slice(0, s.indexOf(" ")); //'Ana'  
var fnume=s.slice(s.lastIndexOf(" ")+1,s.length); //'Popescu'
```

Proprietăți și metode: `length`, `charAt()`, `indexOf()`, `lastIndexOf()`, `replace()`, `split()`, `toLowerCase()`, `toUpperCase()`, `concat()`, ..

Concatenarea: `"numarul" + "1"`, `"id"+1`

Caractere speciale: `\'` `\"` `\n` `\t` `\v` `\b` `\\`

Accesarea unui caracter: `s[0]`, `s.charAt(0)`, `s.charAt[s.length-1]`

```
var x = "abcde";  
alert(x[0]);  
x[0] = 'v';  
alert(x[0]); // => a
```

Un string nu este un array de caractere

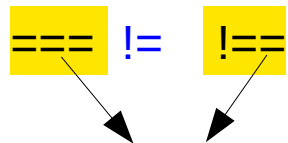
## Tipul boolean: true si false

Orice valoare poate fi convertita explicit folosind obiectul predefinit Boolean

```
var nume = Boolean(valoare);  
  
true === Boolean("adevarat") // => true  
false === Boolean("") // => true
```

Operatori logici pentru tipuri primitive: > < <= >= && || ! == === != !==

Alte valori pot fi folosite ca si Boolene:  
pentru fals: 0, "", NaN, null, undefined  
pentru adevarat: orice alta valoare



verifica si tipul operanzilor

### Operatorul conditional

*conditie ? expr1 : expr2*

```
function fact(n){  
  return (n <= 2) ? n: n*fact(n-1);  
}
```

## Tipurile undefined si null

variabilele care nu au primit încă o valoare au tipul undefined

```
var x  
x == undefined // true  
typeof(x) // undefined
```

null lipsa unei valori (intentionata)

```
var x=null  
typeof(x) //object  
null == undefined // true  
null === undefined // false
```

# Tipul object

Un obiect este o colectie de perechi nume-valoare.  
Daca valoarea este o functie atunci proprietatea se numeste metoda.

```
var ob = {prop1: val1, prop2: val2, ... ,prop-n: val-n};
```

Accesarea proprietatilor:

```
ob.prop1; // val1
```

```
ob["prop1"]; // val1
```

## Exemple:

```
var complex = { re: 5, im: 3}; \\ complex.re, complex.im
var student = {nume: "Ionescu",
               nota1:9,
               nota2:10,
               media:function(){
                   return (this.nota1 + this.nota2)/2;
               }
            }
```

```
student.nume    \\ Ionescu
student.nota1   \\ 9
student.nota2   \\ 10
student.media() \\ 9.5
student.media   \\ functia
```

**this** este obiectul cu care se va  
apela functia

# Tipul object

Toate datele din tipurile primitive in afara de tipul **object** sunt transmise prin valoare.

Datele de tip **object** sunt transmise prin referinta.

```
var a = {nume: "Ana"} // object  
var b = a; // a și b refera aceeași zona  
b.nume = b.nume + " Popescu"; // se modifica și b și a  
alert(a.nume ); // "Ana Popescu"
```

```
var s = "Ana"; // string  
var t = s; // t copiaza valoarea lui s  
t = t + " Popescu" // se modifica doar t  
alert(s) // => "Ana"
```

# Crearea obiectelor

- Prin obiect literal:

proprietatile, metodele, împreună cu valorile lor sunt enumerate între acolade; se creează un singur obiect.

```
var pers= {nume:"Popescu",prenume:"Andrei",vârsta:20}
```

# Crearea obiectelor

- Cu ajutorul obiectului generic

se apeleaza constructorul `new Object()` și se adauga apoi proprietatile și metodele; se creeaza un singur obiect

```
var pers= new Object();  
pers.num="Popescu";  
pers.prenume="Andrei";  
pers.varsta=20;
```



# Crearea obiectelor

- Cu ajutorul unui constructor de obiecte

Se definește o funcție `constructor(parametrii)` care apoi va fi apelată cu `new constructor(parametrii)` pentru fiecare obiect care va fi creat

```
function pers(n,p,v) { this.num= n;  
                        this.prenume=p;  
                        this.varsta=v;  
                        }
```

```
var p1=new pers("Popescu","Andrei",20);  
var p2=new pers("Ionescu","Bogdan",20);
```

# Proprietăți și metode globale

Pot fi folosite împreună cu orice variabilă și obiect creat în JavaScript

Proprietate	Descriere
<b>Infinity</b>	O valoare numerică care reprezintă infinitiv pozitiv/negativ
<b>NaN</b>	O valoare "Not-a-Number"
<b>undefined</b>	Indică o variabilă căreia nu i-a fost atribuită o valoare

## Metode

```
isNaN() // Determină dacă valoarea este un număr invalid  
parseInt() //converteste un sir într-un intreg  
parseFloat() //converteste un sir într-un număr zecimal  
Number() //converteste un obiect într-un număr  
String() //converteste un obiect într-un sir
```

# Obiecte predefinite în JavaScript

Obiecte corespunzătoare  
tipurilor primitive

Boolean, String, Number

se pot crea obiecte noi cu  
new

Object

Array, Set, Map

Function

RegExp

Date

se pot crea obiecte noi cu  
new

```
var y = new Number(123);  
typeof(y) // object
```

```
var ob = new Object();  
ob.x =1; ob.y=2; // ob ={x:1, y:2}
```

```
var d= new Date(2015,3,1);  
alert(d.getUTCDay()); // 2 (ziua din  
săptămâna (0-6))
```

# Array

```
var v = new Array();  
v[0] = "a"; v[1] = "b"; c  
  
var v = new Array("a","b");  
  
var v = [6,4,7,3];
```

The diagram shows three lines of code. The first line is red, the second is blue, and the third is purple. Arrows point from the end of each line to the word 'create'.

## Proprietati si metode

```
v=[6,4,7,3];  
v.length // 4  
v.push(10); // => v=[6,4,7,3,10]  
v.pop(); // => v=[6,4,7,3]  
v.shift(); // => v=[4,7,3]  
v.unshift(10); // => v=[10,4,7,3]  
v.sort(); // => v=[3,4,6,7]
```

tipul elementelor nu e fixat

```
v=["hi",2,[5,7]]
```

```
var s = "azi este joi";  
  
var a = s.split(" ");  
// a = ["azi", "este", "joi"]  
  
a.reverse();  
// a = ["joi", "este", "azi"]  
  
var s = a.join('/');  
// s = "joi/este/azi"
```

## Exemplu

Generarea random a unei culori dintr-un vector de culori și colorarea body-ului în culoarea respectiva

```
function getRandomInt(max) {  \generarea random a unui întreg între 0 și max-1
    return Math.floor(Math.random() * Math.floor(max));
}
var culori=["blue","green","yellow","black","red","orange","white","pink"];
document.body.style.backgroundColor=culori[getRandomInt(culori.length)];
```

**random(), floor()** : metode ale obiectului **Math**

**length:** proprietate a obiectului **Array**

## Array: metoda **filter()**

- creaza un nou array cu elementele care verifica conditia implementata de functia data ca parametru
- nu modifica array-ul initial

Sinaxa

```
array.filter(function(currentValue, index, arr), thisValue)
```



optionale

Exemplu

```
function check(cuvant) {  
  return cuvant.length > 4;  
}  
  
var cuvinte = ["pisica", "cal", "caine", "oaie"];  
var rez=cuvinte.filter(check);  
console.log(rez);    //rez=["pisica", "caine"]
```

## Array: metoda `reduce()`

- executa o funcție de reducere (data ca parametru) pe fiecare element al array-ului rezultand o singura valoare de ieșire
- nu modifica array-ul initial

### Sinaxa

```
array.reduce(function(total, currentValue, currentIndex, arr), initialValue)
```

### Exemplu

```
function suma(total,val) {  
  return total+val;  
}  
  
var array = [1,2,3,4];  
var rez=array.reduce(suma);  
console.log(rez);    //rez=10;
```



optionale

## Set (introduces in ECMAScript6)

Metode: add(val), has(val), size(), values(), keys(), delete(val), clear()

```
var m = new Set();  
m.add(1).add(2).add(3);  
if (m.has(3)) {let s=0;  
                for (e of m.values()) s=s+e;  
                alert(s);}
```



# Instructiuni: for (exista 2 variante noi ale instructiunii for)

```
for (initializare; conditie; update) {  
  instructiuni;  
}
```

```
for (var i = 0; i < 9; i++) {  
  console.log(i);}
```

//afiseaza la consola nr. de la 0-8

```
for (variabila in obiect) {  
  instructiuni  
}
```

```
var pers = {nume:"Popescu", prenume:"Andrei", varsta:30};  
var text = " ";  
var x;  
for (x in pers) {  
  text += pers[x] + " ";  
}  
//text va conține „Popescu Andrei 30”
```

```
for (item of iteratii) {  
  instructiuni  
}
```

```
var tablou = [10, 20, 30];  
for (var x of tablou) {  
  x += 1;  
  console.log(x);  
}  
//se va afisa la consola 11, 21, 31
```

# Instrucțiuni: while, do, if, switch (întâlnite și în alte limbaje de programare)

```
while (conditie) {  
    instructiuni;  
}
```

```
do {  
    instructiuni  
} while (conditie);
```

```
if (conditie) instructiune;
```

```
if (conditie) instructiune  
else instructiune;
```

```
switch (expresie)  
{  
    case 1:  
        bloc 1  
        break;  
    case 2:  
        bloc 2  
        break;  
    .....  
    default : bloc  
}
```

# Funcții

Sintaxa:

```
function nume(arg1, arg2,..., argn) {  
    instructiuni;  
    return valoare; // nu neaparat  
}
```

În Java Script o funcție poate fi apelata cu un numar variabil de parametrii

```
function suma(a,b) {  
    return a+b;  
}
```

```
suma(2,3); // 5  
suma(); // NaN  
suma(2); // NaN  
suma(3,4,1,5,6,7) // 7
```

```
var x = 0;  
var y = {n:0};  
function f(a,b){  
    a = 1;  
    b.n = 1;}  
f(x,y);  
alert(x); //=> 0  
alert(y.n); //=> 1
```

Parametrii de tip primitiv se transmit prin valoare;  
Parametrii de tip obiect se transmit prin referinta.

# Funcții

În Java Script orice funcție poate accesa un obiect notat „arguments” (asemănător unui array) care conține valorile argumentelor cu care se apelează funcția;

arguments.length va calcula numărul argumentelor

```
function fun() {  
  return arguments.length;  
}  
  
fun(2, "sss", 5); // 3
```

```
function func1(a, b, c) {  
  console.log(arguments[0]);  
  // expected output: 1  
  
  console.log(arguments[1]);  
  // expected output: 2  
  
  console.log(arguments[2]);  
  // expected output: 3  
}  
  
func1(1, 2, 3);
```

Exemplu (developer.mozilla.org)

## Exemplu folosind obiectul arguments:

Rescrierea functiei **suma** care calculează suma argumentelor functiei indiferent de numarul de argumente cu care se va apela functia

```
function suma()  
{  
    var s=0;  
    for(var i=0; i<arguments.length; i++)  
        s+=arguments[i];  
    return s;  
}
```

```
suma(); // 0
```

```
suma(5); // 5
```

```
suma(2,3,4,5); //14
```

## Functii anonime

```
function (arg1,...,argn){  
    instructiuni;  
}  
  
var fun = function () {  
    return arguments.length;  
}
```

```
function f(x){return x+1};  
var f1 = f;  
var x=f(3);  
var x1=f1(3);
```

```
var g = function (x){return x+1};  
var y = g(3);  
typeof(g) // "function"
```

```
var h = (x)=>{x+1}
```



arrow functions  
ECMAScript 6

Orice tab al unui browser contine un obiect **window** (din clasa Window)

## Metodele **prompt** si **alert**

**prompt(text, default-text)**: afiseaza o caseta de dialog care cere utilizatorului sa introduca informatii

**alert(mesaj)**: afiseaza o caseta de alertare care contine un mesaj și un buton OK

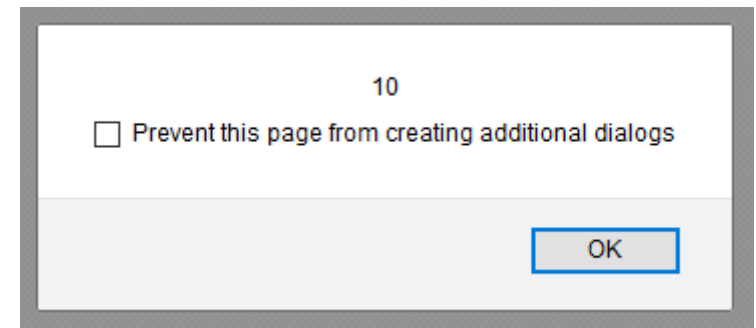
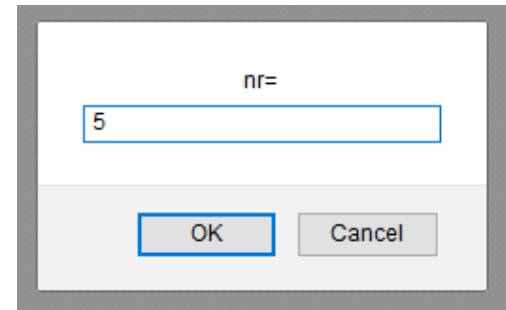
```
var x = prompt("nr1");  
var y = prompt("nr2");  
alert(typeof(x));  
alert(x+y);  
alert(parseInt(x)+parseInt(y));
```

```
<script
type="text/javascript" >

var n=prompt("nr= ");
alert(suma(n));

function suma(x) {
var i;
var s = 0;
for (i=1; i< x ; i++) s=s+i;
return s;};

</script>
```





## Exercițiu propus de Irina Ciocan

Consideram variabila `v=["abc#bbb","a#b#c","zzz#q"]`.

Realizati toate operatiile cerute mai jos, intr-un singur rand, fara a folosi instructiuni conditionale (if) sau repetitive (for, while, do...while) ci doar metode ale array-urilor sau stringurilor:

- concatenati toate elementele din vector, obtinand un singur sir (rezultat partial: "abc#bbba#b#czzz#q")
- aduceti toate literele la litere mari (rezultat partial: "ABC#BBBA#B#CZZZ#Q")
- obtineti un vector de siruri impartind sirul de mai sus in bucati, considerand drept separator caracterul # (rezultat partial: ["ABC","BBBA","B","CZZZ","Q"])
- din vectorul obtinut stergeti cuvintele care au mai putin de 3 elemente (rezultat partial: ["ABC","BBBA","CZZZ"] )
- din vectorul ramas obtinem cel mai mare sir din punct de vedere al ordinii alfabetice: (rezultat: "CZZZ")

### Rezolvare:

```
var res=v.join("").toUpperCase().split("#").filter(a => a.length>=3).reduce((a,b) => a>b?a:b);
```