**Curs si Tutoriale JavaScript**

**Curs Javascript**

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

**Lectiile si tutorialele din acest curs JavaScript sunt gratuite** si se adreseaza in special incepatorilor in web development si programare site-uri web.  
Incepe cu lucrurile de initiere si de baza ale programarii JavaScript, instructiuni, cod si exemple simple, continuand cu elemente mai complexe, obiecte JavaScript, evenimente, functii, clase, si altele.

# intaxa JavaScript

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

### Adaugarea JavaScript intr-o pagina HTML

Codul JavaScript se adauga in paginile web folosind elementul HTML: **<script> </script>**.  
In interiorul etichetei **<script> ... </script>** se scrie codul JS.  
Pentru scrierea si executarea programelor JavaScript avem nevoie de un editor simplu de texte (cum ar fi Notepad++) si un browser (ex. Mozilla Firefox, Google Chrome).  
  
Putem, de asemenea, sa introducem instructiunile JavaScript intr-un alt fisier, extern, care va avea extensia "**.js**", pentru editarea acestui fisier este nevoie la fel de un editor simplu de texte. Avantajul fiind ca putem folosi acelasi cod in mai multe pagini HTML si in cazul necesitatii unei modificari in codul JavaScript, modificam doar datele dintr-un singur fisier (cel cu extensia ".js").  
In cazul in care codul JavaScript se afla intr-un fisier extern, eticheta **<script>** din pagina HTML va trebui sa contina atributul "**src**" a carui valoare e locatia fisierului in care se afla codul JavaScript.  
- In fisierul extern cu extensia "js" nu trebuie sa scriem eticheta "<script>", scriem direct instructiunile scriptului.  
  
Iata un exemplu de script JavaScript scris in interiorul unei pagini web (HTML):

<!DOCTYPE html>

<html>

<head>

<title>Title</title>

</head>

<body>

<h3>Continut HTML ..</h3>

<script>

document.write('Textul afisat cu document.write() din JS.');

</script>

</body>

</html>

Incercati codul

Instructiunea **document.write** este folosita pentru a tipari ceva in pagina.  
  
Daca dorim sa incarcam scriptul dintr-un fisier extern (de exemplu "*cod.js*"), codul nostru in documentul HTML va arata astfel:

<!DOCTYPE html>

<html>

<head>

<title>Title</title>

</head>

<body>

<h3>Continut HTML ..</h3>

<script async src='cod.js'></script>

</body>

</html>

*Atributul "****async****" specifica incarcarea script-ului in mod asincron; adica, restul codului din pagina continua sa se incarce, fara a fi intrerupt de scriptul din 'cod.js' (e necesar cand instructiunile din script incarca alte fisiere externe, cum ar fi imagini, multiedia, etc.).*

- Iar fisierul "cod.js" va contine:

document.write('Textul afisat cu document.write() din JS.');

Rezultatul afisat in pagina web va fi acelasi ca-n exemplu anterior.

### Comentarii in codul JavaScript

Comentariile in interiorul codului sunt necesare cand dorim sa specificam rolul anumitor functii si variabile, pentru o mai usoara intelegere ulterioara a scriptului.  
Pentru a adauga un comentariu, pe o singura linie, in interiorul codului, incepem scrierea acestuia cu succesiunea //.  
Daca dorim sa scriem comentarii pe mai multe randuri, se foloseste /\* la inceputul comentariului si \*/ la sfarsitul acestuia.

<script>

// Comentariu pe o singura linie

let var1 ='MarPlo.net';

/\*

Comentarii pe mai multe linii

Alt rand,

Viata e stare de Fericire, Bucurati-va

\*/

var var2 ='Happiness';

</script>

### Conventii de sintaxa

In orice limbaj, scrierea are conventii si reguli de sintaxa. Chiar si scrierea în limba româna are regulile ei de sintaxa: orice propozitie incepe cu litera mare, se termina cu un semn de punctuatie, etc.! La fel si limbajele de programare au regulile si sintaxa lor.  
In continuare vor fi prezentate regulile de sintaxa din limbajul JavaScript.

1. **Case-sensitive** - se face diferenta intre literele mari si mici, astfel cuvinte precum "exemple, Exemple" vor fi tratate diferit.
2. **Punct si virgula (;)** - Declaratiile /liniile de cod trebuie sa se termine cu un caracter "punct si virgula" (;).  
   Exemplu:
3. var1 = 8; var2 ='abc';

var3 ='JavaScript';

1. **Spatiile libere** - JavaScript ignora spatiile libere, tab-urile si liniile libere care apar în instructiuni, acestea sunt utile pentru a face codul mai bine structurat si usor de citit. Recunoaste doar spatiile care apar în string-uri (sirurile de caractere).  
   Exemplu:
2. var1 = 'Tutorial JS' ;
3. // E la fel cu

var1='Tutorial JS';

1. **Ghilimelele** - Ghilimelele simple ('') si duble ("") sunt folosite pentru a delimita sirurile de caractere (string). (Exemplu: "Invat JavaScript" sau '*String JavaScript*' ).
2. **Caractere speciale** - cand scriem scripturi, apare necesitatea de a folosi in cod sau in datele de iesire, un caracter specia, sau o linie noua. Pentru aceasta folosim caracterul backslash "\" in fata unuia din codurile Escape , astfel :
   * **\b** - backspace  
     **\f** - indica o pagina noua  
     **\n** - linie noua  
     **\r** - indica un retur de car  
     **\t** - indica spatiu de tasta TAB  
     **\\** - caracter backslash  
     **\'** - indica un apostrof (ghilimele simple)  
     **\"** - indica ghilimele duble
   * De exemplu, daca dorim sa afisam un text, folosind **document.write()**, iar acel text trebuie sa contina ghilimele si caracter backslash "\", si anume textul (*Curs "JavaScript" \ MarPlo.net*), pentru a nu "deruta" scriptul in interpretarea codului, deoarece ghilimelele si backslash fac parte din sintaxa, adaugam **\** in fata acestor caractere din interiorul sirului. Comanda de afisare a sirului va fi astfel:

document.write("Curs \"JavaScript\" \\ MarPlo.net");

1. **Numele variabilelor si functiilor** - numele dat variabilelor si functiilor trebuie sa respecte urmatoarele reguli:
   * - primul caracter trebuie sa fie o litera, un caracter de subliniere (\_) sau semnul $.  
     - primul caracter nu poate fi un numar.  
     - numele nu trebuie sa contina spatii libere.  
     - nu se folosesc cuvinte rezervate, care fac parte din limbajul JavaScript (cum sunt "array", "status", "alert", "script"), deoarece interpretorul programului nu va face diferenta intre aceste nume si comenzile JavaScript cu aceleasi nume.

# Definire Variabile si Constante

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Tipuri de valori si Variabile](https://marplo.net/javascript/definire-variabile-constante" \l "hshtv" \o "Tipuri de valori si Variabile)
* [Definire si utilizare Constante](https://marplo.net/javascript/definire-variabile-constante" \l "hshdc" \o "Definire si utilizare Constante)

In aceasta lectie veti invata despre variabile si constante, elemente de baza ale programari, care sunt necesare pentru a scrie un script. Aceste elemente sunt similare cu cele din PHP si alte limbaje de programare.

### Definire si utilizare variabile

Variabilele sunt nume stocate in memorie la care se atribue /memoreaza diferite date.  
In JavaScript variabilele se pot defini folosind declaratia: **var** sau **let**.

**var name ='Value';**

**//Sau**

**let name ='Value';**

- Unde 'name' este numele variabilei, iar 'Value' este valoarea atribuita.  
  
Variabilele se folosesc specificand numele lor, si rezulta valoarea care este atribuita.

<script>

let tjs = 'Tutorial JavaScript - Variabile';

document.write(tjs);

</script>

Incercati codul

#### Diferenta dintre var si let

Cu **var** se definesc variabile Globale, valabile in tot scriptul si care pot fi apelate si modificate in alte sectiuni (corpuri separate) din script; sau Locale, valabile in corpul functiilor.  
Declaratia **let** permite definirea de variabile care sunt limitate la sectiunea /scopul, expresia unde au fost definite.  
  
- Se poate intelege diferenta dintre 'var' si 'let' din aceste doua exemple similare.  
1. Exemplu cu "let":

<script>

let xn = 1;

if(xn ==1){

let xn =2;

}

document.write(xn); // 1

</script>

Incercati codul

2. Exemplu cu "var":

<script>

var xn = 1;

if(xn ==1){

var xn =2;

}

document.write(xn); // 2

</script>

Incercati codul

#### Tipuri de valori si Variabile

Sunt mai multe tipuri de date care se pot atribui la variabile. Tipul valorii adaugate determina tipul variabilei.

* **string (sir)** - Valori de tip text, adaugate intre ghilimele simple sau duble.

var x = 'Site MarPlo.net';

* **number** - Numere intregi sau cu zecimale.
* var x = 8;

var y = 7.95;

* **booleen** - Valoarea **true** sau **false**.
* var x = true;

var y = false;

* **array** - Mai multe valori ordonate cu index numeric.
* let x = ['ab', 78, 'xy'];

alert(x[1]); // 78

* **object** - Un obiect de date care poate contine proprietati si functii.

var x = {prop1:'ab', p2:78};

* **function** - O functie, care poate returna o valoare (cu return).
* var fun = function(){
* return 'Value';

};

* **undefined** - Variabila declarata fara vreo valoare.

let x;

* **null** - Valoare nula, anulata.

var x = null;

In JavaScript tipul unei variabile poate fi schimbat pe parcursul codului.

<script>

var x; //acum e undefined

x = 5; //acum e number

document.write(x);

x = '<h4>MarPlo</h4>'; //acum e string

document.write(x);

</script>

Incercati codul

### Definire si utilizare Constante

Constante se definec cu declaratia: **const**.  
Spre deosebire de variabile, valoarea unei constante nu poate fi schimbata si nu poate fi redeclarata, valoarea lor ramane aceeasi, fixa.

const X = 'MarPlo';

//schimband valoarea, rezulta eroare in consola

X ='abc';

//redeclarare, rezulta eroare in consola

const X = 123;

Ca si variabilele, constantele se folosesc specificand numele lor, si rezulta valoarea care este atribuita.

<script>

const TJC = 'Tutorial JavaScript - Constante';

document.write(TJC);

</script>

# Operatori in JavaScript

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Operatori de atribuire](https://marplo.net/javascript/operatori.html" \l "hshoa" \o "Operatori de atribuire)
* [Operatori de comparare](https://marplo.net/javascript/operatori.html" \l "hshoc" \o "Operatori de comparare)
* [Operatori logici](https://marplo.net/javascript/operatori.html" \l "hshol" \o "Operatori logici)
* [Operator de concatenare pentru siruri](https://marplo.net/javascript/operatori.html" \l "hshocs" \o "Operator de concatenare pentru siruri)
* [Operatorul conditional (Ternary)](https://marplo.net/javascript/operatori.html" \l "hshoct" \o "Operatorul conditional (Ternary))
* [Precedenta operatorilor](https://marplo.net/javascript/operatori.html" \l "hshpo" \o "Precedenta operatorilor)

Operatorii se folosesc pentru a manipula, combina si modifica datele dintr-un program sau script. Acestia sunt de mai multe feluri, in aceasta lectie sunt prezentati **tipurile de operatori folositi in JavaScript**.

### Operatori aritmetici

Operatorii aritmetici sunt principalii operatori folositi cu numere, acestia efectueaza operatiile aritmetice cunoscute:

|  |  |
| --- | --- |
| **Operator** | **Exemplu** |
| **Adunare: +** | let a = 8;  let b = 3;  document.write(a + b); // 11  Incercati codul |
| **Scadere: -** | let a = 8;  let b = 3;  document.write(a - b); // 5  Incercati codul |
| **Inmultire: \*** | let a = 8;  let b = 3;  document.write(a \* b); // 24  Incercati codul |
| **Impartire: /** | let a = 9;  let b = 3;  document.write(a / b); // 3  Incercati codul |

Pe langa acesti patru operatori, in programare sunt folositi inca trei operatori aritmetici:

|  |  |
| --- | --- |
| **Operator** | **Descriere /Exemplu** |
| **Modul: %** | - Determina restul impartirii a doua numere.  let a = 8;  let b = 3;  document.write(a % b); // 2  Incercati codul |
| **Incrementare: ++** | - Creste valoarea cu o unitate.  let a = 9;  a++;  document.write(a); // 10  Incercati codul |
| **Decrementare: --** | - Scade valoarea cu o unitate.  let a = 9;  a--;  document.write(a); // 8  Incercati codul |

### Operatori de atribuire

Operatorul de atribuire actioneaza de la dreapta la stanga; se evalueaza operandul din dreapta iar valoarea se atribuie variabilei din stanga semnului "=".

|  |  |
| --- | --- |
| **Operator** | **Exemplu** |
| **=** | let a = 8;  document.write(a); // 8  Incercati codul |
| **+=** | let a = 8;  a += 3; //La fel cu: a = a+3  document.write(a); // 11  Incercati codul |
| **-=** | let a = 8;  a -= 3; //La fel cu: a = a-3  document.write(a); // 5  Incercati codul |
| **\*=** | let a = 8;  a \*= 3; //La fel cu: a = a\*3  document.write(a); // 24  Incercati codul |
| **/=** | let a = 84;  a /= 4; //La fel cu: a = a/4  document.write(a); // 21  Incercati codul |
| **%=** | let a = 8;  a %= 3; //La fel cu: a = a%3  document.write(a); // 2  Incercati codul |

### Operatori de comparatie

Operatorii de comparatie se folosesc pentru a evalua doua valori pe care le compara. Raspunsul poate fi TRUE sau FALSE.

*Un operator de comparatie des folosit este operatorul de egalitate, reprezentat prin doua semne egal "==". Este diferit de simplul "=" (acesta atribuie o valoare).  
Operatorul '==' compara doua valori determinand daca acestea sunt identice, adica egale ca valoare.*

- Lista cu operatorii de comparare:

|  |  |
| --- | --- |
| **Operator** | **Descriere /Exemplu** |
| **==** | - Egalitate ca valoare.  let a = '4'; //string  let b = 4; //number  document.write( (a == b) ); //true  Incercati codul |
| **===** | - Identic, egal ca valoare si ca tip de date.  let a = '4'; //string  let b = 4; //number  document.write( (a === b) ); //false  Incercati codul |
| **!=** | - Inegalitate ca valoare.  let a = 9;  let b = 4;  document.write( (a != b) ); //true  Incercati codul |
| **!==** | - Inegalitate ca valoare sau ca tip de date.  let a = 4;  let b = 4;  document.write( (a !== b) ); //false  Incercati codul |
| **>** | - Mai mare.  let a = 9;  let b = 4;  document.write( (a > b) ); //true  Incercati codul |
| **<** | - Mai mic.  let a = 9;  let b = 4;  document.write( (a < b) ); //false  Incercati codul |
| **>=** | - Mai mare sau egal.  let a = 4;  let b = 4;  document.write( (a >= b) ); //true  Incercati codul |
| **<=** | - Mai mic sau egal.  let a = 5;  let b = 4;  document.write( (a <= b) ); //false  Incercati codul |

### Operatori logici

Similar cu operatori de comparatie, operatorii logici compara doua expresii si returneaza TRUE sau FALSE.

|  |  |
| --- | --- |
| **Operator** | **Descriere /Exemplu** |
| **&&** | - Denumit AND (Si), compara doua expresii si returneaza TRUE daca amandoua sunt adevarate, in caz contrar returneaza FALSE.  var x = 9;  var y = 4;  document.write( (x>7 && y<8) ); //true  Incercati codul |
| **||** | - Denumit OR (sau), compara doua expresii si returneaza TRUE daca cel putin una din ele este adevarata, in caz contrar returneaza FALSE.  var x = 9;  var y = 4;  document.write( (x>12 || y<8) ); //true  Incercati codul |
| **!** | - not, este operator unar, foloseste o singura expresie si returneaza TRUE daca expresia este falsa, daca expresia este adevarata returneaza FALSE.  var x = 9;  var y = 9;  document.write( !(x == y) ); //false  Incercati codul |

### Operator de concatenare pentru siruri

Operatorul de concatenare (unire) pentru siruri este caracterul plus (+). Acesta uneste doua siruri, iar rezultatul este tot un sir.  
- Exemplu:

let str1 ='Site: ';

let str2 ='MarPlo.net';

document.write(str1 + str2); //Site MarPlo.net

Incercati codul

Daca operatorul de concatenare se foloseste cu un numar si un sir, rezultatul e un sir cu cele doua valori unite.  
- Exemplu:

let x = 8; //number

let y ='9' //string

document.write(x + y); //89

Incercati codul

### Operatorul conditional (Ternary)

Operatorul conditional (denumit si Ternary) atribuie o valoare unei variabile pe baza unei conditii.  
Sintaxa de folosire

**variabila = (conditie) ? val1 :val2**

- Se evalueaza conditia, daca este adevarata atunci variabila ia valoarea 'val1', altfel ia valoarea 'val2'.  
Iata un exemplu:

let age = 17;

let copt = (age < 18) ? 'Tanar' :'Matur';

document.write(copt); // Tanar

Incercati codul

### Precedenta operatorilor

Cand in expresii se folosesc mai multi operatori, JavaScript tine cont de precedenta (importanta) predefinita a fiecarui operator.  
Precum in aritmetica, intr-o ecuatie cu adunare si inmultire ( 2+3\*4 ), daca nu sunt paranteze, se executa intai inmultirea, aceasta avand precedenta superioara fata de adunare. La fel e si cu operatorii in programare.  
Daca apar mai multi operatori cu aceeasi precedenta, JavaScript ii va evalua de la stanga spre dreapta.  
In tabelul urmator sunt prezentati operatorii in ordinea precedentei lor, unde operatorii din partea de sus a tabelului au precedenta maxima:

|  |  |
| --- | --- |
| **Operator** | **Nume operator** |
| **()** | de grupare |
| **!   ++   --** | de negare, incrementare, decrementare |
| **\*   /   %** | de inmultire, impartire |
| **+   -** | de adunare, scadere |
| **<   <=   >   >=** | de comparatie |
| **==   !=** | de egalitate |
| **&&** | SI logic |
| **||** | SAU logic |
| **? :** | conditional /Ternary |
| **=   +=   -=   \*=   /=   %=** | de atribuire |

# nstructiuni conditionale if() else, switch

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Instructiunea if().. else](https://marplo.net/javascript/instructiuni_conditionale.html" \l "hshie" \o "Instructiunea if.. else)
* [Instructiunea else if()](https://marplo.net/javascript/instructiuni_conditionale.html" \l "hshei" \o "Instructiunea else if())
* [Instructiunea switch](https://marplo.net/javascript/instructiuni_conditionale.html" \l "hshis" \o "Instructiunea switch)

Cu ajutorul instructiunilor conditionale putem face programele sa testeze diferite conditii dupa care sa decida datele care vor fi executate.  
In Javacript sunt urmatoarele instructiuni conditionale:

* **if()** - executa comenzile dorite cand o conditie este adevarata.
* **if().. else** - executa anumite comenzi cand o conditie este adevarata si alte comenzi cand aceasta este falsa.
* **if().. else if()** - executa comenzile dorite cand prima conditie e adevarata sau alte comenzi cand aceea e falsa si urmatoarea e adevarata.
* **switch().. case** - selecteaza care comanda va fi executata in functie de valoarea unei variabile.

### Instructiunea if()

Sintaxa:

**if(conditie){**

**//Codul care va fi executat daca este adevarata conditia**

**}**

- Unde 'conditie' poate fi orice expresie logica.  
Daca rezultatul conditiei este TRUE se executa codul dintre acolade, in caz contrar, cand conditia returneaza FALSE, se trece peste acest cod.  
  
- Exemplu, urmatorul script afiseaza "Buna ziua" daca ora este mai mare decat 11.

<script>

// daca ora > 11, va scrie in fereastra: Buna ziua

let ora = 12;

if(ora >11){

document.write('<h4>Buna ziua!</4>');

}

</script>

Incercati codul

### Instructiunea if().. else

In exemplul anterior am vazut ca este afisat "Buna ziua" daca ora > 11, iar in caz contrar nu se intampla nimic.  
Folosind instructiunea "**if().. else**" putem stabili comenzi care sa fie executate cand conditia instructiunii "if" este FALSE.  
  
Sintaxa:

**if(conditie){**

**//codul care va fi executat daca este adevarata conditia**

**}**

**else {**

**//codul care va fi executat daca conditia este falsa**

**}**

- Unde 'conditie' poate fi orice expresie logica.  
Daca rezultatul conditiei este TRUE se executa codul dintre primele acoladele, care apartin de "if", in caz contrar, cand conditia returneaza FALSE, sunt executate comenzile din a doua grupa de acolade (de la else).  
  
- Iata un exemplu practic. Urmatorul script afiseaza "Buna ziua" daca ora este mai mare decat 11, in caz contrar afiseaza 'Este ora ...'  
Se foloseste obiectul **Date()** care determina data si ora curenta (va fi explicat mai amanuntit in alta lectie).

<script>

// daca ora > 11, va scrie in fereastra: Buna ziua

// Alfel afiseaza: Este ora ...

var d = new Date();

let ora = d.getHours();

if(ora >11){

document.write('<h4>Buna ziua</h4>');

}

else {

document.write('<h4>Este ora '+ora+'</h4>');

}

</script>

Incercati codul

### Instructiunea else if()

Instructiunea **else if()** se adauga dupa ***if()***, si inainte de '***else***'.  
Aceasta instructiune se foloseste pentru a verifica o alta conditie dupa 'if()', ca sa fie rulat un alt cod cand conditia de la 'if()' este FALSE, iar conditia de la **else if()** este TRUE (adevarata).  
  
Sintaxa:

**if(conditie\_1){**

**//codul care va fi executat daca este adevarata conditia**

**}**

**else if(conditie\_2){**

**//cod executat cand conditie\_1 e FALSE iar conditie\_2 e adevarata**

**}**

**else {**

**//codul care va fi executat cand ambele: conditie\_1 si conditie\_2 sunt FALSE**

**}**

- Unde 'conditie\_1' si 'conditie\_2' poate fi orice expresie logica.  
Se evaluata intai 'conditie\_1', daca rezultatul este TRUE se executa codul de la ***if()***. Dar daca este FALSE, programul evalueaza urmatoarea conditie, de la ***else if()*** (conditie\_2). Daca aceasta este adevarata se executa codul dintre acoladele ei, dar daca si aceasta e FALSE, sunt executate comenzile de la '***else***'.  
  
- Exemplu, urmatorul script afiseaza "Buna dimineata" daca ora este mai mare decat 5 si mai mica decat 11, in caz contrar, daca ora este intre 11 si 19 afiseaza "Buna ziua", altfel, afiseaza: 'Este ora ...'

<script>

//daca ora e intre 5 si 11, va scrie: Buna dimineata

//daca ora e intre 11 si 19, va scrie: Buna ziua

//Alfel afiseaza: Este ora ...

var d = new Date();

let ora = d.getHours();

if(ora >5 && ora <11){

document.write('<h4>Buna dimineata</h4>');

}

else if(ora >=11 && ora <19){

document.write('<h4>Buna ziua</h4>');

}

else {

document.write('<h4>Este ora '+ora+'</h4>');

}

</script>

Incercati codul

*Dupa instructiunea****if()****se pot adauga mai multe serii de instructiuni****else if()****; iar '****else****' este optional.*

### Instructiunea switch

Instructiune **switch** e folosita pentru a compara o valoare cu o lista de alte valori.  
Sintaxa:

**switch(expresie) {**

**case valoare1:**

**cod executat daca expresie = valoare1**

**break;**

**case valoare2:**

**cod executat daca expresie = valoare2**

**break;**

**case valoare3:**

**cod executat daca expresie = valoare3**

**break;**

**default :**

**cod executat daca expresie e diferit de valoare1, valoare2 sau valoare3**

**}**

- Prima data este evaluata expresia scrisa intre paranteze rotunde (de la **switch()**), apoi valoarea expresiei este comparata pe rand cu fiecare valoare de la '**case**'. Daca se gaseste o identitate se executa codul asociat acelui '**case**'. Apoi se iese din instructiunea "switch".  
Daca, parcurgand fiecare "case", nu se gaseste nici o egalitate, se executa codul de la '**default**'.  
Prin folosirea lui '**break**' se opreste parcurgerea corpului instructiunii atunci cand s-a gasit o valoare egala cu 'expresie' si se iese din "switch".  
  
- Iata un exemplu de script care afiseaza un mesaj in functie de zilele saptamanii.

<script>

var d = new Date();

var ziua = d.getDay();

switch(ziua){

case 5:

document.write('Astazi e vineri');

break

case 6:

document.write('Astazi e sambata');

break

case 0:

document.write('Astazi e duminica');

break

default:

document.write('Mai e pana sambata');

}

</script>

Incercati codul

- Iata inca un exemplu cu "switch", de data aceasta foloseste valori de tip sir (string).

<script>

var nume ='Marius';

switch(nume){

case 'Cristi':

document.write('Coleg');

break

case 'Marius':

document.write('Frate');

break

case 'Maria':

document.write('Sora');

break

default:

document.write('Altcineva');

}

</script>

# Instructiuni repetitive for()

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Instructiunea for...in](https://marplo.net/javascript/instructiuni-repetitive-for" \l "hshifi" \o "Instructiunea for...in)
* [Instructiunea for...of](https://marplo.net/javascript/instructiuni-repetitive-for" \l "hshifo" \o "Instructiunea for...of)
* [forEach()](https://marplo.net/javascript/instructiuni-repetitive-for" \l "hshfe" \o "forEach())

Instructiunile repetitive se folosesc atunci cand se doreste efectuarea unei comenzi de mai multe ori.  
Instructiunile "for" executa codul de un numar specificat de ori.

## Instructiunea for

Instructiunea **for()** se foloseste pentru a rula un cod de un numar specifitat de repetari.  
Sintaxa:

**for(nr\_start; conditie\_nr, increment\_nr){**

**//cod care va fi executat**

**}**

- "**nr\_start**" e o variabila cu o valoare de inceput a numarului de repetari.  
- "**conditie\_nr**" o conditie (expresie logica) ce verifica daca numarul de repetari se incadreaza intr-o anumita valoare. Daca rezultatul este TRUE se executa inca o data codul dintre acolade.  
- "**increment\_nr** incrementeaza sau decrementeaza valoarea la care a ajuns contoarul repetarilor, apoi aceasta valoare este verificata din nou de "conditie\_nr" pana cand rezultatul este FALSE.

*Intre acoladele de la instructiunea****for****pot fi introduse si alte instructiuni "for", sau alte instructiuni conditionale, acest lucru se numeste****imbricarea instructiunilor****.*

Exemplu:

<script>

//repeta rularea unui cod de 5 ori (de la 0 la 4 inclusiv)

for(var i=0; i<5; i++){

document.write('<br>i este: '+i);

}

</script>

Incercati codul

Se atribue lui 'i' valoarea 0, se verifica conditia (i<5) care este adevarata si se executa corpul instructiunii (dintre acolade, care afiseaza "i este 1) apoi se incrementeaza valoarea lui 'i' cu o unitate (i++), acum 'i' are valoarea 2.  
Se verifica din nou conditia (i<5) si fiindca este adevarata, se executa iar corpul instructiunii apoi se incrementeaza inca o data valoarea lui 'i', ..., si tot asa pana cand 'i' va ajunge sa aibe valoarea 5 care la verificarea conditiei returneaza FALSE, moment in care se termina executia instructiunii "for".

### Instructiunea for...in

Cu '**for...in**' se parcurg proprietatile dintr-un obiect. Pentru fiecare proprietate se poate executa un set de instructiuni, daca obiectul nu are nici o proprietate nu se desfasoara nici un ciclu.  
Acest ciclu se poate executa cu orice obiect JavaScript (inclusiv array).

Sintaxa:

**for(variable in object){**

**//cod ce va fi executat**

**}**

- "***variable***" e o variabila care preia numele (sau index-ul) proprietatii curente din obiectul parcurs (object).  
Exemplu:

<script>

var obj = {nr:100, bool:false, site:'MarPlo.net'};

for(var prop in obj){

document.write('<br>'+ prop +' - '+ obj[prop]);

}

</script>

Incercati codul

### Instructiunea for...of

Cu '**for...of**' se parcurg proprietatile dintr-un obiect sau elementele dintr-un array.  
Diferenta fata de "for...in" e aceasta: la ***for...in*** se retine in 'variabila' numele proprietatii (sau indexul la array), iar la ***for...of*** se retine in 'variabila' valoarea proprietatii (sau a elementului din array).

Sintaxa:

**for(variable of object){**

**//cod ce va fi executat**

**}**

- "***variable***" e o variabila care preia valoarea proprietatii (sau pt. array, a elementului) curente din obiectul parcurs (object).  
Exemplu:

<script>

var arr =['MarPlo.net', 'GamV.eu', 80];

for(var elm of arr){

document.write('<br>'+ elm);

}

</script>

Incercati codul

### forEach()

**forEach()** este o metoda JavaScript care se foloseste pentru a parcurge elementele dintr-un Array.  
Sintaxa:

**Array.forEach(callbackF)**

- '***callbackF***' este o functie care va fi apelata pentru fiecare element din Array.  
Aceasta functie poate avea trei parametri: **callbackF(val, index, Array)**

* ***val*** - valoarea elementului curent.
* ***index*** - index-ul elementului curent.
* ***Array*** - array-ul care e parcurs (optional).

Exemplu, afiseaza index-ul si valoarea fiecarui element dintr-un Array.

<script>

var arr = [100, 'GamV.eu', 'MarPlo.net'];

//functia callback

function parseArr(val, index) {

document.write('['+ index +'] = '+ val +'<br>');

}

//parcurge array-ul 'arr', folosind functia parseArr()

arr.forEach(parseArr);

</script>

# Instructiuni repetitive while

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

## Instructiunea while

Instructiunea **while()** repeta un cod atata timp cat o conditie specificata este adevarata.  
Sintaxa:

**while(conditie){**

**//codul care va fi executat**

**}**

- Exemplu:

<script>

var x =0;

while(x <5){

document.write('<br>x este '+ x);

x++;

}

</script>

Incercati codul

- intai am declarat variabila 'x' cu valoarea 0.  
Instructiunea "while" verifica conditia (aici x<5) care este adevarata si permite executarea corpului functiei dintre acolade care afiseaza "x este 1" si incrementeaza valoarea lui 'x' cu o unitate (x++).  
Acum 'x' are valoarea 2, se verifica conditia, care este adevarata si se executa iar codul dintre acolade, ..., si tot asa pana cand la verificarea conditiei rezultatul este FALSE, moment in care se termina rularea instructiunii "while".

### Instructiunea do...while

**do..while()** este o varianta a instructiunii **while()**; dar, spre deosebire de aceasta, **do..while()** executa intai codul din corpul instructiunii (dintre acolade), dupa care verifica conditia, apoi il repeta pana cant conditia returneaza FALSE.  
Astfel, corpul functiei este executat cel putin o data, chiar daca conditia nu este adevarata.

Sintaxa:

**do {**

**//codul care va fi executat**

**} while(conditie)**

- Exemplu:

<script>

var x =8;

do{

document.write('<br>x este '+ x);

x++;

} while(x <5);

</script>

Incercati codul

- Aceasta functie afiseaza "x este 8".  
Observati ca desi conditia este falsa (x<5), codul dintre acolade este totusi executat o singura data.

# break, continue, si eticheta

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Instructiunea continue](https://marplo.net/javascript/break-continue-eticheta" \l "hshic" \o "Instructiunea continue)
* [Instructiunea eticheta (label)](https://marplo.net/javascript/break-continue-eticheta" \l "hshie" \o "Instructiunea eticheta (label))

Comenzile **break** si **continue** pot fi folosite in codul JavaScript la instructiuni repetitive **for()** si **while()** pentru a le opri sau sari peste repetari.

### Instructiunea break

Instructiunea **break** intrerupe executia instructiunilor repetitive.  
- Exemplu, rularea lui for() se va opri controlat cu "break":

<script>

//programat sa ruleze pana la 10

for(x=0; x<10; x++){

document.write('<br> X este '+x);

//opreste complet executia cand x este 3

if(x==3) break;

}

</script>

Incercati codul

Similar se poate folosi si cu while():

<script>

var x =0;

//programat sa ruleze pana l0

while(x <10){

document.write('<br> X este '+x);

//opreste complet executia cand x este 3

if(x==3) break;

x++;

}

</script>

Incercati codul

### Instructiunea continue

Instructiunea **continue** sare peste executia repetarii curente, si continua cu celelalte repetari din ciclu.  
- Exemplu, sare peste rularea codului din for() cand 'x' ajunge la valoarea 1 si 3:

<script>

//programat sa ruleze de la 0 la 5

for(x=0; x<5; x++){

//sare peste executia codului cand x este 1 sau 3

if(x ==1 || x ==3) continue;

document.write('<br> X este '+x);

}

</script>

Incercati codul

Similar se poate folosi si cu while():

<script>

var x =0;

//programat sa ruleze pana la 5

while(x <5){

//sare peste executia codului cand x este 1 sau 3

if(x ==1 || x ==3){

x++; //ca sa faca incrementare si cand se sare peste repetare

continue;

}

document.write('<br> X este '+x);

x++;

}

</script>

Incercati codul

### Instructiunea eticheta (label)

Instructiunea **eticheta** (denumita si label) poate fi folosita impreuna cu "**break**" sau "**continue**". Aceasta eticheteaza un grup de instructiuni repetitive imbricate, permitand controlul asupra acelui grup din interiorul instructiunilor imbricate.

Sintaxa:

**nume\_eticheta:**

**for(...){**

**//alte instructiuni for() sau while()**

**}**

- 'nume\_eticheta' poate fi orice nume (fara cele rezervate sintaxei de cod), dupa care se adauga doua-puncte si grupul de imbricari cu for() sau while().  
  
Studiind exemplul urmator, puteti intelege mai bine modul de utilizare a unei '***etichete***':

<script>

loopX: //eticheteaza urmatorul grup de instructiuni imbricate

for(var x=0; x<5; x++){

document.write('<h4>X - '+x+'</h4>');

for(var y=0; y<3; y++){

//opreste executia intregului grup cand x e 2

if(x ==2) break loopX;

document.write('<br>Y = '+y);

}

}

</script>

Incercati codul

- Testand codul, veti observa ca desi "break" este inclus in interiorul celei dea doua instructiune "for()", prin specificarea lui "loopX", care este eticheta intregului ansamblu de instructiuni imbricate, se va intrerupe executia intregului ansamblu, nu numai instructiunea "for()" unde este adaugat "break".

# Ferestre Alert Prompt si Confirm

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Fereastra Prompt](https://marplo.net/javascript/alert_prompt_confirm.html" \l "hshfp" \o "Fereastra Prompt)
* [Fereastra Confirm](https://marplo.net/javascript/alert_prompt_confirm.html" \l "hshfc" \o "Fereastra Confirm)

Alert, Prompt si Confirm sunt ferestre predefinite de dialog, acestea apartin direct obiectului JavaScript "window".

### Fereastra Alert

Probabil ati intalnit pana acum, de mai multe ori exemple cu "Alert".  
Fiind un element simplu si deja cunoscut, voi face o scurta prezentare a sintaxei acestuia.  
Sintaxa:

**window.alert('mesaj')**

- Unde "mesaj" este textul care va apare in fereastra Alert.

*Se poate folosi si direct:****alert('mesaj');****, fara 'window'.*

Urmatorul exemplu deschide o fereastra cu mesajul: "Bine ai venit".

Continut HTML

<script>

alert('Bine ai venit');

</script>

Incercati codul

In browser va apare o fereastra ca in imaginea urmatoare:

### Fereastra Prompt

**Fereastra Prompt** se creaza cu sintaxa:

**window.prompt('mesaj', 'default')**

- Unde "mesaj" este un text care va apare in fereastra, deasupra unei casute de text input; iar "default" este textul care va apare in casuta input.  
  
Urmatorul exemplu deschide o fereastra "Prompt".

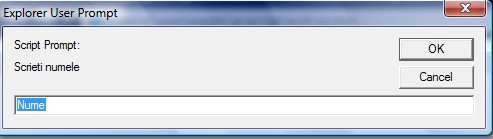
Continut HTML

<script>

window.prompt('Scrieti numele', 'Nume');

</script>

Incercati codul

In browser va apare o fereastra ca in imaginea urmatoare  
  
Textul pe care utilizatorul il adauga in campul din fereastra prompt poate fi preluat intr-o variabila si folosit apoi in script.  
Iata un exemplu in care atribuim deschiderea ferestrei Prompt unei variabile (*aici "nume"*), care va prelua sirul adaugat in caseta de text, apoi valoarea acestei variabile o introducem in mesajul unei ferestre Alert:

Continut HTML

<script>

var nume = window.prompt('Scrieti numele', 'Nume');

alert('Salut '+ nume +'\n Bine ai venit.');

</script>

Incercati codul

Dupa ce utilizatorul scrie numele in fereastra Prompt si apasa OK se va deschide o fereastra Alert care contine in mesaj numele adaugat de utilizator.  
- *Am folosit "\n" pentru a adauga o linie noua, dupa "nume", in textul care apare in fereastra Alert.*

### Fereastra Confirm

**Fereastra de confirmare** se creaza cu sintaxa:

**window.confirm('intrebare')**

In fereastra de confirmare va apare textul "intrebare" si doua butoane "*OK*" si "*Cancel*".  
Aceasta fereastra este folosita pentru a fi executata o comanda cand este apasat butonul "OK" (returneaza TRUE) si alta comanda cand este apasat butonul "Cancel" (returneaza FALSE)  
  
- Urmatorul exemplu deschide o fereastra "Confirm" in care apare intrebarea "*Rezultatul lui 0+0 este 0?*". Daca este apasat butonul "OK" apare o fereastra Alert cu mesajul "Corect", iar daca este apasat butonul "Cancel" apare o fereastra Alert cu mesajul "Incorect".

Continut HTML

<script>

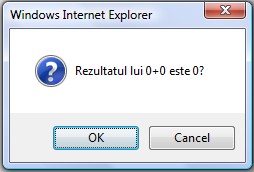
var intrebare = window.confirm('Rezultatul lui 0+0 este 0?');

if(intrebare) alert('Corect');

else alert('Incorect');

</script>

Incercati codul

In browser va apare o fereastra ca in imaginea urmatoare

# Definire si Utilizare Functii in JS

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Instructiunea return](https://marplo.net/javascript/definire-utilizare-functii" \l "hshir" \o "Instructiunea return)
* [Apelarea functiilor](https://marplo.net/javascript/definire-utilizare-functii" \l "hshap" \o "Apelarea functiilor)
* [Functii si domeniul variabilelor](https://marplo.net/javascript/definire-utilizare-functii" \l "hshfdd" \o "Functii si domeniul variabilelor)
* [Functii imbricate](https://marplo.net/javascript/definire-utilizare-functii" \l "hshfi" \o "Functii imbricate)
* [Functii Anonime](https://marplo.net/javascript/definire-utilizare-functii" \l "hshfa" \o "Functii Anonime)

O functie poate contine mai multe instructiuni si comenzi care ulterior pot fi utilizate usor si de mai multe ori prin apelarea functiei care le contine.  
Daca un program necesita multe linii de cod, folosind functiile putem imparti codul in parti mai mici pe care le putem utiliza separat acolo unde este nevoie.  
Exista **functii predefinite**, adica cele care apartin direct programului JavaScript (precum: alert(), parseInt(), trim(), etc.); si functii create in script (cele definite de programator).  
- In aceasta lectie este prezentat modul de creare si utilizare a functiilor in script.

### Creare (definire) functii

Exista mai multe moduri de a crea o functie in JavaScript.

1. ***Modul standard*** - folosind cuvantul **function** urmat de numele functiei si doua paranteze rotunde.  
   Sintaxa:
2. **function nameFun(p1, p2){**
3. **//codul care va fi executat**
4. **}**

- 'nameFun' reprezinta numele functiei. Poate fi orice nume ca si la variabile.  
- 'p1, p2' sunt parametri fuunctiei, si reprezinta valorile care trebuie sa le primeasca functia cand este apelata. Acestia sunt optionali (poate fi definita si fara parametri).  
Parametri sunt necesari cand se doreste transmiterea de date care sa fie utilizate in codul functiei. Se pot adauga unul sau mai multi parametri, cat sunt necesari.  
- Exemplu, fara parametru:

function fun1(){

alert('Love yourself; love everyone.');

}

1. ***Functii atribuite la variabile*** - se adauga ca valori la variabile, si se apeleaza cu numele variabilei.  
   Sintaxa:
2. **var nameVar = function(p1, p1){**
3. **//codul care va fi executat**
4. **}**

- 'nameVar' e numele variabilei dar si al functiei utilizat la apelarea ei.  
- 'p1, p2' sunt parametri functiei (optionali).  
- Exemplu:

var fun1 = function(){

alert('Love yourself; love everyone.');

}

*Diferenta fata da modul standard e aceasta: o functie definita in mod standard nu mai poate fi redefinita, dar o functie atribuita unei variabile poate fi redefinita deoarece reprezinta o valoare a variabilei iar valorile de variabile pot fi schimbate.*

1. ***Arrow functions*** - functii definite fara cuvantul "function"; se foloseste o prescurtare: "()=>".  
   Diferit la acestea e faptul ca nu au propriul **this** (care face referire la obiectul functiei).  
   Sintaxa:
2. **var nameVar = (p1, p1)=>{**
3. **//codul care va fi executat**
4. **}**

- 'p1, p2' sunt parametri functiei (optionali).  
- Exemplu:

var fun1 = ()=>{

alert('Love yourself; love everyone.');

}

#### Instructiunea return

O functie care returneaza un rezultat foloseste pentru aceasta instructiunea **return**. Aceasta specifica valoarea pe care o returneaza functia cand este apelata.

Sintaxa:

function nameFun(p1, p2)

//codul care va fi executat

return 'Valoare';

*Instructiunea****return****se poate aplica in oricare tip de functie; la cele atribuite ca valori de variabile, si la 'arrow functions'.  
Pe langa faptul ca 'return' returneaza o valoare, acesta opreste executia functiei; codul din functie adaugat dupa 'return' nu este executat.*

- Exemplu:

//returneaza suma parametrilor a si b

function fun2(a, b){

return a + b;

}

### Apelarea functiilor

Dupa ce am creat o functie, pentru a fi folosita, functia trebuie **apelata**.

O functie fara parametri se apeleaza in felul urmator:

**nume\_functie();**

- Exemplu, se defineste o functie care adauga un text intr-un element HTML:

<h3 id='hid'>Default text</h3>

<script>

//definim o functie care scrie un text HTML in elementul cu id #hid

function textHid(){

document.getElementById('hid').innerHTML ='<h3>Mereu e o zi frumoasa.</h3>';

}

//se apeleaza functia

textHid();

</script>

Incercati codul

O functie care contine parametri se apeleaza in felul urmator:

**nume\_functie(v1, v2);**

- 'v1, v2' sunt valorile care se transmit functiei si sunt asociate in aceeasi ordine cu parametri adaugati la definirea ei.  
  
O functie o data creata poate fi utilizata (apelata) de mai multe ori.  
Iata un exemplu cu o functie care adauga in elementul HTML cu id-ul transmis un text adaugat la apelare.

<h3 id='hid1'>Tag H3</h3>

<div id='dv1'>Continut Div</div>

<script>

//adauga in elementul HTML cu id-ul de la 'id' textul de la 'str'

function addText(id, str){

document.getElementById(id).innerHTML = str;

}

//se apeleaza functia de 2 ori, cu valori diferite

addText('hid1', 'Tutorial JavaScript');

addText('dv1', 'Text adaugat prin apelarea functiei.');

</script>

Incercati codul

- Iata si un exemplu cu **return**:

<h3 id='sum'>Tag H3</h3>

<script>

//returneaza un text cu suma lui x si y

function sumXY(x, y){

let re = x+y;

return 'Suma lui x si y este: '+ re;

}

//preia valoarea returnata de sumXY

let sum1 = sumXY(7, 12);

//adauga in elementul HTML cu id 'sum' valoarea de la sum1

document.getElementById('sum').innerHTML = sum1;

</script>

Incercati codul

### Functii si domeniul variabilelor

Functiile si variabilele definite in afara unei functii pot fi utilizate in interiorul ei.  
Valoare unei variabile definite in afara functiei poate fi modificata in codul ei, iar modificare ramane si in afara functiei dupa apelarea ei.

- Exemplu, se creaza o functie si o variabila care vor fi apelate in cadrul altei functii:

<div id='dv1'>Continut Div</div><br>

<div id='dv2'>Arata nr</div>

<script>

var nr =8;

//dubleaza valoare lui x

function setVal(x){

return x\*2;

}

//adauga valoarea returnata de setVal() in elementul cu id-ul 'id'

//apoi modifica valoarea variabilei 'nr'

function showVal(id){

document.getElementById(id).innerHTML ='Valoare lui "nr" dublat este: '+ setVal(nr);

nr =99;

}

//apeleaza showVal() cu id-ul pt. parametru

showVal('dv1');

//adauga valoare curenta a lui 'nr' in Div-ul #dv2

document.getElementById('dv2').innerHTML ='Valoare lui "nr" dupa modificare in functie este: '+nr;

</script>

Incercati codul

Valoarea unei variabile creata in afara functiei nu e afectata daca in interiorul unei functii se defineste (cu 'let' sau 'var') o variabila cu acelasi nume.  
- Exemplu:

<div id='dv1'>Div-ul #dv1</div><br>

<div id='dv2'>Div-ul #dv2</div>

<script>

var nr =8;

//defineste in functie o variabila 'nr' cu valoarea parametrului

function f1(x){

var nr = x;

//adauga valoarea lui 'nr' in elementul #dv1

document.getElementById('dv1').innerHTML ='Valoare lui "nr" din functie este: '+nr;

}

//apeleaza f1() cu valoarea pt. 'nr' din functie

f1(5);

//adauga valoare lui 'nr' in Div-ul #dv2. 'nr' extern nu a fost modificat

document.getElementById('dv2').innerHTML ='Valoare lui "nr" extern este: '+nr;

</script>

Incercati codul

### Functii imbricate

In cadrul unei functii se poate crea si apela alta functie.  
Functiile si variabilele create direct intr-o functie nu pot fi utilizate in afara ei

- Exemplu, apelare functie imbricata:

<div id='dv1'>Continut HTML</div>

<script>

//o functie definita standard in care se creaza si se apeleaza un 'arrow function'

function f1(x){

//arrow function imbricata

let f2 =(x)=>{

return 'X dublat este: '+ (x\*2);

}

//daca x < 4 adauga in #dv1 rezultat returnat de f2()

if(x <4) document.getElementById('dv1').innerHTML = f2(x);

}

f1(3);

//daca se incearca apelarea lui f2(), apare eroare in consola din browser

f2(3); //ReferenceError: f2 is not defined

</script>

Incercati codul

### Functii Anonime

Functiile anonime nu au un nume, ele nu pot fi apelate, ci se creaza direct ca argument transmis la apelarea altei functii.  
Functiile anonime pot fi definite standard (utilizand cuvantul 'function') sau cu modul 'arrow function'.

- Exemplu, functia predefinita **setTimeout()** din JavaScript necesita doua argumente la apelare, primul e o functie, al doilea e o valoare numeria (milisecunde).

<div id='dv1'>Div-ul #dv1</div><br>

<div id='dv2'>Div-ul #dv2</div>

<script>

//primul argument transmis e o functie anonima, care va fi executata de setTimeout() dupa 2 secunde

window.setTimeout(

function(){

document.getElementById('dv1').innerHTML ='<h3>Viata e Fericire; Bucurati-va.</h3>';

},

2000);

//primul argument e o functie anonima (arrow function), executata de setTimeout() dupa 3 secunde

window.setTimeout(

()=>{

document.getElementById('dv2').innerHTML ='<h3>Fiecare zi e placuta; Bucurati-va.</h3>';

},

3000);

</script>

# Utilizare Functii si Parametri lor

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Definire parametri cu valoare implicita](https://marplo.net/javascript/functii2.html" \l "hshdpi" \o "Definire parametri cu valoare implicita)
* [Proprietatea arguments](https://marplo.net/javascript/functii2.html" \l "hshpa" \o "Proprietatea arguments)
* [Functia callback](https://marplo.net/javascript/functii2.html" \l "hshfc" \o "Functia callback)
* [Functii recursive](https://marplo.net/javascript/functii2.html" \l "hshfr" \o "Functii recursive)

### Apelare functie cu numar diferit de argumente

Cand se creaza o functie ii putem defini si un numar de parametri (sau argumente) pe care le accepta, lucru de care se tine cont in momentul apelarii ei.  
Sunt situatii in care dorim sa transmitem functiei un numar diferit de argumente, care poate fi mai mic sau mai mare decat numarul de parametri ai functiei.  
Intr-o astfel de situatie se verifica in corpul functiei daca a fost transmisa valoare pentru parametri definiti, daca nu, acel parametru e **undefined** (**evaluat ca False**) si se poate da o valoare implicita (default).

- Exemplu, o functie cu un parametru care returneaza un salut cu numele transmis ca argument. Daca acesta nu e transmis, seteaza o valoare implicita.

<div id='dv1'>Content dv1</div>

<div id='dv2'>Content dv2</div>

<script>

function sayHi(name){

//daca nu e primita valoare pt. 'name', seteaza un text implicit

if(!name) var name ='Vizitator';

return 'Salut '+ name;

}

//adauga in #dv1 rezultatul transmis de apelarea functiei fara argument

document.getElementById('dv1').innerHTML = sayHi();

//a doua apelare, cu argument

document.getElementById('dv2').innerHTML = sayHi('MarPlo');

</script>

Incercati codul

### Definire parametri cu valoare implicita

La crearea unei functii se pot defini parametri cu o valoare implicita, astfel se evita necesitatea verificarii daca argumentul e transmis la apelare.  
Sintaxa:

**function numeF(p1=v1, p2=v2){**

**//codul care va fi executat**

**}**

- 'p1' si 'p2' sunt parametrii; iar 'v1' si 'v2' reprezinta valorile implicite pentru fiecare.  
  
Cand functia este apelata fara argumentul respectiv, parametru va avea valoarea data la definire, altfel preia valoarea transmisa.  
Iata exemplu prezentat anterior, dar aici cu valoare implicita definita la parametru.

<div id='dv1'>Div dv1</div>

<div id='dv2'>Div dv2</div>

<script>

//creaza functia cu o valoare default pt. parametru

function sayHi(name='Vizitator'){

return 'Salut '+ name;

}

//adauga in #dv1 rezultatul transmis de apelarea functiei fara argument

document.getElementById('dv1').innerHTML = sayHi();

//a doua apelare, cu argument

document.getElementById('dv2').innerHTML = sayHi('MarPlo');

</script>

Incercati codul

### Proprietatea arguments

Proprietatea **arguments** retine intr-un obiect tip array argumentele (valorile) transmise la apelarea unei functii, in ordinea adaugarii lor (cu primul index 0).  
Aceasta proprietate nu poate fi aplicata la '***arrow functions***'.  
Valoarea ei nu este un array, ci un obiect cu indexi numerici. Dar i-se poate aplcica **length** pentru aflarea numarului de elemente.

Sintaxa:

**function numeF(){**

**var args = numeF.arguments;**

**//codul care va fi executat**

**}**

- Variabila 'args' contine obiectul cu argumentele transmise la apelarea functiei numeF().  
- Aici, primul argument poate fi preluat cu: ***args[0]***  
  
Aceasta proprietate este utila cand o functie se apeleaza cu un numar de argumente mai mare decat sunt la definirea ei.  
- Exemplu, o functie cu un parametru. Daca e apelata fara argument, seteaza valoarea 0; daca e un singur argument, il dubleaza; daca sunt mai multe le aduna.

<div id='dv1'>Div-ul #dv1</div>

<div id='dv2'>Div-ul #dv2</div>

<div id='dv3'>Div-ul #dv3</div>

<script>

const f1 = function(x){

//retine obiectul-ul cu argumentele transmise la apelare f1()

var args = f1.arguments;

//daca nu e nici un argument, seteaza valoarea 0 pt. 're'

//daca e un singur argument, il dubleaza

//daca sunt mai multe, le aduna

if(args.length ==0) var re =0;

else if(args.length ==1) var re = args[0]\*2;

else if(args.length >1){

var re =0;

for(var i=0; i<args.length; i++) re += args[i];

}

return re;

}

//adauga in elemente HTML rezultatul apelarii lui f1() de trei ori, cu nr. diferit de argumente

document.getElementById('dv1').innerHTML ='f1() fara argument returneaza: '+ f1();

document.getElementById('dv2').innerHTML ='f1(3) returneaza: '+ f1(3);

document.getElementById('dv3').innerHTML ='f1(2, 5, 3) returneaza: '+ f1(2, 5, 3);

</script>

Incercati codul

### Functia callback

Functia callback e o functie transmisa ca argument la alta functie, unde va fi apelata cu numele parametrului la care e asociata, si argumentele necesare.  
- Se poate intelege mai bine din urmatorul exemplu:

<div id='dv1'>Callback function.</div>

<script>

//functia callback, adauga 'Salut name' in #dv1

const fcb = (name)=>{

document.getElementById('dv1').innerHTML ='Salut '+ name;

}

//functia la care va fi transmisa si utilizata functia callback

function f1(id, callback){

if(id >1) callback('MarPlo.net');

}

//apeleaza f1() cu un Id si functia callback fcb()

f1(8, fcb);

</script>

Incercati codul

Adesea se folosesc ***functiile anonime*** ca functii callback. Functiile anonime sunt cele care se definesc direct ca argument la apelarea altei functii.  
- Iata exemplu anterior, aici cu functie callback anonima.

<div id='dv1'>Anonimous callback function.</div>

<script>

//functia la care va fi transmisa si utilizata functia callback

function f1(id, callback){

if(id >1) callback('GamV.eu');

}

//apeleaza f1() cu un Id si functie callback anonima

f1(8, (name)=>{

document.getElementById('dv1').innerHTML ='Salut '+ name;

});

</script>

Incercati codul

### Functii recursive

O functie JavaScript poate fi recursiva, adica se poate autoapela.  
- Exemplu, o functie JavaScript recursiva care afla factorialul unui numar precizat "n".

<div id='dv1'>Exemplu functie recursiva</div>

<script>

//returneaza factorial de n

function factorial(n){

var re;

if(n >0) re = n\*factorial(n-1); //auto-apelare

else if(n ==0) re =1;

return re;

}

//afisaza in #dv1 factorial de 8

document.getElementById('dv1').innerHTML ='Factorial de 8 este: '+ factorial(8);

</script>

Incercati codul

Functia factorial() verifica intai daca "n" este mai mare decat 0, apoi in caz afirmativ, "n" se inmulteste cu rezultatul returnat de auto-apelarea ei cu argumentul "n-1". Cand "n" ajunge 0 se incheie repetarea auto-apelarii si returneaza valoare lui 're'.  
JavaScript incheie toate auto-apelarile in corpul functiei si retine de la o auto-apelare la urmatoarea ultimul rezultat returnat pana ce ajunge la apelarea initiala a functiei, apoi intoarce rezultatul final.

# Obiectul String - Sir

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Utilizare Proprietate si Metode ale obiectului String](https://marplo.net/javascript/obiectul-string" \l "hshpms" \o "Utilizare Proprietate si Metode ale obiectului String)
* [Adaugare ghilimele si caractere speciale in sir](https://marplo.net/javascript/obiectul-string" \l "hshgcs" \o "Adaugare ghilimele si caractere speciale in sir)
* [Sir Template](https://marplo.net/javascript/obiectul-string" \l "hshst" \o "Sir Template)

Toate elementele dintr-o pagina sunt vazute de JavaScript ca fiind obiecte.  
Obiectele sunt structuri compacte de date care pot sa contina mai multe proprietati si functii (denumite Metode).  
Apelarea proprietatilor si metodelor unui obiect se face cu operatorul punct (.)

- Sintaxa:

**obiect.prorietate**

**obiect.metoda()**

## Obiectul string

**String** (sau sir) e obiectul care contine text adaugat intre ghilimele simple sau duble.  
Un obiect de tip String se poate crea folosind ghilimele sau cu instructiunea **new String()**.

- Sintaxa:

**//definire String cu new**

**var str = new String('text..');**

**//definire String direct cu ghilimele**

**var str ='Text..';**

**//sau**

**var str ="Text..";**

• In general se folosesc direct ghilimele, iar toate valorile incadrate intre ghilimele sunt vazute in JavaScript ca 'String' si pot folosi proprietatile si metodele obiectului String.

### Utilizare Proprietate si Metode ale obiectului String

Obiectul String are o singura proprietate: **length**, aceasta contine numarul de caractere din sir.

let str ='Tutorial JS';

let nr\_chr = str.length; // 11

Se poate citi orice caracter din sir folosind numarul de ordine (indexul) care incepe de la 0. Primul caracter are index 0 (**str[0]**), al doilea are index 1 (**str[1]**), etc.  
  
- Exemplu, se afiseaza intr-un element HTML numarul de caractere dintr-un sir, si al treilea caracter.

<div id='dv1'>HTML coontent.</div>

<script>

let str ='some\_str';

//afiseaza in #dv1 numarul de caractere din 'str' si al treilea caracter

document.getElementById('dv1').innerHTML ='Sirul "some\_str" are '+ str.length +' caractere; al treilea e: '+ str[2];

</script>

Incercati codul

#### Utilizare Metode

Obiectul String contine numeroase metode care pot fi utile in lucru cu siruri. O lista cu metode pentru siruri gasiti la pagina: **[Metode ale obiectului String in JS](https://marplo.net/javascript/metode-obiect-string-js" \o "Metode ale obiectului String in JS)**.  
- Iata exemple cu unele din cele mai folosite metode pentru siruri.

#### Exemplu cu indexOf()

Metoda **indexOf()** e utila pentru a verifica daca un sir contine un anumit subsir.  
Aceasta returneaza pozitia primei aparitii a unui subsir intr-un sir, sau valoarea -1 daca subsirul nu e gasit.  
- Exemplu, daca sirul de la variabila 'str' contine cuvantul 'spirit', afiseaza indexul de inceput a subsirului.

<div id='dv1'>Exemplu cu indexOf().</div>

<script>

let str ='In spiritul libertatii.';

//daca str contine 'spirit', adauga indexul de inceput in #dv1

var ix = str.indexOf('spirit');

if(ix !=-1){

document.getElementById('dv1').innerHTML ='Cuvantul "spirit" incepe de la index: '+ ix;

}

</script>

Incercati codul

#### Exemplu cu match()

Metoda **match()** e utila pentru a compara un sir cu o expresie regulata (RegExp). Returneaza un array cu elementele din sir care se potrivesc cu expresia regulata, sau 'null'.  
- Exemplu, se verifica daca un sir e o adresa valida de email si preia numele din adresa.

<div id='dv1'>Exemplu JavaScript cu match().</div>

<script>

const email ='some\_name@domain.net';

//is\_em e un array de forma ['email', 'nume', 'domeniu'] sau null

var is\_em = email.match(/^([A-z0-9]+[A-z0-9.\_%-]\*)@([A-z0-9\_\-\.]+\.[A-z]{2,5})$/i);

//daca la email e o adresa valida de email, adauga in #dv1 numele din adresa

if(is\_em){

document.getElementById('dv1').innerHTML ='Numele din adresa "'+email+'" este: '+ is\_em[1];

}

</script>

Incercati codul

#### Inlantuire Metode

Se pot aplica mai multe metode (inlantuite) la un sir, in aceeasi linie de cod.  
**str.metoda1().metoda2()**  
  
- Exemplu cu **replace()** si **trim()**.  
Metoda **replace()** inlocuieste intr-un sir unele caractere specificate (sau expresie RegExp) cu altele noi specificate.  
Metoda **trim()** sterge spatiile goale de la inceputul si sfarsitul unui sir.

In urmatorul exemplu se inlocuiesc intr-un sir cu cod HTML caracterele '<' si '>' cu echivalentele pentru afisare in HTML, si se adauga sirul rezultat intr-un element HTML.

<div id='dv1'>Exemplu JavaScript cu metode inlantuite.</div>

<script>

let htm ='<h3>Astazi e o zi Fericita</h3>';

//inlocuieste < si > din htm cu entitatile html, aplica trim() si adauga sirul in #dv1

htm = htm.replace(/</ig, '&lt;').replace(/>/ig, '&gt;').trim();

document.getElementById('dv1').innerHTML = htm;

</script>

Incercati codul

### Adaugare ghilimele si caractere speciale in sir

La adaugarea de ghilimele in textul dintr-un String se tine cont de ghilimelele de inceput (prin care e definit sirul).  
Ghilimelele duble se pot adauga normal intr-un sir definit cu ghilimele simple (si invers).  
- Sintaxa:

**var str ='Un text cu "ghilimele" duble';**

**var str ="Un text cu 'ghilimele' simple";**

Acelasi tip de ghilimele cu cele folosite la definirea sirului se pot adauga doar escapate (precedate de '\').  
- Sintaxa:

**var str ='Un text cu \'ghilimele simple\' ca cele la definire';**

**var str ="Alt text cu \"ghilimele duble\", ca si cele de definire";**

In afara de ghilimele sunt si alte caractere speciale care trebuie escapate cand se folosesc in sir.  
Iata o lista cu aceste caractere:

* **\0** - caracterul NULL.
* **\'** - ghilimea simpla (in sir definit intre ghilimele simple).
* **\"** - ghilimea dubla (in sir definit intre ghilimele duble).
* **\\** - backslash.
* **\n** - linie noua.
* **\r** - carriage return.
* **\v** - tab vertical.
* **\t** - spatiu tab.
* **\b** - backspace.
* **\uXXXX** - cod unicode.

- Exemplu, se adauga intr-un tag <pre> un sir cu mai multe linii, care contine ghilimele si backslash.

<pre id='pr1'>HTML content, object string.</pre>

<script>

let str ='Sir definit intre \'ghilimele simple\', \n adaugate cu backslash \\, \n dar contine si "ghilimele duble".';

document.getElementById('pr1').innerHTML = str;

</script>

Incercati codul

### Sir Template

**Sirul Template se adauga intre caracterele de accent ( `...` )**.  
Acest tip de String permite crearea de siruri cu linii noi adaugate normal (fara \n) si redarea de expresii de cod JavaScript care sunt executate direct in el.  
- Sintaxa:

**var tmp =`string text line 1**

**Line 2 si ${js\_expression} expresie de cod.`;**

- ***js\_expression*** poate fi o variabila JavaScript, apelarea unei functii sau o expresie JS cu valori. Acestea se adauga intre: ***${}*** si sunt executate ca orice cod JavaScript.  
  
Exemplu, un sir ce contine mai multe linii si cod html cu continut dinamic.

<pre id='pr1'>Template string, HTML content</pre>

<script>

//se definesc variabile si o functie (arrow) care vor fi utilizate in sirul template

let str\_st ='Sir Template';

let arr =[8, 0];

const sumXY =(x, y)=>(x+y);

//sirul template

var tmp =`<h3>Acesta e ${str\_st} din JavaScript</h3>

Apelare functie sumXY(): suma 7+8 = ${sumXY(7,8)}

Si o linie cu <strong>expresie JS, 8\*0 = ${arr[0] \* arr[1]}</strong>.`;

//adauga sirur rezultat in #pr1

document.getElementById('pr1').innerHTML = tmp;

</script>

# Numere in JavaScript

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Functii pentru obiectul Number](https://marplo.net/javascript/numere-js" \l "hshmn" \o "Functii pentru obiectul Number)

Operatiile aritmetice simple cu numere se pot efectua cu operatorii specifici: scadere (-), adunare (+), inmultire (\*), impartire (/) si modul (%) restul impartirii. Vedeti lectia de la adresa: **[marplo.net/javascript/operatori.html](https://marplo.net/javascript/operatori.html" \o "Operatori in JavaScript)**  
  
- Exemplu, se afla suma a trei numere, apoi restul impartirii la 3.

const n1 =2;

const n2 =9;

//suma

let sum\_n = n1 +n2 +15; // 26

//restul impartirii la 3

let rest3 = sum\_n %3; // 2

document.write('Suma (n1+n2+15) = '+sum\_n+'<br>Restul sumei impartit la 3 este '+rest3);

Incercati codul

### Siruri numerice

Cu numerele adaugate intre ghilimele nu se pot efectua corect operatiuni matematice, valoarea dintre ghilimele e considerata ca tip **String** (sir).

- Exemplu, se incearca adunarea unui sir numeric cu un numar, rezulta un sir cu ele unite:

let n1 ='21';

let n2 =9;

//suma

let sum\_n = n1 +n2; //219

document.write("Suma ('21'+9) ="+sum\_n);

Incercati codul

Inainte de a efectua operatiuni matematice cu numere de tip sir, acestea trebuie transformate in tip **Number**.  
Modul cel mai simplu de a transforma un sir numeric in numar e: inmultind sirul cu valoarea 1, sau cu functia **Number()**.  
- Exemplu, se transforma un sir numeric in numar, apoi se aduna:

let n1 ='21';

let n2 =9;

n1 = n1\*1; //transforma in numar

let sum\_n = n1 +n2; //30

document.write('Suma (21+9) = '+sum\_n);

Incercati codul

Acelasi exemplu, aplicand functia Number():

let n1 ='21';

let n2 =9;

n1 = Number(n1); //transforma in numar

let sum\_n = n1 +n2; //30

document.write('Suma (21+9) = '+sum\_n);

Incercati codul

### Functii pentru obiectul Number

Numerele fara zecimale sunt considerate intregi (Integer), iar cele cu zecimale (adaugate dupa punct) sunt considerate de tip Float.  
Numerele negative se adauga cu semnul minus (-) in fata.  
Obiectul Number are cateva metode pentru numere.

* **Number.isInteger(nr)** - returneaza True daca 'nr' e numar intreg, altfel False.
* var n1 = 90;
* var n2 = -25;
* var n3 = 90.23;
* var n4 ='23';
* console.log(Number.isInteger(n1)); // true
* console.log(Number.isInteger(n2)); // true
* console.log(Number.isInteger(n3)); // false
* console.log(Number.isInteger(n4)); // false

Incercati codul

* **Number(sn)** - transforma un sir numeric 'sn' in numar. Returneaza NaN (Not a Number) daca sirul nu e numeric.
* var n1 ='57.98';
* var n2 ='23 str';
* console.log(Number(n1)); // 57.98
* console.log(Number(n2)); // NaN

Incercati codul

* **parseFloat(sn)** - returneaza un numar cu zecimale dintr-un sir numeric 'sn' cu zecimale.
* var nr ='13.56';
* nr = parseFloat(nr); //numar 13.56
* var sum\_n = nr +8.2;
* document.write('- Suma: nr +8 = '+sum\_n); //21.759999999999998

Incercati codul

* **parseInt(sn)** - returneaza un numar intreg (primul) dintr-un sir numeric 'sn'.
* var n1 ='25.89';
* var n2 ='-34.8 23';
* console.log(parseInt(n1)); // 25
* console.log(parseInt(n2)); // -34

Incercati codul

* **nr.toFixed(d)** - returneaza un sir cu numarul 'nr' formatat cu valoarea rotunjita la numarul de zecimale specificat 'd'.  
  - Aceasta metoda se aplica la numere, nu la 'siruri numerice'.
* var n1 = 92;
* var n2 = -25.5689;
* var n3 = 90.2378;
* console.log(n1.toFixed(2)); // 92.00
* console.log(n2.toFixed(1)); // -25.6
* console.log(n3.toFixed(2)); // 90.24

Incercati codul

* **nr.toPrecision(n)** - returneaza un sir cu numarul 'nr' avand:  
  - pentru 'nr'>1, lungimea de cifre specificata 'n'.:  
  - pentru 'nr'<1, lungimea de zecimale specificata 'n'.
* var n1 = 24;
* var n2 = -3.23;
* var n3 = 0.46;
* console.log(n1.toPrecision(3)); // 24.0
* console.log(n2.toFixed(1)); // -3.2
* console.log(n3.toFixed(3)); // 0.460

Incercati codul

* **nr.toString()** - returneaza un Sir cu numarul de la 'nr'.
* var n1 = 12.5;
* var sum\_n = n1.toString() +3;
* var ex2 = (12 + 9).toString();
* console.log(sum\_n); // 12.53
* console.log(ex2); // 21

Incercati codul

#### Functia Number() cu obiectul Date

Functia **Number()** aplicata cu obiectul Date returneaza numarul de milisecunde incepand de la 1.1.1970 pana la data /timpul din obiectul Date.

var dt1 = new Date('2018-07-15');

var dt2 = new Date('2018-07-15 11:13:00');

console.log(Number(dt1)); // 1531612800000

console.log(Number(dt2)); // 1531642380000

# Obiectul Math - Metode pentru operatiuni matematice

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Proprietati ale obiectului Math](https://marplo.net/javascript/obiect-math-operatiuni-matematice" \l "hshpom" \o "Proprietati ale obiectului Math)
* [Metode ale obiectului Math](https://marplo.net/javascript/obiect-math-operatiuni-matematice" \l "hshmom" \o "Metode ale obiectului Math)

Obiectul **Math** contine proprietati si metode pentru constante si operatiuni matematice.  
Nu este nevoie sa fie creat in script un obiect Math inainte de a fi folosit, acesta se aplica direct, urmat de proprietate sau metoda.

- Exemplu:

const x = Math.PI; //valoarea constantei PI

var y = Math.sqrt(16); //radical din 16

- Exemplu util, se creaza o functie care returneaza un numar aleator intre doua numere intregi.

<script>

//returneaza un numar aleator intre min si max (inclusive)

function randomInt(min, max){

min = Math.ceil(min);

max = Math.floor(max);

return Math.floor(Math.random() \*(max - min + 1)) +min;

}

var nr1 = randomInt(1, 10); //numar intre 1 si 10

var nr2 = randomInt(-10, 20); //numar intre -10 si 20

document.write('<br> Numar aleator intre 1 si 10: '+ nr1 +'<br> Numar aleator intre -10 si 20: '+ nr2);

</script>

Incercati codul

#### Extindere obiect Math

Obiectul Math poate fi extins, se pot adauga direct in el noi proprietati si metode, folosind sintaxa:

**//definire proprietate**

**Math.prop\_name = prop\_val;**

**//definire metoda**

**Math.method\_name = function(){**

**//codul metodei**

**}**

- Exemplu, se creaza la obiectul Math o metoda care returneaza cel mai mare divizor comun la o lista de numere.

<script>

//returneaza cel mai mare divizor comun la o lista de numere transmise ca argumente

Math.cmmdc = function() {

if(arguments.length ==2){

if(arguments[1] ==0) return arguments[0];

else return Math.cmmdc(arguments[1], arguments[0] % arguments[1]);

}

else if(arguments.length >2){

var result = Math.cmmdc(arguments[0], arguments[1]);

for(var i=2; i<arguments.length; i++) result = Math.cmmdc(result, arguments[i]);

return result;

}

};

let mdv = Math.cmmdc(20, 30, 15, 70, 40); // 5

document.write('<br> Cel mai mare divizor comun al numerelor: (20, 30, 15, 70, 40) este: '+ mdv);

</script>

Incercati codul

### Proprietati ale obiectului Math

Proprietatile obiectului Math reprezinta constante matematice.

* **Math.E** - constanta lui Euler (aproximativ 2.718)
* **Math.LN2** - logaritm natural din 2 (aproximativ 0.693)
* **Math.LN10** - logaritm natural din 10 (aproximativ 2.303)
* **Math.LOG2E** - logaritm in baza 2 din E (aproximativ 1.443)
* **Math.LOG10E** - logaritm in baza 10 din E (aproximativ 0.434)
* **Math.PI** - constanta PI (aproximativ 3.14159)
* **Math.SQRT1\_2** - radical din 0.5 (aproximativ 0.707)
* **Math.SQRT2** - radical din 2 (aproximativ 1.414)

### Metode ale obiectului Math

* **Math.abs(x)** - returneaza valoarea absoluta (pozitiva) a numarului x.
* var n1 = 18;
* var n2 = -25.3;
* console.log(Math.abs(n1)); // 18
* console.log(Math.abs(n2)); // 25.3

Incercati codul

* **Math.acos(x)** - returneaza arccosinus din x.
* var n1 = -1;
* var n2 = 0.5;
* console.log(Math.acos(n1)); // 3.141592653589793
* console.log(Math.acos(n2)); // 1.0471975511965979

Incercati codul

* **Math.asin(x)** - returneaza arcsinus din x.
* var n1 = -1;
* var n2 = 0.5;
* console.log(Math.asin(n1)); // -1.5707963267948966 (-pi/2)
* console.log(Math.asin(n2)); // 0.5235987755982989

Incercati codul

* **Math.atan(x)** - returneaza arctangenta din x.
* var n1 = 1;
* var n2 = 0.5;
* console.log(Math.atan(n1)); // 0.7853981633974483
* console.log(Math.atan(n2)); // 0.4636476090008061

Incercati codul

* **Math.atan2(y, x)** - returneaza unghiul dintre axa si un punct de coordonate (x,y).
* var v1 = Math.atan2(90, 15);
* var v2 = Math.atan2(15, 90);
* console.log(v1); // 1.4056476493802699
* console.log(v2); // 0.16514867741462683

Incercati codul

* **Math.cbrt(x)** - returneaza radacina cubica (radical de ordin 3) a numarului x.
* var n1 = 64;
* console.log(Math.cbrt(n1)); // 4

Incercati codul

* **Math.ceil(x)** - returneaza cel mai apropiat intreg mai mare sau egal cu x.
* var n1 = 18.7;
* var n2 = -25.3;
* console.log(Math.ceil(n1)); // 19
* console.log(Math.ceil(n2)); // -25

Incercati codul

* **Math.cos(x)** - returneaza cosinus din x.
* var n1 = 0;
* var n2 = 1;
* console.log(Math.cos(n1)); // 1
* console.log(Math.cos(n2)); // 0.5403023058681398

Incercati codul

* **Math.exp(x)** - returneaza valoarea lui E la puterea x.
* var n1 = 2;
* var n2 = -1;
* console.log(Math.exp(n1)); // 7.38905609893065
* console.log(Math.exp(n2)); // 0.36787944117144233

Incercati codul

* **Math.floor(x)** - returneaza cel mai apropiat intreg mai mic sau egal cu x.
* var n1 = 18.7;
* var n2 = -25.3;
* console.log(Math.floor(n1)); // 18
* console.log(Math.floor(n2)); // -26

Incercati codul

* **Math.hypot(x1, x2, ...)** - returneaza radicalul sumei patratelor numerelor x1, x2, ...
* var n1 = 3;
* var n2 = 4;
* console.log(Math.hypot(n1, n2)); // 5

Incercati codul

* **Math.log(x)** - returneaza logaritm natural din x.
* var n1 = 1;
* var n2 = 10;
* console.log(Math.log(n1)); // 0
* console.log(Math.log(n2)); // 2.302585092994046

Incercati codul

* **Math.log10(x)** - returneaza logaritm in baza 10 din x.
* var n1 = 100000;
* var n2 = 2;
* console.log(Math.log10(n1)); // 5
* console.log(Math.log10(n2)); // 0.3010299956639812

Incercati codul

* **Math.max(x1, x2, ...)** - returneaza maximul dintre x1, x2, ...
* var n1 = 61.5;
* var n2 = -65.3;
* var n3 = 53;
* console.log(Math.max(n1, n2, n3)); // 61.5

Incercati codul

* **Math.min(x1, x2, ...)** - returneaza minimul dintre x1, x2, ...
* var n1 = 61.5;
* var n2 = -65.3;
* var n3 = 53;
* console.log(Math.min(n1, n2, n3)); // -65.3

Incercati codul

* **Math.pow(x, y)** - returneaza valoarea lui x la puterea y.
* var x = 4;
* var y = 3;
* console.log(Math.pow(x, y)); // 64

Incercati codul

* **Math.random()** - returneaza un numar aleator intre 0 si 1.
* document.write('<br> - Numar aleator intre 0 si 1: '+ Math.random());

Incercati codul

* **Math.round(x)** - rotunjeste pe x la cel mai apropiat intreg.
* var n1 = 5.8;
* var n2 = -25.38;
* console.log(Math.round(n1)); // 6
* console.log(Math.round(n2)); // -25

Incercati codul

* **Math.sin(x)** - returneaza sinus din x.
* var n1 = 0;
* var n2 = 1;
* var n3 = Math.PI /2;
* console.log(Math.sin(n1)); // 0
* console.log(Math.sin(n2)); // 0.8414709848078965
* console.log(Math.sin(n3)); // 1

Incercati codul

* **Math.sqrt(x)** - returneaza radical din x.
* var n1 = 4;
* var n2 = 81.25;
* console.log(Math.sqrt(n1)); // 2
* console.log(Math.sqrt(n2)); // 9.013878188659973

Incercati codul

* **Math.tan(x)** - returneaza tangenta din x.
* var n1 = 1;
* var n2 = 90;
* console.log(Math.tan(n1)); // 1.5574077246549023
* console.log(Math.tan(n2)); // -1.995200412208242

Incercati codul

* **Math.trunc(x)** - returneaza partea intreaga a numarului x (eliminand zecimalele).
* var n1 = 18.78;
* var n2 = -25.3;
* console.log(Math.trunc(n1)); // 18

console.log(Math.trunc(n2)); // -25

# Obiectul Array

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Accesare element din Array](https://marplo.net/javascript/obiectul-array" \l "hshaea" \o "Accesare element din Array)
* [Adaugare si Modificare elemente in Array](https://marplo.net/javascript/obiectul-array" \l "hshama" \o "Adaugare si Modificare elemente in Array)
* [Parcurgere Array](https://marplo.net/javascript/obiectul-array" \l "hshpa" \o "Parcurgere Array)
* [Lucru cu elemente din Array](https://marplo.net/javascript/obiectul-array" \l "hshlea" \o "Lucru cu elemente din Array)
* [Array Multidimensional](https://marplo.net/javascript/obiectul-array" \l "hsham" \o "Array Multidimensional)

## Definire si Utilizare Array

Array este un obiect care poate sa contina mai multe valori intr-o singura variabila. Aceste valori sunt ordonate cu un index numeric incepand de la 0.  
Un Array se poate crea adaugand valorile intre paranteze patrate, sau cu formula: **new Array()**.  
Valorile se pot adauga si dupa definirea array-ului.

- Sintaxa cu paranteze patrate:

**let nume\_array =['v1', 'v2', 'v3'];**

**//sau**

**let nume\_array =[];**

**nume\_array[0] ='v1';**

**nume\_array[1] ='v2';**

**nume\_array[2] ='v3';**

- Sintaxa cu 'new':

**let nume\_array = new Array('v1', 'v2', 'v3');**

**//sau**

**let nume\_array = new Array();**

**nume\_array[0] ='v1';**

**nume\_array[1] ='v2';**

**nume\_array[2] ='v3';**

- 'nume\_array' poate fi orice nume de variabila.  
- valorile adaugate pot fi orice tip de valoare: string (sir), numar, boolean (true /false), obiect (chiar si un alt array).

### Accesare element din Array

Fiecare valoare adaugata intr-un Array devine un element al Array-ului si are asociat un index numeric. Cu acest index se poate accesa oricare element din array.

- Sintaxa:

**nume\_array[index]**

Indexul incepe de la 0. Primul element are index 0, al doilea are index 1, al treilea index 2, etc.  
  
- Exemplu, se defineste un Array cu trei elemente si se scrie in pagina valoarea de la al doilea element (cu index 1).

<script>

//se defineste un array cu 3 elemente si se acceseaza elementul al doilea

let arr =['HTML', 'JavaScript', 890];

var v2 = arr[1];

document.write(v2); //JavaScript

</script>

Incercati codul

### Adaugare si Modificare elemente in Array

Dupa ce a fost creat un array, se poat adauga elemente noi in el sau modifica cele deja adaugate; folosind acceeasi formula ca la accesare, cu indexul intre paranteze patrate.  
- Sintaxa:

**nume\_array[index] ='valoare';**

Un mod simplu de a adauga un nou element la sfarsitul unui array (dupa ultimul adaugat) e cu metoda **push()**, aceasta adauga unul sau mai multe elemente la sfarsitul unui array.  
- Sintaxa:

**var nume\_array =['v1'];**

**nume\_array.push('v2', 'v3');**

Numarul de elemente dintr-un array se poate afla cu proprietatea **length**.  
Indexul ultimului element se poate afla cu: **(array.length -1)**.  
  
- Exemplu, la un Array cu doua elemente se modifica valoarea de la al doilea element si se mai adauga unul.

<script>

//array cu 2 elemente

let arr =['HTML', 'PHP'];

//modifica al doilea element

arr[1] ='Tutorial JS';

//adauga un nou element dupa ultimul

arr.push('New item');

//test, arr are acum 3 elemente

document.write('<br>Array-ul arr are '+ arr.length +' elemente. Al doilea e: '+ arr[1]);

</script>

Incercati codul

### Parcurgere Array

Un Array poate fi parcurs cu instructiune **for()** sau cu metoda **forEach()**.  
- Exemplu cu for(), parcurge si afiseaza vloarea elementelor dintr-un array.

<script>

//array cu 4 elemente

let arr =['HTML', 'CSS', 'JavaScript', 'PHP'];

//parcurge array-ul si scrie valoarea fiecarui element

for(var i=0; i<arr.length; i++){

document.write('<br>'+ arr[i]);

}

</script>

Incercati codul

Metoda **forEach()** preia ca argument o functie 'callback' cu 3 argumente.  
- Sintaxa:

**array.forEach(callbackF)**

- 'callbackF' e o functie care se executa pt. fiecare element, si poate avea trei argumente.

**callbackF(val, index, array)**

* val - valoarea elementului curent.
* index - indexul elementului
* array - (optional) array-ul parcurs.

- Exemplu cu forEach(), parcurge si afiseaza indexul si vloarea elementelor dintr-un array.

<script>

//functia callback pt. forEach()

let parseArr =(v, i)=>{

//scrie indexul si valoarea elementului

document.write('<br>'+ i +' - '+ arr[i]);

}

//array cu 4 elemente

let arr =['HTML', 'CSS', 'JavaScript', 'PHP'];

//parcurge array-ul cu forEach() si functia parseArr()

arr.forEach(parseArr);

</script>

Incercati codul

*Ca performanta,****for()****e mai rapid decat****forEach()****.  
- Un array se poate parcurge si cu instructiunile:****for..in****si****for..of****, vedeti lectia de la: [marplo.net/javascript/instructiuni-repetitive-for](https://marplo.net/javascript/instructiuni-repetitive-for" \o "Instructiuni repetitive for())*

### Lucru cu elemente din Array

JavaScript contine numeroase metode pentru lucru cu elementele unui array. O lista cu aceste metode gasiti la pagina de la adresa: **[marplo.net/javascript/metode-obiect-array](https://marplo.net/javascript/metode-obiect-array" \o "Metode ale obiectului Array in JS)**  
Iata cateva exemple.

- Verificare daca o variabila e array. Se aplica metoda: **Array.isArray()**.

let arr =['MarPlo.net', 'GamV.eu', 90];

//daca arr e array, afiseaza primul element

if(Array.isArray(arr)) document.write('- Primul elm.: '+arr[0]);

Incercati codul

- Aranjare array in ordine alfabetica. Se aplica metoda **sort()**.

let arr =['ef', 'a8', 9, 78];

arr.sort(); //aranjeaza in ordine alfabetica

console.log(arr); //[78, 9, 'a8', 'ef']

Incercati codul

- Suma si Produsul elementelor dintr-un array. Se aplica metoda **reduce()** cu o functie callback.

const arr =[1, 2, 3, 4];

//suma elementelor din arr

const ar\_sum = arr.reduce((a, b)=>a+b);

//produsul elementelor din arr

const ar\_prd = arr.reduce((a, b)=>a\*b);

document.write('<p>Suma numerelor din [1, 2, 3, 4] e: '+ar\_sum+', Produsul: '+ar\_prd+'</p>');

Incercati codul

### Array Multidimensional

La elementele unui array se poate adauga ca valoare un alt array.  
Elementele array-ului interior se acceseaza ierarhic incepand de la array-ul principal.  
- Sintaxa:

**array[index\_array\_intern][index\_elm]**

- Exemplu cu un array Bidimensional:

<div id='dv1'>JS, array Bidimensional</div>

<script>

//se defineste un array bidimensional (elementul al doilea e un array)

let arr =[

'Web develop',

['html', 'css', 'javascript'],

'xyz'

];

//indexul ultimului element din array-ul interior (care e la arr[1])

let ix = arr[1].length -1;

//preia valoarea primului element din 'arr' si a ultimului din array interior

let title = arr[0];

let tutor = arr[1][ix];

//adauga in #dv1 valorile preluate

document.getElementById('dv1').innerHTML ='<h3>'+ title +'</h3>Tutorial '+ tutor;

</script>

# Obiectul Date - Lucru cu Data si Timp

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Obiect Date cu data si ora specifice](https://marplo.net/javascript/obiect-date-timp" \l "hshods" \o "Obiect Date cu data si ora specifice)

Obiectul **Date** se foloseste pentru a lucra cu data zilei si timpul (ora).

1. Data initiala 0 (de referinta) este 1-01-1970. Pentru date anterioare are valori negative.
2. JavaScript retine data si timpul ca un numar de milisecunde incepand de la 1-01-1970.
3. Cand creati un obiect "Date", ora folosita de obiect este aceea de pe calculatorul client (al vizitatorului).

## Creare obiect Date

Data si ora curenta pot fi retinute intr-un obiect Date in JavaScript folosind sintaxa:

**var dt = new Date();**

- 'dt' poate fi orice nume de variabila.  
Iata ce contine acest obiect; testati urmatorul cod:

var dt = new Date();

document.write(dt);

//similar cu: Mon Jul 09 2018 17:54:20 GMT+0300 (GTB Daylight Time)

Incercati codul

Dupa ce a fost creat obiectul, se pot folosi metodele acestuia pentru a lucra cu componentele lui referitoare la an, luna, zi, ora, minute, secunde.  
Iata doua exemple.  
  
- Preia numarul de milisecunde (cu **getTime()**) din data curenta si afiseaza timpul Timestamp, adica numarul de secunde de la 1-01-1970 pana acum.

//defineste un obiect Date cu timpul curent

const dt = new Date();

//preia numarul de milisecunde, apoi secundele

let mils = dt.getTime();

let sec = Math.trunc(mils /1000);

//scrie numarul Timestamp

document.write('<p>Numar de secunde de la 1-01-1970 pana acum: '+ sec +'</p>');

Incercati codul

- Preia anul (cu **getFullYear()**), luna (cu **getMonth()**) si numarul curent al zilei (cu **getDate()**), apoi le adauga intr-un element HTML in pagina (prima luna are valoarea 0, ultima are valoarea 11).

<p id='dv1'>Anul, luna si numarul silei.</p>

<script>

//array cu numele lunilor

const ar\_month =['Ianuarie', 'Februarie', 'Martie', 'Aprilie', 'Mai', 'Iunie', 'Iulie', 'August', 'Septembrie', 'Octombrie', 'Noiembrie', 'Decembrie'];

//defineste un obiect Date cu timpul curent

var dt = new Date();

//preia anul, numarul lunii si al zilei din luna

let year = dt.getFullYear();

let month = dt.getMonth();

let day = dt.getDate();

//adauga in #dv1 luna curenta din 'ar\_month' si numarul zilei

document.getElementById('dv1').innerHTML ='Data curenta: '+ year +'-'+ ar\_month[month] +'-'+ day;

</script>

Incercati codul

## Obiect Date cu data si ora specifice

Pentru alte date si ore specifice (fixe), un obiect Date se poate crea in 3 moduri:

**new Date(year, month, day, hours, minutes, seconds, milliseconds)**

**new Date(milliseconds)**

**new Date('date string')**

- In continuare e prezentat pe rand fiecare mod, cu exemple.

### new Date(year, month, ...)

Formula **new Date(year, month, ...)** creaza un obiect Date cu data si timp specific (fix). Poate primi 7 argumente (numere intregi), ultimele 5 sunt optionale.  
Sintaxa:

**new Date(year, month, day, hours, minutes, seconds, milliseconds)**

* ***year*** - anul, un numar cu 4 sa 2 cifre. Cu doua caractere reprezinta anul intre 1900 si 1999 (ex.: 84 = 1984).
* ***month*** - numarul lunii din an, incepand cu 0 pt. Ianuarie pana la 11 pt. Decembrie.
* ***day*** - (optional) numarul zilei din luna.
* ***hour*** - (optional) ora.
* ***minutes*** - (optional) minute.
* ***seconds*** - (optional) secunde.
* ***milliseconds*** - (optional) milisecunde.

- Exemple:

var dt = new Date(2018, 7, 21, 11, 35, 20); //Reprezinta: 21 August 2018, Ora 11:35:20

var dt = new Date(98, 0, 8, 9); //Reprezinta: 8 Ianuarie 1998, Ora 9:00:00

var dt = new Date(2019, 11, 25); //Reprezinta: 25 Decembrie 2019

- Exemplu practic. Preia numarul de milisecunde dintr-un obiect cu data fixa si afiseaza timpul Timestamp (numarul de secunde de la 1-01-1970 pana la acea data si ora).

//defineste un obiect Date pt. data 23 Martie 2018, Ora 14:35

const dt = new Date(2018, 2, 23, 14, 35);

//preia numarul de milisecunde, apoi secundele

let mils = dt.getTime();

let sec = Math.trunc(mils /1000); // 1521808500

//scrie numarul Timestamp

document.write('<p>Numar de secunde de la 1-01-1970 pana la 23-Martie-2018, 14:35 este: '+ sec +'</p>');

Incercati codul

### new Date(milliseconds)

Formula **new Date(milliseconds)** creaza un obiect Date cu **timpul 0 (1970-01-1) plus numarul de milisecunde specificat**.  
Daca numarul de milisecunde e negativ, data e anterioara anului 1970.  
- Exemple:

//obiect Date cu numar de secunde ori 1000 (pt. milisecunde)

const dt = new Date(1521808500 \*1000);

document.write('<p>'+ dt +'</p>');

// Fri Mar 23 2018 14:35:00 GMT+0200 (GTB Standard Time)

Incercati codul

Cu valoare negativa.

//obiect Date cu numar de secunde ori 1000 (pt. milisecunde)

const dt = new Date(-152180850 \*1000);

document.write('<p>'+ dt +'</p>');

// Sat Mar 06 1965 17:32:30 GMT+0200 (GTB Standard Time)

Incercati codul

### new Date('date string')

Formula **new Date('date string')** creaza un obiect Date dintr-un sir cu data si ora specificate literar.  
Sirul de data poatea avea unul din aceste formate:

**new Date('Month DD, YYYY hh:mm:ss'); // Aug 05, 2018 09:25:30**

**new Date('Month DD YYYY'); // Aug 05 2018**

**new Date('DD Month YYYY'); // 05 Aug 2018**

**new Date('YYYY-MM-DD'); // 2018-08-05**

**new Date('YYYY-MM-DDThh:mm:ss'); // 2018-08-05T09:25:30**

* **Month** - numele lunii in engleza (january, february, march, ...), poate fi si prescurtat cu primele trei litere (Jan, Feb, Mar).
* **YYYY** - reprezinta anul. Daca e cu doua caractere (YY) reprezinta anul intre 1900 si 1999.
* **MM** - numarul lunii din an, indicat cu doua caractere: 01 (pt. Ianuarie), 02 (pt. Februarie), ... 12 (Decembrie).
* **DD** - numarul zilei din luna, 1...31
* **hh** - ora.
* **mm** - minute.
* **ss** - secunde.
* **T** - se adauga ca separator de Timp intre data si ora (ex.: '2018-12-17T03:24:00').

- Exemplu, preia si afiseaza numarul de secunde dintre data curenta si un sir cu o alta data si ora.

//obiect cu data curenta

const dt1 = new Date();

//defineste un obiect Date pt. data 23 August 2018, Ora 14:35

const dt2 = new Date('2018-08-25T14:35:00');

//preia diferenta de milisecunde dintre dt1 si dt2

let mils = dt1.getTime() - dt2.getTime();

//retine secundele

let sec = Math.trunc(mils /1000);

//scrie numarul Timestamp

document.write('<p>Numar de secunde de la 23-August-2018, 14:35 pana acum: '+ sec +'</p>');

Incercati codul

Obiectul Date contine metode pentru preluarea componentelor din data (an, luna, zi, etc.), dar si pentru adaugare /setare.  
O lista cu metode ale obiectului Date gasiti la pagina: **[marplo.net/javascript/metode-obiect-date](https://marplo.net/javascript/metode-obiect-date" \o "Metode ale obiectului Date)**  
  
- Exemplu, adauga o ora si 30 minute la data curenta, apoi preia fi afiseaza intr-un element HTML numele zilei din saptamana si ora.

<p id='p1'>Example Date objet in JavaScript</p>

<script>

//array cu zilele din saptamana

const days =['Duminica', 'Luni', 'Marti', 'Miercuri', 'Joi', 'Vineri', 'Sambata'];

var ad\_m = 90\*60\*1000; //o ora si 30 minute in milisecunde

const dt = new Date();

//preia milisecundele din data curenta si adauga ad\_m

let mils = dt.getTime();

dt.setTime(mils +ad\_m);

//preia ziua din saptamana, ora, minute, secunde

let wday = dt.getDay();

let hour = dt.getHours();

let mins = dt.getMinutes();

let sec = dt.getSeconds();

//afiseaza ora in #p1

document.getElementById('p1').innerHTML ='Ziua si ora peste o ora si 30 minute: <strong>'+ days[wday] +', '+ hour +':'+ mins +':'+ sec +'</strong>';

</script>

# Obiectul window

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Ferestre Pop-up](https://marplo.net/javascript/windows.html" \l "hshfp" \o "Ferestre Pop-up)
* [Bara de derulare - scroll](https://marplo.net/javascript/windows.html" \l "hshbs" \o "Bara de derulare - scroll)
* [Salvare date in browser](https://marplo.net/javascript/windows.html" \l "hshsb" \o "Salvare date in browser)

Obiectul **window** reprezinta fereastra din browser in care e afisat continutul.  
Obiectul **window** se creaza automat pentru fiecare fereastra deschisa in browser, inclusiv pentru elemente <iframe> din pagina. Acesta contine proprietati si metode pentru lucru cu elemente ale ferestrei si ale paginii web.  
- O lista cu proprietati si metode gasiti la pagina:  
**[marplo.net/javascript/proprietati-metode-window](https://marplo.net/javascript/proprietati-metode-window" \o "Proprietati si Metode obiect window)**

#### Cateva exemple

Proprietatea **parent** face referire la fereastra parinte (daca acel continut e intr-un <iframe>).

<h4>Exemplu parent</h4>

<p>La clic pe buton modifica culoarea background a ferestrei parinte, in care e aceasta fereastra (iframe).</p>

<button id='btn1'>Parent background</button>

<script>

document.getElementById('btn1').addEventListener('click', (ev)=>{

parent.document.body.style.background ='#bfbffe';

});

</script>

Incercati codul

Proprietatea **self** face referire la fereastra curenta.  
- Cu ea se poate verifica daca continutul e intr-un <iframe> sau in fereastra principala.Proprietatea **top** face referire la fereastra primara din browser.**outerHeight** - returneaza inaltimea ferestrei (incluzand toolbar), in pixeli.**outerWidth** - returneaza lungimea ferestrei (in pixeli).

### Ferestre Pop-up

Ferestrele Pop-up se pot crea din JavaScript cu dimensiuni si pozitie pe ecran stabilite de dv., folosind aceasta sintaxa

**window.open('URL', 'name', 'props')**

- 'URL' - adresa pentru continutul paginii din fereastra. Daca se adauga un sir gol, va afisa o pagina goala.  
- 'name' - numele ferestrei.  
- 'props' - (optional) perechi ***nume=valoare*** (separate prin virgula) cu proprietati pentru definirea ferestrei.

* ***height*** - inaltimea ferestrei (in pixeli).
* ***left*** - distanta ferestrei fata de marginea din stanga ecranului (in pixeli).
* ***menubar*** - daca sa afiseze sau nu bara de meniu, folosind valoarea 1 sau 0.
* ***titlebar*** - daca sa afiseze sau nu bara de titlu, folosind valoarea 1 sau 0.
* ***top*** - distanta ferestrei fata de marginea de sus a ecranului (in pixeli).
* ***width*** - lungimea ferestrei (in pixeli).

Daca argumentul 'props' nu e adaugat, se va deschide fereastra ca un nou tab in browser.  
  
- Exemplu, la clic pe un buton se deschide o fereastra cu dimensiuni si pozitie specificate.

<h4>Exemplu window.open()</h4>

<p>La clic pe buton se deschide o fereastra pop-up, cu aceste proprietati: <em>'width=650, height=450, left=30, top=50, menubar=0, titlebar=0'</em>.</p>

<button id='btn1'>Open pop-up</button>

<script>

document.getElementById('btn1').addEventListener('click', (ev)=>{

var popup = window.open('//gamv.eu/', 'TestWin', 'width=650, height=450, left=30, top=50, menubar=0, titlebar=0');

});

</script>

Incercati codul

#### Ferestre cu continut din JavaScript

Metoda **window.open()** returneaza o referinta la fereastra deschisa cu ea. Folosind aceasta referinta, se pot deschide ferestre pop-up cu continut creat direct din JavaScript. In acest caz, la URL se adauga un sir gol.  
In fereastra pop-up se poate adauga buton pentru inchiderea ferestrei cu metoda **close()**.

- Exemplu, la clic pe un buton se deschide o fereastra fara adresa URL, cu continut adaugat din JavaScript; si cu buton de inchidere.

<h4>Exemplu fereastra cu continut din JS si buton Close</h4>

<p>La clic pe buton deschide o fereastra pop-up in care se adauga elemente HTML din JavaScript si un buton Close.</p>

<button id='btn1'>Open pop-up</button>

<script>

document.getElementById('btn1').addEventListener('click', (ev)=>{

//deschide fereastra pop-up

var popup = window.open('', 'TestWindow', 'width=350, height=300, left=20, top=25');

//adauga html si text in ea

popup.document.write('<h1>Bucura-te, fii Fericit</h1>');

popup.document.write(`<blockquote style="color:#0000e0; font-size:18px;">Cea mai pretioasa avere pe care o ai, nimeni nu ti-o poate lua, o poti darui si transmite fara sa scada;<br>

Esti tu insuti, starea ta de a fi.<br>

Daruieste Bucurie si Pace prin starea ta,<br>

Ce si cum daruiesti, tu primesti.</blockquote>

<h4>Transmite Pace si Bucurie, fii Fericit!</h4>`);

popup.document.write('<p> - Nu spun ce sa faci, lumea te reflecta, Tu iti transmiti.</p>');

//adauga buton de inchidere

popup.document.write('<button onclick="window.close()">Close</button>');

//seteaza un background la Body din pagina deschisa

popup.document.body.style.background ='#fefee9';

});

</script>

Incercati codul

### Bara de derulare - scroll

Bara de derulare a paginii (scroll) apartine obiectului **window**. Acesta contine proprietati si metode pentru citirea si setarea pozitiei barelor de derulare a paginii.

* **scrollX** - returneaza pozitia barei orizontale de derulare fata de marginea din stanga (in pixeli).
* **scrollY** - returneaza pozitia barei verticale de derulare fata de marginea de sus (in pixeli).
* **scrollBy(x, y)** - deruleaza continutul cu numarul de pixeli specificati la x si y (orizontal, vertical).
* **scrollTo(x, y)** - deruleaza continutul (barele de scroll) la pozitiile specificate la x si y (orizontal, vertical).

• La obiectul window se poate inregistra evenimentul '**scroll**' prin care se poate detecta cand se deruleaza pagina.  
  
- Exemplu, buton de derulare cu scrollTo() si afisare coordonate ale barelor de derulare.

<h4>Exemplu scrollTo()</h4>

<p style='background:#fdfddb; height:1500px; width:130%;'>La click pe buton, bara de derulare orizontala se va muta la pozitia 300, iar cea verticala se va derula in jos la pozitia 1100 (pixeli).<br>

Se afiseaza in buton coordonatele barelor de derulare, x si y (cu evenimentul 'scroll').<br>

- In partea de jos e un buton pentru scroll-top.</p>

<button id='btn1' style='position:fixed;left:20%; top:170px;'>Scroll</button>

<button onclick='window.scrollTo(0,0)' style='display:block; margin:5px 25px 8px auto;'>Go-Top</button>

<script>

var btn = document.getElementById('btn1');

btn.addEventListener('click', (ev)=>{

window.scrollTo(300, 1100);

});

//detecteaza cand se face scroll la pagina

window.addEventListener('scroll', (ev)=>{

//afiseaza in btn coordonatele barelor de derulare

btn.textContent = window.scrollX +', '+ window.scrollY;

});

</script>

Incercati codul

### Salvare date in browser

Cu proprietatea **localStorage** din obiectul window se pot salva date in browser-ul utilizatorului pe timp nelimitat, pana cand acestea sunt sterse de utilizator sau de alt script.  
Datele stocate in browser cu **localStorage** sunt salvate ca ***string*** (siruri), si pot fi accesate in site-ul din care au fost adaugate de fiecare data cand utilizatorul viziteaza acel site.  
- Aceasta proprietate returneaza un obiect cu metode pentru adaugare si preluare date stocate in browser.

* **localStorage.setItem('key', 'val');** - stocheaza in browser valoarea 'val' ca sir asociat la numele 'key'
* **localStorage.getItem('key');** - returneaza sirul retinut la 'key'
* **localStorage.removeItem('key');** - sterge din localStorage elementul cu numele 'key'

- Exemplu, la clic pe buton salveaza in browser textul adaugat in caseta de text. Iar la click pe alt buton afiseaza din localStorage textul retinut in browser.

<h4>Exemplu localStorage</h4>

<p>La clic pe buton salveaza in browser textul adaugat in caseta de text.<br>

La click pe urmatorul buton care apare, afiseaza din localStorage textul retinut in browser.<br>

- Daca in browser sunt deja salvate date de pe acest site la 'some\_key' in localStorage, le afiseaza in elementul #resp.</p>

<div id='dv1'>

Text: <input type='text' value='marplo.net' id='txt1'><br>

<button id='btn1'>Add in storage</button>

</div>

<button id='btn2' style='display:none'>Get from storage</button>

<blockquote id='resp'>#resp</blockquote>

<script>

//verifica daca sunt date in localStorage la 'some\_key'

var ls\_val = localStorage.getItem('some\_key');

if(ls\_val) document.getElementById('resp').innerHTML ='Din localStorage:<br>'+ ls\_val;

//salveaza in browser valoarea de la #txt1, cu cheie 'some\_key'

document.getElementById('btn1').addEventListener('click', (ev)=>{

var str = document.getElementById('txt1').value;

localStorage.setItem('some\_key', str);

//ascunde #dv1 si afiseaza al doilea buton

document.getElementById('dv1').style.display ='none';

# JavaScript HTML DOM - Obiectul document

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Utilizare proprietati din obiectul document](https://marplo.net/javascript/dom-html-obiect-document" \l "hshpd" \o "Utilizare proprietati din obiectul document)
* [Utilizare metode din obiectul document](https://marplo.net/javascript/dom-html-obiect-document" \l "hshmd" \o "Utilizare metode din obiectul document)
* [Lucru cu Proprietati si Metode ale elementelor HTML in JS](https://marplo.net/javascript/dom-html-obiect-document" \l "hshpmh" \o "Lucru cu Proprietati si Metode ale elementelor HTML in JS)

In JavaScript toate elementele dintr-o pagina web sunt considerate obiecte, si sunt organizate in mod ierarhic, pe sistemul Parinte-Copil.  
Parinte este obiectul care contine alte obiecte, iar cele din interiorul lui sunt obiecte Copil. Acestea la randul lor pot contine alte elemente si astfel devin Parinti pentru acelea.  
Aceasta structura ierarhica, de continere pe nivele, e cunoscuta ca DOM (Document Object Model), unde fieare obiect poate fi accesat in mod ierarhic de la primul nivel pana la el, separate cu punct ( ***parinte.copil*** ).  
Obiectul **window** e considerat pe primul nivel, acesta contine elementele /obiectele ferestrei: **Navigator, Location, History, Document**.  
Obiectul **document** contine elementele HTML din pagina, tag-urile, proprietatile CSS, atributele si textul, dupa cum se poate vedea in urmatoarea imagine.  


## Obiectul document

**Obiectul document** reprezinta pagina web, in el se gasesc elementele (tag-urile HTML), atributele si continutul dintr-o pagina web.  
Acesta contine proprietati si metode prin care se pot adauga, accesa, edita, sterge in mod dinamic elemente si continut in pagina.  
Document apartine obiectului **window** si se poate accesa prin: **window.document**; dar fiindca "window" este obiectul principal, poate fi omis cand se folosesc proprietati si metode ale obiectului "document", si pot fi apelate folosind urmatoarea sintaxa.

**document.nume\_proprietate  
document.numeMetoda()**

- De exemplu: **document.write('String')** scrie sirul 'String' in pagina.

***O lista cu proprietati si metode utile ale obiecului document gasiti la pagina:  
[marplo.net/javascript/proprietati-metode-obiect-document](https://marplo.net/javascript/proprietati-metode-obiect-document" \o "Proprietati si Metode din obiectul document)***

### Utilizare proprietati din obiectul document

Obiectul **document** contine numeroase proprietati pentru accesarea elementelor din pagina. Majoritatea sunt doar pentru citirea elementelor (Read only), dar sunt cateva si pentru scriere.  
Iata cateva exemple.

• **document.URL** - returneaza un sir cu adresa paginii.

<script>

var url = document.URL;

document.write('URL:<br>'+ url);

</script>

Incercati codul

• **document.title** - returneaza si modifica titlul paginii (din tag-ul <title>).

<script>

var title = document.title;

document.write('Titlu:<br>'+ title);

//modifica titlul (se vede la titlu din browser)

document.title ='Pacea-i buna';

</script>

Proprietatile pot sa contina la randul lor obiecte care au propriile lor proprietati si metode.

• De exemplu: **document.links** - contine un array cu toate link-urile (tag-uri <a>) din pagina, sub forma de obiecte. Acestea au proprietati specifice elementelor HTML, care reprezinta partile componente (atributele, textul).  
Urmatorul exemplu afiseaza adresa 'href' a link-urilor din pagina.

<a href='//marplo.net' title='MarPlo.net'>MarPlo.net</a>

<ul><li><a href='//gamv.eu' title='GamV.eu'>GamV.eu</a></li></ul>

<div><a href='//coursesweb.net' title='CoursesWeb.net'>CoursesWeb.net</a></div>

<hr>

<script>

var alnk = document.links; //array cu link-urile din pagina

//parcurge array-ul si afiseaza valoarea href de la fieare

for(var i=0; i<alnk.length; i++) document.write('<br>'+ alnk[i].href);

</script>

Incercati codul

• Proprietatea **innerHTML** apartine obiectelor HTML din document. Aceasta returneaza sau scrie continutul din elementul HTML la care e aplicata.

- Urmatorul exemplu preia continutul din primul link din pagina, apoi il modifica.

<a href='//marplo.net' title='MarPlo.net'>MarPlo.net</a>

<ul><li><a href='//gamv.eu' title='GamV.eu'>GamV.eu</a></li></ul>

<div><a href='//coursesweb.net' title='CoursesWeb.net'>CoursesWeb.net</a></div>

<script>

var alnk = document.links; //array cu link-urile din pagina

//preia textul din primul link

var txta1 = alnk[0].innerHTML;

//modifica textul din primul link

alnk[0].innerHTML ='Text modificat din JS';

//arata in consola textul initial

console.log('Textul din primul link era: '+ txta1);

</script>

Incercati codul

### Utilizare metode din obiectul document

Obiectul document contine numeroase metode pentru preluarea si modificarea elementelor HTML din pagina in script-uri JS.  
In continuare sunt prezentate cateva exemple cu accesare si manipulare elemente HTML folosind metodele JavaScript.

• **document.getElementById('id')** - preia elementul HTML care are id-ul specificat.

*Mai multe detalii si exemple cu getElementById() gasiti la Lectia de la pagina: [marplo.net/javascript/getelementbyid.html](https://marplo.net/javascript/getelementbyid.html" \o "getElementById)*

Exemplu, preia continutul dintr-un Div si il adauga cu proprietatea **textContent** in alt element HTML.

<p id='prg1'>Exemplu cu <em>getElementById()</em></p>

Continutul din #prg1 este: <span id='sp1'>str</span>

<script>

//preia elementul html cu id-ul prg1

var elm = document.getElementById('prg1');

//preia continutul html din elm

var elm\_cnt = elm.innerHTML;

//adauga sirul din elm\_cnt ca text in #sp1

document.getElementById('sp1').textContent = elm\_cnt;

</script>

Incercati codul

• **getElementsByTagName('tag')** - returneaza un array cu elementele HTML care au tag-ul specificat.

*Mai multe detalii si exemple cu getElementsByTagName() gasiti la Lectia de la pagina: [marplo.net/javascript/getelementsbytagname.html](https://marplo.net/javascript/getelementsbytagname.html" \o "getElementsByTagName)*

Exemplu, afiseaza in pagina numarul de tag-uri H3 si id-ul de la primul.

<h3 id='id\_elm'>Unde-s doi sunt doi de unu.</h3>

<div>Div, HTML content</div>

<h3>Alt tag H3</h3>

<blockquote id='resp'>Pt. raspuns din JS.</blockquote>

<script>

//array cu tag-urile H3

var ar\_h3 = document.getElementsByTagName('h3');

//adauga in #resp nr. elemente H3 si id-ul de la primul

if(ar\_h3.length >0){

var idh = ar\_h3[0].id; //id-ul primului element din ar\_h3

document.getElementById('resp').innerHTML ='Sunt '+ ar\_h3.length +' elemente H3, primul are id: '+ idh;

}

</script>

Incercati codul

### Lucru cu Proprietati si Metode ale elementelor HTML in JS

Elementele HTML preluate in JavaScript au proprietati si metode pentru accesarea si manipularea partilor componente (atribute, stiluri css, continut).  
Acestea se aplica direct la elementul respectiv.

**elmHtml.proprietate  
elmHtml.metoda()**

- Unde 'elmHtml' este elementul HTML preluat cu metode specifice DOM (din ***document***).

***O lista cu proprietati si metode utile ale elementelor HTML gasiti la pagina:  
[marplo.net/javascript/proprietati-metode-element-html](https://marplo.net/javascript/proprietati-metode-element-html" \o "Proprietati si Metode utile ale elementelor HTML in JavaScript)***

#### Exemple

• **elmHtml.addEventListener('event', callback)** - apeleaza functia de la **callback** cand evenimentul de la 'event' este efectuat (vedeti si tutorialul de la: [marplo.net/javascript/detectare-stergere-evenimente-js](https://marplo.net/javascript/detectare-stergere-evenimente-js" \o "Detectare si Stergere Evenimente in JS)).  
- Exemplu, cand se apasa clic pe butonul cu id #btn1 afiseaza o fereastra alert() cu textul din #prg1.

<p id='prg1'>Ex. addEventListener(), clic.</p>

<button id='btn1'>Clic</button>

<script>

//la click pe #btn1 se apeleaza o functie

document.getElementById('btn1').addEventListener('click', (ev)=>{

var cnt = document.getElementById('prg1').textContent;

alert(cnt);

});

</script>

Incercati codul

• **elmHtml.parentNode** - returneaza elementul parinte in care e inclus elmHtml.  
• **elmHtml.className** - returneaza clasa (valoarea atributului ***class***) din elmHtml.  
- Exemplu, cand se apasa clic pe butonul cu id #btn1 afiseaza o fereastra alert() cu clasa din elementu parinte al butonului.

<div class='cls\_1'>Div in care e un buton: <button id='btn1'>Clic</button></div>

<script>

//la click pe #btn1 se apeleaza o functie (ev reprezinta un obiect cu evenimentul)

document.getElementById('btn1').addEventListener('click', (ev)=>{

//preia elementul parinte (ev.target reprezinta elementul care a actionat evenimentul)

var parent = ev.target.parentNode;

//preia clasa

var cls = parent.className;

alert('Elementul parinte are class='+ cls);

});

</script>

Incercati codul

• **elmHtml.setAttribute('attr', 'val')** - seteaza atributul specificat 'attr' cu valoarea 'val'.  
• **elmHtml.outerHTML** - returneaza un sir HTML cu tot elementul elmHtml (toata sructura si continutul). Sau inlocuieste elmHtml cu altceva (transmis ca sir).  
- Exemplu, cand se apasa clic pe butonul cu id #btn1 seteaza atributul 'style' cu proprietati CSS la #prg1, apoi sterge butonul din pagina.

<p id='prg1'>Pacea, Bucuria si Bunatatea construiesc Sanatatea.</p>

<button id='btn1'>Clic</button>

<script>

//la click pe #btn1 se apeleaza o functie (ev reprezinta un obiect cu evenimentul)

document.getElementById('btn1').addEventListener('click', (ev)=>{

//seteaza style la #prg1

document.getElementById('prg1').setAttribute('style', 'color:#0000e0; font-size:22px;');

//sterge butonul (ev.target reprezinta elementul care a actionat evenimentul)

ev.target.outerHTML ='';

});

</script>

# Utilizare getElementById

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [innerHTML](https://marplo.net/javascript/getelementbyid.html" \l "hshinh" \o "innerHTML)
* [getAttribute(attr) si setAttribute(attr, val)](https://marplo.net/javascript/getelementbyid.html" \l "hshgsa" \o "getAttribute si setAttribute)
* [parentNode](https://marplo.net/javascript/getelementbyid.html" \l "hshpn" \o "parentNode)
* [Proprietatea style](https://marplo.net/javascript/getelementbyid.html" \l "hshps" \o "Proprietatea style)
* [value](https://marplo.net/javascript/getelementbyid.html" \l "hshval" \o "value)

ID-ul este un atribut care poate fi adaugat in tag-urile HTML. Prin valoarea data acestui atribut se atribuie un nume unic acelui element. Acest "id" poate fi folosit in stilurile CSS pentru a defini aspectul grafic si aranjarea in pagina a tag-ului respectiv, dar poate fi folosit si in scripturi JavaScript pentru a lucra cu elementele si continutul lor.  
Cu valoarea id-ului se poate prelua elementul HTML in JavaScript foarte usor, folosind metoda **getElementById()**cu urmatoarea sintaxa:

**document.getElementById('id')**

- returneaza un obiect JavaScript cu elementul care are id-ul specificat.

Obiectul returnat de **getElementById()** contine proprietati si metode prin care se poate lucra cu partile componente ale acelui element (atribute, continut). O lista cu acestea gasiti la pagina:  
**[marplo.net/javascript/proprietati-metode-element-html](https://marplo.net/javascript/proprietati-metode-element-html" \o "Proprietati si Metode utile ale elementelor HTML in JavaScript)**

- In continuare sunt prezentate cateva exemple cu unele din aceste proprietati si metode pe care le puteti testa direct pe acest site.

### addEventListener()

**addEventListener()** e o metoda prin care se poate inregistra detectarea de evenimente efectuate pe un element HTML (*click, mouseenter, focus, input*, etc.).  
Sintaxa:

**elm.addEventListener('event', callF)**

Unde '***elm***' reprezinta elementul HTML. Cand are loc evenimentul '***event***' la acel element, se apeleaza functia de la ***callF***.  
  
Mai multe detalii in tutorialul de la: **[marplo.net/javascript/detectare-stergere-evenimente-js](https://marplo.net/javascript/detectare-stergere-evenimente-js" \l "hsheal" \o "Detectare eveniment cu addEventListener)**  
  
- Iata un exemplu, cand mouse-ul intra pe suprafata unui Div, se schimba culoarea background; iar la click pe el se afiseaza o fereastra alert.

<h4>Exemplu cu addEventListener()</h4>

<p id='pr1'>Cand mouse-ul intra pe suprafata Div-ului cu id 'dv1', se schimba culoarea background; iar la click pe el se afiseaza o fereastra alert.</p>

<div id='dv1' style='background:#b8eeb9; height:100px; font-weight:700; width:150px;'>Div #dv1<br>

- Click Aici -</div>

<script>

var dv1 = document.getElementById('dv1');

//detecteaza mouseenter

dv1.addEventListener('mouseenter', (ev)=>{

// ev.target reprezinta elementul la care sa declansat evenimentul (aici acelasi cu dv1 )

ev.target.style.background ='#ced0fe';

});

//detecteaza click

dv1.addEventListener('click', (ev)=>{

alert('Pacea-i Buna');

});

</script>

Incercati codul

### innerHTML

Returneaza continutul elementului, sau il inlocuieste cu alt continut HTML.

<h4>Exemplu innerHTML</h4>

<a id='lnk1' href='//marplo.net' title='MarPlo.net'><em>MarPlo.net</em></a>

<ul><li><a href='//gamv.eu' title='GamV.eu'>GamV.eu</a></li></ul>

<p>La clic pe urmatorul button preia continutul din primul link (cu id 'lnk1'), apoi il inlocuieste cu altceva si afiseaza continutul initial in consola.</p>

<button id='btn1'>Test innerHTML</button>

</div>

<script>

document.getElementById('btn1').addEventListener('click', (ev)=>{

var lnk1 = document.getElementById('lnk1');

//preia continutul html din #lnk1

var cnt = lnk1.innerHTML;

//modifica continutul din lnk1

lnk1.innerHTML ='Continut adaugat cu <em>innerHTML</em>';

//arata in consola continutul initial

console.log('Continut din #lnk1 era: '+ cnt);

});

</script>

Incercati codul

### getAttribute(attr) si setAttribute(attr, val)

**getAttribute(attr)** returneaza valoarea atributului 'attr' specificat.  
**setAttribute(attr, val)** defineste atributul specificat la 'attr' cu valoarea de la 'val'.

<h4>Exemplu getAttribute() si setAttribute()</h4>

<p id='pr1' style='background:#fbfbbb; font-size:18px;'>La clic pe urmatorul button se afiseaza la #resp valoarea atributului 'style' din acest paragraf (cu getAttribute() ), apoi defineste alta valoare la atributul 'style' cu setAttribute().</p>

<button id='btn1'>Set style</button>

<blockquote id='resp'>#resp</blockquote>

<script>

var prg = document.getElementById('pr1');

document.getElementById('btn1').addEventListener('click', (ev)=>{

document.getElementById('resp').textContent ='style precedent: '+ prg.getAttribute('style');

prg.setAttribute('style', 'background:#dee0fe; font-weight:700;');

});

</script>

Incercati codul

### parentNode

Returneaza elementul parinte in care se afla elementul la care e atasat.

<h4>Exemplu parentNode</h4>

<div>

<p>La clic pe urmatorul button se afiseaza la #resp numele tag-ului elementului parinte (cu tagName) in care e adugat butonul.</p>

<button id='btn1'>Parent Tag</button>

</div>

<blockquote id='resp'>#resp</blockquote>

<script>

var btn = document.getElementById('btn1');

btn.addEventListener('click', (ev)=>{

document.getElementById('resp').textContent = btn.parentNode.tagName;

});

</script>

Incercati codul

### Proprietatea style

Cu proprietatea **style** se pot defini din JavaScript proprietati de stil CSS la elementele HTML, folosind aceasta sintaxa.

**elm.style.propName ='value';**

Unde 'elm' reprezinta elementul HTML, 'propName' e proprietatea CSS, iar 'value' e valoarea care i-se atribuie.

*Diferenta apare la proprietatile CSS compuse, cum e "font-weight", "margin-top" sau "border-top-width" si altele similare.  
In JavaScript dispare liniuta '-' si cuvintele urmatoare se scriu cu primul caracter majuscula, astfel pentru "font-weight" din CSS in JS este:****fontWeight****, iar pentru "border-top-width" in JS este:****borderTopWidth***

<h4>Exemplu cu proprietatea style</h4>

<p id='pr1'>La clic pe urmatorul button se definesc la acest paragraf proprietatile CSS 'color' si 'font-size' folosind proprietatea JS 'style'.</p>

<button id='btn1'>Set style</button>

<script>

var prg = document.getElementById('pr1');

document.getElementById('btn1').addEventListener('click', (ev)=>{

prg.style.color ='#0000d0';

prg.style.fontSize ='20px';

});

</script>

Incercati codul

### value

Proprietatea **value** se foloseste la elemente de formular, aceasta preia sau atribue valori in campurile din formular.  
- Pentru a folosi proprietatea "***value***" impreuna cu getElementById('id'), campul sau caseta de formular la care se face referire trebuie sa aibe un "id".  
Iata un exemplu simplu care, la clic pe un buton afiseaza intr-un element HTML textul scris intr-o caseta de tip 'password'.

<h4>Exemplu cu value</h4>

<p id='pr1'>La clic pe urmatorul button se afiseaza la #resp valoarea /parola din caseta de tip 'password'.</p>

Password: <input type='password' id='inp1' value='marplo.net'><br>

<button id='btn1'>Show pass</button>

<blockquote id='resp'>#resp</blockquote>

<script>

var inp = document.getElementById('inp1');

document.getElementById('btn1').addEventListener('click', (ev)=>{

document.getElementById('resp').textContent = inp.value;

});

</script>

# window.navigator

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

Obiectul **navigator** apartine direct obiectului "window".  
- **navigator** contine informatii despre programul /aplicatia browser.  
Acest obiect are cateva proprietati si metode care pot fi apelate cu sintaxa:

**navigator.nume\_proprietate  
navigator.numeMetoda()**

### Proprietati si Metode obiect navigator

* **cookieEnabled** - returneaza True daca cookie e activat in browser, in caz contrar, False.
* var msg = (navigator.cookieEnabled) ? 'Cookie poate fi folosit in browser' :'Nu se poate utiliza cookie in browser';
* document.write('<p>'+ msg +'</p>');

Incercati codul

* **geolocation** - returneaza un obiect **Geolocation** care permite aflarea pozitiei (latitudine /longitudine) navigatorului (functioneaza in pagini cu adresa 'secure': HTTPS).
* <p id='prg1'>Clic pe buton pt. aflare coordonate.</p>
* <button onclick='getLocation()'>Get coords</button>
* <script>
* var prg1 = document.getElementById('prg1');
* //functie apelata de la buton
* function getLocation(){
* var options = { enableHighAccuracy: true, timeout: 5000};
* var error =(err)=>{
* prg1.innerHTML ='ERROR: '+ err.code +' - '+err.message;
* }
* if(navigator.geolocation) navigator.geolocation.getCurrentPosition(showPosition, error, options);
* else prg1.innerHTML ='Geolocation nu e valabil in acest browser.';
* }
* function showPosition(pos){
* prg1.innerHTML = 'Latitude: '+ pos.coords.latitude +'<br>Longitude: '+ pos.coords.longitude;
* }
* </script>

Incercati codul

* **language** - limbajul setat in browser.
* document.write('<p>Limbajul setat in browser e: '+ navigator.language +'</p>');
* //Ex.: en-US

Incercati codul

* **onLine** - returneaza True daca browser-ul e online, in caz contrar False.
* document.write('<p>Browser-ul e online: '+ navigator.onLine +'</p>');

Incercati codul

* **oscpu** - returneaza un sir cu sistemul de operare curent, sau 'undefined'.
* document.write('<p>Sistemul dv. de operare e:<br> '+ navigator.oscpu +'</p>');
* //ex.: Windows NT 6.1

Incercati codul

* **userAgent** - returneaza un sir cu antetul /tipul browser-ului.
* document.write('<p>location.userAgent a returnat:<br> '+ navigator.userAgent +'</p>');
* //ex.: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.87 Safari/537.36

Incercati codul

- Multe din metodele obiectului navigator au fost deprecate in versiunea JS mai noua, sau nu sunt standard.  
O metoda din acest obiect e:

* **vibrate(ms)** - determina vibrarea dispozitivului (la dispozitive care au sistem de vibratie) pt. un numar de milisecunde 'ms' specificat.

navigator.vibrate(500); // vibreaza 500ms

# window.history

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

Navigatoarele web au o caracteristica numita "History" care permite urmarirea locatiilor pe care le-ati vizitat.  
Acest obiect apartine direct obiectului "window"; contine un Array cu paginile vizitate de client, in cadrul unei ferestre, un istoric al paginilor vizitate.  
  
- Obiectul **history** permite navigarea inapoi in lista istoric la paginile care au mai fost vizitate in cadrul aceleasi ferestre.  
Acest obiect are cateva proprietati si metode care pot fi apelate cu sintaxa:

**window.history.nume\_proprietate  
window.history.numeMetoda()**

• Obiectul **history** are doua proprietati: **length** si **state**.  
**window.history.length** - returneaza numarul de adrese URL din 'history'.  
**window.history.state** - returneaza obiectul adaugat in history cu **pushState()** sau **replaceState()**.

### Metode obiect history

* **back()** - incarca in pagina adresa URL precedenta din lista istoric.

<button onClick='window.history.back()'>Back</button>

* **forward()** - incarca urmatoarea adresa URL din lista istoric.

<button onClick='window.history.forward()'>Forward</button>

* **go(x)** - incarca o adresa URL, echivalenta cu un salt la numarul "x" (pozitiv sau negativ) din lista istoric.
* window.history.go(-1); //echivalent cu: window.history.back()
* window.history.go(-2); //a doua adresa inapoi
* window.history.go(1); //echivalent cu: window.history.forward()
* **pushState(state\_obj, title, url)** - adauga in istoric o noua intrare pt. pagina curenta, cu adresa de la 'url' si titlul de la 'title'.  
  In plus, modifica adresa din bara de adrese cu 'url', fara sa incarce acea adresa (nu conteaza daca e adresa valida sau nu).  
  - 'state\_obj' e un mic obiect JS asociat cu noua modificare in istoric, care apoi poate fi preluat cu: **history.state**.
* <button onClick='addAdr()'>Adauga adresa</button>
* <script>
* var stob ={prop:'some str'};
* function addAdr(){
* window.history.pushState(stob, 'New Title', 'some\_url.html');
* //foloseste cu proprietatea state obiectul adaugat in history
* alert(history.state.prop);
* }
* </script>

- Demo:

- Clic pe urmatorul buton, vedeti modificarea la adresa din browser.  
Adauga adresa

* **replaceState(state\_obj, title, url)** - modifica in '**history**' istoricul pagini curente cu argumentele transmise: adresa de la 'url' si titlul cu 'title'.  
  La fel ca ***pushState()***, modifica adresa din bara de adrese cu 'url', fara sa incarce acea adresa (nu conteaza daca e adresa valida sau nu).  
  - 'state\_obj' e un mic obiect JS asociat cu noua modificare in istoric, care apoi poate fi preluat cu: **history.state**.
* <button onClick='replaceAdr()'>Modifica adresa</button>
* <script>
* var stob ={prop:'some str'};
* function replaceAdr(){
* window.history.replaceState(stob, 'New Title', 'change\_url.html');
* //foloseste cu proprietatea state obiectul adaugat in history
* alert(history.state.prop);
* }
* </script>

- Demo:

- Clic pe urmatorul buton, vedeti modificarea la adresa din browser.  
Modifica adresa

# Obiectul location

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

Obiectul **location** apartine obiectului '***window***' dar si obiectului '***document***'.  
- **location** contine date din adresa URL curenta.  
  
• **window.location** (sau **document.location** ) returneaza adresa URL completa, dar daca i-se atribue o valoare (un sir cu adresa URL) va deschide in fereastra acea adresa (face redirect la adresa adaugata).

- Exemplu, afiseaza adresa completa a paginii, iar dupa 5 secunde face redirect la alta adresa (in iframe daca script-ul e intr-un <iframe>).

document.write('<p>Adresa curenta:<br>'+window.location+'<br><br>- Dupa 5 secunde redirect la: //gamv.eu</p>');

window.setTimeout(()=>{window.location ='//gamv.eu';}, 5000);

Incercati codul

Acest obiect are proprietati si metode care pot fi apelate cu sintaxa:

**window.location.nume\_proprietate  
window.location.numeMetoda()**

### Proprietati obiect location

* **hash** - returneaza sau seteaza sirul din adresa URL care urmeaza dupa caracterul diez (#).
* //URL: https://marplo.net/javascript?id=9#abc\_xy
* var hash = window.location.hash; // #abc\_xy
* //adauga alt hash
* window.location.hash ='xy\_890';
* alert(hash);
* **host** - contine numele domeniului si numarul de port (daca e specificat) al adresei URL.
* //URL: https://bfie.marplo.net/page1
* var str = window.location.host; // bfie.marplo.net
* **hostname** - contine numele domeniului din adresa URL (fara port).
* //URL: https://marplo.net/javascript?id=9#abc\_xy
* var str = window.location.hostname; // marplo.net
* **href** - seteaza sau returneaza adresa URL completa.
* <p>Exemplu: location.href<br>
* - La clic pe buton afiseaza fereastra alert cu adresa paginii, apoi incarca alta adresa (in iframe daca script-ul e intr-un &lt;iframe&gt;).</p>
* <button id='btn1'>Click</button>
* <script>
* document.getElementById('btn1').addEventListener('click', (ev)=>{
* var str = window.location.href;
* alert(str);
* //incarca alta adresa in fereastra
* window.location.href ='//gamv.eu/';
* });
* </script>

Incercati codul

* **origin** - contine partea din adresa URL cu protocol, hostname (domeniu) si port (daca e specificat).
* <p>Exemplu: location.origin<br>
* - <strong>window.location.origin</strong> a returnat:<br>
* <span id='resp'>window.location.origin</span></p>
* <script>
* document.getElementById('resp').innerHTML = window.location.origin;
* </script>

Incercati codul

* **pathname** - returneaza sau seteaza partea din adresa URL dupa numele domeniului, cu '/' la inceput.
* //URL: https://marplo.net/html/div\_span.html
* var str = window.location.pathname; // /html/div\_span.html
* alert(str);
* //incarca alta pagina din aceleasi site
* window.location.pathname ='/javascript/sintaxajs.html';
* **port** - returneaza sau seteaza numarul port-ului din URL (daca e specificat).
* //URL: https://marplo.net:443/javascript
* var port = window.location.pathname; // 443
* **protocol** - returneaza sau seteaza partea de protocol din adresa URL, cu ':' la sfarsit.
* //URL: https://marplo.net/javascript
* var prot = window.location.protocol; // https:
* **search** - returneaza sau seteaza sirul 'query' (parametri dupa '?') din adresa URL, inclusiv '?'.
* //URL: //https://domain.net/page?id=2&src=abcd
* var query = window.location.search; // ?id=2&src=abcd

### Metode obiect location

* **assign(url)** - incarca si afiseaza documentul de la adresa 'url' transmisa.
* <p>Exemplu window.location.assign().<br>
* - La clic pe buton incarca o adresa cu o imagine.</p>
* <button id='btn1'>Click</button>
* <script>
* document.getElementById('btn1').addEventListener('click', (ev)=>{
* window.location.assign('//marplo.net/imgs/smile\_gift.png');
* });
* </script>

Incercati codul

* **reload(arg)** - reincarca adresa URL curenta in fereastra de browser. Parametrul 'arg' e optional, poate fi **true** sau **false**.  
  Daca e specificat 'true', reincarca continutul de pe server; daca e 'false' sau nu e specificat, poate sa reincarce continutul din cache-ul browser-ului.
* <p>Exemplu window.location.reload().<br>
* - La clic pe buton reincarca pagina de pe server.</p>
* <button id='btn1'>Click</button>
* <script>
* document.getElementById('btn1').addEventListener('click', (ev)=>{
* window.location.reload(true);
* });
* </script>

Incercati codul

* **replace(url)** - incarca si afiseaza in fereastra continutul de la adresa 'url'.  
  - Diferenta fata de ***assign()*** e faptul ca **replace()** inlocuieste in 'history' pagina curenta cu pagina de la noua adresa, adica butonul Back din browser nu mai intoarce la pagina initiala.
* <p>Exemplu window.location.replace().<br>
* - La clic pe buton incarca o imagine.</p>
* <button id='btn1'>Replace</button>
* <script>
* document.getElementById('btn1').addEventListener('click', (ev)=>{
* window.location.replace('//marplo.net/imgs/smile\_gift.png');
* });

</script>

# Obiectul anchor si Lucru cu adrese din link

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

Obiectul **anchor** reprezinta elementele HTML <a> (folosite pentru link-uri).  
In afara de proprietatile si metodele specifice elementelor html (prezentate la: **[marplo.net/javascript/proprietati-metode-element-html](https://marplo.net/javascript/proprietati-metode-element-html" \o "Proprietati si Metode utile ale elementelor HTML in JavaScript)** ), obiectul anchor contine si proprietati proprii, utile pentru lucru cu adrese URL (link-uri adaugate in elemente <a>).

### Proprietati obiect Anchor

Proprietatile obiectului **anchor** sunt similare cu cele de la obiectul ***navigator***, dar se aplica la elementele <a> preluate /create in JavaScrip.

**download** - returneaza sau seteaza valoarea atributului 'download' dintr-un link.  
- Acest atribut indica faptul ca link-ul e pt. download, resursa va fi descarcata cand se da clic pe link.

<p>Exemplu link cu atribut download.<br>

- La clic pe link, resursa de la adresa 'href' va fi descarcata.</p>

<a href='/imgs/smile\_gift.png' title='Image' id='lnk1' download='smile'><img src='/imgs/smile\_gift.png' alt='Smile' width='40' height='35'> Link image</a>

<p>La clic pe urmatorul buton va fi adaugat la #resp valoarea atributului download.</p>

<button id='btn1'>Attr download</button>

<blockquote id='resp'>#resp</blockquote>

<script>

document.getElementById('btn1').addEventListener('click', (ev)=>{

var dwl = document.getElementById('lnk1').download;

document.getElementById('resp').innerHTML = dwl;

});

</script>

Incercati codul

**hash** - returneaza sau seteaza sirul din adresa de la 'href' care urmeaza dupa caracterul diez (#).**host** - contine numele domeniului si numarul de port (daca e specificat) din adresa de la 'href'.**hostname** - contine numele domeniului din adresa de la 'href'.**href** - seteaza sau returneaza adresa completa de la 'href'.**origin** - contine partea din adresa de la 'href' cu protocol, hostname (domeniu) si port (daca e specificat).**password** - returneaza sau seteaza partea 'password' (parola) din adresa de la 'href' (daca e specificata, inainte de numele domeniului).**pathname** - returneaza sau seteaza partea din adresa de la 'href' dupa numele domeniului, cu '/' la inceput.**port** - returneaza sau seteaza numarul port-ului din adresa link-ului (daca e specificat).**protocol** - returneaza sau seteaza partea de protocol din adresa link-ului, cu ':' la sfarsit.**referrerPolicy** - returneaza sau seteaza valoarea atributului **referrerpolicy**. Se pot seta urmatoarele valori:  
- **no-referrer** - referrer HTTP header nu va fi transmis.  
- **origin** - referrer-ul transmis va fi originea paginii (protocol, domeniu si port).  
- **unsafe-url** - referrer-ul va include originea si 'pathname' din adresa paginii.**rel** - returneaza sau seteaza valoarea atributului **rel**.**search** - returneaza sau seteaza sirul 'query' (parametri dupa '?') din adresa link-ului, inclusiv '?'.**target** - returneaza sau seteaza valoarea atributului 'target'.**title** - returneaza sau seteaza valoarea atributului 'title' din link.**text** - returneaza sau seteaza textul link-ului.**username** - returneaza sau seteaza partea 'username' (nume\_user) din adresa de la 'href' (daca e specificat, inainte de numele domeniului).

**<<-[Obiectul location](https://marplo.net/javascript/obiectul-location" \o "Obiectul location) -- [Obiectul form in JavaScript](https://marplo.net/javascript/obiect-form" \o "Obiectul form in JavaScript)->>**

* **[Tutoriale JavaScript](https://marplo.net/javascript/tutoriale-js" \o "Tutoriale JavaScript)**
* **[Vue.js](https://marplo.net/vuejs" \o "Vue.js)**
* **[Curs jQuery](https://marplo.net/javascript/curs-jquery-tutoriale-js" \o "Curs jQuery)**
* **[Tutoriale Node.js](https://marplo.net/javascript/tutoriale-nodejs-js" \o "Tutoriale Node.js)**
* **[Coduri si Functii JavaScript](https://marplo.net/javascript/cod-functii-javascript-js" \o "Coduri si Functii JavaScript)**
* **[Scripturi](https://marplo.net/javascript/scripts" \o "Scripturi)**
* **[JavaScript](https://marplo.net/javascript" \o "JavaScript)**

[Utilizare EventSource pentru evenimente de la server](https://marplo.net/javascript/eventsource-evenimente-server" \o "Utilizare EventSource pentru evenimente de la server)[JavaScript Worker](https://marplo.net/javascript/worker-js" \o "JavaScript Worker)[Creare clase in JavaScript cu Metode care pot fi inlantuite](https://marplo.net/javascript/creare-clase-metode-inlantuite" \o "Creare clase in JavaScript cu Metode care pot fi inlantuite)[Subclase cu extends si Mostenire](https://marplo.net/javascript/subclase-extends-mostenire" \o "Subclase cu extends si Mostenire)[Definire si Utilizare Clase in JavaScript](https://marplo.net/javascript/definire-utilizare-clase-js" \o "Definire si Utilizare Clase in JavaScript)[classList - Lucru cu clase css](https://marplo.net/javascript/classlist-lucru-clase-css" \o "classList - Lucru cu clase css)[Creare scripturi avansate Javascript - PHP](https://marplo.net/javascript/javascript_php.html" \o "Creare scripturi avansate Javascript - PHP)[createElement si insertBefore](https://marplo.net/javascript/createelement_insertbefore.html" \o "createElement si insertBefore)[querySelector si querySelectorAll](https://marplo.net/javascript/queryselector_queryselectorall.html" \o "querySelector si querySelectorAll)[Utilizare getElementsByTagName in JS](https://marplo.net/javascript/getelementsbytagname.html" \o "Utilizare getElementsByTagName in JS)[Utilizare Cookie in JS](https://marplo.net/javascript/cookie.html" \o "Utilizare Cookie in JS)[Lucru cu Imagini in JavaScript](https://marplo.net/javascript/lucru-imagini-js" \o "Lucru cu Imagini in JavaScript)[Detectare si Stergere Evenimente in JS](https://marplo.net/javascript/detectare-stergere-evenimente-js" \o "Detectare si Stergere Evenimente in JS)[Evenimente JavaScript](https://marplo.net/javascript/evenimente.html" \o "Evenimente JavaScript)[Accesare elemente din form](https://marplo.net/javascript/accesare-elemente-form" \o "Accesare elemente din form)[Obiectul form in JavaScript](https://marplo.net/javascript/obiect-form" \o "Obiectul form in JavaScript)[Obiectul location](https://marplo.net/javascript/obiectul-location" \o "Obiectul location)[window.history](https://marplo.net/javascript/history" \o "window.history)[window.navigator](https://marplo.net/javascript/navigator" \o "window.navigator)[Utilizare getElementById](https://marplo.net/javascript/getelementbyid.html" \o "Utilizare getElementById)[JavaScript HTML DOM - Obiectul document](https://marplo.net/javascript/dom-html-obiect-document" \o "JavaScript HTML DOM - Obiectul document)[Obiectul window](https://marplo.net/javascript/windows.html" \o "Obiectul window)[Obiectul Date - Lucru cu Data si Timp](https://marplo.net/javascript/obiect-date-timp" \o "Obiectul Date - Lucru cu Data si Timp)[Obiectul Array](https://marplo.net/javascript/obiectul-array" \o "Obiectul Array)[Obiectul Math - Metode pentru operatiuni matematice](https://marplo.net/javascript/obiect-math-operatiuni-matematice" \o "Obiectul Math - Metode pentru operatiuni matematice)[Numere in JavaScript](https://marplo.net/javascript/numere-js" \o "Numere in JavaScript)[Obiectul String - Sir](https://marplo.net/javascript/obiectul-string" \o "Obiectul String - Sir)[Utilizare Functii si Parametri lor](https://marplo.net/javascript/functii2.html" \o "Utilizare Functii si Parametri lor)[Definire si Utilizare Functii in JS](https://marplo.net/javascript/definire-utilizare-functii" \o "Definire si Utilizare Functii in JS)[Ferestre Alert Prompt si Confirm](https://marplo.net/javascript/alert_prompt_confirm.html" \o "Ferestre Alert Prompt si Confirm)[break, continue, si eticheta](https://marplo.net/javascript/break-continue-eticheta" \o "break, continue, si eticheta)[Instructiuni repetitive while](https://marplo.net/javascript/instructiuni-repetitive-while" \o "Instructiuni repetitive while)[Instructiuni repetitive for()](https://marplo.net/javascript/instructiuni-repetitive-for" \o "Instructiuni repetitive for())[Instructiuni conditionale if() else, switch](https://marplo.net/javascript/instructiuni_conditionale.html" \o "Instructiuni conditionale if() else, switch)[Operatori in JavaScript](https://marplo.net/javascript/operatori.html" \o "Operatori in JavaScript)[Definire Variabile si Constante](https://marplo.net/javascript/definire-variabile-constante" \o "Definire Variabile si Constante)[Sintaxa JavaScript](https://marplo.net/javascript/sintaxajs.html" \o "Sintaxa JavaScript)[Donation](https://www.paypal.com/cgi-bin/webscr?cmd=_s-xclick&hosted_button_id=KEHLERNATG54U&source=url)

#### Un Test simplu in fiecare zi

##### HTML

##### CSS

##### JavaScript

##### PHP-MySQL

##### Engleza

##### Spaniola

***Clic pe metoda ce creaza un array cu toate elementele din pagina cu un anumit nume de tag.***

**getElementsByName()getElementById()getElementsByTagName()**

[Sharing Twitter](https://twitter.com/intent/tweet?text=Obiectul%20anchor%20si%20Lucru%20cu%20adrese%20din%20link&url=https://marplo.net/javascript/obiectul-anchor-adrese-link&via=CoursesWeb)

#### Last accessed pages

1. **Curs PHP MySQL, Tutoriale si Scripturi PHP (41499)**
2. **Obiectul location (284)**
3. **Blog si Cugetari Personale (28582)**
4. **Pronumele direct si indirect (1189)**
5. **window.history (352)**

#### Popular pages this month

1. **Cursuri si Tutoriale: Engleza, Spaniola, HTML, CSS, Php-Mysql, JavaScript, Ajax (4004)**
2. **Curs HTML gratuit Tutoriale HTML5 (3576)**
3. **Curs si Tutoriale JavaScript (2686)**
4. **Curs PHP MySQL, Tutoriale si Scripturi PHP (2650)**
5. **Curs si Tutoriale Ajax (2527)**

##### Site access statistics

Chat

**[Home](https://marplo.net/" \o "Home)** **[Contact](https://marplo.net/contact" \o "Contact)** **[Engleza](https://marplo.net/engleza" \o "Engleza)** **[Spaniola](https://marplo.net/spaniola" \o "Spaniola)** **[Html](https://marplo.net/html" \o "Html)** **[CSS](https://marplo.net/css" \o "CSS)** **[JavaScript](https://marplo.net/javascript" \o "JavaScript)** **[PHP-MySQL](https://marplo.net/php-mysql" \o "PHP-MySQL)** **[Ajax](https://marplo.net/ajax" \o "Ajax)** **[Flash-AS3](https://marplo.net/flash" \o "Flash-AS3)** **[PHP Laravel - Tutoriale](https://marplo.net/php-mysql/laravel-tutoriale" \o "PHP Laravel - Tutoriale)** **[Vue.js](https://marplo.net/vuejs" \o "Vue.js)** **[Tutoriale Node.js](https://marplo.net/javascript/tutoriale-nodejs-js" \o "Tutoriale Node.js)** **[Blog](https://marplo.net/blog" \o "Blog)**- WebMaster, WebDesigner : MarPlo -

# Obiectul form in JavaScript

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Proprietati obiect form](https://marplo.net/javascript/obiect-form" \l "hshpf" \o "Proprietati obiect form)
* [Metode obiect form](https://marplo.net/javascript/obiect-form" \l "hshmf" \o "Metode obiect form)
* [Evenimente obiect form](https://marplo.net/javascript/obiect-form" \l "hshef" \o "Evenimente obiect form)

Obiectul **form** reprezinta elementele HTML <form> si sunt in general utilizate pentru formulare de trimitere date de la o aplicatie la alta; date care pot fi adaugate /selectate de utilizator.

## Preluare element <form> in JavaScript

Sunt mai multe moduri prin care elementele HTML <form> pot fi preluate ca obiecte in JavaScript.  
In general se folosesc metodele: **getElementById()**, **querySelector()** sau proprietatea: **document.forms**.

* Daca elementul <form> are un ID, se poate accesa cu oricare din aceste sintaxe:

**document.getElementById('id\_form')  
document.querySelector('#id\_form')  
document.forms['id\_form']**

- Exemplu, preia un <form> dupa ID si afiseaza valoarea de la atributul 'action'.

<p>Acceseaza un &lt;form&gt; dupa ID si afiseaza valoarea de la atributul 'action'.</p>

<form id='frm1' action='some\_page.php' method='post'>

<p>Acest formular cu id '#frm1' are action: <em id='resp'>#resp</em></p>

Text: <input type='text'/>

</form>

<script>

var frm = document.getElementById('frm1');

document.getElementById('resp').innerHTML = frm.action;

</script>

Incercati codul

* Daca elementul <form> are un atribut ***name***, se poate accesa cu aceasta sintaxa:

**document.forms['name\_form']**

- Exemplu, preia un <form> dupa 'name' si afiseaza valoarea de la atributul 'action'.

<p>Acceseaza un &lt;form&gt; dupa 'name' si afiseaza valoarea de la atributul 'action'.</p>

<form name='frm2' action='some\_adr.php' method='post'>

<p>Acest formular cu name 'frm2' are action: <em id='resp'>#resp</em></p>

Text: <input type='text'/>

</form>

<script>

var frm = document.forms['frm2'];

document.getElementById('resp').innerHTML = frm.action;

</script>

Incercati codul

* Daca elementul <form> are o clasa css (atribut ***class***), se poate accesa cu aceasta sintaxa:

**document.querySelector('form.clas\_form')**

*Daca sunt mai multe form-uri cu acelasi 'class', metoda****querySelector()****va returna pe primul.*

- Exemplu, preia un <form> dupa 'class' si afiseaza valoarea de la atributul 'action'.

<p>Acceseaza un &lt;form&gt; dupa 'class' si afiseaza valoarea de la atributul 'action'.</p>

<form class='frm3' action='some\_page.php' method='post'>

<p>Acest formular cu class 'frm3' are action: <em id='resp'>#resp</em></p>

Text: <input type='text'/>

</form>

<script>

var frm = document.querySelector('form.frm3');

document.getElementById('resp').innerHTML = frm.action;

</script>

Incercati codul

In afara de proprietatile si metodele specifice elementelor html (prezentate la: [marplo.net/javascript/proprietati-metode-element-html](https://marplo.net/javascript/proprietati-metode-element-html" \o "Proprietati si Metode utile ale elementelor HTML in JavaScript) ), obiectul form contine proprietati si metode proprii.

### Proprietati obiect form

**eform.acceptCharset** - returneaza sau seteaza valoarea atributului **accept-charset**.  
- Atributul 'accept-charset' seteaza tipul de caractere (codarea) valabil pt. acel formular. Daca nu e specificat se aplica codarea documentului.

<h4>Exemplu form.acceptCharset</h4>

<form id='frm1' action='#' accept-charset='utf-8'>

Email: <input type='email' name='email'><br>

Pass: <input type='password' name='password'>

</form>

<p>Afiseaza valoarea proprietatii acceptCharset.</p>

<blockquote id='resp'>#resp</blockquote>

<script>

var frm1 = document.getElementById('frm1');

document.getElementById('resp').innerHTML ='accept-charset = '+ frm1.acceptCharset;

</script>

Incercati codul

**eform.action** - returneaza sau seteaza valoarea atributului **action**.**eform.autocomplete** - returneaza sau seteaza valoarea atributului **autocomplete** din <form>.  
- Cand valoarea e '**on**' (default), browser-ul completeaza automat valori in casete bazat pe valori adaugate anterior de utilizator.  
- Valoarea '**off**' anuleaza aceasta functionalitate.**eform.elements** - contine un obiect tip array cu elementele din formular, in ordinea in care sunt adaugate. acestea pot fi accesate dupa index-ul de ordine (incepand de la 0), sau dupa nume (care e adaugat la atributul 'name').**eform.enctype** - returneaza sau seteaza valoarea atributului 'enctype'.  
- Atributul '***enctype***' indica tipul de continut folosit pentru trimiterea formularului la server; valoarea default e '***application/x-www-form-urlencoded***'.  
- Valoarea '***multipart/form-data***' permite casetei <input type='file'> sa uploadeze fisiere.**eform.length** - contine numarul de elemente din formular.**eform.method** - returneaza sau seteaza valoarea atributului 'method' din <form>.**eform.name** - returneaza sau seteaza valoarea atributului 'name' din <form>.**eform.noValidate** - returneaza True daca atributul 'novalidate' este adaugat in <form>, in caz contrar, False. Defaul e ***true***  
- Daca atributul 'novalidate' e true, datele din formular vor fi validate inainte de a fi transmise.**eform.target** - returneaza sau seteaza valoarea atributului 'target' din <form>.  
- Atributul 'target' indica in ce fereastra sa fie afisate datele returnate din trimiterea formularului (poate fi utilizat atributul 'name' al unui <iframe>).

### Metode obiect form

**eform.reportValidity()** - returneaza True daca datele din formular sunt valide (conform setarilor definite), in caz contrar False.

<h4>Exemplu form.reportValidity()</h4>

<form id='frm1' action='#'>

Nume: <input type='text' pattern='[A-z0-9]{3,}' required placeholder='Minim 3 litere / numere'><br>

Email: <input type='email' required>

</form>

<p> Casetele din formular au atributul 'required' (trebuie completate).<br>

- La clic pe buton se verifica valoarea returnata de metoda reportValidity() si afiseaza raspuns la #resp.</p>

<button id='btn1'>Trimite</button>

<blockquote id='resp'>#resp</blockquote>

<script>

var frm1 = document.getElementById('frm1');

document.getElementById('btn1').addEventListener('click', (ev)=>{

var res = frm1.reportValidity() ?'Date completate corect' :'Completati corect toate casetele.<br>Numele cel putin 3 caractere, doar litere si numere';

document.getElementById('resp').innerHTML = res;

});

</script>

Incercati codul

**eform.reset()** - reseteaza datele din casetele din formular la valorile initiale (la fel ca butonul reset).**eform.submit()** - trimite datele din formular la adresa de la 'action' (la fel ca butonul Submit).

### Evenimente obiect form

Evenimentele se declansaza la efectuarea anumitor actiuni. Pot fi utilizate pentru executia unor functii la actiunea respectiva.

**reset** - se executa cand se reseteaza datele din formular (cu butonul 'reset', sau metoda reset() ).

<h4>Exemplu reset</h4>

<form id='frm1' action='#'>

Text: <input type='text' value='some-val'><br>

Select: <select><option value='gamv'>gamv.eu</option><option selected value='marplo'>marplo.net</option></select><br>

<input type='reset' value='Reset'>

</form>

<p> Modificati datele din formular apoi apasati butonul urmator.</p>

<blockquote id='resp'>#resp</blockquote>

<script>

var frm1 = document.getElementById('frm1');

//se executa cand se reseteaza #frm1

frm1.addEventListener('reset', (ev)=>{

document.getElementById('resp').innerHTML ='Formularul e resetat.';

});

</script>

Incercati codul

**submit** - se executa cand se trimite formular (cu butonul 'submit').  
- E utila pentru validarea datelor din formular sau efectuarea anumitor instructiuni inainte de trimiterea datelor.  
- Cu metoda **ev.preventDefault()** se blocheaza trimiterea automata a formularului ('ev' reprezinta evenimentul declansat, la care e asociata).

<h4>Exemplu submit</h4>

<form id='frm1' action='#'>

Text: <input type='text' name='txt1' placeholder='Minim 3 caractere'><br>

<input type='submit' value='Submit'>

</form>

<p> La trimiterea formularului (clic pe Submit), daca in caseta de text sunt mai putin de 3 caractere, afiseaza mesaj la #resp.<br>

Daca sunt cel putin 3, trimite datele cu metoda submit().</p>

<blockquote id='resp'>#resp</blockquote>

<script>

var frm1 = document.getElementById('frm1');

//se executa cand se trimite #frm1

frm1.addEventListener('submit', (ev)=>{

//opreste trimiterea automata a form-ului

ev.preventDefault();

var tval = frm1['txt1'].value;

if(tval.length <3) document.getElementById('resp').innerHTML ='Adaugati cel putin 3 caractere.';

else frm1.submit();

});

</script>

Incercati codul

**<<-[Obiectul anchor si Lucru ..](https://marplo.net/javascript/obiectul-anchor-adrese-link" \o "Obiectul anchor si Lucru cu adrese din link) -- [Accesare elemente din form](https://marplo.net/javascript/accesare-elemente-form" \o "Accesare elemente din form)->>**

* **[Tutoriale JavaScript](https://marplo.net/javascript/tutoriale-js" \o "Tutoriale JavaScript)**
* **[Vue.js](https://marplo.net/vuejs" \o "Vue.js)**
* **[Curs jQuery](https://marplo.net/javascript/curs-jquery-tutoriale-js" \o "Curs jQuery)**
* **[Tutoriale Node.js](https://marplo.net/javascript/tutoriale-nodejs-js" \o "Tutoriale Node.js)**
* **[Coduri si Functii JavaScript](https://marplo.net/javascript/cod-functii-javascript-js" \o "Coduri si Functii JavaScript)**
* **[Scripturi](https://marplo.net/javascript/scripts" \o "Scripturi)**
* **[JavaScript](https://marplo.net/javascript" \o "JavaScript)**

[Utilizare EventSource pentru evenimente de la server](https://marplo.net/javascript/eventsource-evenimente-server" \o "Utilizare EventSource pentru evenimente de la server)[JavaScript Worker](https://marplo.net/javascript/worker-js" \o "JavaScript Worker)[Creare clase in JavaScript cu Metode care pot fi inlantuite](https://marplo.net/javascript/creare-clase-metode-inlantuite" \o "Creare clase in JavaScript cu Metode care pot fi inlantuite)[Subclase cu extends si Mostenire](https://marplo.net/javascript/subclase-extends-mostenire" \o "Subclase cu extends si Mostenire)[Definire si Utilizare Clase in JavaScript](https://marplo.net/javascript/definire-utilizare-clase-js" \o "Definire si Utilizare Clase in JavaScript)[classList - Lucru cu clase css](https://marplo.net/javascript/classlist-lucru-clase-css" \o "classList - Lucru cu clase css)[Creare scripturi avansate Javascript - PHP](https://marplo.net/javascript/javascript_php.html" \o "Creare scripturi avansate Javascript - PHP)[createElement si insertBefore](https://marplo.net/javascript/createelement_insertbefore.html" \o "createElement si insertBefore)[querySelector si querySelectorAll](https://marplo.net/javascript/queryselector_queryselectorall.html" \o "querySelector si querySelectorAll)[Utilizare getElementsByTagName in JS](https://marplo.net/javascript/getelementsbytagname.html" \o "Utilizare getElementsByTagName in JS)[Utilizare Cookie in JS](https://marplo.net/javascript/cookie.html" \o "Utilizare Cookie in JS)[Lucru cu Imagini in JavaScript](https://marplo.net/javascript/lucru-imagini-js" \o "Lucru cu Imagini in JavaScript)[Detectare si Stergere Evenimente in JS](https://marplo.net/javascript/detectare-stergere-evenimente-js" \o "Detectare si Stergere Evenimente in JS)[Evenimente JavaScript](https://marplo.net/javascript/evenimente.html" \o "Evenimente JavaScript)[Accesare elemente din form](https://marplo.net/javascript/accesare-elemente-form" \o "Accesare elemente din form)[Obiectul anchor si Lucru cu adrese din link](https://marplo.net/javascript/obiectul-anchor-adrese-link" \o "Obiectul anchor si Lucru cu adrese din link)[Obiectul location](https://marplo.net/javascript/obiectul-location" \o "Obiectul location)[window.history](https://marplo.net/javascript/history" \o "window.history)[window.navigator](https://marplo.net/javascript/navigator" \o "window.navigator)[Utilizare getElementById](https://marplo.net/javascript/getelementbyid.html" \o "Utilizare getElementById)[JavaScript HTML DOM - Obiectul document](https://marplo.net/javascript/dom-html-obiect-document" \o "JavaScript HTML DOM - Obiectul document)[Obiectul window](https://marplo.net/javascript/windows.html" \o "Obiectul window)[Obiectul Date - Lucru cu Data si Timp](https://marplo.net/javascript/obiect-date-timp" \o "Obiectul Date - Lucru cu Data si Timp)[Obiectul Array](https://marplo.net/javascript/obiectul-array" \o "Obiectul Array)[Obiectul Math - Metode pentru operatiuni matematice](https://marplo.net/javascript/obiect-math-operatiuni-matematice" \o "Obiectul Math - Metode pentru operatiuni matematice)[Numere in JavaScript](https://marplo.net/javascript/numere-js" \o "Numere in JavaScript)[Obiectul String - Sir](https://marplo.net/javascript/obiectul-string" \o "Obiectul String - Sir)[Utilizare Functii si Parametri lor](https://marplo.net/javascript/functii2.html" \o "Utilizare Functii si Parametri lor)[Definire si Utilizare Functii in JS](https://marplo.net/javascript/definire-utilizare-functii" \o "Definire si Utilizare Functii in JS)[Ferestre Alert Prompt si Confirm](https://marplo.net/javascript/alert_prompt_confirm.html" \o "Ferestre Alert Prompt si Confirm)[break, continue, si eticheta](https://marplo.net/javascript/break-continue-eticheta" \o "break, continue, si eticheta)[Instructiuni repetitive while](https://marplo.net/javascript/instructiuni-repetitive-while" \o "Instructiuni repetitive while)[Instructiuni repetitive for()](https://marplo.net/javascript/instructiuni-repetitive-for" \o "Instructiuni repetitive for())[Instructiuni conditionale if() else, switch](https://marplo.net/javascript/instructiuni_conditionale.html" \o "Instructiuni conditionale if() else, switch)[Operatori in JavaScript](https://marplo.net/javascript/operatori.html" \o "Operatori in JavaScript)[Definire Variabile si Constante](https://marplo.net/javascript/definire-variabile-constante" \o "Definire Variabile si Constante)[Sintaxa JavaScript](https://marplo.net/javascript/sintaxajs.html" \o "Sintaxa JavaScript)[Donation](https://www.paypal.com/cgi-bin/webscr?cmd=_s-xclick&hosted_button_id=KEHLERNATG54U&source=url)

#### Un Test simplu in fiecare zi

##### HTML

##### CSS

##### JavaScript

##### PHP-MySQL

##### Engleza

##### Spaniola

***Clic pe metoda ce creaza un array cu toate elementele din pagina cu un anumit nume de tag.***

**getElementsByName()getElementById()getElementsByTagName()**

[Sharing Twitter](https://twitter.com/intent/tweet?text=Obiectul%20form%20in%20JavaScript&url=https://marplo.net/javascript/obiect-form&via=CoursesWeb)

#### Last accessed pages

1. **Curs PHP MySQL, Tutoriale si Scripturi PHP (41500)**
2. **PHP MySQL - Introducere si Tipuri de Date (1573)**
3. **Obiectul anchor si Lucru cu adrese din link (246)**
4. **Obiectul location (284)**
5. **Blog si Cugetari Personale (28582)**

#### Popular pages this month

1. **Cursuri si Tutoriale: Engleza, Spaniola, HTML, CSS, Php-Mysql, JavaScript, Ajax (4004)**
2. **Curs HTML gratuit Tutoriale HTML5 (3576)**
3. **Curs si Tutoriale JavaScript (2686)**
4. **Curs PHP MySQL, Tutoriale si Scripturi PHP (2651)**
5. **Curs si Tutoriale Ajax (2527)**

##### Site access statistics

Chat

**[Home](https://marplo.net/" \o "Home)** **[Contact](https://marplo.net/contact" \o "Contact)** **[Engleza](https://marplo.net/engleza" \o "Engleza)** **[Spaniola](https://marplo.net/spaniola" \o "Spaniola)** **[Html](https://marplo.net/html" \o "Html)** **[CSS](https://marplo.net/css" \o "CSS)** **[JavaScript](https://marplo.net/javascript" \o "JavaScript)** **[PHP-MySQL](https://marplo.net/php-mysql" \o "PHP-MySQL)** **[Ajax](https://marplo.net/ajax" \o "Ajax)** **[Flash-AS3](https://marplo.net/flash" \o "Flash-AS3)** **[PHP Laravel - Tutoriale](https://marplo.net/php-mysql/laravel-tutoriale" \o "PHP Laravel - Tutoriale)** **[Vue.js](https://marplo.net/vuejs" \o "Vue.js)** **[Tutoriale Node.js](https://marplo.net/javascript/tutoriale-nodejs-js" \o "Tutoriale Node.js)** **[Blog](https://marplo.net/blog" \o "Blog)**- WebMaster, WebDesigner : MarPlo -

# Accesare elemente din form

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

In **<form>** se pot adauga numeroase tipuri de elemente HTML de formular pentru diferite tipuri de date care pot fi adaugate sau selectate de utilizator.  
- Vedeti tutorialele HTML: ***[Creare formulare](https://marplo.net/html/formulare.html" \o "Creare formulare)*** si ***[HTML5 - Elemente si atribute noi in form](https://marplo.net/html/html5-formular-input-atribute.html" \o "HTML5 - Elemente si atribute noi in formular)***.

### Preluare elemente din <form> in JavaScript

Sunt mai multe moduri prin care elementele HTML dintr-un <form> pot fi preluate ca obiecte in JavaScript.  
In general se folosesc metodele: **getElementById()**, **querySelector()**, formula: **eform['elm\_name']**, sau proprietatea: **eform.elements[index]**.

* Daca elementul din <form> are un ID, se poate accesa cu oricare din aceste sintaxe:

**document.getElementById('id\_elm')  
document.querySelector('#id\_elm')**

- Exemplu, preia dupa ID caseta de text dintr-un <form> si afiseaza valoarea de la 'value'.

<h4>Accesare input dupa ID</h4>

<form action='#' method='post'>

Text: <input type='text' value='some-val' id='txt1'/>

</form>

<p>La clic pe buton, acceseaza dupa ID caseta text din Form si afiseaza la #resp valoarea de la 'value'.</p>

<button id='btn1'>Click</button>

<blockquote id='resp'>#resp</blockquote>

<script>

document.getElementById('btn1').addEventListener('click', (ev)=>{

var ftxt = document.getElementById('txt1');

document.getElementById('resp').innerHTML = ftxt.value;

});

</script>

Incercati codul

* Daca elementul din <form> are un atribut ***name***, se poate accesa cu aceasta sintaxa:

**eform['elm\_name']**

Unde ***eform*** reprezinta obiectul formularului, iar '***elm\_name***' numele elementului din acel formular.  
- Exemplu, preia dupa '***name***' caseta de text dintr-un <form> si afiseaza valoarea de la 'value'.

<h4>Accesare input dupa name</h4>

<form id='frm1' action='#' method='post'>

Text: <input type='text' value='some-val' name='txt1'/>

</form>

<p>La clic pe buton, acceseaza dupa 'name' caseta text din Form si afiseaza la #resp valoarea de la 'value'.</p>

<button id='btn1'>Click</button>

<blockquote id='resp'>#resp</blockquote>

<script>

document.getElementById('btn1').addEventListener('click', (ev)=>{

var frm1 = document.getElementById('frm1');

document.getElementById('resp').innerHTML = frm1['txt1'].value;

});

</script>

Incercati codul

* Daca elementul din <form> are o clasa css (atribut ***class***), se poate accesa cu oricare din aceste sintaxe:

**document.querySelector('#id\_form .clas\_elm')  
eform.querySelector('.clas\_elm')**

Unde '***id\_form***' este id-ul formularului in care e acel element, iar ***clas\_elm*** este clasa elementului respectiv.  
- ***eform*** reprezinta obiectul formularului.

*Daca in acel <form> sunt mai multe elemente cu acelasi 'class', metoda****querySelector()****va returna pe primul.*

- Exemplu, preia dupa '***class***' caseta de text dintr-un <form> si afiseaza valoarea de la 'value'.

<h4>Accesare input dupa class</h4>

<form id='frm1' action='#' method='post'>

Text: <input type='text' value='some-val' class='ftxt'/>

</form>

<p>La clic pe buton, acceseaza dupa 'class' caseta text din Form si afiseaza la #resp valoarea de la 'value'.</p>

<button id='btn1'>Click</button>

<blockquote id='resp'>#resp</blockquote>

<script>

document.getElementById('btn1').addEventListener('click', (ev)=>{

var ftxt = document.querySelector('#frm1 .ftxt');

document.getElementById('resp').innerHTML = ftxt.value;

});

</script>

Incercati codul

* Pentru accesarea unui element din <form> in functie de numarul de ordine (index care incepe de la 0), se pot folosi oricare din aceste sintaxe:

**eform[index]  
eform.elements[index]  
eform.item(index)**

Unde '***eform***' reprezinta formularul in care e acel element, iar ***index*** e numarul de ordine (in functie de ordinea adaugarii elementelor in <form>).  
- Exemplu, preia primul element dintr-un <form> si afiseaza tipul lui (valoarea 'type').

<h4>Preluare input dupa index de ordine</h4>

<form id='frm1' action='#' method='post'>

Check: <input type='checkbox' value='some-val'/><br>

Text: <input type='text' value='txt-val'/>

</form>

<p>La clic pe buton, acceseaza primul element din Form si afiseaza la #resp tipu lui (valoarea de la 'type').</p>

<button id='btn1'>Click</button>

<blockquote id='resp'>#resp</blockquote>

<script>

document.getElementById('btn1').addEventListener('click', (ev)=>{

var ftxt = document.getElementById('frm1');

document.getElementById('resp').innerHTML = ftxt[0].type;

});

</script>

Incercati codul

Elementele de formular au proprietati si metode specifice care pot fi utilizate in JavaScript pentru interactiunea cu utilizatorul.  
- O lista cu proprietati si metode utile ale elementelor din <form> gasiti la pagina de la adresa:  
**[marplo.net/javascript/proprietati-metode-elemente-form](https://marplo.net/javascript/proprietati-metode-elemente-form" \o "Proprietati si Metode elemente din form)**

# Evenimente JavaScript

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Utilizare Evenimentele in cod JavaScript](https://marplo.net/javascript/evenimente.html" \l "hshuej" \o "Utilizare Evenimentele in cod JavaScript)

Evenimentele reprezinta actiuni provocate de browser sau de vizitatorul paginii.  
De exemplu: cand pagina incepe sa se incarce in fereastra are loc evenimentul "Load", daca vizitatorul apasa un buton din pagina se provoaca evenimentul "Click". Cand mouse-ul este deasupra unui element, se declanseaza un eveniment "mouseover".

O lista cu evenimente JavaScript organizate pe categorii gasiti la pagina de la adresa:  
**[marplo.net/javascript/lista-evenimente-js](https://marplo.net/javascript/lista-evenimente-js" \o "Lista evenimente JavaScript)**

### Atribute HTML pentru evenimente

JavaScript poate reactiona la evenimente cu ajutorul "**event-handlers**" (manageri sau gestionar de evenimente).  
Handlerele de evenimente se pot adauga ca atribute in tag-urile HTML, folosind urmatoarea sintaxa.

**<tag event='cod-JS'>**

Unde '***event***' reprezinta numele evenimentului, cu 'on' la inceput (onclick, onload, ..); iar '***cod-JS***' poate fi in general o functie JavaScript care se executa la declansarea evenimentului respectiv.  
  
- Exemplu, cand mouse-ul e pozitionat deasupra unui Div se apeleaza o functie care schimba culoarea 'background', iar cand se apasa clic pe acel Div se executa alta functie care adauga alt text in el.

*Cuvantul****this****din cod reprezinta elementul pe care se actioneaza evenimentul, si e transmis ca argument la functia apelata.*

<style>

#dv1 {

background:#b0deb0;

font-size:20px;

height:50px;

padding:3% 8px 5px 8px;

text-align:center;

width:120px;

}

</style>

<h4>Exemplu onmouseover si onclick</h4>

<div id='dv1' onmouseover='changeBgr(this)' onclick='changeTxt(this)'>Click Aici.</div>

<script>

function changeBgr(elm){

elm.style.background ='#bebefe';

}

function changeTxt(elm){

elm.innerHTML ='Pacea-i Buna.';

}

</script>

Incercati codul

- Demo:

Click Aici.

Evenimentele pot fi utilizate si la elemente de formular.  
Daca dorim sa apelam o functie de fiecare data cand se adauga ceva intr-o caseta <input>, se poate folosi evenimentul '***oninput***'.  
- Exemplu, cand se adauga caractere intr-o caseta de text, se apeleaza o functie care copie acel text in alt element HTML, si afiseaza numarul de caractere adaugate.

<h4>Exemplu oninput</h4>

<p>Cand se scrie ceva in caseta de text, e apelata o functie care afiseaza numarul de caractere si adauga textul la #resp.</p>

Text: <input type='text' value='Liniste' oninput='addTxt(this)'/>

<blockquote id='resp'>#resp</blockquote>

<script>

var resp = document.getElementById('resp');

function addTxt(elm){

var txt = elm.value;

resp.innerHTML ='<p>Nr. caractere: '+ txt.length +'</p>'+ txt;

}

</script>

Incercati codul

### Utilizare Evenimentele in cod JavaScript

Pentru a nu amesteca tag-urile HTML cu coduri JS, evenimentele se pot adauga in codul JavaScript, asociate la elementul HTML preluat in JS; folosind aceasta sintaxa.

**function funcName(ev){**

**// ev reprezinta obiectul cu evenimentul declansat**

**//codul functiei**

**}**

**elm.event = funcName;**

Unde ***elm*** reprezinta elementul HTML la care se adauga evenimentul '***event***'.  
La definire, functia ***funcName*** poate avea un parametru (aici 'ev'); si reprezinta obiectul cu evenimentul care apeleaza functia. Aceasta se apeleaza doar cu numele, fara paranteze rotunde.  
  
• Sau cu functie anonima:

**elm.event = function(ev){**

**// ev reprezinta obiectul cu evenimentul declansat**

**//codul functiei**

**};**

Iata un exemplu util cu '***onkeyup***'.  
- Muta automat cursorul in urmatoarea caseta <input> atunci cand s-a completat tot campul precedent (cand se ajunge la lungimea '***maxlength***').

<style>

#frm1 input {

background:#b0dfb0;

font-size:16px;

text-align:center;

width:50px;

}

#frm1 input:focus {

background:#ededaf;

}

</style>

<h4>Exemplu onkeyup</h4>

<p>Cursorul se muta automat in urmatoarea caseta de text cand s-a completat tot campul precedent (la lungimea 'maxlength' a campului, aici 4 caractere).</p>

<form id='frm1'>

Adaugati text: <input maxlength='4' autofocus> / <input maxlength='4'> / <input maxlength='4'>

</form>

<script>

//functia apelata de onkeyup; primeste indexul de ordine al elementului in form

function nextElm(ev){

var elm = ev.target; //elementul curent, care declanseaza evenimentul

# Detectare si Stergere Evenimente in JS

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Cuvantul this](https://marplo.net/javascript/detectare-stergere-evenimente-js" \l "hshct" \o "Cuvantul this)
* [Detectare eveniment cu addEventListener()](https://marplo.net/javascript/detectare-stergere-evenimente-js" \l "hsheal" \o "Detectare eveniment cu addEventListener)
* [Stergere evenimente inregistrate cu addEventListener()](https://marplo.net/javascript/detectare-stergere-evenimente-js" \l "hsheals" \o "Stergere evenimente inregistrate cu addEventListener)

Evenimentele sunt actiuni efectuate de utilizator in pagina web.  
De exemplu, click-ul sau pozitionarea cursorului pe un element, scrierea intr-o caseta text input, sau derularea paginii, sunt evenimente efectuate de utilizator.  
Pentru a putea executa un anumit cod JS cand un eveniment e declansat, JavaScript contine asa numitele '**event handlers**' (manipulatoare de evenimente), precum: click, mouseover, keypress, scroll, etc.

- O lista cu evenimente JavaScript grupate pe categorii gasiti la pagina de la adresa:  
**[marplo.net/javascript/lista-evenimente-js](https://marplo.net/javascript/lista-evenimente-js" \o "Lista evenimente JavaScript)**

## Inregistrare event handlers

Metoda veche de a executa anumite instructiuni, sau o functie cand un eveniment e declansat, e prin adaugarea acestuia in tag-ul HTML in care evenimentul trebuie detectat.  
De exemplu, cand un utilizator apasa click pe acest link, 'onclick' detecteaza aceasta actiune si apeleaza functia *oFunctie()*:

<a href='#' onclick='oFunctie();' title='Text'>Link</a>

• Se pot detecta evenimente fara a-le adauga in tag-ul HTML, ci direct in codul JavaScript, folosind aceasta sintaxa:

**function funcName(ev){**

**// ev reprezinta obiectul cu evenimentul declansat**

**//codul functiei**

**}**

**elm.event = funcName;**

Unde ***elm*** reprezinta elementul HTML la care se adauga evenimentul '***event***'.  
La definire, functia ***funcName*** poate avea un parametru (aici 'ev'); si reprezinta obiectul cu evenimentul care apeleaza functia. Observati ca aceasta se apeleaza doar cu numele, fara paranteze rotunde.  
  
- Exemplu, la click pe un anumit DIV, e afisata o fereastra Alert cu ID-ul acelui Div:

<h4>Exemplu onclick</h4>

<div id='dv\_1' style='background:#a8eda9; cursor:pointer; padding:8px; 12px; width:125px'>Click here</div>

<script>

var elm = document.getElementById('dv\_1'); // preia elementul

// functie executata cand se produce evenimentul

function oFunctie(ev){

// ev.target reprezinta elementul care a declansat evenimentul

let id = ev.target.id;

alert('DIV id: '+id);

}

// inregistrare event onclick

elm.onclick = oFunctie;

</script>

Incercati codul

#### Evenimente cu functie anonima

Functia apelata la declansarea evenimentului poate fi creata direct in expresia de inregistrare a eveniment.  
Sintaxa:

**elm.event =(ev)=>{**

**// ev reprezinta obiectul cu evenimentul declansat**

**//codul functiei**

**};**

- Iata exemplu anterior, aici cu functie anonima:

<h4>Exemplu onclick cu functie anonima</h4>

<div id='dv\_1' style='background:#a8eda9; cursor:pointer; padding:8px; 12px; width:125px'>Click here</div>

<script>

var elm = document.getElementById('dv\_1'); // preia elementul

// inregistrare event onclick

elm.onclick =(ev)=>{

// ev.target reprezinta elementul care a declansat evenimentul

let id = ev.target.id;

alert('DIV id: '+id);

};

</script>

Incercati codul

#### Stergere detectare eveniment cu 'on'

• Pentru a anula detectarea unui eveniment care e atasat la elementul HTML (cele adaugate cu prefixul 'on'), i-se atribuie valoarea **null**.

**element.event\_handler = null;**

- Exemplu. Cand utilizatorul plaseaza cursorul peste DIV-ul cu id='dvid', se afiseaza o fereastra Alert, apoi se sterge detectarea 'onmouseover' la acel element, astfel, cand utilizatorul muta a doua oara cursorul peste acel DIV nu se intampla nimic.

<h4>Exemplu anulare eveniment</h4>

<p>Dupa prima emitere a evenimentului 'onmouseover', acesta e anulat.</p>

<div id='dvid' style='background:#a8eda9; cursor:pointer; padding:8px; 12px; width:125px'>Mutati cursorul aici.</div>

<script>

var elm = document.getElementById('dvid'); // preia elementul

// inregistrare onmouseover

elm.onmouseover =(ev)=>{

alert('Plasati inca o data cursorul pe acel text');

elm.onmouseover = null; // sterge inregistrarea evenimentului

};

</script>

Incercati codul

### Cuvantul this

In JavaScript, cuvantul **this** (acesta) reprezinta mereu 'proprietarul' functiei. In cazul evenimentelor, **this** reprezinta elementul HTML care declanseaza evenimentul.  
  
- Exemplu. La click pe tag-ul cu id='dvid', JavaScript preia continutul din el, il copie intr-un <textarea>, apoi ii modifica culoarea background.

<h4>Exemplu cu this</h4>

<p>La clic pe Div, copie continutul in textarea apoi schimba culoarea background.</p>

<div id='dvid' style='background:#a8eda9; cursor:pointer; width:125px; padding:8px 12px;'>Div, Exemplu cu this</div>

<textarea id='txta'></textarea>

<script>

var elm = document.getElementById('dvid');

// inregistrare onclick

elm.onclick = function(){

// preia continutul HTML din elementul curent (cel ce declanseaza evenimentul)

let cnt = this.innerHTML;

document.getElementById('txta').value = cnt;

// setare culoare background la elementul curent

this.style.background ='#bebefe';

};

</script>

Incercati codul

#### this si parametru event in functii arrow

Cuvantul **this** nu e recunoscut in ***functii arrow*** (definite cu: **()=>{}** ) cu aceeasi valoare.  
In functia arrow, ***this*** reprezinta obiectul parinte al functiei (in care e definita).  
In functiile arrow (ca in toate celelalte functii) se poate folosi parametru **event** (aici definit cu 'ev'). Acesta contine un obiect cu evenimentul respectiv; si are o proprietate: **target** ce contine obiectul care a declansat acel eveniment.  
- In functiile arrow se poate folosi urmatoarea sintaxa.

**var elm = event.target;**

- Iata exemplu precedent, aici cu functie arrow si parametru event (ev):

<h4>Exemplu cu functie arrow si parametru event</h4>

<p>La clic pe Div, copie continutul in textarea apoi schimba culoarea background.</p>

<div id='dvid' style='background:#a8eda9; cursor:pointer; width:125px; padding:8px 12px;'>Div, Exemplu cu event.target</div>

<textarea id='txta'></textarea>

<script>

var elm = document.getElementById('dvid');

// inregistrare onclick cu functie arrow

elm.onclick =(ev)=>{

// preia continutul HTML din elementul care declanseaza evenimentul

let cnt = ev.target.innerHTML;

document.getElementById('txta').value = cnt;

// setare culoare background la elementul declansator

ev.target.style.background ='#bebefe';

};

</script>

Incercati codul

### Detectare eveniment cu addEventListener()

Exista inca un mod de a inregistra detectare de evenimente in JS, si anume cu metoda **addEventListener()**.  
La aceasta metoda evenimentul se adauga fara prefixul'on' (*'click', 'mouseenter', 'mouseleave', ...*), folosind urmatoarea sintaxa:

**element.addEventListener('event', oFunctie, use\_capture);**

- *event* - e un sir cu tipul de eveniment ce trebuie detectat, fara prefixul '*on*'.  
- *oFunctie* - o functie accesata cand 'event' e declansat.  
- *use\_capture* - (optional) o valoare booleana (true sau false). Determina daca evenimentul trebuie executat in faza de capturare (true) sau in faza de 'bubbling' (false). Default e: false.  
  
- Exemplu. Cand mouse-ul e deasupra unui element LI, capata o culoare de fundal verde, cand cursorul iese din zona acelui <li> se elimina culoarea de fundal.

<h4>Exemplu cu addEventListener()</h4>

<p>Miscati mouse-ul pe listele urmatoare.</p>

<ul>

<li>WebDevelopment - //coursesweb.net </li>

<li>Cursuri gratuite - //marplo.net </li>

<li>Games - //gamv.eu </li>

</ul>

<script>

var elm\_li = document.getElementsByTagName('li'); // preia toate tag-urile LI

// functie executata la mouseenter

function mEnter(ev){

// seteaza o culoare de fundal

ev.target.style.background ='#07da08';

}

// functie executata la mouseleave

function mLeave(ev){

// sterge culoarea de fundal

this.style.background ='none';

}

// parcurge obiectul cu elementele LI

for(var i=0; i<elm\_li.length; i++){

// inregistrare mouseenter si mouseleave la fiecare LI

elm\_li[i].addEventListener('mouseenter', mEnter);

elm\_li[i].addEventListener('mouseleave', mLeave);

}

</script>

Incercati codul

### Stergere evenimente inregistrate cu addEventListener()

• Pentru a anula detectare unui eveniment inregistrat cu **addEventListener()**, se foloseste metoda **removeEventListener()**.  
Sintaxa:

**element.removeEventListener('event', oFunctie);**

- **oFunctie** trebuie sa fie aceeai care e adaugata la addEventListener().  
  
Iata un exemplu, dupa un anumit numar de declansari se anuleaza detectarea evenimentului 'mousemove' inregistrat la un Div.

<h4>Exemplu sterge detectare eveniment cu addEventListener()</h4>

<p>Cand mouse-ul e miscat pe urmatorul Div, se afiseaza numarul de emiteri a evenimentului 'mousemove'<br>

Cand contoarul ajnge la 12 se anuleaza detectarea acestui eveniment.</p>

<div id='dvid' style='background:#a8eda9; padding:8px; 12px; text-align:center; width:130px'>

Div - Miscati cursorul aici.<br>

<strong id='resp'>0</strong>

</div>

<script>

var elm = document.getElementById('dvid');

var resp = document.getElementById('resp');

var nre =0;

//apelata la mousemove

function mOver(ev){

nre++;

resp.innerHTML = nre;

if(nre ==12) elm.removeEventListener('mousemove', mOver); // sterge inregistrarea evenimentului

}

// inregistrare mousemove

elm.addEventListener('mousemove', mOver);

</script>

# Lucru cu Imagini in JavaScript

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Obiectul Image](https://marplo.net/javascript/lucru-imagini-js" \l "hshoi" \o "Obiectul Image)
* [Proprietati obiect Image](https://marplo.net/javascript/lucru-imagini-js" \l "hshpoi" \o "Proprietati obiect Image)
* [Preluare mai multe imagini dintr-un element HTML](https://marplo.net/javascript/lucru-imagini-js" \l "hshpmi" \o "Preluare mai multe imagini dintr-un element HTML)
* [Schimbare Imagine cu alta](https://marplo.net/javascript/lucru-imagini-js" \l "hshsi" \o "Schimbare Imagine cu alta)
* [Preincarcare imagini](https://marplo.net/javascript/lucru-imagini-js" \l "hshpi" \o "Preincarcare imagini)

#### Array-ul images

In JavaScript toate imaginile dintr-o pagina web se gasesc intr-un array denumit **images**, care apartine obiectului **document**.  
Imaginile sunt adaugate in array-ul **images** in ordinea in care sunt adaugate in pagina, cu un index de ordine incepand de la 0.  
Prima imagine din pagina se afla in array-ul **images** cu index 0, a doua imagine are index 1, si tot asa.  
Astel, se poate face referire la prima imagine folosind urmatoarea expresie.

var img1 = document.images[0];

Numarul de imagini se poate afla cu proprietatea **length**.  
Astel, se poate face referire la ultima imagine folosind urmatorul cod.

var nr\_imgs = document.images.length;  
var last\_img = document.images[nr\_imgs -1];

### Preluare imagini cu metode JavaScript

O anumita imagine din pagina poate fi preluata mai clar in codul JavaScript folosind metoda **getElementById()** sau **querySelector()**.

• Daca tag-ul <img> are un ID, se poate folosi **getElementById()**:

**var img = document.getElementById('id\_img');**

Unde '***id\_img***' este id-ul imaginii (adaugat la atributul **id**).  
  
• Daca tag-ul <img> are o clasa (adaugata la atributul **class**), se poate folosi **querySelector()**:

**var img = document.querySelector('css\_sel');**

Unde '***css\_sel***' reprezinta un selector CSS care face referire la imagine.  
  
- Exemplu, preia imaginea cu class 'cls\_im' care se afla intr-un Div cu id 'dv1', iar la clic pe ea afiseaza adresa de la 'src' intr-un element HTML.

<h4>Exemplu preluare imagine cu querySelector</h4>

<p>La click pe imagine adauga la #resp adresa ei de la 'src'.</p>

<div id='dv1'> Div<br>

<img src='javascript/imgs/smile\_gift.png' height='115' width='130' alt='Smile' class='cls\_im'/>

</div>

<blockquote id='resp'>#resp</blockquote>

<script>

//preia prima imagine cu class .cls\_im din #dv1

var img = document.querySelector('#dv1 img.cls\_im');

//detecteaza click pe imaginea preluata

img.addEventListener('click', (ev)=>{

document.getElementById('resp').innerHTML = img.src;

});

</script>

Incercati codul

*Daca sunt mai multe imagini cu acelasi 'class' in elementul specificat, querySelector() va prelua pe prima.*

### Obiectul Image

Fiecare imagine dintr-un document HTML reprezinta in JavaScript un obiect **Image**.  
Obiectul **Image** are proprietati specifice pentru lucru cu imagini.  
- De exemplu, cu proprietatea **height** se poate prelua sau modifica inaltimea imaginii afisata in pagina, iar cu proprietatea **width** se preia sau se modifica din JS lungimea imaginii.

<h4>Exemplu dimensiuni imagine</h4>

<p>La clic pe butonul 'Set size' mareste dimensiunile imaginii (height si width) cu 50%.</p>

<img src='javascript/imgs/smile\_gift.png' alt='Smile' height='115' width='130' id='img1'/><br>

<button id='btn1'>Set size</button>

<script>

var img = document.getElementById('img1');

document.getElementById('btn1').addEventListener('click', (ev)=>{

//preia inaltimea si lungimea imaginii

let h = img.height;

let w = img.width;

//seteaza inaltimea si lungimea cu 50% mai mari

img.height = h \*1.5;

img.width = w \*1.5;

});

</script>

Incercati codul

#### Proprietati obiect Image

**alt** - returneaza sau seteaza valoarea atributului 'alt'.

<h4>Exemplu alt</h4>

<p>La clic pe imagine, afiseaza la #resp valoarea atributului 'alt'.</p>

<img src='javascript/imgs/smile\_gift.png' height='115' width='130' alt='Zambeste' id='im1'/>

<blockquote id='resp'>#resp</blockquote>

<script>

var img = document.getElementById('im1');

img.addEventListener('click', (ev)=>{

document.getElementById('resp').innerHTML = ev.target.alt;

});

</script>

Incercati codul

**complete** - returneaza True daca imaginea s-a incarcat complet, in caz contrar, False.**height** - returneaza sau seteaza valoarea atributului 'height'.**isMap** - returneaza True daca imaginea are setat atributul 'ismap', in caz contrar, False.**naturalHeight** - returneaza inaltimea originala a imaginii.**naturalWidth** - returneaza lungimea originala a imaginii.**src** - returneaza sau seteaza valoarea atributului 'src'.**useMap** - returneaza sau seteaza valoarea atributului 'usemap' dintr-o imagine.**width** - returneaza sau seteaza valoarea atributului 'width'.

### Preluare mai multe imagini dintr-un element HTML

Daca intr-un element HTML aveti mai multe imagini si vreti sa le preluati in JavaScript, se poate folosi una din aceste metode: **getElementsByTagName('img')** sau **querySelectorAll('css\_sel')** ('*css\_sel*' e selectorul CSS ce reprezinta imaginile).  
Ambele metode returneaza un array cu elementele preluate, si pot fi parcurse cu instructiunea **for()**.

- Exemplu cu **getElementsByTagName('img')**:

//preia toate imaginile dintr-un element cu id 'dv1'

var imgs = document.getElementById('dv1').getElementsByTagName('img');

//parcurge imaginile

for(var i=0; i<imgs.length; i++){

//cod executat la fiecare imagine parcursa

}

- Exemplu cu **querySelectorAll()**:

//preia toate imaginile cu class 'cls\_im' dintr-un element cu id 'dv1'

var imgs = document.querySelectorAll('#dv1 img.cls\_im');

//parcurge imaginile

for(var i=0; i<imgs.length; i++){

//cod executat la fiecare imagine parcursa

}

### Schimbare Imagine cu alta

Daca doriti sa schimbati o imagine din pagina cu una de la o alta adresa, se poate simplu, modificand adresa de la '**src**' cu cea noua.

**var img = document.getElementById('id\_img');  
img.src ='adresa/alta\_imagine.jpg';**

- Iata un exemplu practic. La clic pe un buton se schimba in acelasi loc imagini cu adrese dintr-un array.

<h4>Exemplu schimbare imagini</h4>

<p>La click pe butonul 'Schimba imaginea' se afiseaza o noua imagine cu adresa preluata dintr-un array cu 4 adrese.</p>

<img src='javascript/imgs/sunshine.jpg' id='im1' height='235' width='340' alt='Image'/><br>

<button id='btn1'>Schimba imaginea</button>

<script>

//array cu imagini

var imgs =['sunshine.jpg', 'spring\_dream.jpg', 'spring\_dream1.jpg', 'waterfall.jpg'];

var icm =0; //idexul imaginii curente

var img = document.getElementById('im1');

document.getElementById('btn1').addEventListener('click', (ev)=>{

icm++;

if(icm >= imgs.length) icm =0; //trece la 0 dupa ce ajunge la ultima

img.src ='javascript/imgs/'+ imgs[icm];

});

</script>

Incercati codul

### Preincarcare imagini

In exemplul precedent, la clic pentru schimbare imagine browserul trebuie sa astepte afisare noi imagini pana cand aceasta este incarcata; iar in cazul unor imagini mari asteptarea poate dura prea mult. Acest lucru poate fi evitat prin "preincarcarea imaginilor", folosind o instanta de obiect **Image**.  
Sintaxa:

**var obimg = new Image();  
obimg.src ='adresa\_img.jpg'; //preincarca imaginea**

- Acest cod incarca imaginea in JavaScript, fiind valabila spre afisare rapida din obiectul 'obimg'.  
  
- Iata un exemplu practic. Cand mouse-ul intra pe suprafata unei imagini se schimba imaginea, cand mouse-ul iese din suprafata ei se adauga o alta imagine.  
Pentru o afisare imediata, imaginile sunt preincarcate cu obiectul ***Image***.

<h4>Exemplu schimbare imagini la trecere cu mouse-ul</h4>

<p>Treceti cu mouse-ul pe urmatoarea imagine, apoi mutati-l in afara ei.</p>

<img src='javascript/imgs/spring\_dream.jpg' id='im1' height='235' width='340' alt='Image'/>

<script>

//obiect cu imagini pt. enter si leave

var imgs ={enter:'spring\_dream1.jpg', leave:'waterfall.jpg'};

var obimg ={}; //va contine imaginile preincarcate

//pt. preincarcare imaginilor din imgs

function preloadImgs(imgs){

//parcurge imgs si seteaza in obimg obiecte Image cu fiecare (cu adresa preincarcata)

for(var k in imgs){

obimg[k] = new Image();

obimg[k].src ='javascript/imgs/'+ imgs[k];

}

}

//apeleaza functia cu imaginile ce trebuie preincarcate

preloadImgs(imgs);

//inregistreaza eveniente 'mouseenter' si 'mouseleave' la img

var img = document.getElementById('im1');

# Utilizare Cookie in JS

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Preluare valoare din cookie](https://marplo.net/javascript/cookie.html" \l "hshpc" \o "Preluare valoare din cookie)
* [Creare cookie](https://marplo.net/javascript/cookie.html" \l "hshsc" \o "Creare cookie)
* [Stergere cookie](https://marplo.net/javascript/cookie.html" \l "hshdc" \o "Stergere cookie)
* [Exemplu complet utilizare cookie](https://marplo.net/javascript/cookie.html" \l "hshexc" \o "Exemplu complet utilizare cookie)

Valorile majoritatii variabilelor dintr-un script dispar atunci cand fereastra navigatorului este inchisa. Spre deosebire de acestea, valorile variabilelor cookie se pot pastra un timp indefinit. Pentru ca valorile lor sa se poata pastra, browserul utilizatorului stocheaza variabilele cookie in calculatorul utilizatorului.  
Astfel, cookie-urile sunt fisiere care contin date din site salvate pe calculatorul vizitatorului.

### Despre cookie

Un cookie consta in principal dintr-o pereche **nume=valoare**, iar caracteristici mai avansate permit stabilirea unei date de expirare si pot preciza ce pagini web vad informatia cookie.  
Un fisier cookie poate persista luni de zile, sau ani (daca nu e sters), simplificand vizitele ulterioare ale utilizatorului pe site, daca informatii referitoare la vizite si preferintele utilizatorului sunt salvate si preluate din cookie de fiecare data cand va reveni la site.  
JavaScript are functii pentru citirea, adaugarea si editarea fisierelor cookie.

#### Dezavantaje cookie

Fisierele cookie au si unele dezavantaje:  
1. Fiind stocate pe calculatorul utilizatorului, acestea poat fi sterse din gresela (sau intentionat).  
2. Browserul impune restrictii privind dimensiunea si numarul de fisiere cookie care pot fi stocate, iar fisierele cookie mai noi le pot suprascrie pe cele vechi.  
Daca aceste limite sunt depasite, browserul va sterge cele mai vechi cookie-uri si nefolosite.

- Numarul total de cookie-uri pentru un server sau domeniu este de 200.  
- Marimea maxima a unui fisier cookie este 4 kb.

3. Cand un utilizator trece de la un browser la altul, fisierele cookie salvate de un browser nu sunt recunoscute de celalalt.  
4. Daca mai multi utilizatori folosesc acelasi calculator si acelasi browser ei pot folosi fisiere cookie care apartin altcuiva.  
5. Informatiile private precum parole, numere personale nu trebuie stocate direct intr-un cookie deoarece fisierele cookie pot fi citite si de alte programe.  
6. Utilizatorul poate configura browserul sa interzica anexarea de cookie, in acest caz aplicatia care foloseste cookie ar putea sa nu functioneze.

### Preluare valoare din cookie

Numele si valorile dintr-un cookie sunt stocate si stabilite utilizand proprietatea **cookie** a obiectului **document**.  
Pentru a prelua sirul cookie intr-o variabila se foloseste o instructiune ca aceasta:

**var myCookie = document.cookie;**

**document.cookie** returneaza un sir cu toate cookie-urile valabile in acea pagina. Un sir ca acesta:

nume1=valoare1; nume2=valoare2; nume3=valoare3

Unde *'nume1', 'nume2', 'nume3'* reprezinta numele fiecarui cookie, iar *'valoare1', 'valoare2', 'valoare3'* reprezinta valorile stocate in fiecare cookie.  
  
- Exemplu:

<h4>Exemplu document.cookie</h4>

<p>La click pe buton se afiseaza la #resp sirul returnat de <em>document.cookie</em>.</p>

<button id='btn1'>Cookie</button>

<blockquote id='resp'>#resp</blockquote>

<script>

document.getElementById('btn1').addEventListener('click', (ev)=>{

document.getElementById('resp').innerHTML = document.cookie;

});

</script>

Incercati codul

*Perechile****nume=valoare****sunt separate prin caracterul punct si virgula ";" si un spatiu, iar ultima pereche nu are caracterul punct si virgula.*

Pentru extragerea si utilizarea valorilor din fisierele cookie, trebuie sa prelucrati in JavaScript sirul obtinut cu: **document.cookie**.  
Pentru a simplifica preluarea valorii unui cookie, puteti folosi functia **getCookie()** din urmatorul cod JS:

function getCookie(name){

var re ='';

var str\_c =' '+ document.cookie +';'; //pune sirul de cookie intre un spatiu si ;

var name =' '+ name +'='; //sa caute ce e intre name si =

var start\_c = str\_c.indexOf(name); //unde incepe name

var end\_c;

if(start\_c != -1) { // daca exista numele transmis

start\_c += name.length; // unde incepe valoarea

end\_c = str\_c.indexOf(';', start\_c); //unde se termina valoarea

re = decodeURIComponent(str\_c.substring(start\_c, end\_c)); //valoarea

}

return re;

}

var val\_c = GetCookie('nume\_cookie');

document.write(val\_c); // Afiseaza valoarea cookie-ului cautat, sau sir gol ''

### Creare cookie

Perechea **nume=valoare** este informatia minima de care aveti nevoie pentru a defini un cookie, dar se pot adauga si alti parametri.  
Iata o lista de parametrii ce pot fi utilizati pentru a seta un cookie:

**nume=valoare** - numele si valoarea salvata in cookie (numele sa contina doar litere, numere, \_ ).- Exemplu:

var val\_c ='value';

document.cookie ='some\_name='+ encodeURIComponent(val\_c);

- Se aplica functia **encodeURIComponent()** la valorea adaugata in cookie ca sa codeze caractere care fac parte din sintaxa ( =;.).**expires=date** - '**date**' e un sir cu data si timpul cand cookie-ul va expira si va fi automat sters.  
Perioada de expirare trebuie transformata din milisecunde in format: *Wdy, DD-Mon-YYYY HH:MM:SS GMT*.  
Daca nu este specificat nici un timp (expires sau max-age), cookie-ul va dispare la inchiderea browser-ului.**max-age=seconds** - se poate folosi in loc de ***expires***, si e mai simplu.  
**seconds** reprezinta durata de existenta a cookie-ului, in secunde.  
Daca nu este specificat nici un timp (expires sau max-age), cookie-ul va dispare la inchiderea browser-ului.- Exemplu:

var one\_week = 7\*24\*60\*60; //Valabil o saptamana in secunde

document.cookie ='food=fruits; max-age='+ one\_week;

**path=path** - stabileste paginile din site in care se poate folosi acel cookie.  
Daca nu e specificat, cookie-ul poate fi accesat in directorul si subdirectoarele paginii in care e setat.- Exemplu:

//cookie-ul 'food' e disponibil in toate paginile din directorul '/blog' (inclusiv subdirectoare)

document.cookie ='food=fruits; path=/blog';

//cookie-ul 'color' e disponibil in toate paginile din directorul 'javascript/test' (inclusiv subdirectoare)

document.cookie ='color=blue; path=/javascript/test';

- Valoarea pentru ca fisierul cookie sa poata fi folosit in toate paginile unui domeniu, din toate directoarele, este calea radacina '/' ( **path=/** ).**domain=nume\_domeniu** - domeniul si subdomeniu in care cookie-ul e accesibil.  
Daca nu e specificat, cookie-ul e valabil in domeniul unde e creat.- Exemplu:

//cookie-ul 'food' e disponibil in toate paginile din site-ul marplo.net (inclusiv in subdomenii)

document.cookie ='food=fruits; path=/; domain=marplo.net';

//cookie-ul 'color' e disponibil numai in paginile din subdomeniu 'bfie.marplo.net/'

document.cookie ='color=blue; path=/; domain=bfie.marplo.net';

**secure** - daca e adaugat, cookie-ul va fi transmis doar la paginile cu adresa cu HTTPS.- Exemplu:

var one\_week = 7\*24\*60\*60; //Valabil o saptamana in secunde

document.cookie ='food=fruits; path=/; secure; max-age='+ one\_week;

#### Functie pentru creare cookie

In urmatorul exemplu este prezentata o functie utila pentru creare cookie in JavaScript cu parametri necessari.

function setCookie(name, value, maxage, path){

var maxage =(!maxage) ? '' :'; max-age='+ maxage;

var path =(!path) ? '' :'; path=' + path;

document.cookie = name + '='+ encodeURIComponent(value) + maxage + path;

}

- Aceasta functie se poate folosi ca in acest cod:

//creaza un cookie valabil o saptamana in tot site-ul

var m\_age = 7\*24\*60\*60;

setCookie('un\_nume', 'o-valoare', m\_age, '/');

*Un alt mod de a salva date in browser-ul utilizatorului pe timp nelimitat e cu metoda****localStorage****.  
- Vedeti:****[marplo.net/javascript/windows.html#hshsb](https://marplo.net/javascript/windows.html" \l "hshsb" \o "Salvare date in browser)***

### Stergere cookie

Pentru a sterge un cookie, numele si calea (daca a fost specificat parametrul "path") trebuie sa fie aceleasi cu numele si calea folosite la crearea lui.  
Stergerea unui cookie se face prin stabilirea datei de expirare ('expires') la un moment in trecut, sau definirea unei valori negetive (cu minus) la 'max-age'.  
- Pentru stergere cookie puteti folosi functia **delCookie()** din acest cod.

//sterge cookie-ul name adaugand valoare negativa la 'max-age'

//path e necesar doar daca a fost folosit la crearea cookie-ului

function delCookie(name, path){

var path =(!path) ? '' :'; path=' + path;

var two\_d = 2\*24\*60\*60; //doua zile in secunde

document.cookie = name + "=bye; max-age=-" + two\_d + path;

}

### Exemplu complet utilizare cookie

Pentru a intelege mai bine cum puteti crea si folosi fisierele cookie, studiati exemplul urmator.  
Exemplu foloseste functii de creare, citire si stergere cookie prezentate in acest tutorial.  
Scopul acestui exemplu este de a intelege cum functioneaza fisierele cookie. Intr-o parte sunt casete input pentru scrierea unor preferinte care vor fi trimise la scriptul JS pentru inregistrarea lor in cookie. In alta coloana sunt doua butoane, unul pentru afisarea preferintelor inregistrate in cookie; al doilea buton pentru a sterge acele date din cookie.  
- Codul complet este urmatorul, dati clic pe butonul de jos ca sa-l testati:

<style>

#add\_fav, #show\_fav {

display:inline-block;

font-size:18px;

margin:5px 8px;

padding:4px 5px;

}

#add\_fav {

background:#dee0fe;

width:350px;

}

#show\_fav {

border:1px solid #000;

background:#b0eeb0;

vertical-align:top;

width:250px;

}

#show\_fav em {

font-weight:700;

}

#set\_c {

display:block;

margin:8px auto;

}

# Utilizare getElementsByTagName in JS

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [getElementsByTagName() la getElementById()](https://marplo.net/javascript/getelementsbytagname.html" \l "hshgti" \o "getElementsByTagName la getElementById)
* [getAttribute() si getElementsByTagName()](https://marplo.net/javascript/getelementsbytagname.html" \l "hshgat" \o "getAttribute si getElementsByTagName)
* [Inregistrare evenimente la taguri HTML cand documentul este incarcat](https://marplo.net/javascript/getelementsbytagname.html" \l "hshpti" \o "Inregistrare evenimente la taguri HTML cand documentul este incarcat)

**getElementsByTagName('tag')** este o functie, sau metoda JavaScript care obtine si face referire la toate elementele HTML cu 'tag' specificat ca argument intre paranteze.  
De exemplu, urmatoarea instructiune face referire la toate tag-urile DIV din documentul HTML.

**document.getElementsByTagName('div')**

Aceasta functie returneaza un array (cu index de la 0) cu elementele pe care le-a obtinut.  
Array-ul returnat poate fi parcus cu instructiunea **for()**, iar la elementele din el se pot aplica proprietatile si metodele specifice obiectelor HTML in JavaScript. O lista cu acestea gasiti la pagina:  
**[marplo.net/javascript/proprietati-metode-element-html](https://marplo.net/javascript/proprietati-metode-element-html" \o "Proprietati si Metode utile ale elementelor HTML in JavaScript)**

### Aplicarea unei proprietati elementelor cu acelasi tag

Pentru aplicarea unei proprietati la elementele HTML care au acelasi tag, trebuie parcurs array-ul returnat de metoda **getElementsByTagName()**. Ca formula generala poate fi folosit urmatorul cod:

**var elms = document.getElementsByTagName('numeTag');**

**for(var i=0; i<elms.length; i++) {**

**elms[i].proprietate;**

**}**

- Unde "elms" este o variabila in care e stocat array-ul cu tag-urile obtinute.  
- Functia for() parcurge fiecare element din array-ul 'elms'.  
  
Iata un exemplu in care atunci cand se da click pe un buton, continutul din fiecare tag SPAN va fi subliniat si albastru.

<h4>Exemplu getElementsByTagName</h4>

<p>Continut cu text incadrat in etichete SPAN:<br>

marplo.net : <span>Cursuri</span> si <span>tutoriale</span> web <span>gratuite</span>.

</p>

<p>- La clic pe urmatorul buton se vor aplica stilurile CSS: 'color' si 'text-decoration' la toate tag-urile SPAN din pagina.</p>

<button id='btn1'>Style Span</button>

<script>

function adStyle(tag){

//array cu toate tag-urile precizate

var elms = document.getElementsByTagName(tag);

// Parcurge array-ul cu tag-urile

for(var i=0; i<elms.length; i++){

//aplica proprietati de stil

elms[i].style.color ='blue';

elms[i].style.textDecoration ='underline';

}

}

//la click pe #btn1 se apeleaza functia adStyle() cu argument 'span'

document.getElementById('btn1').addEventListener('click', (ev)=>{

adStyle('span');

});

</script>

Incercati codul

### getElementsByTagName() la getElementById()

Cand aplicati **getElementsByTagName(numeTag)** direct la obiectul **document**, se preia tag-urile 'numeTag' din tot documentul HTML, dar sunt situatii cand doriti sa preluati doar tag-uri dintr-o anumita portiune a paginii, cum ar fi de exemplu dintr-un DIV. In acest caz, adaugati acelui element HTML un ID (in care se afla tag-urile respective) si in codul JS il preluati cu metoda **getElementById(ID)**, iar la acest obiect aplicati **getElementsByTagName()**.  
Sintaxa generala e urmatoarea:

**var elms = document.getElementById('id').getElementsByTagName('numeTag')**

- se va obtine un array cu elementele 'numeTag' care sunt in obiectul cu id-ul de la 'id'.  
  
Iata si un exemplu in care sunt doua liste <ul> diferite, iar cand este apasat un buton, va fi aplicat un stil CSS doar la tag-urile <li> de la al doilea <ul>.

<h4>Exemplu getElementsByTagName() la getElementBiId()</h4>

<ul id='ul1'>

<li>ul1 - Vine urmatorul.</li>

<li>ul1 - Am venit degeaba.</li>

</ul>

<ul id='ul2'>

<li>ul2 - Eu sunt cel ce vine,</li>

<li>ul2 - La mine.</li>

</ul>

<p>- La clic pe butonul de mai jos se vor aplica stilurile CSS: 'color' si 'text-decoration' la tag-urile LI din elementul cu id 'ul2'.</p>

<button id='btn1'>Style Li</button>

<script>

function adStyle(id, tag){

//array cu tag-urile 'tag' din elementul cu id-ul 'id'

var elms = document.getElementById(id).getElementsByTagName(tag);

// Parcurge array-ul cu tag-urile

for(var i=0; i<elms.length; i++){

//aplica proprietati de stil

elms[i].style.color ='blue';

elms[i].style.textDecoration ='underline';

}

}

//la click pe #btn1 se apeleaza functia adStyle() cu argumentele pt id si tag

document.getElementById('btn1').addEventListener('click', (ev)=>{

adStyle('ul2', 'li');

});

</script>

Incercati codul

### getAttribute() si getElementsByTagName()

Sunt situatii in care trebuie aplicate efecte sau proprietati JavaScript doar la unele elemente (care au acelasi atribut) dintr-o grupa cu aceleasi tag-uri.  
Daca doriti sa adaugati un efect sau proprietate doar elementelor HTML care au acelasi atribut (si acelasi tag), folositi metoda **getAttribute('attr\_name')** in cadrul parcurgerii array-ului returnat de **getElementsByTagName()**.  
- Ca formula generala ar fi urmatoarea:

**var elms = document.getElementsByTagName('numeTag');**

**for(var i=0; i<elms.length; i++) {**

**if(elms[i].getAttribute('attr\_name')=='val') {**

**// Executati functia sau codul dorit**

**}**

**}**

- Unde "val" e valoarea atributului 'attr\_name' din tag-urile "numeTag" la care doriti sa aplicati un anume cod sau functie.  
  
Ca sa intelegeti mai bine, studiati exemplul urmator in care sunt trei etichete <span> intr-un paragraf, iar la apasarea pe un buton, va fi adaugata o culoare de fundal doar la continutul din etichete SPAN care au atributul 'data-bgr' cu aceeasi valoare.

<h4>Exemplu getElementsByTagName() si getAttribute()</h4>

<p>Continut cu text incadrat in etichete SPAN:<br>

<span data-bgr='yellow'>Cursuri</span>, si <span data-bgr='yellow'>tutoriale</span> web <span data-bgr='green'>gratuite</span>.</p>

<p>- La clic pe butonul de mai jos se va aplica o culoare background la tag-urile SPAN care au atributul 'data-bgr' cu valoarea 'yellow'.</p>

<button id='btn1'>Add Bgr</button>

<script>

function adBgr(tag){

//array cu tag-urile de la 'tag'

var elms = document.getElementsByTagName(tag);

// Parcurge array-ul cu tag-urile

for(var i=0; i<elms.length; i++){

//verifica atributul data-bgr

if(elms[i].getAttribute('data-bgr') =='yellow'){

elms[i].style.background ='yellow';

}

}

}

//la click pe #btn1 se apeleaza functia adBgr() cu argumentele pt. tag

document.getElementById('btn1').addEventListener('click', (ev)=>{

adBgr('span');

});

</script>

Incercati codul

### Inregistrare evenimente la taguri HTML cand documentul este incarcat

Instructiunile JavaScript care lucreaza cu obiecte HTML din pagina e bine sa fie executate dupa incarcarea documentului HTML in DOM, deoarece JavaScript poate lucra cu elementele care sunt deja incarcate.  
Puteti sa adaugati script-ul respectiv la sfarsitul documentului HTML (inainte de tag-ul de inchidere </body>), sau puteti sa folositi evenimentul **DOMContentLoaded**.  
**DOMContentLoaded** se ataseaza la obiectul **document**, si se executa cand toate elementele din pagina (documentul HTML) s-au incarcat in DOM, fara sa mai astepte incarcarea continutului extern (fisiere css, imagini, iframe).  
- Nu functioneaza in <iframe>.

• De exemplu, daca doriti sa preluati anumite tag-uri din pagina intr-un script JS, ori adaugati acel script dupa tag-urile respective, ori folositi ***DOMContentLoaded*** ca in urmatorul exemplu.  
  
- In acest exemplu, dupa incarcarea documentului HTML se preiau in JS un element <input> si tag-urile LI din pagina si inregistreaza click la fiecare. Apoi, cand se da clic pe un LI, se adauga in caseta de text continutul din el.

<h4>Exemplu DOMContentLoaded</h4>

<p>- Dupa incarcarea documentului HTML se preia un element Input si tag-urile LI din pagina si inregistreaza click la fiecare.<br>

La clic pe oricare din listele de mai jos, se va adauga in caseta input textul din el.</p>

Input Text: <input type='text' id='inp1' readonly/>

<ul>

<li>Pacea-i Buna.</li>

<li>Iertarea Fericeste.</li>

<li>Iubirea-i Sfanta</li>

</ul>

<script>

var inp1; //va contine elementul #inp1

//functie apelata la DOMContentLoaded

function liClick(){

inp1 = document.getElementById('inp1'); //caseta de text #inp1

//array cu tag-urile LI

var elms = document.getElementsByTagName('li');

// Parcurge array-ul cu tag-urile si inregistreaza click la fiecare

for(var i=0; i<elms.length; i++){

# querySelector si querySelectorAll

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [querySelectorAll()](https://marplo.net/javascript/queryselector_queryselectorall.html" \l "hshqsa" \o "querySelectorAll)

**querySelector()** si **querySelectorAll()** sunt doua functii JavaScript foarte utile cand se lucreaza cu elemente HTML in JavaScript.  
Cu aceste functii se pot prelua in JavaScript elemente HTML selectate cu selectori CSS ("id", "class").

### querySelector()

**querySelector()** returneaza primul element din documentul HTML care se potriveste cu grupul de selectori specificati, sau ***null*** daca nu e gasit.  
Sintaxa:

**var elm = document.querySelector('selectori');**

- ***"selectori"*** e un sir cu unul sau mai multi selectori CSS, separati prin virgula.  
- ***elm*** e variabila ce va contine obiectul cu elementul HTML returnat.  
  
Exemplu, codul de mai jos afiseaza continutul din primul LI cu class='aclass'.

<h4>Exemplu querySelector()</h4>

<ul>

<li>CoursesWeb.net</li>

<li class='aclass'>MarPlo.net</li>

<li class='aclass'>GamV.eu</li>

</ul>

<p style='background:#fbfbb0;'>La clic pe urmatorul buton (preluat cu querySelector) se afiseaza la #resp continutul din primul LI cu class='aclass'.</p>

<button id='btn1'>Click</button>

<blockquote id='resp'>#resp</blockquote>

<script>

var btn = document.querySelector('#btn1');

btn.addEventListener('click', (ev)=>{

document.getElementById('resp').innerHTML = document.querySelector('li.aclass').innerHTML;

});

</script>

Incercati codul

### querySelectorAll()

Metoda **querySelectorAll()** returneaza un array cu elementele care se potrivesc cu grupul de selectori specificati.  
Sintaxa:

**var elms = document.querySelectorAll('selectori');**

- ***"selectori"*** e un sir cu unul sau mai multi selectori CSS, separati prin virgula.  
- ***elms*** e un array cu obiectele HTML selectate.  
  
Exemplu, se preiau intr-un array tag-urile LI cu class='sites', si tag-urile cu class='note' din elementul cu id='dv1' (adica selctorul: '*li.sites, #dv1 .note*'); apoi afiseaza continutul din ele.

<h4>Exemplu querySelectorAll()</h4>

<ul>

<li class='sites'>CoursesWeb.net</li>

<li class='sites'>MarPlo.net</li>

<li>GamV.eu.net</li>

</ul>

<div id='dv1'>

Cursuri Web Development - <span class='note'>querySelector si querySelectorAll</span>

</div>

<p style='background:#fbfbb0;'>Se preiau intr-un array tag-urile LI cu class='sites', si tag-urile cu class='note' din #dv1 (cu selctorul: 'li.sites, #dv1 .note'), apoi, la clic pe urmatorul buton se afiseaza la #resp continutul lor.</p>

<button id='btn1'>Click</button>

<blockquote id='resp'>#resp</blockquote>

<script>

//array cu tag-urile LI cu class='sites', si tag-urile cu class='note' din elementul cu id='dv1'

var elms = document.querySelectorAll('li.sites, #dv1 .note');

var btn = document.querySelector('#btn1');

btn.addEventListener('click', (ev)=>{

var re ='';

// parcurge array-ul elms si preia in re un sir cu continutul din fiecare element

for(var i=0; i<elms.length; i++) {

re += elms[i].innerHTML +'<br>';

}

//afiseaza sirul in #resp

document.getElementById('resp').innerHTML = re;

});

</script>

Incercati codul

*Diferenta dintre*querySelector() si querySelectorAll()*este aceea ca, querySelector() returneaza un singur obiect cu primul element HTML care se potriveste cu "selectori", dar querySelectorAll() returneaza un array cu toate elementele HTML care se potrivesc cu "selectori".*

• Pentru a limita cautarea obiectelor HTML la un singur element (nu in tot documentul), se pot aplica aceste functii la un obiect cu elementul de interes.  
Sintaxa:

**elm.querySelector('selectori')  
  
elm.querySelectorAll('selectori')**

- Iata un alt exemplu cu querySelectorAll(), selectarea se face pe un anumit element, nu pe "document".

<p style='background:#fbfbb0;'>Se preiau cu querySelectorAll() elementele din tag-ul cu id 'sites' returnate cu selctorul: 'li a'.<br>

- Cand mouse-ul e deasupra urmatoarelor link-uri, se adauga in caseta de text adresa de la 'href'.</p>

Adresa URL: <input type='text' id='txt1' />

<ul id='sites'>

<li><a href='//coursesweb.net/javascript' title='Curs JavaScript'>Curs JavaScript</a></li>

<li><a href='//marplo.net/' title='Cursuri online'>Cursuri online</a></li>

</ul>

<script>

var txt1 = document.getElementById('txt1');

// preia tag-urile A din LI adaugate in elementul cu id='dv1'

var elms = document.getElementById('sites').querySelectorAll('li a');

// parcurge array-ul elms si inregistreaza 'mouseenter' la fiecare

for(var i=0; i<elms.length; i++){

elms[i].addEventListener('mouseenter', (ev)=>{

txt1.value = ev.target.href;

});

}

</script>

# createElement si insertBefore

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [insertBefore()](https://marplo.net/javascript/createelement_insertbefore.html" \l "hshib" \o "insertBefore)
* [Adaugare automata casete input in formular](https://marplo.net/javascript/createelement_insertbefore.html" \l "hshacf" \o "Adaugare automata casete input in formular)

**createElement()** si **insertBefore()** sunt doua functii (metode) JavaScript care sunt folosite pentru a adauga elemente noi in pagina HTML, create dinamic cu JavaScript. De obicei aceste functii sunt folosite impreuna.

### createElement('tag')

Metoda **createElement('tag')** creaza in JS un obiect cu elementul HTML specificat la 'tag'.  
Sintaxa generala este urmatoarea:

**document.createElement('tag')**

- Unde 'tag' este denumirea tag-ului pentru elementul care va fi creat (pentru <p> paramerul va fi 'p', pentru <div> parametrul va fi 'div', iar pentru o casuta <input> se specifica 'input').  
  
Obiectul creat cu **createElement()** reprezinta un element HTML gol in JavaScript. La acesta se poate adauga continut cu **innerHTML** si atribute cu **setAttribute()**.  
Iata un exemplu in care este creat tag-ul <h3> la care se adauga un atribut 'class' si continut.

var elm = document.createElement('h3');

elm.setAttribute('class', 'o\_clasa')

elm.innerHTML ='Textul din eticheta H3 creata dinamic';

console.log(elm);

- Acest cod va crea un obiect in JS ce contine urmatorul cod HTML

<h3 class='o\_clasa'>Textul din eticheta H3 creata dinamic</h3>

*Similar pot fi create si alte tag-uri (elemente) HTML, pentru un <div>, in loc de 'h3' scrieti 'div', si tot asa pt. 'span', 'br', 'li', ... etc.*

Acum trebuie doar adaugat acest obiect in pagina, in locul unde dorim. Pentru aceasta se poate folosi metoda **insertBefore()**, prezentata in contnuare.

### insertBefore()

Metoda **insertBefore()** adauga un obiect HTML din JS imediat inaintea altui element din pagina luat ca referinta, din interiorul unui element parinte.  
Sintaxa generala este urmatoarea:

**parinte.insertBefore(element\_nou, referinta);**

- 'element\_nou' este obiectul din JS cu elementul HTML pe care dorim sa-l adaugam.  
- 'referinta' este elementul HTML din pagina inaintea caruia va fi adaugat 'element\_nou'.  
- 'parinte' este elementul parinte, in care se afla 'referinta' si unde va fi adaugat si 'element\_nou'

*Daca 'referinta' are valoarea****null****, 'element\_nou' va fi adaugat la sfarsitul listei nod-urilor copil din 'parent', devenind astfel ultimul nod copil al acestuia.*

Iata un exemplu din care sa intelegeti mai bine.  
La click pe un buton, va fi adaugat un tag H3 inaintea unui aume DIV (cu id='rpr').

<h4>Exemplu cu insertBefore()</h4>

<p>La click pe urmatorul buton, va fi adaugat un tag H3 creat cu createElement(), inaintea unui DIV cu id='rpr'.</p>

<button id='btn1'>Add H3</button>

<div id='rpr'>#rpr, elementul de reper.</div>

<script>

// Functia ce creaza noul element si-l adauga inaintea unui cadru cu id='rpr'

function add\_h3(){

// Creaza noul element H3 si ii adauga o clasa si continut

var elm = document.createElement('h3');

elm.className ='o\_clasa';

elm.innerHTML = 'Textul din eticheta H3 creata dinamic';

// Creaza obiectul cu elementul de reper (Adaugati-i si valoarea null pentru a vedea diferenta)

var reper = document.getElementById('rpr');

// Adauga elementul nou inaintea celui de reper (Parinte este body)

document.body.insertBefore(elm, reper);

}

document.getElementById('btn1').addEventListener('click', add\_h3);

</script>

Incercati codul

- Demo:

Elementul de reper

*Daca elementul parinte nu e BODY, ci un alt DIV, TABLE ori altceva, il preluati mai intai intr-un obiect (ex.: parinte=document.getElementById('id\_parinte');) apoi il includeti in formula cu 'parinte.insertBefore()'*

### Adaugare automata casete input in formular

Iata un exemplu practic si util de folosire a metodelor **createElement()** si **insertBefore()** pentru a adauga automat casute text intr-un formular.  
Explicatiile necesare sunt in codul scriptului.

<h4>Exemplu createElement() si insertBefore()</h4>

<p>La clic pe butonul 'Adauga caseta' se adauga automat cate o casete de text in formular.</p>

<form action='#'>

Text: <input type='text' name='nume[]' />

<input type='submit' value='Submit' id='submit' /><br><br>

<input type='button' id='btn1' value='Adauga caseta' />

</form>

<script>

// Functia creaza elementul input si-l adauga inaintea butonului Submit

function add\_input(){

// Seteaza noul element input, cu atributul type=text si name=nume[]

var elm = document.createElement('input');

elm.setAttribute('type', 'text');

elm.setAttribute('name', 'nume[]');

elm.style.display = 'block'; // Seteaza display:block; pt. a afisa casutele unele sub altele

// Seteaza obiectele cu elementul de reper (Submit) si cadru parinte

var reper = document.getElementById('submit');

var parinte = reper.parentNode;

// Adauga elementul nou inaintea celui de reper

parinte.insertBefore(elm, reper);

}

//la click pe #btn1 apeleaza add\_input()

document.getElementById('btn1').addEventListener('click', add\_input);

</script>

Incercati codul

# reare scripturi avansate Javascript - PHP

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Alert cu nume utilizator](https://marplo.net/javascript/javascript_php.html" \l "hshau" \o "Alert cu nume utilizator)
* [Ceas cu ora de pe server](https://marplo.net/javascript/javascript_php.html" \l "hshcos" \o "Ceas cu ora de pe server)
* [Afisare date cu JavaScript de la PHP, in functie de o adresa URL](https://marplo.net/javascript/javascript_php.html" \l "hshajp" \o "Afisare date cu JavaScript de la PHP, in functie de o adresa URL)

Codul PHP este executat pe server iar datele de iesire sunt transmise la navigatorul web.  
Codul JavaScript este executat de browser pe calculatorul utilizatorului, dupa excutia codului de pe server.  
Combinand aceste doua limbaje de programare web, se pot obtine scripturi JavaScript dinamice, cu rezultate in functie de datele primite si procesate de server. Astfel, aceeasi pagina de site poate contine un cod JavaScript pentru un utilizator si alt cod JS in cazul altui utilizator.  
  
- Sunt 2 modalitati de a combina JavaScript cu PHP ca sa se obtina un rezultat dinamic sau personalizat:

1. Prin scrierea intregului script JS in codul PHP si adaugarea acestuia in pagina web cu functia PHP "echo" (sau "print").
2. <?php
3. echo '<script> var str ="Cod JS"; </script>';
4. ?>

*Atentie, trebuie sa tineti cont de modul de lucru al ghilimelelor din cod, astfel incat ceea ce va fi returnat de "echo" sa fie un script JS ca si cum a fost scris in codul HTML.*

1. Prin adaugarea in scriptul JS din codul HTML doar a variabilelor PHP necesare scriptului JavaScript (returnand valoarea cu "echo").
2. <script>
3. var str ='<?php echo $var\_php; ?>';
4. </script>

- Ambele variante de cod trebuie scrise in fisiere php ca sa poata fi procesate de modulul PHP.  
  
In continuare sunt explicate cateva exemple practice din care sa intelegeti modul de combinare PHP-JavaScript si utilitatea acestuia.

### Alert cu nume utilizator

De exemplu, intr-o pagina web cu sistem de autentificare sa fie afisata o fereastra Alert cu numele utilizatorului dupa ce s-a autentificat.  
In acest caz, presupunem ca numele de utilizator e stocat in PHP intr-o variabila de sesiune: $\_SESSION['nume'].  
Folosind prima metoda, codul PHP ar fi urmatorul:

<?php

session\_start(); // Aceasta functie trebuie scrisa la inceputul fisierului php

// Cod php ...

echo '<script>alert("Bine ai venit '. $\_SESSION['nume']. '");</script>';

?>

Sau a doua varianta:

<?php

session\_start(); // La inceputul paginii php

?>

<!-- Cod HTML -->

<script>

var user ="<?php echo $\_SESSION['nume']; ?>";

alert('Bine ai venit '+ user);

</script>

- O astfel de fereastra Alert poate fi utila in cazul notificarii utilizatorului cu date ce pot fi extrase de PHP dintr-o baza de date sau fisier, cum ar fi instiintarea unui nou mesaj privat.

### Ceas cu ora de pe server

Un ceas facut si afisat in pagina cu JavaScript va afisa ora de la calculatorul vizitatorului. Daca se doreste afisarea unei aceleasi date (de pe server) pentru toti vizitatorii, cum ar fi in cazul unor jocuri, trebuie adaugat in scriptul JS timpul de pe server, dupa cum e si in urmatorul exemplu:

<div id='tag\_ora'></div>

<script>

// Script ora server-time, de la https://marplo.net

// Preia data serverului prin PHP

var dt\_serv = new Date(<?php echo date('y,n,j,G,i,s'); ?>);

// Defineste variabilele JS de lucru

var ore = dt\_serv.getHours() // Preia ora

var minute = dt\_serv.getMinutes() // Preia minutele

var secunde = dt\_serv.getSeconds() // Preia secundele

// Functia de prelucrare si afisare a datelor

function ceas(){

secunde++;

if(secunde>59){

secunde = 0;

minute++;

}

if(minute>59){

minute = 0;

ore++;

}

if(ore>23) ore = 0;

var output ='<h4>Ora server - '+ore+':'+minute+':'+secunde+'</h4>';

document.getElementById('tag\_ora').innerHTML = output;

}

// Apeleaza functia la fiecare secunda

setInterval('ceas()', 1000);

</script>

- Testati singuri pt. a vedea rezultatul.

### Afisare date cu JavaScript de la PHP, in functie de o adresa URL

Ati vazut in site-urile pt. trafic sau afisare banner ca e necesar adaugarea in pagina a unui mic cod JS cu o anume adresa URL la atributul "src".  
Acestea folosesc acelasi principiu, de combinare PHP cu JavaScript.  
Adresa din codul ce trebuie adaugat in pagina apeleaza un script PHP, acesta preia cu **$\_GET** parametri din adresa URL primita si in functie de acestia prelucreaza datele pe server si construieste pentru returnarea rezultatului un cod JavaScript si HTML (*fara tagul <script>*) care va determina afisarea in pagina (cu **document.write**, sau **innerHTML** intr-un anume tag) a unui cod HTML pt. banner, trafic, top, ... etc.  
Ca sa intelegeti mai bine, experimentati urmatorul exemplu:  
  
1. Creati pe server un fisier php, cu denumirea "phpjs\_test.php", si adaugati urmatorul cod:

<?php

// Exemplu test, de la https://marplo.net

// Creaza o un array cu denumiri

$ids =[1=>'php-mysql', 2=>'javascript', 3=>'html'];

// Preia id-ul de la adresa URL

// Va returna cu echo un link (cu denumirea asociata din array) intr-un cod specific JavaScript (document.write)

if(isset($\_GET['id'])){

if($sir = $ids[$\_GET['id']]){

// Va determina afisarea unui link in pagina apelanta

echo 'document.write("<a href=\'https://marplo.net/'. $sir. '\'>Curs '. $sir. '</a>");';

}

}

?>

2. In acelasi director de pe server creati un fisier html (ex. "test\_jsphp.html") in care adaugati urmatorul cod:

<script src='phpjs\_test.php?id=2'></script>

- Apelati acest fisier html de pe server. Rezultatul va fi afisarea (*determinata de "document.write()"*) in pagina a unui link ce a fost definit in scriptul php in functie de 'id' din adresa de la "src".  
  
Similar se pot determina spre afisare banere sau orice altceva construit cu html in interiorul comezii "document.write()".

# classList - Lucru cu clase css

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

**classList** e o proprietate JavaScript prin case se poate lucra cu clase CSS la elementele HTML, utila pentru a adauga, elimina, inlocui si comuta clase CSS la elemente HTML. Contine un obiect-lista a atributului class.  
Sintaxa:

**element.classList**

- Daca atributul 'class' nu e setat, sau e gol, codul: **element.classList.length** returneaza 0.

## Metode classList

Obiectul continut de proprietatea classList are aceste metode:

* **add(class1, class2, ..)** - adauga unul sau mai multe clase la un element. Daca clasa specificata exista deja, nu va fi adaugata.
* **contains(class)** - Pentru a verifica daca un element contine sau nu o anumita clasa css. Returneaza True daca elementul contine numele de clasa specificat; altfel, False.
* **item(index)** - returneaza numele clasei cu numarul de index specificat (indexul incepe de la 0) sau Null daca indicele specificat este in afara intervalului.
* **remove(class1, class2, ..)** - elimina unul sau mai multe clase css dintr-un element.
* **replace(oldClass, newClass)** - inlocuieste o clasa existenta cu alta noua.
* **toggle(class, [true|false])** - Cand e adaugat un singur argument, comuta acea clasa. Daca exista deja, ***o sterge si returneaza false***, daca nu exista, ***o adauga si returneaza true***.  
  Al doilea parametru e optional (true sau false) forteaza clasa sa fie adaugata sau eliminata, indiferent daca aceasta exista deja sau nu.  
  De exemplu:  
  - Elimina o clasa:

element.classList.toggle('classToRemove', false);

- Adauga o clasa:

element.classList.toggle('classToAdd', true);

### Exemple cu classList

1. Adauga doua clase css la un <div>:

var div1 = document.getElementById('div1');

div1.classList.add('cls\_1', 'cls\_2');

2. Elimina o clasa css de la un <div>:

var div1 = document.getElementById('div1');

div1.classList.remove('cls\_2');

3. Verifica daca un <div> contine sau nu o anumita clasa css:

var div1 = document.getElementById('div1');

if(div1.classList.contains('cls\_1')) alert('.cls\_1 in #div1');

else alert('.cls\_1 not in #div1');

4. Inlocuieste clasa 'foo' cu 'bar':

var div1 = document.getElementById('div1');

div1.classList.replace('foo', 'bar');

5. Daca 'visible' e setat, il elimina; altfel, il adauga:

var div1 = document.getElementById('div1');

div1.classList.toggle('visible');

# Definire si Utilizare Clase in JavaScript

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Clase JS - Proprietati si Metode](https://marplo.net/javascript/definire-utilizare-clase-js" \l "hshpmc" \o "Clase JS - Proprietati si Metode)
* [Metode Statice](https://marplo.net/javascript/definire-utilizare-clase-js" \l "hshsm" \o "Metode Statice)
* [Mai multe instante de obiect ale unei clase](https://marplo.net/javascript/definire-utilizare-clase-js" \l "hshmic" \o "Mai multe instante de obiect ale unei clase)
* [Metode de clasa Getter si Setter](https://marplo.net/javascript/definire-utilizare-clase-js" \l "hshgsm" \o "Metode de clasa Getter si Setter)

Ca sa definiti o clasa in JavaScript, se poate folosi direct declaratia **clss** cu numele clasei.

**class numeClasa {**

**constructor(params){**

**//proprietati**

**}**

**//metode**

**}**

- Un alt mod, adaugarea clasei la o variabila:

**let varClasa = class {**

**constructor(params){**

**//proprietati**

**}**

**//metode**

**}**

Diferenta este faptul ca "varClasa" este o variabila, valoarea ei poate fi schimbata.  
  
- "params" reprezinta argumentele care se transmit cand e creat un obiect al clasei (cu cuvantul **new**).

**let obj = new numeClasa(argumente);**

**//sau**

**let ob2 = new varClasa(argumente);**

## Clase JS - Proprietati si Metode

Proprietatile se definesc in **constructor()**, folosind cuvantul: **this**.  
constructor() e o metoda speciala care e automat apelata cand se creaza o instanta de obiect al clasei (prin cuvantul: ***new***).  
In corpul clasei, proprietatile si metodele se apeleaza aplicand cuvantul: ***this*** (sau numele clasei pentru ***metode statice***).  
In afara clasei, proprietatile si metodele se apeleaza folosind numele instantei de obiect (sau numele clasei pentru ***metode statice***).  
  
- Exemplu de clasa simpla JavaScript:

<div id='ex\_res'>Aici e afisat rezultatul.</div>

<script>

class Rectangle {

//primeste doua argumente la crearea unui obiect al clasei

constructor(width, height){

//setare proprietati

this.width = width;

this.height = height;

}

//o metoda simpla, foloseste proprietatile definite in constructor

area(){

return this.width \* this.height;

}

//alta metoda (primeste un argument); ca sa vedeti cum se apeleaza o metoda in alta

multipliArea(n){

return this.area() \*n;

}

}

//utilizare, creare instanta de obiect a clasei, cu new

let ob\_rect = new Rectangle(5, 3);

//preia valorile returnate de metodele area() si multipliArea()

var area = ob\_rect.area();

var m\_area = ob\_rect.multipliArea(3);

//adauga in elementul html #ex\_res valorile de la area si m\_area

document.getElementById('ex\_res').innerHTML ='area() method returned: '+ area +'<br>multipliArea(3) returned: '+ m\_area;

</script>

Incercati codul

### Metode Statice

Metodele statice din corpul clasei se creaza folosind cuvantul: **static**.  
Atat in interiorul cat si in exteriorul clasei, metodele statice se apeleaza cu "numele-clasei".

*O metoda statica nu poate fi apelata cu "this", nici prin numele instantei de obiect.*

- Exemplu:

<div id='ex2\_res'>Aici e afisat rezultatul.</div>

<script>

class User {

//primeste un nume de user cand se creaza un obiect de clasa

constructor(user){

//setare proprietate

this.user = user;

}

//metoda statica

static Hello(name){

return 'Hello '+ name;

}

//metoda standard, ca sa vedeti cum se apeleaza o metoda statica in clasa

titleHello(){

var re = User.Hello(this.user);

return '<h2>'+ re +'</h2>';

}

}

//utilizare, creaza un obiect al clasei

let ob\_usr = new User('MarPlo');

//adauga in elementul html #ex2\_res valoarea returnata de titleHello()

document.getElementById('ex2\_res').innerHTML = ob\_usr.titleHello();

//preia valoarea returnata de metoda statica (folosind numele clasei)

var hello = User.Hello('Peace');

alert(hello);

</script>

Incercati codul

### Mai multe instante de obiect ale unei clase

Se pot crea mai multe instante de obiect cu aceeasi clasa, cu valori diferite.  
- Exemplu:

<div id='ex\_hi1'>#ex\_hi1 div</div>

<div id='ex\_hi2'>#ex\_hi2 div</div>

<script>

class Test {

constructor(name){

this.name = name;

}

//adauga 'Helo name' in elementul HTML #id

addHello(id){

document.getElementById(id).innerHTML ='<h3>Hello '+ this.name +'</h3>';

}

}

//prima instanta de obiect

let ob1 = new Test('Happines');

ob1.addHello('ex\_hi1'); //apeleaza metoda cu ID-ul unui Div

//a doua instanta de obiect

let ob2 = new Test('Peace');

var hi2 = ob2.addHello('ex\_hi2'); //apeleaza metoda cu alt ID

</script>

Incercati codul

### Metode de clasa Getter si Setter

Sintaxa getter/setter exista pentru proprietati care se definesc bazate pe anumite conditii sau alte proprietati.  
Metoda "getter" se defineste cu cuvantul: **get**.  
Metoda "setter" se defineste cu cuvantul: **set**.  
"getter" si "setter" trebuie sa aibe acelasi nume.  
Metoda setter trebuie sa aibe un argument. Este apelata automat cand se atribuie o valoare unei proprietati cu acelasi nume ca 'setter'; valoarea atribuita va fi argumentul transmis metodei.  
  
- Puteti intelege din codul din acest exemplu:

<div id='ex3\_res'>Aici e afisat rezultatul.</div>

<script>

class Article {

constructor(){

this.th1 ='No title';

}

//getter, returneaza valoarea proprietatii th1 setata in setter

get title(){

return this.th1;

}

//setter, e apelata cand o valoare (val) e atribuita proprietatii: title

set title(val){

if(val !='') this.th1 ='<h1>'+ val +'</h1>';

}

}

//creaza obiect al clasei

let ob\_art = new Article();

//seteaza o valoare proprietatii: title

//aceasta va apela automat metoda setter title() cu valoarea ca argument

ob\_art.title ='MarPlo.net - Clase JS';

//acest cod apeleaza automat metoda getter: title()

var th1 = ob\_art.title;

//adauga in elementul html #ex3\_res valoarea de la th1

document.getElementById('ex3\_res').innerHTML = th1;

</script>

Incercati codul

# Subclase cu extends si Mostenire

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Clase JS - Mostenire, super](https://marplo.net/javascript/subclase-extends-mostenire" \l "hsh_cis" \o "Clase JS - Mostenire, super)
* [Metode de clase si super](https://marplo.net/javascript/subclase-extends-mostenire" \l "hsh_cms" \o "Metode de clase si super)
* [super la metode statice](https://marplo.net/javascript/subclase-extends-mostenire" \l "hsh_ssm" \o "super la metode statice)

## Creare subclase cu extends

In JavaScript se pot crea subclase (ca o clasa copil al altei clase, denumita parinte) folosind cuvantul: **extends**.  
Subclasa mosteneste si poate extinde proprietatile si metodele din clasa parinte.  
Daca subclasa nu are constructor(), va folosi metoda constructor() din clasa parinte.  
  
- Exemplu:

<div id='ex\_res'>Shows response</div>

<script>

//Clasa parinte

class User {

constructor(name){

//seteaza proprietate

this.name = name;

}

//metoda

sayHi(){

return 'Hello '+ this.name;

}

}

//Subclasa

class Gamer extends User {

//metoda foloseste proprietatea name definita in clasa parinte

sayBye(){

return 'Goodbye '+ this.name;

}

}

//Utilizare

//creaza un obiect al subclasei

//transmite un argument deoarece foloseste constructor() din clasa parinte

let ob\_gbr = new Gamer('MarPlo');

//apeleaza metoda sayHi() definita in clasa parinte (User)

var hi = ob\_gbr.sayHi();

//apeleaza metoda definita in subclasa

var bye = ob\_gbr.sayBye();

//adauga valorile de la 'hi' si 'bye' in Div #ex\_res

document.getElementById('ex\_res').innerHTML ='<h3>'+hi+'<br>'+bye+'</h3>';

</script>

Incercati codul

### Clase JS - Mostenire, super

Subclasa mosteneste proprietatile si metodele definite in clasa parinte (inclusiv metodele statice).  
Daca exista constructor() in subclasa, inainte de a folosi cuvantul "**this**" se aplica: **super()**.  
**Pentru a rescrie proprietati si metode din clasa parinte**, se definesc iar in subclasa.  
  
- Exemplu:

<div id='ex\_res'>Shows response</div>

<script>

//Clasa parinte

class Parent {

constructor(name){

//seteaza proprietate

this.name = name;

}

//metoda

sayHi(){

return 'Hello '+ this.name;

}

}

//Subclasa

class Child extends Parent {

constructor(name){

//aplica super ca sa includa constructor() din clasa parinte, transmitand argumentul necesar

super(name);

//aici se pot rescrie sau adauga proprietati noi

this.site ='CoursesWeb.net';

}

//rescrie metoda din parinte

sayHi(){

return 'From Subclass, site: '+ this.site +'<br>Hello '+ this.name;

}

}

//Utilizare

//creaza un obiect al subclasei Child

let obj = new Child('MarPlo');

//apeleaza metoda sayHi(), rescrisa in clasa copil

var hi = obj.sayHi();

//afisaza valoarea de la 'hi' in Div #ex\_res

document.getElementById('ex\_res').innerHTML ='<h2>'+hi+'</h2>';

</script>

Incercati codul

### Metode de clase si super

Cuvantul **super** se foloseste pentru a apela metodele corespunzatoare din clasa de baza (parinte). Se foloseste cand nu se vrea rescrierea completa a unei metode, ci adaugarea de instructiuni noi la cele deja existente in metoda din clasa de baza.  
Clasele furnizeaza cuvantul "super" pentru:

* **super(parametri)** - apeleaza constructor() din parinte, cu parametri necesari (numai in constructor()).
* **super.metoda(parametri)** - awpeleaza o metoda din clasa de baza. Returneaza valoarea returnata de metoda din clasa parinte.

- Exemplu:

<div id='ex\_res'>Shows response</div>

<script>

//Clasa parinte

class Parent {

constructor(name){

//seteaza proprietate

this.name = name;

}

//metoda

sayHi(){

return 'Hello '+ this.name;

}

}

//Subclasa

class Child extends Parent {

constructor(name, site){

//aplica super ca sa includa constructor() din clasa parinte, transmitand argumentul necesar

super(name);

//aici se pot rescrie sau adauga proprietati noi

this.site = site;

}

//refoloseste metoda din parinte

sayHi(){

var hi = super.sayHi(); //retine datele returnate de metoda

//aici se pot adauga noi instructiuni

return hi +'<br>From: '+ this.site;

}

}

//Utilizare

//creaza un obiect al subclasei Child

let obj = new Child('MarPlo', 'CoursesWeb.net');

//apeleaza metoda sayHi() redefinita in clasa copil

var hi = obj.sayHi();

//afisaza valoarea de la 'hi' in Div #ex\_res

document.getElementById('ex\_res').innerHTML ='<h2>'+hi+'</h2>';

</script>

Incercati codul

#### super la metode statice

Instructiunea "super" se poate aplica si la metode statice.  
- Exemplu:

<script>

//clasa de baza

class Parent {

//metoda statica

static sayHi(name){

alert('Hello '+ name +' de la Parinte');

}

}

//Subclasa

class Child extends Parent {

//redefineste metoda statica

static sayHi(name, name2){

super.sayHi(name); //include metoda de la parent

alert('Hello '+ name2 +' de la Subclasa');

}

}

//apeleaza metoda statica din subclasa

Child.sayHi('MarPlo', 'Gamv');

</script>

# Creare clase in JavaScript cu Metode care pot fi inlantuite

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

In aceasta lectie puteti invata cum se pot crea clase in JavaScript **cu metode care pot fi inlantuite**.  
Inlantuirea metodelor inseamna accesarea mai multor metode, sau functii ale unui obiect, intr-o singura instructune.

**obiect.metoda\_1().metoda\_2()**

Cerinta prin care se pot accesa metode inlantuite este ca metoda precedenta (*aici metoda\_1()*) sa returneze instanta de obiect (**this**).  
• Sintaxa:

**class NumeClasa {**

**constructor(){**

**//aici se pot defini proprietati**

**}**

**metoda\_1(){**

**// un cod JavaScript**

**return this; // returneaza instanta de obiect**

**}**

**metoda\_2(){**

**// instructiuni JavaScript**

**}**

**}**

- Iata un exemplu, un o clasa JavaScript prin care se pot calcula aria si perimetrul dreptunghiului (vedeti comentariile din cod si testati-l).

class Rectangle {

constructor(){

//proprietati

this.a =0;

this.b =0;

}

// seteaza valorile pt. a si b

setAB(a1, b1){

this.a = a1;

this.b = b1;

return this; // returneaza instanta de obiect

}

// returneaza aria

area(){

return this.a \* this.b;

}

// returneaza perimetru

perimeter(){

return 2 \* (this.a + this.b);

}

}

// creaza o instanta de obiect a clasei

var obR = new Rectangle();

// seteaza valorile laturilor si obtine aria si perimetru

var area = obR.setAB(7, 8).area();

var perimeter = obR.setAB(7, 8).perimeter();

// test

document.write('Aria = '+ area +'<br>Perimetru = '+ perimeter);

Incercati codul

### Multiple metode inlantuite

Se pot apela mai mult de doua metode inlantuite, folosind acelasi principiu: toate metodele accesate precedent trebuie sa returneze instanta de obiect (**this**).

- Iata un exemplu de inlantuire trei metode. O clasa JavaScript ce poate crea un tag HTML, cu ID, atribut "class" si continut (studiati codul si testati-l).

class setTag {

constructor(){

//proprietati

this.id = ''; // atributul id

this.cls = ''; // atributul class

}

// seteaza id-ul

setId(id1){

this.id = ' id="'+ id1 +'"';

return this; // returneaza instanta de obiect

}

// seteaza atributul class

setClass(cls1){

this.cls =' class="'+ cls1 +'"';

return this; // returneaza instanta de obiect

}

// returneaza tag-ul HTML si continutul

getTagCnt(tag, cnt){

return '<'+ tag + this.id + this.cls +'>'+ cnt +'</'+ tag+ '>';

}

}

// creaza o instanta de obiect a clasei

var obTag = new setTag();

// variabile cu tipul tag-ului si continut

var tag = 'div';

var cnt = 'https://marplo.net';

// apeleaza metodele inlantuite pt. setare ID, "class", si obtine un <div> cu aceste atribute si continut

var getTag = obTag.setId('some\_id').setClass('a\_class').getTagCnt(tag, cnt);

// test

document.write(getTag); //<div id="some\_id" class="a\_class">https://marplo.net</div>

Incercati codul

Instructiunea cu metodele inlantuite din codul de mai sus:

var getTag = obTag.setId('some\_id').setClass('a\_class').getTagCnt(tag, cnt);

E la fel cu acest cod:

// seteaza pe rand ID-ul, "class", apoi obtine tag-ul cu continutul

obTag.setId('some\_id');

obTag.setClass('a\_class');

var getTag = obTag.getTagCnt(tag, cnt);

# avaScript Worker

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Primire date in fisierul worker](https://marplo.net/javascript/worker-js" \l "hsh2" \o "Primire date in fisierul worker)
* [Terminare Worker](https://marplo.net/javascript/worker-js" \l "hsh3" \o "Terminare Worker)
* [Includere scripturi externe in fisier worker](https://marplo.net/javascript/worker-js" \l "hshz" \o "Includere scripturi externe in fisier worker)

Web Worker e un obiect JavaScript care ruleaza in fundal, fara sa afecteze performantele paginii. Codul din worker poate efectua diferite sarcini fara sa interfereze cu pagina din browser.  
In pagina, un script JS poate face diferite instructiuni: click-uri, selectare, afisare date, etc., in timp ce codul din worker efectueaza alte instructiuni.

### Creare Web Worker

In general, un web worker se creaza intr-un fisier JS separat pe server, de exemplu ***worker.js***. Acest fisier contine codul care ruleaza in fundal.  
  
Se poate executa orice cod JS in fisierul worker, cu unele exceptii.  
Nu se poate manipula DOM direct din worker, nuci nu se pot folosi proprietati si metode implicite din obiectele **window**, **document**, si **parent**. Dar se pot folosi multe obiecte din window, incluzand: **XMLHttpRequest** (Ajax), **WebSockets**, **Navigator** si **IndexedDB**.  
  
In pagina in care vreti sa utilizati codul dintr-un web worker, creati o instanta de obiect **Worker** cu aceasta sintxa.

**if(window.Worker){**

**var wrk = new Worker('worker.js');**

**}**

- Dupa ce e creat un cod JavaScript in fisierul **worker.js** (sau orice alt nume vreti), poate sa trimita date la obiectul **Worker**.  
  
Datele sunt trimise intre codul din fisierul worker si cel din pagina printr-un sistem de mesagerie — ambele parti trimit date cu metoda **postMessage()**, si pot receptiona datele prin evenimentul **onmessage**. Datele sunt stocate in proprietatea **event.data**.

### Exemplu simplu cu JavaScript Worker exampe

Urmatorul exemplu afiseaza un contoar care ruleaza continuu intr-un fisier worker. Scriptul e salvat intr-un fisier "workers\_ex.js" si folosit intr-o pagina wen.

#### Cod in workers\_ex.js

var ic =0;

function timedCount(){

ic++;

postMessage(ic);

setTimeout('timedCount()', 500);

}

timedCount();

#### Cod in pagina web

<h4>Exemplu JavaScript Worker</h4>

<blockquote>Contor: <b id='resp'></b></blockquote>

<button id='bt\_startw'>Start Worker</button> -

<button id='bt\_stopw'>Stop Worker</button> -

<script>

var wrk; //pt. obiectul Worker

var resp = document.getElementById('resp');

function startWorker(){

if(window.Worker){

//daca wrk e undefined sau false, il defineste

if(!wrk) wrk = new Worker('javascript/worker\_ex.js');

//addauga in #resp datele primite de la worker

wrk.addEventListener('message', (ev)=>{

resp.innerHTML = ev.data;

});

}

}

function stopWorker(){

wrk.terminate();

wrk = false;

}

//click-ul la butoanele Start /Stop Worker

document.getElementById('bt\_startw').addEventListener('click', startWorker);

document.getElementById('bt\_stopw').addEventListener('click', stopWorker);

</script>

Incercati codul

### Primire date in fisierul worker

Scriptul din pagina poate sa trimita date la worker cu metoda **postMessage()**.

**var wrk = new Worker('worker.js');  
wrk.postMessage('Date trimise la worker');**

Scriptul din worker receptioneaza datele prin evenimentul **onmessage**, in proprietatea **event.data**.

**addEventListener('message', (ev)=>{**

**var data = ev.data;**

**}**

*Se pot trimite orice tip de date cu****postMessage()****: string, number, array, object, boolean.*

- Exemplu, cand utilizatorul tasteaza un text intr-o caseta input, textul e trimis ca sir la un worker care trimite inapoi la pagina un obiect cu textul inversat si numarul de caractere.

#### Cod in worker\_ex.js

//la receptionare date din pagina

addEventListener('message', (ev)=>{

var str = ev.data;

//obiect cu datele care sa fie trimise cu postMessage()

var re ={s:str.split('').reverse().join(''), n:str.length};

postMessage(re);

}

#### Cod in webpage

<h4>Exemplu date catre worker</h4>

<p>Tastati orice text in caseta urmatoare:</p>

Text: <input type='text' id='inp1'/>

<blockquote id='resp'>Aici afiseaza textul inversat si numarul de caractere.</blockquote>

<script>

var resp = document.getElementById('resp');

//defineste obiect Worker, daca e valabil in browser

var wrk = (window.Worker) ? new Worker('javascript/worker\_ex.js') :false;

//daca wrk defint

if(wrk){

//la tastare text in input

document.getElementById('inp1').addEventListener('keyup', (ev)=>{

wrk.postMessage(ev.target.value); //trimite date la worker

});

//preia date de la worker

wrk.addEventListener('message', (ew)=>{

if(ew.data.n) resp.innerHTML ='Text inversat: '+ ew.data.s +'<br>Nr. caractere: '+ ew.data.n;

});

}

</script>

Incercati codul

*In mod normal, web-worker nu se foloseste pentru astfel de scripturi simple, ci mai mult pentru sarcini intensive ale procesorului, in aplicatii precum jocuri, WebSockets si solicitari ajax repetitive.  
Pentru depanarea codului din fisierul worker, se poate folosi metoda****console.log()****in acel fisier.*

### Terminare Worker

Cand se creeaza un obiect Worker, acesta va continua sa detecteze mesajele chiar si dupa terminarea scriptului extern, pana cand acesta va fi terminat /inchis.  
Pentru terminarea unui worker (eliberand astfel resursele), se foloseste metoda **terminate()**.

**workerObj.terminate()**

In fisierul worker se poate inchide singur cu metoda **close()**.

**//fisier worker**

**// codul..**

**if(conditie) close();**

### Includere scripturi externe in fisier worker

Cu functia **importScripts()** se pot include unul sau mai multe scripturi externe in fisierul worker.

**importScripts('foo.js'); //include foo.js**

**importScripts('foo.js', 'bar.js'); //include doua scripturi**

**importScripts('//example.com/hello.js'); //include script din alt domeniu**

Browserul incarca si executa fiecare script inclus. Obiectele globale din fiecare script pot fi apoi folosite de worker.

# Utilizare EventSource pentru evenimente de la server

## Curs Javascript

[**Home**](https://marplo.net/)[**Engleza**](https://marplo.net/engleza)[**Spaniola**](https://marplo.net/spaniola)[**Html**](https://marplo.net/html)[**CSS**](https://marplo.net/css)[**PHP-MySQL**](https://marplo.net/php-mysql)[**Ajax**](https://marplo.net/ajax)[**Blog**](https://marplo.net/blog)[**Forum**](https://marplo.net/forum/)[**Games**](https://gamv.eu/)

Top of Form



Bottom of Form

* [Trimitere eveniment message de la server](https://marplo.net/javascript/eventsource-evenimente-server" \l "hsh1" \o "Trimitere eveniment message de la server)
* [Exemplu cu eveniment de la server in JS](https://marplo.net/javascript/eventsource-evenimente-server" \l "hshex" \o "Exemplu cu eveniment de la server in JS)
* [Utilizare nume personalizat pentru evenimentul trimis de server](https://marplo.net/javascript/eventsource-evenimente-server" \l "hshz" \o "Utilizare nume personalizat pentru evenimentul trimis de server)

Cu SSE (Server-Sent Events - Evenimente trimise de server) pagina web poate sa primeasca automat date de la server fara solicitari de trimitere.

### Primirea de evenimente de la server

Evenimentele SSE sunt unidirectionale; mesajele sunt transmise intr-un singur sens, de la server la pagina web.  
Pentru a primi automat in JavaScript notificari de la server, definiti un obiect **EventSource**, cu adresa URL a fisierului de pe server.

Sintaxa:

**var evsource = new EventSource('sse\_file.php');**

Daca fisierul e pe alt domeniu, definiti **EventSource** cu adresa acelui fisier, si un al doilea argument cu un obiect cu proprietatea **withCredentials** setata **true** (default e **false**).  
Sintaxa:

**var evsource = new EventSource('//api.example.com/sse\_file.php', {withCredentials: true});**

Dupa ce ati creat obiectul cu **EventSource**, se poate receptiona mesajul de la server prin evenimentul **message**:

**var evsource = new EventSource('sse\_file.php');**

**evsource.addEventListener('message', (ev)=>{**

**//preia notificarea de la server din eveniment (ev)**

**let data = ev.data;**

**});**

Fiecare eveniment **message** are aceste proprietati:

* **data** - textul cu notificarea de la server. Poate fi orice sir, de exemplu un sir cu obiect JSON.
* **lastEventId** - id-ul ultimului mesaj (daca e transmis de la server).

### Trimitere eveniment message de la server

Scriptul de la server care trimite evenimente stream trebuie sa transmita header-ul **Content-Type: text/event-stream**.  
Fiecare eveniment '**message**' e transmis ca text cu unul sau mai multe linii, terminat cu doua linii goale.  
Sintaxa (in PHP):

**header('Content-Type: text/event-stream');**

**echo "data:$data\n";**

**echo "retry:$retry\n";**

**echo "id=$id \n\n";**

* **$data** - o variabila cu sirul din proprietatea **data**.
* **$retry** - (optional) durata, in milisecunde dupa care va fi transmisa urmatoarea notificare. Seteaza intervalul de timp dintre notificarile de la server. Daca nu e specificat, default e 3000 (3 secunde).
* **$id** - (optional) un ID pentru notificarea curenta. Valoarea care poate fi citita in JavaScript la proprietatea **ev.lastEventId**.

*Sirul intreg cu mesajul trebuie sa se termine cu doua linii goale '\n\n'.  
Daca sirul incepe cu caracter doua-puncte (:), e considerat un comentariu si e ignorat.*

***echo ': Text ignored \n\n';***

*Daca sirul nu contine caracter doua-puncte, intreaga linie este tratata ca numele campului cu un sir de valori goale.*

### Exemplu cu eveniment de la server in JS

Iata un exemplu simplu care ajuta sa intelegeti cum functioneaza evenimentele de la server cu obiectul **EventSource**.  
- Acest exemplu afiseaza intr-un element HTML timpul (ora:minute:secunde) transmis de la server.

<h3>Exemplu eveniment de la server cu EventSource</h3>

<h4>Server-Time: <span id='ss\_time'></span></h4>

<script>

var ss\_time = document.getElementById('ss\_time');

//daca browser-ul suporta EventSource

if(window.EventSource){

//defineste un obiect EventSource object care sa primeasca date de la sse\_ex.php

var evsource = new EventSource('javascript/sse\_ex.php');

//receptioneaza datele de la server

evsource.addEventListener('message', (ev)=>{

let id = ev.lastEventId; //in caz ca e necesar id-ul mesajului

//preia si adauga data in #ss\_time

let data = ev.data;

ss\_time.innerHTML = data;

});

}

else ss\_time.innerHTML ='No server-sent events support.';

</script>

Incercati codul

Sintaxa pentru evenimentele stream de la server e simpla. Intai setati header-ul **'Content-Type'** cu valoarea **'text/event-stream'**, apoi se pot transmite evenimentele stream cu '**echo**' (in PHP).  
- Fisierul **sse\_ex.php** contine acest cod:

<?php

header('Content-Type: text/event-stream');

header('Cache-Control: no-cache'); //ca sa nu adauge datele in cache la browser

header('Access-Control-Allow-Origin: \*'); //permite acces stream doar de pe domeniu actual

$time = date('h:i:s');

$id = time(); //sa seteze id-ul cu timestamp-ul curent

//transmite evenimentul cu un interval de timp de 1 secunda (1000 milisecunde)

echo "data:$time\n";

echo "retry:1000\n";

echo "id=$id \n\n";

//trimite datele la browser

ob\_flush();

flush();

#### Terminare evenimente streams

In mod implicit, daca conexiunea dintre client si server se inchide, conexiunea este resetata.  
Pentru a termina complet conexiunea de pe partea de client, aplicati metoda **close()**.

**var evsource = new EventSource('sse\_file.php');**

**//restul codului..**

**if(conditie) evsource.close();**

*Pentru a termina complet conexiunea de pe partea de server, transmiteti fara header****"text/event-stream" Content-Type****, sau returnati un status HTTP diferit de 200 OK (ex. 404 Not Found).*

***header('HTTP/1.0 404 Not Found');  
exit();***

*- Va impiedica browserul sa restabileasca conexiunea.*

### Utilizare nume personalizat pentru evenimentul trimis de server

Cuvantul '**message**' e numele implicit pentru evenimente stream de la server, dar se poate specifica si alt nume, adaugand campul **event** in textul transmis la browser.  
Sintaxa (in PHP):

**echo 'event:event\_name\n';**

**echo "data:$data\n";**

**echo "retry:$retry\n";**

**echo "id=$id \n\n";**

Iar pe partea de client, scriptul JS poate receptiona evenimentul specificat:

**var evsource = new EventSource('sse\_file.php');**

**evsource.addEventListener('event\_name', (ev)=>{**

**//preia notificarea de la server din eveniment (ev)**

**let data = ev.data;**

**});**

- Acest cod va fi apelat automat cand serverul trimite un mesaj cu numele de eveniment "event\_name".