CUPRINS

1. Introducere în JavaScript

JavaScript este cel mai popular limbaj pentru scripturi şi funcţionează în toate browserele importante, cum ar fi Internet Explorer, Firefox, Chrome, Opera şi Safari.

Ce este JavaScript?

- JavaScript a fost proiectat pentru a adăuga interactivitate paginilor HTML
- JavaScript este un limbaj pentru scripturi
- Un limbaj pentru scripturi este un limbaj de programare simplificat
- JavaScript este, în general, înglobat direct în paginile HTML
- JavaScript este un limbaj interpretat (adică scriptul este executat direct, fără compilare prealabilă)
- JavaScript poate fi folosit fără licenţă

Java şi JavaScript sunt două limbaje complet diferite.
Java (dezvoltat de Sun Microsystems) este un limbaj mult mai puternic şi mai complex, din aceeaşi categorie cu C sau C++.

Ce poate face JavaScript?

- **JavaScript oferă proiectanţilor HTML un instrument de programare –** în general proiectanţii paginilor HTML nu sunt programatori, dar JavaScript este un limbaj cu o sintaxă foarte simplă şi aproape oricine poate insera mici secvenţe de cod în paginile HTML
- **JavaScript poate insera în mod dinamic text într-o pagină HTML** – O instrucţiune JavaScript ca aceasta: `document.write("<h1>" + name + "</h1>")` poate scrie un text variabil în pagina HTML
- **JavaScript poate reacţiona la evenimente –** Un cod JavaScript poate fi proiecta să se execute când se întâmplă ceva, spre exemplu când pagina s-a încărcat complet sau utilizatorul acţionează un element HTML
- **JavaScript poate citi şi scrie elementele HTML –** Un cod JavaScript poate citi şi modifica conţinutul unui element HTML
- **JavaScript poate fi folosit pentru a valida datele –** Un cod JavaScript poate fi folosit pentru a valida datele înainte de a fi trimise către server. În acest fel serverul nu mai face procesări suplimentare.
- **JavaScript poate fi folosit pentru a detecta browserul utilizatorului** – Un cod JavaScript poate detecta tipul browserului şi poate încărca o pagină proiectată special pentru tipul respectiv de browser

- **JavaScript poate fi folosit pentru a crea cookies** – Un cod JavaScript poate fi utilizat pentru a stoca şi extrage informaţii pe calculatorul vizitatorului paginii HTML

Cum se inserează JavaScript într-o pagină HTML

Pentru a insera JavaScript într-o pagină HTML se utilizează tagul **<script>**. În exemplul următor, JavaScript este utilizat pentru a scrie un text într-o pagină web:

```
<html>
<body>
<h3>Afisarea unui mesaj cu JavaScript</h3> <hr/>
<script type="text/javascript">
document.write("Bine ati venit!");
</script>
</body>
</html>
```

Exemplul următor ilustrează cum pot fi adăugate taguri HTML pentru a formata textul afişat cu JavaScript:

```
<html>
<body>
<h3>Utilizarea tagurilor HTML in mesajul afisat cu
JS</h3> <hr/>
<script type="text/javascript">
document.write("<h1>Bine ati venit!</h1>");
</script>
</body>
</html>
```

Explicaţii:

Pentru a insera JavaScript într-o pagină HTML, folosim tagul **<script>** şi în interiorul acestui tag folosim atributul type pentru a defini limbajul în care este scris scriptul. Deci tagurile **<script type="text/javascript">** şi **</script>** marchează locul în care începe, respectiv se sfârşeşte scriptul:

```
<html>
<body>
<script type="text/javascript">
...
</script>
</body>
</html>
```

Comanda **document.write** reprezintă modalitatea JS standard pentru a scrie un text într-o pagină . Deoarece această comandă este inclusă între

tagurile **<script>** şi **</script>**, browserul o va recunoaşte drept comandă JS şi va executa respectiva linie de cod. Pentru exemplul considerat, browserul va scrie în pagină textul **Bine ati venit!**

Obs: Dacă comanda **document.write** nu este inclusă între tagurile de script, browserul o va interpreta ca text obişnuit şi va afişa pe ecran linia de cod.

Browsere care nu recunosc JavaScript

Dacă browserul nu recunoaşte JS, liniile de cod vor fi afişate ca atare în pagină. Pentru a evita acest lucru, scriptul ar trebui „ascuns" în taguri de comentariu. În exemplul următor, scriptul este scris între tagurile de comentariu:

```
<html>
<body>
<script type="text/javascript">
<!--
document.write("Bine ati venit!");
//-->
</script>
</body>
</html>
```

Ultimele două caractere **//** reprezintă simbolul JS pentru comentariu şi sunt scrise pentru a împiedica JS să execute tagul **-->**.

2. Inserarea scripturilor JS

Dacă scriptul este inclus în secţiunea **body**, el va fi executat cât timp se încarcă pagina. Dacă scriptul este inclus în secţiunea **head**, el va fi executat numai când este apelat..

Scripturi în <head>

Scripturile care trebuie executate când sunt apelate sau când are loc un eveniment, trebuie scrise în secţiunea **head**. În acest fel, scriptul va fi sigur încărcat înainte de a fi utilizat.

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("Aceasta caseta de alertare este apelata si
afisata cand are loc evenimentul onload");
```

```
}
</script>
</head>

<body onload="message()">
<h3>Casetele de alertare</h3> <hr/>
</body>
</html>
```

Scripturi în <body>

Scripturile care trebuie executate când pagina se încarcă trebuie scrise în secțiunea body și vor genera conținutul paginii:

```
<html>
<head>
</head>

<body>
<h3>Afisarea textului cu JavaScript</h3> <hr/>
<script type="text/javascript">
document.write("Acest mesaj este scris cu
JavaScript");
</script>
</body>

</html>
```

Scripturi în <head> și <body>

Puteți include un număr nelimitat de scripturi JS în document, deci puteți avea scripturi și în **head** și în **body**:

```
<html>
<head>
<script type="text/javascript">
....
</script>
</head>
<body>
<script type="text/javascript">
....
</script>
</body>
```

Folosirea unui script extern

Dacă doriți să utilizați același script în mai multe pagini web fără a rescrie codul, trebuie să scrieți scriptul JS într-un fișier extern. Fișierul trebuie să aibă extensia **.js** și nu poate conține tagul **<script>**. Pentru a utiliza fișierul extern, includeți-l în atributul **src** al tagului **<script>**:

```
<html>
<head>
<script type="text/javascript" src="…….js"></script>
</head>
<body>
</body>
</html>
```

Obs: Scriptul trebuie plasat în locul în care ar fi fost scris în mod normal.

3. Instrucțiuni JavaScript

JavaScript este o secvență de declarații, instrucțiuni și comenzi care vor fi executate de către browser. Spre deosebire de HTML, Java Script este case-sensitive, deci aveți grijă când scrieți instrucțiuni, declarați variabile sau apelați funcții. O instrucțiune JavaScript este o comandă către browser și are rolul de a spune browserului ce trebuie să facă. Următoarea instrucțiune JS spune browserului să scrie în pagină textul "Buna ziua":

```
document.write("Buna ziua");
```

Fiecare instrucțiune se încheie cu punct și virgulă (;).
Codul JavaScript este o secvență de instrucțiuni JS. Fiecare instrucțiune este executată de browser în ordinea în care a fost scrisă.
Exemplul următor va scrie un titlu și două paragrafe într-o pagină web:

```
<html>
<body>

<h3>Utilizarea tagurilor HTML in mesajele afisate cu
JS</h3> <hr/>
<script type="text/javascript">
document.write("<h1>Acesta este un titlu</h1>");
document.write("<p>Acesta este un paragraf.</p>");
document.write("<p>Acesta este un alt
paragraf.</p>");
</script>

</body>
</html>
```

Blocuri JavaScript

Instrucțiunile JavaScript pot fi grupate în blocuri care se scriu între acolade. Instrucțiunile dintr-un bloc vor fi executate împreună.

În acest exemplu, instrucțiunile care scriu un titlu şi două paragrafe, au fost grupate împreună într-un bloc:

```
<html>
<body>

<script type="text/javascript">
{
document.write("<h1>Acesta este un titlu</h1>");
document.write("<p>Acesta este un paragraf.</p>");
document.write("<p>Acesta este un alt
paragraf.</p>");
}
</script>

</body>
</html>
```

În mod normal, un bloc este folosit pentru a grupa un grup de instrucțiuni într-o funcție sau într-o condiție (blocul va fi executat dacă o anumită condiție este satisfăcută).

4. Comentarii JavaScript

Comentariile pot fi adăugate pentru a explica codul sau a-l face mai uşor de citit. Comentariile care se scriu pe o singură linie încep cu **//**. În exemplul următor, comentariile tip linie sunt folosite pentru a explica codul:

```
<html>
<body>

<script type="text/javascript">
// Scrie un titlu:
document.write("<h1>Acesta este un titlu</h1>");
// Scrie doua paragrafe:
document.write("<p>Acesta este un paragraf.</p>");
document.write("<p>Acesta este un alt
paragraf.</p>");
</script>

</body>
</html>
```

Comentarii multi-linie

Aceste comentarii încep cu **/*** şi se încheie cu ***/**, ca în exemplul următor:

```
<html>
<body>

<script type="text/javascript">
/*
Codul urmator va scrie in pagina
un titlu si doua paragrafe
*/
document.write("<h1>Acesta este un titlu</h1>");
document.write("<p>Acesta este un paragraf.</p>");
document.write("<p>Acesta este un alt
paragraf.</p>");
</script>

</body>
</html>
```

Folosirea comentariilor pentru a preveni execuția

În exemplul următor, comentariul este utilizat pentru a împiedica executarea unei linii de cod (metoda poate fi utilizată pentru a depana codul):

```
<html>
<body>

<script type="text/javascript">
//document.write("<h1>Acesta este un titlu</h1>");
document.write("<p>Acesta este un paragraf.</p>");
document.write("<p>Acesta este un alt
paragraf.</p>");
</script>

</body>
</html>
```

În exemplul următor, comentariul este utilizat pentru a împiedica execuția unui bloc de cod (util pentru depanarea codului):

```
<html>
<body>

<script type="text/javascript">
/*
document.write("<h1>Acesta este un titlu</h1>");
```

```
document.write("<p>Acesta este un paragraf.</p>");
document.write("<p>Acesta este un alt
paragraf.</p>");
*/
</script>

</body>
</html>
```

În exemplul următor, comentariul este plasat la sfârşitul liniei de cod:

```
<html>
<body>

<script type="text/javascript">
document.write("Salutare"); // scrie in pagina textul
"Salutare"
document.write(" prieteni!"); // scrie in pagina
textul " prieteni!"
</script>

</body>
</html>
```

5. Variabile JavaScript

În JS, variabilele sunt folosite pentru a păstra valori sau expresii. O variabilă poate avea un nume scurt, de exemplu x, sau mai descriptiv, de exemplu prenume.
Reguli pentru numele variabilelor JavaScript:

- numele este case-sensitive (y şi Y sunt două variabile diferite)
- numele trebuie să înceapă cu o literă sau cu liniuța de subliniere (underscore)

Exemplu

Valoarea unei variabile se poate modifica în timpul execuției scriptului. Puteți referi variabila prin nume pentru a-i afişa sau modifica conținutul, ca în exemplul următor:

```
<html>
<body>
```

```
<h3>Declararea, initializarea, atribuirea si afisarea
unei variabile</h3> <hr/>
<script type="text/javascript">
var prenume;
prenume="Mihai";
document.write("<b>Numele variabilei</b>: prenume");
document.write("<br/>");
document.write("<b>Valoare initiala</b>: "+prenume);
document.write("<br/>");
prenume="Adrian";
document.write("<b>Valoare dupa atribuire</b>:
"+prenume);
</script>

</body>
</html>
```

Declararea variabilelor JavaScript

Puteți crea variabile cu sintaxa :

```
var nume_variabila;
```

După declarare, variabila nu conține valori (este vidă). Puteți să inițializați o variabilă chiar în momentul declarării:

```
var x=8;
var prenume="Matei";
```

Obs: Când atribuiți unei variabile o valoare de tip text, textul trebuie scris între ghilimele.

Dacă atribuiți valori unei variabile care nu a fost încă declarată, ea va fi declarată automat.
Declarațiile:

```
x=8;
prenume="Matei";
```

au acelaşi efect cu declarațiile:

```
var x=8;
var prenume="Matei";
```

Dacă redeclarați o variabilă JavaScript, ea va păstra valoarea inițială:

```
var x=7;
var x;
```

După execuția instrucțiunilor de mai sus, variabila x are valoarea 7 care nu a fost resetată la redeclarare.

6. Operatorii JavaScript

Operatorii aritmetici

Sunt folosiți pentru a efectua operații aritmetice cu variabile şi/sau valori.
Dacă **y=5**, tabelul următor prezintă operatorii aritmetici:

Operator	Descriere	Exemplu	Rezultat
+	adunare	x=y+2	x=7
-	scădere	x=y-2	x=3
*	înmulțire	x=y*2	x=10
/	împărțire	x=y/2	x=2.5
%	modulo (restul împărțirii întregi)	x=y%2	x=1
++	incrementare	x=++y	x=6
--	decrementare	x=--y	x=4

Operatorii de atribuire

Sunt folosiți pentru a atribui valori variabilelor JavaScript. Dacă **x=10** şi **y=5**, tabelul următor prezintă operatorii de atribuire:

Operator	Exemplu	Echivalent cu	Rezultat
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Operatorul + utilizat pentru şiruri de caractere

Acest operator poate fi utilizat şi pentru a concatena variabile tip şir de caractere (string sau text).
Exemplu:

```
t1="Ce mai";
t2="faci azi?";
t3=t1+t2;
```

După execuție, variabila **t3** conține șirul „**Ce maifaci azi?**”. Pentru a adăuga un spațiu între cele două șiruri, inserați un spațiu la sfârșitul primului șir sau la începutul celui de-al doilea șir sau între cele două șiruri:

```
t1="Ce mai";
t2="faci azi?";
t3=t1+" "+t2;
```

Adunarea şirurilor şi a numerelor

Regulă: **Dacă adunați un număr cu un şir de caractere, veți obține un şir de caractere.**

```
<html>
<body>

<h3>Adunarea sirurilor de caractere si a numerelor cu
siruri de caractere</h3> <hr/>
<script type="text/javascript">
x=6+7;
document.write("6+7="+x);
document.write("<br />");
x="6"+"7";
document.write('"6"+"7"='+x);
document.write("<br />");
x=6+"7";
document.write('6+"7"='+x);
document.write("<br />");
x="6"+7;
document.write('"6"+7='+x);
document.write("<br />");
</script>

</body>
</html>
```

7. Operatorii de comparare şi operatorii logici

Operatorii de comparare sunt utilizați în construcții logice pentru a verifica egalitatea sau diferența dintre două variabile sau valori.
Dacă **x=5**, tabelul următor prezintă operatorii de comparare:

Operator	Descriere	Exemple
==	Egal cu	**x==8** este fals

`===`	Este egal exact (valoare şi tip)	`x===5` este adevărat `x==="5"` este fals
`!=`	Diferit	`x!=8` este adevărat
`>`	Mai mare decât	`x>8` este fals
`<`	Mai mic decât	`x<8` este adevărat
`>=`	Mai mare sau egal cu	`x>=8` este fals
`<=`	Mai mic sau egal cu	`x<=8` este adevărat

Operatorii logici sunt utilizaţi pentru a determina relaţia logică dintre variabile sau valori.
Dacă **x=6 şi y=3**, tabelul următor prezintă operatorii logici:

Operator	Descriere	Exemple
`&&`	şi	`(x < 10 && y > 1)` este adevărat
`||`	sau	`(x==5 || y==5)` este fals
`!`	not	`!(x==y)` este adevărat

Operatorul condiţional
Acest operator atribuie o valoare unei variabile în funcţie de o anumită condiţie.
Sintaxă:

```
variabila=(conditie)?valoare1:valoare2
```

Exemplu:

```
salut=(visitor=="FEM")?"Doamna ":"Domnul";
```

8. Instrucţiunile condiţionale

Adesea, când scrieţi cod JS, trebuie să realizaţi operaţii diferite în funcţie de decizii diferite. Pentru a realiza acest lucru, folosiţi în cod instrucţiunile condiţionale.
În JavaScript există următoarele instrucţiuni condiţionale:

- **if** – folosiţi această instrucţiune dacă un cod trebuie executat când o condiţie este adevărată
- **if...else** - folosiţi această instrucţiune pentru a executa un cod când o condiţie este adevărată şi alt cod dacă condiţia este falsă
- **if...else if....else** – folosiţi această instrucţiune pentru a selecta unul din mai multe blocuri de cod pentru a fi executat
- **switch** - folosiţi această instrucţiune pentru a selecta unul din mai multe blocuri de cod pentru a fi executat

Instrucțiunea `if`

Sintaxă

```
if (conditie)
  {
  cod ce trebuie executat daca conditia este
adevarata
  }
```

Exemplu:

```
<html>
<body>

<h3>Scriptul va afisa un mesaj daca ora<10 folosind
instructiunea if</h3> <hr/>
<script type="text/javascript">
var d = new Date();
var time = d.getHours();

if (time < 10)
  {
  document.write("<b>Buna dimineata</b>");
  }
</script>

</body>
</html>
```

Instrucțiunea `if…else`

Sintaxă:

```
if (conditie)
  {
  cod executat daca conditia este adevarata
  }
else
  {
  cod executat daca conditia este falsa
  }
```

Exemplu:

```
<html>
<body>
```

```
<h3>Scriptul va afisa un mesaj sau altul in functie
de ora, cu instructiunea if..else</h3> <hr/>
<script type="text/javascript">
var d = new Date();
var time = d.getHours();

if (time < 10)
{
document.write("<b>Buna dimineata</b>");
}
else
{
document.write("<b>Buna ziua</b>");
}
</script>

</body>
</html>


</html>
```

Instrucțiunea `if...else if...else`

Sintaxă:

```
if (conditia1)
  {
  cod executat daca conditia1 este adevarata
  }
else if (conditia2)
  {
  cod executat daca conditia2 este adevarata
  }
else
  {
  cod executat daca nici conditia1, nici conditia2 nu
sunt adevarate
  }
```

Exemplu:

```
<html>
<body>
```

```
<h3>Scriptul va afisa unul din trei mesaje in functie
de ora, cu instructiunea if-else-if-else</h3> <hr/>
<script type="text/javascript">
var d = new Date();
var time = d.getHours();
if (time<10)
{
document.write("<b>Buna dimineata</b>");
}
else if (time>=10 && time<17)
{
document.write("<b>Buna ziua</b>");
}
else
{
document.write("<b>Buna seara</b>");
}
</script>

</body>
</html>
```

Exemplu

Link-ul din exemplul următor va deschide Google sau Yahoo.

```
<html>
<body>

<h3>Scriptul afiseaza in mod aleator unul din doua
link-uri, folosind if..else</h3> <hr/>
<script type="text/javascript">
var r=Math.random();
if (r>0.5)
{
document.write("<a
href='http://www.google.com'>Google!</a>");
}
else
{
document.write("<a
href='http://www.yahoo.com>Yahoo!</a>");
}
</script>

</body>
```

```
</html>
```

Instrucțiunea switch

Sintaxă:

```
switch(n)
{
case 1:
  executa bloc 1
  break;
case 2:
  executa bloc 2
  break;
......
default:
  cod executat daca n este diferit de case 1, case
2,....
}
```

Exemplu:

```
<html>
<body>

<h3>Scriptul utilizeaza instructiunea switch</h3>
<hr/>
<script type="text/javascript">
var d = new Date();
theDay=d.getDay();
switch (theDay)
{
case 5:
  document.write("<b>Vineri</b>");
  break;
case 6:
  document.write("<b>Sambata</b>");
  break;
case 0:
  document.write("<b>Duminica</b>");
  break;
default:
  document.write("<b>Astept weekend-ul!</b>");
}
</script>
```

```
</body>
</html>
```

9. Casete popup

JavaScript are trei tipuri de casete popup: caseta Alert, caseta Confirm şi caseta Prompt.

Caseta Alert

O casetă de alertă se utilizează atunci doriți să fiți siguri că o anumită informație ajunge în atenția utilizatorului. Când o casetă de alertă este afişată, utilizatorul va trebui să acționeze butonul "**OK**" pentru a putea continua.

Sintaxă:

```
alert("...un text....");
```

Exemplu:

```
<html>
<head>
<script type="text/javascript">
function afiseaza_alert()
{
alert("Sunt o caseta de alertare!");
}
</script>
</head>
<body>

<h3>La apasarea butonului va fi apelata o functie
care afiseaza caseta alert</h3> <hr/>
<input type="button" onclick="afiseaza_alert()"
value="Apasa" />

</body>
</html>
```

Caseta Confirm

O casetă de confirmare se utilizează atunci când doriți ca utilizatorul să verifice sau să accepte ceva. Când caseta de confirmare este afişată, utilizatorul va trebui să acționeze butonul "**OK**" sau butonul "**Cancel**" pentru a putea

continua. Dacă utilizatorul acționează butonul "**OK**", caseta returnează valoarea **true**, dacă acționează butonul "**Cancel**", caseta returnează valoarea **false**.
Sintaxă:

```
confirm("....un text....");
```

Exemplu:

```
<html>
<head>
<script type="text/javascript">
function afiseaza_confirm()
{
var r=confirm("Apasati un buton");
if (r==true)
  {
  document.write("Ati apasat butonul OK!");
  }
else
  {
  document.write("Ati apasat butonul Cancel!");
  }
}
</script>
</head>
<body>

<h3>La apasarea butonului va fi apelata o functie
care afiseaza caseta confirm si verifica ce buton ati
apasat</h3> <hr/>
<input type="button" onclick="afiseaza_confirm()"
value="Apasa" />

</body>
</html>
```

Caseta Prompt

Această casetă se utilizează atunci când doriți ca utilizatorul să introducă o anumită valoare înainte de a accesa pagina. Când caseta prompt este afişată, utilizatorul va trebui să acționeze butonul "**OK**" sau butonul "**Cancel**" pentru a putea continua după ce introduce valoarea solicitată. Dacă utilizatorul acționează butonul "**OK**", caseta returnează valoarea **true**, dacă acționează butonul "**Cancel**", caseta returnează valoarea **false**.
Sintaxă:

```
prompt("....un text....","valoare_implicita");
```

Exemplu:

```
<html>
<head>
<script type="text/javascript">
function afiseaza_prompt()
{
var name=prompt("Va rog sa va introduceti
numele","");
if (name!=null && name!="")
  {
  document.write("Buna ziua " + name + "! Ce mai
faci?");
  }
}
</script>
</head>
<body>

<h3>La apasarea butonului va fi apelata o functie
care afiseaza caseta prompt</h3> <hr/>
<input type="button" onclick="afiseaza_prompt()"
value="Apasa" />
</body>
</html>
```

Obs. Dacă doriți ca textul dintr-o casetă să fie afişat pe mai multe linii, procedați ca în exemplul următor:

```
<html>
<head>
<script type="text/javascript">
function afiseaza_alert()
{
alert("Buna! Asa se adauga" + '\n' + "o intrerupere
de linie" + '\n' + "intr-o caseta de alertare!");
}
</script>
</head>
<body>

<h3>Caseta alert cu textul scris pe mai multe
linii</h3> <hr/>
```

```
<input type="button" onclick="afiseaza_alert()"
value="Apasa" />

</body>
</html>
```

10. Funcții

O funcție va fi executată când are loc un eveniment sau când este apelată.
Dacă doriți ca browserul să nu execute un script atunci când pagina se încarcă, puteți scrie scriptul într-o funcție. O funcție poate fi apelată din orice punct al paginii, sau chiar din alte pagini, dacă funcția este scris într-un fişier JS extern. Funcțiile JS pot fi scrise în secțiunea **<head>** sau în secțiunea **<body>** a documentului. Totuşi, pentru a fi siguri că funcția este încărcată în browser înainte de a fi apelată, este recomandat să o scrieți în secțiunea **<head>**.

Definirea unei funcții

Sintaxă:

```
function nume_functie(var1,var2,...,varX)
{
codul functiei
}
```

Parametrii **var1**, **var2**, etc. sunt variabile sau valori transmise funcției. Acoladele marchează începutul şi sfârşitul corpului funcției.
Obs: Chiar dacă funcția nu are parametri, parantezele rotunde de după numele funcției trebuie să fie prezente.
Exemplu:

```
<html>
<head>
<script type="text/javascript">
function afiseaza_mesaj()
{
alert("Bine ati venit!");
}
</script>
</head>
<body>
<h3>La apasarea butonului este apelata o functie JS
care afiseaza caseta alert</h3> <hr/>
<form>
```

```
<input type="button" value="Apasati!"
 onclick="afiseaza_mesaj()" />
</form>
</body>
</html>
```

Dacă linia de cod **alert("Bine ati venit!")** din exemplul anterior nu ar fi fost scrisă în corpul unei funcţii, codul ar fi fost executat imediat ce linia respectivă ar fi fost încărcată în browser. Deoarece codul a fost inclus într-o funcţie, el nu va fi executat decât atunci când utilizatorul acţionează butonul şi este apelată funcţia **afiseaza_mesaj()**.

Instrucţiunea `return`

Instrucţiunea **return** este folosită pentru a specifica valoarea returnată de o funcţie şi trebuie inclusă în orice funcţie care returnează o valoare.
În exemplul următor, funcţia suma returnează **suma** celor doi parametri de intrare:

```
<html>
<head>
<script type="text/javascript">
function suma(a,b)
{
return a+b;
}
</script>

</head>
<body>

<h3>Suma urmatoare este calculata si returnata de o
functie</h3> <hr/>
<script type="text/javascript">
document.write("7+9="+suma(7,9));
</script>

</body>
</html>
```

Durata de viaţă a variabilelor JavaScript

Dacă declaraţi o variabilă în interiorul unei funcţii, ea poate fi accesată numai din interiorul funcţiei. Când funcţia se încheie, variabila este distrusă. Variabilele declarate în corpul unei funcţii se numesc **variabile locale**. Puteţi avea variabile locale cu acelaşi nume în funcţii diferite, deoarece fiecare

variabilă locală este recunoscută numai în interiorul funcției în care este declarată. Dacă declarați o variabilă în afara tuturor funcțiilor (**variabilă globală**), ea poate fi accesată de toate funcțiile din pagină. O variabilă globală este distrusă numai atunci când pagina este închisă.

Exemple

Exemplul 1
Ilustrează cum se poate transmite o variabilă unei funcții şi cum poate fi folosită respectiva variabilă în corpul funcției.

```
<html>
<head>
<script type="text/javascript">
function functia_1(text)
{
alert(text);
}
</script>
</head>
<body>

<h3>Functii JavaScript</h3> <hr/>
<form>
<input type="button" onclick="functia_1('Bune ati
venit!')" value="Apasati">
</form>
<p>Cand apasati butonul, va fi apelata o functie cu
textul "Bine ati venit!" drept parametru. Functia va
afisa parametrul cu o caseta de alertare.</p>

</body>
</html>
```

Exemplul 2
Ilustrează cum poate fi folosit rezultatul returnat de o funcție.

```
<html>
<head>
<script type="text/javascript">
function functie_2()
{
return ("Bine ati venit!");
}
</script>
```

```
</head>
<body>

<h3>Textul urmator este returnat de o functie apelata
direct din document.write()</h3> <hr/>
<script type="text/javascript">
document.write(functie_2())
</script>

</body>
</html>
```

Exemplul 3

```
<html>
<head>
<script type="text/javascript">
function salut(txt)
{
alert(txt);
}
</script>
</head>

<body>
<h3>Utilizarea functiilor JavaScript</h3> <hr/>
<form>
<input type="button"
onclick="salut('Buna dimineata!')"
value="Dimineata">

<input type="button"
onclick="salut('Buna seara!')"
value="Seara">
</form>

<p>
Cand apasati unul dintre butoane, va fi apelata o
functie care afiseaza mesajul primit ca
parametru.</p>

</body>
</html>
```

11. Instrucțiunea `for`

Instrucțiunile repetitive sunt utilizate pentru a executa o secvență de cod în mod repetat. În JS sunt două tipuri diferite de instrucțiuni repetitive:

- **`for`** – repetă o secvență de cod de un număr precizat de ori
- **`while`** – repetă o secvență de cod cât timp o condiție este adevărată

Instrucțiunea **`for`** se utilizează când se cunoaşte dinainte numărul de iterații dorit pentru secvența de cod.

Sintaxă:

```
for
(var=val_initiala;var<=val_finala;var=var+increment)
{
codul ce trebuie executat
}
```

Exemplu:

În exemplul următor, instrucțiunea **`for`** începe cu **i=0**, corpul instrucțiunii se repetă cât timp **i≤50** şi contorul **i** este incrementat cu 2 la fiecare iterație. Vor fi afişate numerele pare ≤50.

Obs: Valoarea **increment** poate fi şi negativă şi comparația se poate realiza şi cu orice alt operator de comparare.

```
<html>
<body>
<h3>Utilizarea instructiunii for</h3> <hr/>
<script type="text/javascript">
document.write("Numerele pare mai mici sau egale cu
50: "+"<br/>");
var i=0;
for (i=0;i<=50;i+=2)
{
document.write(i +"  ");
}
</script>
</body>
</html>
```

Exemplu:

În acest exemplu, instrucțiunea **`for`** este utilizată pentru a afişa cele 6 titluri HTML.

```
<html>
<body>
```

```
<h3>Afisarea titlurilor HTML cu instructiunea
for</h3> <hr/>
<script type="text/javascript">
for (i = 1; i <= 6; i++)
{
document.write("<h" + i + ">Aceste este un titlu " +
i);
document.write("</h" + i + ">");
}
</script>

</body>
</html>
```

12. Instrucțiunea `while`

Instrucțiunea execută în mod repetat o secvență de cod, cât timp o condiție este adevărată.

Sintaxă:

```
while (var<=val_finala)
  {
  codul ce trebuie executat
  }
```

Obs: Operatorul <= poate fi înlocuit cu orice alt operator de comparare.

Exemplu:

```
<html>
<body>
<h3>Utilizarea instructiunii while</h3> <hr/>
<script type="text/javascript">
document.write("Numerele naturale mai mici egale cu
5:"+"<br/>");
var i=0;
while (i<=5)
  {
  document.write(i);
  document.write("<br/>");
  i++;
  }
</script>
```

```
</body>
</html>
```

Instrucțiunea do...while

Este o variantă a instrucțiunii while. Secvența de instrucțiuni va fi executată în mod sigur o dată, apoi în mod repetat, cât timp condiția specificată este adevărată.

Sintaxă:

```
do
  {
  codul ce trebuie executat
  }
while (var<=val_finala);
```

Exemplu:

Vor fi afișate numerele impare ≤50.

```
<html>
<body>
<h3>Utilizarea instructiunii do-while</h3> <hr/>
<script type="text/javascript">
document.write("Numerele impare mai mici sau egale cu
50:"+"<br/>");
var i=1;
do
  {
  document.write(i + " ");
   i+=2;
  }
while (i<=50);
</script>
</body>
</html>
```

13. Instrucțiunile `break` şi `continue`

Instrucțiunea `break`

Este utilizată pentru a întrerupe în mod forțat execuția unei bucle. În exemplul următor, bucla va fi întreruptă când **i=3**.

```
<html>
```

```
<body>
<h3>Utilizarea instructiunii break</h3> <br/>
<script type="text/javascript">
document.write("Numerele naturale mai mici egale cu
5:"+"<br/>");
var i=0;
for (i=0;i<=10;i++)
  {
  if (i==3)
    {
    document.write("Ciclu oprit cu break"); break;
    }
  document.write(i);
  document.write("<br />");
  }
</script>
</body>
</html>
```

Instrucţiunea `continue`

Instrucţiunea întrerupe execuţia iteraţiei curente şi sare la următoarea iteraţie. În exemplul următor, valoarea 3 nu va fi afişată (este sărită cu instrucţiunea **`continue`**).

```
<html>
<body>
<h3>Utilizarea instructiunii continue</h3> <hr/>
<script type="text/javascript">
document.write("Numerele naturale mai mici sau egale
cu 10:"+"<br/>");
var i=0
for (i=0;i<=10;i++)
  {
  if (i==3)
    {
     document.write("valoare sarita cu
continue"+"<br/>");continue;
    }
  document.write(i);
  document.write("<br />");
  }
</script>
</body>
</html>
```

14. Instrucțiunea `for...in`

Această instrucțiune este utilizată pentru a parcurge elementele unui tablou sau a enumera proprietățile unui obiect.

Sintaxă:

```
for (variabila in obiect)
  {
  cod ce trebuie executat
  }
```

Obs: Codul din corpul instrucțiunii este executat câte o dată pentru fiecare element din tablou sau proprietate.

Obs: Argumentul **variabila** poate fi o variabilă, un element de tablou sau o proprietate a unui obiect.

Exemplu

Instrucțiunea **`for..in`** este utilizată pentru a parcurge elementele unui tablou:

```
<html>
<body>
<h3>Parcurgerea elementelor unui tablou cu
instructiunea for..in</h3> <hr/>
<script type="text/javascript">
var x;
var pets = new Array();
pets[0] = "Pisica";
pets[1] = "Caine";
pets[2] = "Papagal";
pets[3] = "Hamster";
document.write("Valorile memorate in tablou
sunt:"+"<br/>");
for (x in pets)
  {
  document.write(pets[x] + "<br />");
  }
</script>

</body>
</html>
```

15. Evenimentele JavaScript

Utilizând JavaScript, putem crea pagini web dinamice. Evenimentele sunt acțiuni ce pot fi detectate de JavaScript. Fiecare element dintr-o pagină web are un anumit număr de evenimente care pot declanşa un script. Spre exemplu, putem utiliza evenimentul onClick al unui buton pentru a indica ce funcție va fi executată dacă utilizatorul acționează butonul respectiv. Evenimentele sunt definite în tagurile HTML.
Exemple de evenimente

- Un click de mouse
- Încărcarea unei pagini web sau a unei imagini
- Mişcarea mouse-ului peste o anumită zonă din pagina web
- Selectarea unui câmp de intrare dintr-un formular HTML
- Submiterea unui formular HTML
- Apăsarea unei taste

Obs: Evenimentele sunt în mod normal asociate cu funcții, care nu vor fi executate înainte de a avea loc evenimentul.

Evenimentele `onLoad` şi `onUnload`

Aceste evenimente sunt declanşate când utilizatorul întră într-o pagină web, respectiv când părăseşte pagina.
Evenimentul **`onLoad`** este folosit în mod frecvent pentru a detecta tipul şi versiunea browserului utilizatorului şi a încărca varianta de pagină potrivită cu aceste informații.
Ambele evenimente sunt folosite frecvent pentru a stabili ce cookies vor fi setate când utilizatorul întră în sau părăseşte pagina. Spre exemplu, puteți întreba care este numele utilizatorului când acesta vizitează prima dată pagina. Numele oferit de utilizator este memorat într-un cookie. Data viitoare când utilizatorul vă vizitează pagina, puteți să-l întâmpinați cu un mesaj personalizat cu numele său.

Evenimentele `onFocus`, `onBlur` şi `onChange`

Aceste evenimente sunt utilizate frecvent împreună cu validarea câmpurilor unui formular.
Exemplul următor ilustrează utilizarea evenimentului **`onChange`**. Funcția **verificaEmail()** va fi apelată ori de câte ori utilizatorul modifică conținutul unui câmp:

```
<input type="text" size="30" id="email"
onchange="verificaEmail()">
```

Evenimentul `onSubmit`

Acest eveniment este utilizat pentru a valida toate câmpurile unui formular înainte de trimiterea lui către server.

Exemplul următor ilustrează utilizarea evenimentului **`onSubmit`**. Funcția **verificaFormular()** va fi apelată atunci când utilizatorul acționează butonul **submit** din formular. Dacă valorile introduse în câmpuri nu sunt valide, trimiterea formularului va fi anulată. Funcția **verificaFormular()** returnează **true** sau **false**. Dacă funcția va returna valoarea **false**, trimiterea formularului va fi anulată:

```
<form method="post" action="xxx.htm" onsubmit="return
verificaFormular()">
```

Evenimentele `onMouseOver` şi `onMouseOut`

Aceste evenimente sunt utilizate frecvent pentru a crea butoane „animate".

În exemplul următor va fi afişată o casetă de alertă când este detectat un eveniment onMouseOver:

```
<a href="http://www.google.com"
onmouseover="alert('Un eveniment onMouseOver
detectat');return false"><img src="flowers.gif"
alt="Flowers" /></a>
```

16. Instrucțiunea `try...catch`

Când navigăm prin paginile web de pe internet, pot să apară mesaje de eroare la încărcarea unei pagini. În acest caz, uzual, apare o casetă de alertare JavaScript care ne anunță că s-a detectat o eroare de execuție (runtime error) şi ne întreabă dacă dorim să depanăm codul paginii. Aceste mesaje sunt utile pentru proiectanții paginilor web, nu şi pentru vizitatori care, de obicei, părăsesc pagina respectivă. În acest capitol veți învăța cum să gestionați mesajele de eroare JavaScript, astfel încât să nu vă pierdeți audiența.

Instrucțiunea **`try...catch`** vă permite să testați blocurile de cod pentru a depista erorile. Blocul **`try`** conține codul ce trebuie executat, iar blocul **`catch`** conține codul ce va fi executat dacă apare o eroare.

Sintaxă:

```
try
  {
  codul ce trebuie executat
  }
catch(err)
```

```
  {
  gestionarea erorilor
  }
```

Exemple

În exemplul următor ar trebui afişată o casetă de alertare cu mesajul "Bine ati venit!" când butonul este acționat.. Totuşi, în corpul funcției **mesaj()** există o eroare, cuvântul rezervat **alert** este scris greşit. Această eroare va fi detectată de JS. Blocul **catch** sesizează eroarea şi execută un cod special pentru a o rezolva. Acest cod afişează un mesaj de eroare pentru a informa utilizatorul ce se întâmplă. Dacă utilizatorul apasă butonul OK, încărcarea paginii va continua fără probleme:

```
<html>
<head>
<script type="text/javascript">
var txt="";
function mesaj()
{
try
  {
  adddlert("Bine ati venit!");
  }
catch(err)
  {
  text="In aceasta pagina este o eroare.\n\n";
  text+="Descrierea erorii: " + err.description +
"\n\n";
  text+="Pentru a continua apasati OK.\n\n";
  alert(text);
  }
}
</script>
</head>
<body>
<h3>Utilizarea instructiunii try..catch pentru
sesizarea erorilor</h3> <hr/>
<input type="button" value="Vedeti mesajul"
onclick="mesaj()" />
</body>

</html>
```

În exemplul următor `alert` este de asemenea scris greşit. Blocul **`catch`** utilizează o casetă de confirmare pentru a afişa un mesaj care informează utilizatorii că pot apăsa **OK** pentru a continua să viziteze pagina în care a fost depistată eroarea sau pot apăsa **Cancel** dacă doresc să se întoarcă la pagina principală (homepage). Dacă metoda **`confirm`** returnează **false** (utilizatorul a acționat butonul **Cancel**), atunci utilizatorul este redirectat. Dacă confirm returnează **true**, codul din blocul **`catch`** nu are nici-un efect:

```
<html>
<head>
<script type="text/javascript">
var txt="";
function mesaj()
{
try
  {
  adddlert("Bine ati venit!");
  }
catch(err)
  {
  text="In aceasta pagina este o eroare.\n\n";
  text+="Apasati OK daca doriti sa continuati
vizualizarea paginii,\n";
  text+="sau Cancel pentru a va intoarce la pagina
principala.\n\n";
  if(!confirm(text))
    {
    document.location.href="http:
                //carpenmanuela.wik.is/";
    }
  }
}
</script>
</head>

<body>
<h3>Un alt exemplu de utilizare a instructiunii
try..catch</h3> <hr/>
<input type="button" value="Vedeti mesajul"
onclick="mesaj()" />
</body>

</html>
```

17. Instrucțiunea `throw`

Această instrucțiune vă permite să creați o excepție. Dacă o utilizați împreună cu instrucțiunea **`try...catch`**, puteți controla execuția programului şi afişa mesaje de eroare adecvate.

Sintaxă:

```
throw(exceptie)
```

Argumentul **exceptie** poate fi un şir de caractere, un număr întreg, o valoare booleană sau un obiect.

Examplu:

Exemplul următor testează valoarea variabilei **x**. Dacă valoarea este mai mare decât 10, mai mică decât 10 sau nu este un număr, blocul **`throw`** aruncă o eroare. Această eroare este prinsă de blocul **`catch`** care afişează un mesaj corespunzător:

```
<html>
<body>
<h3>Utilizarea instructiunii throw pentru tratarea
corespunzatoare a erorilor</h3> <hr/>
<script type="text/javascript">
var x=prompt("Introduceti un numar cuprins intre 0 si
10:","");
try
  {
  if(x>10)
    {
    throw "Err1";
    }
  else if(x<0)
    {
    throw "Err2";
    }
  else if(isNaN(x))
    {
    throw "Err3";
    }
  }
catch(er)
  {
  if(er=="Err1")
    {
    alert("Eroare! Valoarea este prea mare");
```

```
    }
  if(er=="Err2")
{
    alert("Eroare! Valoarea este prea mică");
    }
  if(er=="Err3")
    {
    alert("Eroare! Valoarea nu este un numar");
    }
  }
</script>
</body>
</html>
```

Exemplu:
Exemplul următor ilustrează utilizarea evenimentului **onerror**.

```
<html>
<head>
<script type="text/javascript">
onerror=handleErr;
var txt="";

function handleErr(msg,url,l)
{
txt="In aceasta pagina este o eroare.\n\n";
txt+="Eroare: " + msg + "\n";
txt+="URL: " + url + "\n";
txt+="Linie: " + l + "\n\n";
txt+="Pentru a continua apasati OK.\n\n";
alert(txt);
return true;
}

function message()
{
adddlert("Bine ai venit!");
}
</script>
</head>

<body>
<h3>Exemplu de utilizare a evenimentului onerror</h3>
<hr/>
```

```
<input type="button" value="Afiseaza mesajul"
onclick="message()" />
</body>

</html>
```

18. Inserarea caracterelor speciale

Pentru a insera într-un şir caractere speciale, cum ar fi apostrof, ghilimele, întrerupere de linie etc., se foloseşte caracterul backslash (\).
Fie următorul cod JavaScript:

```
var txt="Noi suntem echipa "Dinamo" din Bucuresti.";
document.write(txt);
```

În JavaScript, un şir de caractere începe şi termină cu apostrof sau cu ghilimele. Asta înseamnă că şirul de mai sus va fi trunchiat la: Noi suntem echipa
Pentru a rezolva această problemă, trebuie să inserați caracterul backslash (\) înaintea fiecărui caracter special care trebuie afişat, ca în exemplul următor:

```
var txt="Noi suntem echipa \"Dinamo\" din
Bucuresti.";
document.write(txt);
```

Tabelul următor prezintă caracterele speciale ce pot fi inserate într-un text cu ajutorul caracterului backslash:

Cod	Ieşire
\'	apostrof
\"	ghilimele
\&	ampersand
\\	backslash
\n	linie nouă
\r	retur de car
\t	tab
\b	backspace
\f	form feed

19. Obiectele JavaScript

Programarea orientată pe obiecte

JavaScript este un limbaj de programare orientat pe obiecte (POO). Un limbaj POO vă permite să vă definiți propriile obiecte şi propriile tipuri de variabile.
Crearea propriilor obiecte va fi explicată mai târziu, în secțiunea JavaScript avansat. Vom începe prin a examina obiectele încorporate în JS şi cum sunt ele utilizate. Rețineți că un obiect este de fapt, un tip special de date care are proprietăți şi metode.

Proprietăți

Proprietățile sunt valori asociate cu un obiect.
În exemplul următor, utilizăm proprietatea **`length`** a obiectului **`String`** (şir de caractere) pentru a determina numărul de caractere memorate într-un şir:

```
<script type="text/javascript">
var txt="Bine ati venit!";
document.write(txt.length);
</script>
```

Codul de mai sus va afişa valoarea: **`15`**

Metode

Metodele sunt acțiuni ce pot fi realizate cu un obiect.
În exemplul următor, utilizăm metoda UpperCase() a obiectului String pentru a afişa un text cu litere mari:

```
<script type="text/javascript">
var txt="Bine ati venit!";
document.write(txt.toUpperCase());
</script>
```

Codul de mai sus va afişa şirul: **BINE ATI VENIT!**

20. Obiectul `String`

Obiectul String este folosit pentru a manipula secvențe de caractere (text). Un obiect String este creat cu instrucțiunea **`new String()`**.
Sintaxa:

```
var txt = new String(string);
```

sau mai simplu:

```
var txt = string;
```

Proprietățile obiectului `String`

Proprietate	Descriere
`constructor`	Returnează funcția care a creat prototipul obiectului **String**
`length`	Returnează lungimea şirului
`prototype`	Permite adăugarea de proprietăți şi metode unui obiect

Metodele obiectului `String`

Metodă	Descriere
`charAt()`	Returnează caracterul cu indexul specificat
`charCodeAt()`	Returnează codul Unicode al caracterului cu indexul specificat
`concat()`	Concatenează două sau mai multe şiruri şi returnează şirul obținut
`fromCharCode()`	Converteşte valori Unicode în caractere
`indexOf()`	Returnează poziția primei apariții a unui subşir într-un şir
`lastIndexOf()`	Returnează poziția ultimei apariții a unui subşir într-un şir
`match()`	Caută potrivirile dintre un subşir şi un string şi returnează subşirul sau **null** (dacă subşirul nu este găsit)
`replace()`	Caută toate aparițiile unui subşir într-un şir şi le înlocuieşte cu un nou subşir
`search()`	Caută potrivirea dintre un subşir şi un şir şi returnează poziția în care apare potrivirea
`slice()`	Elimină o porțiune din şir şi returnează şirul extras
`split()`	Împarte un şir în subşiruri pe baza unui caracter separator
`substr()`	Extrage dintr-un şir secvența de caractere care începe într-o anumită poziție şi are o anumită lungime
`substring()`	Extrage dintr-un şir caracterele situate între două poziții
`toLowerCase()`	Converteşte un şir în litere mici
`toUpperCase()`	Converteşte un şir în litere mari
`valueOf()`	Returnează valoarea primară a unui obiect **String**

Metode împachetate în taguri HTML

Aceste metode returnează şirul împachetat în tagurile HTML potrivite.

Metodă	Descriere
`anchor()`	Creează o ancoră
`big()`	Afişează şirul cu font mare
`blink()`	Afişează un şir care clipeşte
`bold()`	Afişează şirul cu font bold
`fixed()`	Afişează şirul cu un font cu pas fix
`fontcolor()`	Afişează şirul folosind o anumită culoare
`fontsize()`	Afişează şirul cu o anumită dimensiune a fontului
`italics()`	Afişează şirul cu font italic
`link()`	Afişează şirul ca hiperlegătură
`small()`	Afişează şirul cu font mic
`strike()`	Afişează şirul ca tăiat
`sub()`	Afişează şirul ca subscript (indice)
`sup()`	Afişează şirul ca superscript (exponent)

Exemple

Exemplul 1
Ilustrează utilizarea proprietăţii **`length`** pentru a determina lungimea unui şir.

```
<html>
<body>

<h3>Obiectul String. Determinarea lungimii unui
sir</h3> <hr/>
<script type="text/javascript">
var txt="Bine ati venit!";
document.write("Sirul este: "+txt+"<br/>");
document.write("Are lungimea "+txt.length);
</script>
<p><b>Obs.</b>Sirul nu se modifica.</p>
</body>
</html>
```

Exemplul 2
Ilustrează cum se utilizează tagurile HTML pentru a stiliza un şir.

```
<html>
<body>
```

```
<h3>Obiectul String. Utilizarea tagurilor HTML pentru
stilizarea unui sir.</h3> <hr/>
<script type="text/javascript">

var txt="Bine ati venit!";

document.write("<p>Big: " + txt.big() + "</p>");
document.write("<p>Small: " + txt.small() + "</p>");

document.write("<p>Bold: " + txt.bold() + "</p>");
document.write("<p>Italic: " + txt.italics() +
"</p>");

document.write("<p>Blink: " + txt.blink() + " (nu
functioneaza in IE, Chrome, Safari)</p>");
document.write("<p>Fixed: " + txt.fixed() + "</p>");
document.write("<p>Strike: " + txt.strike() +
"</p>");

document.write("<p>Fontcolor: " +
txt.fontcolor("Blue") + "</p>");
document.write("<p>Fontsize: " + txt.fontsize(14) +
"</p>");

document.write("<p>Subscript: " + txt.sub() +
"</p>");
document.write("<p>Superscript: " + txt.sup() +
"</p>");

document.write("<p>Link: " +
txt.link("http://www.google.com") + "</p>");
</script>
<br/> <br/>
<p><b>Obs.</b>Sirul stilizat nu se modifica!</p>
</body>
</html>
```

Exemplul 3

Ilustrează cum se utilizează metoda **`concat()`** pentru a concatena şiruri. Concatenarea a două şiruri:

```
<html>
<body>
```

```
<h3>Obiectul String. Concatenarea a doua siruri.</h3>
<hr/>
<script type="text/javascript">

var txt1 = "Buna ";
var txt2 = "ziua!";
document.write("Primul sir este: "+txt1+"<br/>");
document.write("Al doilea sir este: "+txt2+"<br/>");
document.write("Sirul concatenat este:
"+txt1.concat(txt2)+"<br/>");

</script>
<p><b>Obs.</b>Sirurile concatenate nu se modifica.
Rezultatul concatenarii poate fi pastrat intr-un nou
sir.</p>
</body>
</html>
```

Concatenarea a trei şiruri:

```
<html>
<body>

<h3>Obiectul String. Concatenarea a trei siruri.</h3>
<hr/>
<script type="text/javascript">
var txt1="Buna ";
var txt2="ziua!";
var txt3=" Bine ati venit!";
document.write("Primul sir este:  "+txt1+"<br/>");
document.write("Al doilea sir este:  "+txt2+"<br/>");
document.write("Al treilea sir este:
"+txt3+"<br/>");
document.write("Sirul concatenat este:
"+txt1.concat(txt2,txt3)+"<br/>");

</script>
<p><b>Obs.</b>Sirurile concatenate nu se modifica.
Rezultatul concatenarii poate fi pastrat intr-un nou
sir.</p>
</body>
</html>
```

Exemplul 4
Ilustrează cum se utilizează metoda **indexOf()** pentru a determina poziția primei apariții a unei valori într-un şir.

```
<html>
<body>

<h3>Obiectul String. Cautarea primei aparitii a unei
valori in sir cu indexof().</h3> <hr/>
<script type="text/javascript">
var str="Buna ziua!";
document.write("Sirul in care se cauta este:
"+str+"<br/>");
document.write("Sirul \"Buna\" apare in sir in
pozitia "+str.indexOf("Buna") + "<br />");
document.write("Sirul \"ZIUA\" apare in sir in
pozitia "+str.indexOf("ZIUA") + "<br />");
document.write("Sirul \"ziua\" apare in sir in
pozitia "+str.indexOf("ziua"));
</script>
<p><b>Obs.</b>Sirul nu se modifica in urma
cautarii!</p>
</body>
</html>
```

Valorile afişate sunt: **0 -1 5**.
Obs. Dacă valoarea nu apare în şir, valoarea returnată este -1. Şirurile sunt indexate de la 0.

Exemplul 5
Ilustrează cum se utilizează metoda **match()** pentru a căuta un subşir într-un şir. Metoda returnează subşirul, dacă este găsit, sau valoarea **null**, dacă subşirul nu este găsit în şir.

```
<html>
<body>

<h3>Obiectul String. Cauta unui subsir intr-un sir cu
match().</h3> <hr/>
<script type="text/javascript">
var str="Hello world!";
document.write("Sirul in care se cauta este:
"+str+"<br/>");
document.write("Sirul cautat:  "+"world"+".  ");
document.write("Valoarea returnata:
"+str.match("world") + "<br />");
```

```
document.write("Sirul cautat:  "+"World"+".  ");
document.write("Valoarea returnata:
"+str.match("World") + "<br />");
document.write("Sirul cautat:  "+"worlld"+".  ");
document.write("Valoarea returnata:
"+str.match("worlld") + "<br />");
</script>
<p><b>Obs.</b>Sirul nu se modifica in urma cautarii.
Rezultatul poate fi memorat intr-o variabila.</p>
</body>
</html>
```

Exemplul 6
Ilustrează cum se utilizează metoda **replace()** pentru a înlocui o secvență din şir cu altă secvență.

```
<html>
<body>

<h3>Obiectul String. Inlocuirea unei secvente din sir
cu replace().</h3> <hr/>
<script type="text/javascript">

var str="Vizitati Microsoft!";
document.write("Sirul initial este:  "+str+"<br/>");
document.write("Subsirul care se modifica este:
"+"Microsoft"+"<br/>");
document.write("Subsirul cu care se inlocuieste este:
"+"Google"+"<br/>");
str.replace("Microsoft","Google");
document.write("Sirul obtinut este:  "+str);

</script>
<p><b>Obs.</b>Sirul se modifica!</p>
</body>
</html>
```

Exemplul 7
Ilustrează cum se foloseşte metoda **slice()** pentru a extrage dintr-un şir o secvență. Metoda returnează şirul extras sau valoarea -1. În mod normal are două argumente: poziția din care începe extragerea (primul caracter are indexul 0) şi, opțional, poziția în care se încheie extragerea. Dacă al doilea argument lipseşte, se vor extrage toate caracterele dintre poziția de început şi sfârşitul

şirului. Dacă se folosesc valori negative, extragerea se va face numărând înapoi de la sfârşitul şirului.

```
<html>
<body>

<h3>Obiectul String. Extragerea unui subsir dintr-un
sir cu slice().</h3> <hr/>
<script type="text/javascript">

var str="Bine ati venit!";
document.write("Sirul initial este: "+str+"<br/>");
// extrage toate caracterele incepand cu pozitia 0:
var txt=str.slice(0);
document.write("Subsirul extras cu slice(0):
"+txt+"<br/>");

//extrage toate caracterele incepand cu pozitia 5:
txt=str.slice(5);
document.write("Subsirul extras cu slice(5):
"+txt+"<br />");

//extrage ultimele 6 caractere de la sfarsitul
sirului:
txt=str.slice(-6);
document.write("Subsirul extras cu slice(-6):
"+txt+"<br />");

//extrage primul caracter din sir:
txt=str.slice(0,1);
document.write("Subsirul extras cu slice(0,1):
"+txt+"<br />");

//extrage caracterele dintre pozitiile 5 si 10
txt=str.slice(5,10);
document.write("Subsirul extras cu slice(5,10):
"+txt+"<br />");

</script>
<p><b>Obs.</b> Sirul nu se modifica in urma
extragerii. Subsirul extras poate fi memorat intr-o
variabila.</p>
</body>
</html>
```

Exemplul 8

Ilustrează utilizarea metodei **split()** pentru a împărți un şir în subşiruri pe baza unui caracter separator. Dacă caracterul separator este omis, se va returna întreg şirul. Dacă caracterul este şirul vid, şirul va fi împărțit caracter cu caracter. Opțional, se poate preciza şi numărul maxim de împărțiri.

```
<html>
<body>

<h3>Obiectul String. Impartirea unui sir in subsiruri
cu split().</h3> <hr/>
<script type="text/javascript">

var str="Bine ati venit";

document.write("Sirul initial:  "+str+"<br/>");

//este returnat intreg sirul
document.write("Sirurile returnate cu split():
"+str.split() + "<br />");

//sunt returnate cele trei cuvinte din sir
document.write("Sirurile returnate cu split(\" \"):
"+str.split(" ") + "<br />");

//sunt returnate caracterele din şir
document.write("Sirurile returnate cu split(\"\"):
"+str.split("") + "<br />");

//sunt returnate numai primele doua cuvinte din sir
document.write("Subsirurile returnate cu split(\"
\",2):  "+str.split(" ",2)+"<br/>");

</script>
<p><b>Obs.</b> Sirul initial nu se modifica.
Rezultatul poate fi memorat intr-o variabila.
</body>
</html>
```

21. Obiectul `Date`

Obiectul **`Date`** este utilizat pentru a lucra cu date calendaristice şi ore. Un obiect de tip **`Date`** este creat cu instrucţiunea **`new Date()`**. Sunt patru metode de a instanţia un obiect **`Date`**:

```
var d = new Date();
var d = new Date(milisecunde);
var d = new Date(dataString);
var d = new Date(an, luna, zi, ore, minute, secunde,
milisecunde);
```

Setarea datei

Putem manevra uşor datele calendaristice folosind metodele obiectului Date.
În exemplul următor, data este setată la 19 februarie 2010:

```
var myDate=new Date();
myDate.setFullYear(2010,1,19);
```

În exemplul următor, data este setată la şapte zile în viitor:

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+7);
```

Compararea a două date calendaristice

Exemplul următor compară data curentă cu 19 februarie 2010:

```
var myDate=new Date();
myDate.setFullYear(2010,1,19);
var today = new Date();

if (myDate>today)
  {
  alert("Astazi este inainte de 19 Februarie 2010");
  }
else
  {
  alert("Astazi este dupa 19 Februarie 2010");
  }
```

Metodele obiectului `Date`

Metoda	Descriere
`getDate()`	Returnează ziua din lună (între 1 şi 31)

getDay()	Returnează ziua din săptămână (0-6)
getFullYear()	Returnează anul (patru cifre)
getHours()	Returnează ora (0-23)
getMilliseconds()	Returnează milisecundele (0-999)
getMinutes()	Returnează minutele (0-59)
getMonth()	Returnează luna (0-11)
getSeconds()	Returnează secundele (0-59)
getTime()	Returnează numărul de milisecunde scurse de la 1.01.1970
getTimezoneOffset()	Returnează diferența dintre GMT şi timpul local, în minute
parse()	Analizează(parsează) o dată ca şir de caractere şi returnează numărul de milisecunde scurse de la 1.01.1970
setDate()	Setează data din lună (1-31)
setFullYear()	Setează anul (patru cifre)
setHours()	Setează ora (0-23)
setMilliseconds()	Setează milisecundele (0-999)
setMinutes()	Setează minutele (0-59)
setMonth()	Setează lunile (0-11)
setSeconds()	Setează secundele (0-59)
setTime()	Setează o dată şi o oră adunând sau scăzând un anumit număr de milisecunde la/din 1.01.1970
toDateString()	Converteşte porțiunea corespunzătoare datei calendaristice dintr-un obiect **Date** intr-un şir de caractere
toString()	Converteşte un obiect **Date** într-un şir de caractere
toTimeString()	Converteşte porțiunea corespunzătoarea timpului dintr-un obiect **Date** într-un şir de caractere
valueOf()	Returnează valoarea primară a unui obiect **Date**

Exemple

Exemplul 1

Ilustrează utilizarea metodei **Date()** pentru a obține data curentă.

```
<html>
<body>

<h3>Obiectul Date. Obtinerea datei curente cu
Date().</h3> <hr/>
<script type="text/javascript">

document.write("Astazi este:  "+Date());

</script>

</body>
</html>
```

Exemplul 2

Ilustrează utilizarea metodei **getTime()** pentru a calcula anii scurşi din 1970 până în prezent.

```
<html>
<body>

<h3>Obiectul Date. Utilizarea metodei getTime().</h3>
<hr/>
<script type="text/javascript">
var d=new Date();
document.write("Au trecut "+d.getTime() + "
milisecunde din 01.01.1970 si pana acum.");
</script>

</body>
</html>
```

Exemplul 3

Ilustrează utilizarea metodei **setFullYear()** pentru a seta o dată specifică.

```
<html>
<body>

<h3>Obiectul Date. Setarea datei cu
setFullYear().</h3> <hr/>
<script type="text/javascript">

var d = new Date();
d.setFullYear(2010,1,19);
```

```
document.write("Data a fost setata la "+d);

</script>

</body>
</html>
```

Exemplul 4
Ilustrează utilizarea metodei **toString()** pentru a converti data curentă într-un şir de caractere.

```
<html>
<body>

<script type="text/javascript">

var d=new Date();
document.write(d.toString());

</script>

</body>
</html>
```

Exemplul 5
Ilustrează utilizarea metodei **getDay()** şi a unui tablou pentru a scrie denumirea zilei din săptămână curente.

```
<html>
<body>

<h3>Obiectul Date. Utilizarea metodei getDay() pentru
a determina ziua din saptamana.</h3> <hr/>
<script type="text/javascript">

var d=new Date();
var weekday=new Array(7);
weekday[0]="Duminica";
weekday[1]="Luni";
weekday[2]="Marti";
weekday[3]="Miercuri";
weekday[4]="Joi";
weekday[5]="Vineri";
weekday[6]="Sambata";
```

```
document.write("Astazi este " + weekday[d.getDay()]);

</script>

</body>
</html>
```

Exemplul 6

Ilustrează cum se poate afişa un ceas într-o pagină web.

```
<html>
<head>
<script type="text/javascript">
function ceas()
{
var today=new Date();
var h=today.getHours();
var m=today.getMinutes();
var s=today.getSeconds();
//functia urmatoare adauga un zero in fata
//numerelor<10
m=verifica(m);
s=verifica(s);
document.getElementById('txt').innerHTML=h+":"+m+":"+
s;
t=setTimeout('ceas()',500);
}

function verifica(i)
{
if (i<10)
  {
  i="0" + i;
  }
return i;
}
</script>
</head>

<body onload="ceas()">
<h3>Obiectul String. Afisarea unui ceas.</h3> <hr/>
<div id="txt"></div>
</body>
</html>
```

22. Obiectul `Array`

Un tablou este o variabilă specială care poate păstra la un moment dat mai multe valori de un anumit tip. Dacă aveți o listă de elemente, animale de companie de exemplu, ați putea păstra valorile în variabile simple, ca în exemplul următor:

```
pet1="Caine";
pet2="Pisica";
pet3="Papagal";
```

Desigur, problema se complică dacă aveți de memorat zeci, sau sute de valori. Cea mai bună soluție este să folosiți tablouri. Un tablou poate reține toate valorile sub un singur nume şi puteți accesa fiecare valoare stocată în tablou folosind numele tabloului şi indexul valorii.

Crearea unui tablou

Un tablou poate fi definit în trei moduri:

```
1:
var pets=new Array();
//tablou obisnuit
pets[0]="Caine";
pets[1]="Pisica";
pets[2]="Papagal";
2:
var pets=new Array("Caine","Pisica","Papagal");
//tablou condensat
3:
var pets=["Caine","Pisica","Papagal"];
//tablou literal
```

Obs: Dacă în tablou stocați valori numerice sau logice, tipul tabloului va fi **Number** sau **Boolean**, în loc de **String**.

Accesarea elementelor dintr-un tablou

Puteți accesa un element dintr-un tablou precizând numele tabloului şi indexul elementului. Primul element din tablou are indexul 0.
Următoarea linie de cod

```
document.write(pets[0]);
```

va afişa şirul: **Caine**

Modificarea valorilor dintr-un tablou

Pentru a modifica o valoare dintr-un tablou, este suficient să atribuiți o nouă valoare elementului respectiv, ca în exemplul următor:

```
pets[0]="Iguana";
```

Proprietățile obiectului Array

Proprietatea	Descriere
`constructor`	Returnează funcția care a creat prototipul obiectului **Array**
`length`	Setează sau returnează numărul elementelor stocate în tablou
`prototype`	Permite adăugare de proprietăți şi metode unui obiect

Metodele obiectului Array

Metoda	Descriere
`concat()`	Concatenează două sau mai multe tablouri şi returnează tabloul obținut
`join()`	Concatenează toate elementele unui tablou într-un şir de caractere
`pop()`	Înlătură ultimul element dintr-un tablou şi returnează respectivul element
`push()`	Adaugă noi elemente la sfârşitul unui tablou şi returnează noua lungime a tabloului
`reverse()`	Răstoarnă ordinea elementelor dintr-un tablou
`shift()`	Înlătură primul element dintr-un tablou şi returnează respectivul element
`slice()`	Selectează o parte dintr-un tablou şi returnează elementele selectate
`sort()`	Sortează elementele unui tablou
`splice()`	Adaugă/Înlătură elemente dintr-un tablou.
`toString()`	Converteşte un tablou în şir de caractere şi returnează rezultatul
`unshift()`	Adaugă noi elemente la începutul unui tablou şi returnează noua lungime a tabloului
`valueOf()`	Returnează valoarea primară a unui tablou

Exemple

Exemplul 1
Ilustrează crearea unui tablou, atribuirea de valori şi afişarea elementelor tabloului.

```
<html>
```

```
<body>

<h3>Obiectul Array. Crearea unui tablou,
initializarea si afisarea elementelor.</h3> <hr/>
<script type="text/javascript">
var pets = new Array();
pets[0] = "Pisica";
pets[1] = "Caine";
pets[2] = "Perus";

document.write("Elementele memorate in tablou
sunt:"+"<br/>");
for (i=0;i<pets.length;i++)
{
document.write(pets[i] + "<br />");
}
</script>

</body>
</html>
```

Exemplul 2
Ilustrează utilizarea instrucțiunii **for...in** pentru a parcurge elementele unui tablou.

```
<html>
<body>
<h3>Obiectul Array. Afisarea elementelor unui tablou
cu instructiunea for..in.</h3> <hr/>
<script type="text/javascript">
var x;
var pets = new Array();
pets[0] = "Pisica";
pets[1] = "Caine";
pets[2] = "Perus";

document.write("Elementele memorate in tablou
sunt:"+"<br/>");
for (x in pets)
{
document.write(pets[x] + "<br />");
}
</script>
</body>
```

```
</html>
```

Exemplul 3
Ilustrează utilizarea metodei **concat()** pentru a concatena trei tablouri.

```
<html>
<body>

<h3>Obiectul Array. Concatenarea a trei tablouri cu
concat().</h3> <hr/>
<script type="text/javascript">

var parinti = ["Maria", "George"];
var copii = ["Elena", "Mihai"];
var frati = ["Paul", "Dan"];
var familie = parinti.concat(copii,frati);
document.write("Parinti:  "+parinti+"<br/>");
document.write("Copii:  "+copii+"<br/>");
document.write("Frati:  "+frati+"<br/>");
document.write("Familia: "+familie);

</script>
<p><b>Obs.</b>Tablourile concatenate nu se modifica.
Rezultatul concatenarii este un nou tablou.</p>
</body>
</html>
```

Exemplul 4
Ilustrează utilizarea metodei **join()** pentru a concatena toate elementele unui tablou într-un şir de caractere.

```
<html>
<body>

<h3>obiectul Array. Concatenarea elementelor unui
tablou intr-un sir de caractere cu join().</h3> <hr/>
<script type="text/javascript">

var fructe = ["Mere", "Pere", "Banane", "Kiwi"];
document.write("Tabloul contine valorile:
"+fructe+"<br/>");
document.write("Sirul concatenat cu join(\" \"):
"+fructe.join(" ") + "<br />");
document.write("Sirul concatenat cu join(\"+\"):
"+fructe.join("+") + "<br />");
```

```
document.write("Sirul concatenat cu join(\" si \"):
"+fructe.join(" si ")+"<br/>");

</script>
<p><b>Obs.</b>Tabloul nu se modifica. Sirul
concatenat poate fi memorat intr-o variabila.
</body>
</html>
```

Exemplul 5

Ilustrează utilizarea metodei **pop()** pentru a înlătura ultimul element dintr-un tablou.

```
<html>
<body>

<h3>Obiectul Array. Eliminarea ultimului element din
tablou cu pop().</h3> <hr/>
<script type="text/javascript">

var pets = ["Pisica", "Caine", "Hamster", "Iguana"];
document.write("Tablou initial contine valorile:
"+pets+"<br/>");

document.write("S-a eliminat valoarea:  "+pets.pop()
+ "<br />");

document.write("Tabloul dupa eliminare:  "+pets +
"<br />");

document.write("S-a eliminat valoarea:  "+pets.pop()
+ "<br />");

document.write("Tabloul dupa eliminare:  "+pets);


</script>
<p><b>Obs.</b> Tabloul se modifica! Valoarea
eliminata poate fi memorata intr-o variabila.</p>
</body>
</html>
```

Exemplul 6
Ilustrează utilizarea metodei **`push()`** pentru a adăuga noi elemente la sfârşitul unui tablou. Pot fi adăugate mai multe valori simultan. Valorile trebuie separate prin virgulă.

```
<html>
<body>

<h3>Obiectul Array. Adaugarea de noi elemente la
sfarsitul tabloului cu push().</h3> <hr/>
<script type="text/javascript">

var pets = ["Pisica", "Caine", "Perus", "Hamster"];

document.write("Tabloul initial:  "+pets+"<br/>");

document.write("Se adauga valorile:
"+"Iguana"+"<br/>");
pets.push("Iguana");
document.write("Tabloul obtinut:  "+pets+"<br/>");

document.write("Se adauga valorile:  "+"Pesti si
Iepure"+"<br/>");
k=pets.push("Pesti","Iepure");
document.write("Tabloul obtinut:  "+pets+"<br/>");

document.write("Tabloul final are "+k+" elemente");

</script>
<p><b>Obs.</b> Tabloul se modifica. Metoda returneaza
noua lungime a tabloului si
valoarea poate fi memorata intr-o variabila.</p>
</body>
</html>
```

Exemplul 7
Ilustrează utilizarea metodei **`reverse()`** pentru a inversa ordinea elementelor dintr-un tablou.

```
<html>
<body>

<h3>Obiectul Array. Inversarea ordinii elementelor
din tablou cu reverse().</h3> <hr/>
<script type="text/javascript">
```

```
var friends = ["Mihai", "Elena", "Andra", "Dan"];
document.write("Tabloul initial:  "+friends+"<br/>");
friends.reverse();
document.write("Tabloul dupa inversare:  "+friends);

</script>

</body>
</html>
```

Exemplul 8
Ilustrează utilizarea metodei **shift()** pentru a elimina primul element dintr-un tablou.

```
<html>
<body>
<h3>Obiectul Array. Eliminarea primului element din
tablou cu shift().</h3> <hr/>
<script type="text/javascript">

var friends = ["Mihai", "Elena", "Andra", "Dan"];

document.write("Tabloul initial:  "+friends+"<br/>");

var x=friends.shift();

document.write("Elementul eliminat:  "+x+"<br/>");
document.write("Tabloul dupa eliminare:  "+friends);

</script>
<p><b>Obs.</b> Tabloul se modifica. Metoda returneaza
elementul eliminat si rezultatul
poate fi memorat intr-o variabila.</p>
</body>
</html>
```

Exemplul 9
Ilustrează cum se selectează elementele unui tablou cu metoda **slice()**. Metoda are două argumente: primul precizează poziția de început a secvenței selectat, iar al doilea poziția de sfârşit. Dacă al doilea argument lipseşte, se vor selecta toate elementele până la sfârşitul tabloului. Dacă argumentul este negativ, se vor selecta elementele de la sfârşitul şirului către început.

```
<html>
```

```
<body>

<h3>Obiectul Array. Selectare elementelor din tablou
cu slice().</h3> <hr/>
<script type="text/javascript">

var pets = ["Caine", "Pisica", "Papagal", "Hamster"];

document.write("Tabloul initial:  "+pets+"<br/>");

var x=pets.slice(0,1);
document.write("Elementele selectate cu slice(0,1):
"+x+"<br/>");


document.write("Elementele selectate cu slice(1):
"+pets.slice(1) + "<br />");

document.write("Elementele selectate cu slice(-2):
"+pets.slice(-2) + "<br />");

</script>
<p><b>Obs.</b> Tabloul nu se modifica. Elementele
selectate pot fi memorate intr-o variabila.</p>
</body>
</html
```

Exemplul 10
Ilustrează utilizarea metodei **sort()** pentru a sorta alfabetic crescător un tablou de şiruri de caractere. Metoda sortează implicit în ordine alfabetică crescătoare.

```
<html>
<body>

<h3>Obiectul Array. Sortarea unui tablou cu
sort().</h3> <hr/>
<script type="text/javascript">

var friends = ["Mihai", "Elena", "Andra", "Dan"];
document.write("Tabloul initial:  "+friends+"<br/>");
document.write("Tabloul sortat:  "+friends.sort());

</script>
```

```
<p><b>Obs.</b> Tabloul se modifica!</p>
</body>
</html
```

Exemplul 11
Ilustrează utilizarea metodei **sort()** pentru a sorta descendent un tablou de numere. Numerele nu vor fi sortate corect. Trebuie adăugată o funcție care să compare numerele.

```
<html>
<body>

<h3>Obiectul Array. Sortarea unui tablou cu valori
numerice.</h3> <hr/>
<script type="text/javascript">

function sortDesc(a, b)
{
return b - a;
}

function sortCresc(a,b)
{
return a-b;
}

var n = new Array(10,5,40,25,100,1);
document.write("Tabloul initial:  "+n+"<br/>");
document.write("Tabloul sortat crescator:
"+n.sort(sortCresc)+"<br/>");
document.write("Tabloul sortat descrescator:
"+n.sort(sortDesc));

</script>
<p><b>Obs.</b> In urma sortarii tabloul se
modifica!</p>
</body>
</html>
```

Exemplul 12
Ilustrează utilizarea metodei **splice()** pentru a adăuga un element în poziția 3 din tablou. Metoda are trei argumente: primul, obligatoriu precizează poziția în care vor fi adăugate/şterse valori, al doilea, obligatoriu, reprezintă numărul

de valori care vor fi şterse (dacă are valoarea 0, nu se vor şterge ci se vor insera valori) şi, al treilea, opţional, care reprezintă noile valori adăugate în tablou.

```
<html>
<body>

<h3>Obiectul Array. Inserarea/stergerea elementelor
dintr-o pozitie a tabloului cu splice().</h3> <hr/>
<script type="text/javascript">

var friends = ["Ana", "Mircea", "Dan", "Maria"];

document.write("Tabloul initial:  "+friends+"<br/>");
document.write("Se adauga valoarea \"Andra\" in
pozitia 3 din tablou cu splice(3,0)"+"<br/>");
friends.splice(3,0,"Andra");
document.write("Tabloul obtinut:  "+friends+"<br/>");
document.write("Se sterg din tablou primele doua
valori cu splice(0,2)"+"<br/>");
document.write("Valorile sterse:
"+friends.splice(0,2)+"<br/>");
document.write("Tabloul obtinut:  "+friends);

</script>
<p><b>Obs.</b> Tabloul se modifica. Daca metoda
elimina elemente din tablou, va returna elementele
eliminate.</p>
</body>
</html>
```

Exemplul 13

Ilustrează utilizarea metodei **`toString()`** pentru a converti un tablou într-un şir de caractere. Metoda returnează şirul de caractere, valorile fiind separate prin virgulă.

```
<html>
<body>

<h3>Obiectul Array. Convertirea unui tablou in sir de
caractere cu toString().</h3> <hr/>
<script type="text/javascript">

var n = new Array(14,0,7,-4,25,13,7);
document.write("Tabloul contine valorile:"+"<br/>");
for(x in n) document.write(n[x]+"<br/>");
```

```
document.write("Sirul de caractere obtinut:"
+n.toString());

</script>
<p><b>Obs.</b> Tabloul nu se modifica. Metoda
returneaza sirul de caractere obtinut.</p>
</body>
</html>
```

Exemplul 14

Ilustrează utilizarea metodei **unshift()** pentru a adăuga noi valori la începutul unui tablou.

```
<html>
<body>

<h3>Obiectul Array. Adaugarea de elemente la
inceputul unui tablou cu unshift().</h3> <hr/>
<script type="text/javascript">

var pets = ["Pisica", "Caine", "Iguana", "Pesti"];
document.write("Tabloul initial:  "+pets+"<br/>");
document.write("Se adauga valoarea \"Papagal\"
"+"<br/>");
pets.unshift("Papagal");
document.write("Tabloul dupa adaugare:
"+pets+"<br/>");
document.write("Se adauga valorile \"Perus\" si
\"Broasca testoasa\" "+"<br/>");
k=pets.unshift("Perus","Broasca testoasa");
document.write("Tabloul dupa adaugare:
"+pets+"<br/>");
document.write("Tabloul final are "+k+" elemente.");
</script>

<p><b>Nota:</b> Metoda unshift() nu lucreaza corect
in Internet Explorer; returneaza undefined!</p>
<p><b>Obs.</b>Tabloul se modifica. Metoda returneaza
numarul de elemente din tablou dupa adaugare.
Valoarea poate fi memorata intr-o variabila.</p>

</body>
</html>
```

23. Obiectul `Boolean`

Obiectul **`Boolean`** este utilizat pentru a converti o valoare ne-booleană într-o valoare booleană (cu valoarea **true** sau **false**).
Crearea unui obiect boolean poate fi realizată ca în secvența de cod următoare:

```
var sem=new Boolean();
```

Obs: Dacă obiectul **`Boolean`** nu are valoare inițială sau are una din valorile **0**, **-0**, **null**, **""**, **false**, **undefined**, sau **NaN (not a number)**, obiectul este inițializat cu valoarea **false**. În caz contrar, valoarea obiectului va fi **true**.

Toate declarațiile din liniile următoare de cod creează un obiect boolean inițializat cu **false**:

```
var sem=new Boolean();
var sem=new Boolean(0);
var sem=new Boolean(null);
var sem=new Boolean("");
var sem=new Boolean(false);
var sem=new Boolean(NaN);
```

Toate declarațiile din liniile următoare de cod crează un obiect boolean inițializat cu **true**:

```
var sem=new Boolean(true);
var sem=new Boolean("true");
var sem=new Boolean("false");
var sem=new Boolean("home");
```

Proprietățile obiectului `Boolean`

Proprietate	Descriere
`constructor`	Returnează funcția care a creat prototipul obiectului
`prototype`	Permite adăugarea de proprietăți şi metode unui obiect

Metodele obiectului `Boolean`

Metodă	Descriere
`toString()`	Convertește o valoare booleană în şir de caractere şi returnează rezultatul
`valueOf()`	Returnează valoarea primară a unui obiect **Boolean**

Exemplu
Ilustrează cum se verifică valoarea unui obiect **`Boolean`**.

```
<html>
<body>

<script type="text/javascript">
var b1=new Boolean( 0);
var b2=new Boolean(1);
var b3=new Boolean("");
var b4=new Boolean(null);
var b5=new Boolean(NaN);
var b6=new Boolean("false");

document.write("0 are valoarea "+ b1 +"<br />");
document.write("1 are valoarea "+ b2 +"<br />");
document.write("Un sir vid are valoarea "+ b3 + "<br
/>");
document.write("null are valoarea "+ b4+ "<br />");
document.write("NaN are valoarea "+ b5 +"<br />");
document.write("Sirul 'false' are valoarea "+ b6
+"<br />");
</script>

</body>
</html>
```

24. Obiectul `Math`

Obiectul Math permite realizarea prelucrărilor matematice. Obiectul include constante matematice şi metode. Obiectul nu are constructor. Toate proprietățile şi metodele pot fi utilizate fără a crea efectiv un obiect.

Sintaxa de utilizare:

```
var pi=Math.PI;
var x=Math.sqrt(16);
```

Constante matematice

În JavaScript se pot utiliza opt constante matematice accesibile prin obiectul Math. Ele pot fi utilizate cu următoarea sintaxă:

```
Math.E
Math.PI
Math.SQRT2
```

```
Math.SQRT1_2
Math.LN2
Math.LN10
Math.LOG2E
Math.LOG10E
```

Metode matematice

Exemplul următor ilustrează utilizarea metodei **round()** pentru a rotunji un număr la cel mai apropiat întreg:

```
document.write(Math.round(4.7));
```

Exemplul următor utilizează metoda **random()** pentru a genera un număr aleator cuprins între 0 şi 1:

```
document.write(Math.random());
```

Exemplul următor utilizează metodele **floor()** şi **random()** pentru a genera un număr aleator cuprins între 0 şi 10:

```
document.write(Math.floor(Math.random()*11));
```

Proprietățile obiectului Math

Proprietate	Descriere
E	Returnează constanta lui Euler (cca. 2.718)
LN2	Returnează logaritm natural din 2 (cca. 0.693)
LN10	Returnează logaritm natural din 10 (cca. 2.302)
LOG2E	Returnează logaritm în baza 2 din E (cca. 1.442)
LOG10E	Returnează logaritm în baza 10 din E (cca. 0.434)
PI	Returnează PI (cca. 3.14159)
SQRT1_2	Returnează rădăcina pătrată din 1/2 (cca. 0.707)
SQRT2	Returnează rădăcina pătrată din 2 (cca. 1.414)

Metodele obiectului `Math`

Metodă	Descriere
abs(x)	Valoarea absolută a lui x
acos(x)	Arccosinus de x, în radiani
asin(x)	Arcsinus de x, în radiani
atan(x)	Arctangentă de x ca valoare numerică între -PI/2 şi

	PI/2 radiani
ceil(x)	Rotunjeşte x la întreg prin adaos
cos(x)	Cosinus de x (x în radiani)
exp(x)	Valoarea lui E^x
floor(x)	Rotunjeşte x la întreg prin lipsă
log(x)	Logaritm natural din x
max(x,y,z,...,n)	Valoarea maximă din şir
min(x,y,z,...,n)	Valoarea minimă din şir
pow(x,y)	Valoarea lui x la puterea y
random()	Un număr aleator între 0 şi 1
round(x)	Rotunjeşte x la cel mai apropiat întreg
sin(x)	Sinus dex (x în radiani)
sqrt(x)	Rădăcină pătrată din x
tan(x)	Tangenta unui unghi

Exemple

Exemplul 1
Ilustrează utilizarea metodei **max()** pentru a determina maximul a două sau mai multe valori.

```
<html>
<body>

<script type="text/javascript">
document.write(Math.max(5,10) + "<br />");
document.write(Math.max(0,150,30,20,38) + "<br />");
document.write(Math.max(-5,10) + "<br />");
document.write(Math.max(-5,-10) + "<br />");
document.write(Math.max(1.5,2.5));
</script>

</body>
</html>
```

Exemplul 2
Ilustrează utilizarea metodei **min()** pentru a determina minimul a două sau mai multe valori.

```
<html>
<body>

<script type="text/javascript">

document.write(Math.min(5,10) + "<br />");
document.write(Math.min(0,150,30,20,38) + "<br />");
document.write(Math.min(-5,10) + "<br />");
document.write(Math.min(-5,-10) + "<br />");
document.write(Math.min(1.5,2.5));

</script>

</body>
</html>
```

Exemplul 3
Convertirea gradelor din Celsius în Fahrenheit şi reciproc.

```
<html>
<head>
<script type="text/javascript">
function convert(degree)
{
if (degree=="C")
 {
 F=document.getElementById("c").value * 9 / 5 + 32;
 document.getElementById("f").value=Math.round(F);
 }
else
 {
 C=(document.getElementById("f").value -32) * 5 / 9;
 document.getElementById("c").value=Math.round(C);
 }
}
</script>
</head>

<body>
<p></p><b>Introduceti un numar in unul din cele doua
campuri:</b></p>
<form>
<input id="c" name="c" onkeyup="convert('C')"> grade
Celsius<br />
egal<br />
```

```
<input id="f" name="f" onkeyup="convert('F')"> grade
Fahrenheit
</form>
</body>

</html>
```

25. Obiectul `RegExp`

Obiectul **`RegExp`** este folosit pentru a realiza căutări şi înlocuiri într-un text. **`RegExp`** este prescurtarea pentru *expresie regulată*. Când realizaţi căutări într-un text, puteţi defini modele de căutare cu ajutorul obiectului **`RegExp`**.
Un model simplu poate fi un singur caracter. Un model mai complicat conţine mai multe caractere şi poate fi utilizat pentru a analiza, verifica formatul, înlocui etc. O expresie regulată este un obiect care descrie modelul căutat în text.

Definire

Un obiect RegExp poate fi definit cu una din următoarele forme:

```
var txt=new RegExp(pattern,modifiers);
```

sau, mai simplu:

```
var txt=/pattern/modifiers;
```

- **pattern** specifică modelul căutat
- **modifiers** specifică dacă cătarea este globală, case-senzitivă etc.

Următoarea linie de cod defineşte un obiect **`RegExp`** numit **m1** cu modelul "**d**":

```
var m1=new RegExp("d");
```

Când folosiţi obiectul **m1** ca să căutaţi într-un şir, va fi găsită litera "**d**".

.

Modificatorii

Modificator	Descriere
`i`	Caută potrivirea fără a face diferenţa între literele mici şi mari
`g`	Realizează o căutare globală (găseşte toate potrivirile, nu numai prima)
`m`	Caută potrivirea pe mai multe linii

Parantezele pătrate

Sunt utilizate pentru a defini un şir de caractere.

Expression	Description
[*abc*]	Găseşte orice caracter precizat între paranteze
[^*abc*]	Găseşte orice caracter diferit de cele precizate
[0-9]	Găseşte o cifră între 0 şi 9
[a-z]	Găseşte orice literă mică
[A-Z]	Găseşte orice literă mare
[a-Z]	Găseşte orice literă, mare sau mică
[*red*|*blue*|*green*]	Găseşte oricare dintre alternativele specificate

Metacaracterele

Sunt caracterele care au o semnificație specială:

Metacaracter	Descriere
.	Găseşte un singur caracter (orice caracter diferit de linie nouă şi sfârşit de linie).
\w	Caută un caracter de cuvânt (litere mici sau mari, cifre şi underscore)
\W	Găseşte un caracter care nu este de cuvânt
\d	Găseşte o cifră
\D	Găseşte un caracter care nu este cifră
\s	Caută un spațiu alb
\S	Caută un caracter diferit de spațiu
\b	Caută o potrivire la începutul/sfârşitul unui cuvânt
\B	Caută o potrivire care nu este la începutul/sfârşitul unui cuvânt
\0	Caută un caracter NUL
\n	Caută un caracter linie nouă
\f	Caută un caracter form feed
\r	Caută un caracter retur de car
\t	Caută un caracter tab
\v	Caută un tab caracter

Cuantificatori

Cuantificator	Descriere
***n*+**	Caută orice şir care conţine cel puţin un caracter *n*
n*	Caută orice şir care conţine 0 sau mai multe apariţii ale caracterului *n*
***n*?**	Caută orice şir care conţine 0 sau o apariţie a caracterului *n*
***n*{X}**	Caută orice şir care conţine o secvenţă de *X* caractere *n*
***n*{X,Y}**	Caută orice şir care conţine o secvenţă de *X* sau *Y* caractere *n*
***n*{X,}**	Caută orice şir care conţine o secvenţă de cel puţin *X* caractere *n*
***n*$**	Caută orice şir care se încheie cu caracterul *n*
^*n*	Caută orice şir care începe cu caracterul *n*
?=*n*	Caută orice şir care este urmat de şirul *n*
?!*n*	Caută orice şir care nu este urmat de şirul *n*

Proprietăţile obiectului `RegExp`

Proprietate	Descriere
`global`	Specifică dacă modificatorul "**g**" este setat
`ignoreCase`	Specifică dacă modificatorul "**i**" este setat
`lastIndex`	Specifică indexul de la care începe căutarea următoarei potriviri
`multiline`	Specifică dacă modificatorul "**m**" este setat
`source`	Textul din modelul **RegExp**

Metodele obiectului `RegExp`

Obiectul **`RegExp`** are trei metode: **`test(), exec()`** şi **`compile().`**

Metoda `test()`

Caută într-un şir un model şi returnează **true** sau **false**.

Exemplu:

```
<html>
<body>

<script type="text/javascript">
var ml=new RegExp("e");
```

```
document.write(m1.test("Cele mai frumoase carti le
pastrez în amintire"));
</script>

</body>
</html>
```

Deoarece modelul "**e**" apare în şir, metoda va returna valoarea **true** care va fi afişată.

Metoda `exec()`

Caută în text un model şi returnează modelul, dacă acesta este găsit, sau valoarea null, dacă modelul nu apare în text.

Exemplu:

```
<html>
<body>

<script type="text/javascript">
var m1=new RegExp("e");

document.write(m1.exec("Cele mai frumoase carti le
pastrez în amintire"));
</script>

</body>
</html>
```

Deoarece modelul "**e**" apare în şir, metoda va returna valoarea **e** care va fi afişată.

Puteți adăuga al doilea parametru obiectului **`RegExp`**, pentru a specifica modul de căutare. Spre exemplu, dacă doriți să găsiți toate aparițiile unui caracter, puteți folosi parametrul "**`g`**" ("**`global`**").

.

Când utilizați parametrul "**`g`**", metoda **`exec()`** lucrează astfel:

- Găseşte prima apariție a modelului "**e**"şi îi memorează poziția
- Dacă executați din nou metoda **`exec()`**, căutarea va începe de la poziția memorată anterior ş.a.m.d.

Exemplu:

```
<html>
<body>

<script type="text/javascript">
var m1=new RegExp("e","g");
```

```
do
{
result=m1.exec("Cele mai frumoase carti le pastrez în
amintire");
document.write(result);
}
while (result!=null)
</script>

</body>
</html>
```

Deoarece modelul "**e**" apare de şase ori în text, programul de mai sus va afişa secvenţa: **eeeeeenull**

Metoda `compile()`

Este utilizată pentru a modifica conţinutul obiectului **`RegExp`**.

Metoda poate schimba modelul căutat şi poate adăuga sau elimina al doilea parametru.

Exemplu:

```
<html>
<body>

<script type="text/javascript">
var m1=new RegExp("e");
document.write(m1.test("Cele mai frumoase carti le
pastrez în amintire"));

m1.compile("d");

document.write(m1.test("Cele mai frumoase carti le
pastrez în amintire"));
</script>

</body>
</html>
```

Deoarece modelul "**e**" apare în şir, dar modelul "**d**" nu apare, programul anterior va afişa valorile: **truefalse**

Exemple

Exemplul 1

Ilustrează utilizarea metodei **match()** a obiectului **String** pentru a găsi toate caracterele precizate în model cu ajutorul parantezelor pătrate.

```
<html>
<body>

<script type="text/javascript">
var str="Ce mai faci?";
var m1=/[a-h]/g;
document.write(str.match(m1));
</script>

</body>
</html>
```

Programul va afişa rezultatul: e,a,f,a,c

Exemplul 2

Ilustrează utilizarea metodei **match()** a obiectului **String** pentru a găsi toate caracterelor diferite de cele din model.

```
<html>
<body>

<script type="text/javascript">
var str="Ce mai faci?";
var m1=/[^a-h]/g;
document.write(str.m1(patt1));
</script>

</body>
</html>
```

Programul anterior va afişa rezultatul: C, ,m,i, ,i,?

Exemplul 3

Ilustrează cum se poate construi un model care să găsească toate secvențele în care un caracter poate avea orice valoare.

```
<html>
<body>

<script type="text/javascript">

var str="Pisica nu are blana tarcata";
var m1=/a.a/g;
document.write(str.match(m1));
```

```
</script>

</body>
</html>
```

Programul va găsi toate secvențele de trei caractere în care primul şi ultimul caracter este „**a”**. Rezultatul afişat pentru şirul de mai sus este: ana,ata

Exemplul 4

Ilustrează cum se poate construi un model care să găsească toate caracterele care nu fac parte dintr-un cuvânt.

```
<html>
<body>

<script type="text/javascript">
var str="Ai obtinut 75%!";
var m1=/\W/g;
document.write(str.match(m1));
</script>

</body>
</html>
```

Programul anterior va afişa rezultatul: , ,%,!

Exemplul 5

Ilustrează cum se poate construi un model cu care să înceapă sau să se sfârşească un cuvânt.

```
<html>
<body>

<script type="text/javascript">
var str="Vizitati Google";
var m1=/\bGo/g;
document.write(str.match(m1));
</script>

</body>
</html>
```

Dacă nu este găsit nici-un cuvânt care începe sau se sfârşeşte cu modelul dat, metoda **match()** va returna valoarea **null**. Pentru exemplul considerat anterior, există în text un cuvânt care se potriveşte şi metoda returnează modelul. Rezultatul afişat este: Go

Exemplul 6
Ilustrează cum se pot găsi toate secvențele dintr-un text, în care un anumit caracter apare cel puțin o dată.

```
<html>
<body>

<script type="text/javascript">
var str="Tu creezi pagini web interesante!";
var m1=/e+/g;
do
{
result=m1.exec(str);
document.write(result); document.write("  ");
}
while(result!=null)
</script>

</body>
</html>
```

Programul anterior va determina toate secvențele din text în care caracterul „e" apare cel puțin o dată (în poziții consecutive). Rezultatul afişat de program este: ee e e e e null

Exemplul 7
Ilustrează cum se pot găsi o secvențele de text în care un anumit caracter apare de minim *x* ori.

```
<html>
<body>

<script type="text/javascript">
var str="Aveti 10, 100, 1000 sau 10000 de lei?";
var m1=/\d{3,}/g;
document.write(str.match(m1));
</script>

</body>
</html>
```

Programul anterior afişează rezultatul: 100,1000,10000, adică secvențele care conțin cel puțin trei cifre zecimale.

Exemplul 8
Ilustrează cum se pot găsi găsi toate subşirurile dintr-un text, care sunt urmate de un subşir dat.

```
<html>
<body>

<script type="text/javascript">
var str="eu am o pisica, dar eu am si un papagal";
var m1=/eu(?= am)/g;
document.write(str.match(m1));
</script>

</body>
</html>
```

Programul anterior determină toate şirurile „**eu**” care sunt urmate de şirul „**am**”. Rezultatul afişat este: eu,eu

26. Obiectul `Number`

Obiectul **`Number`** este un container pentru valorile numerice de bază. Obiectele **`Number`** sunt create cu următoarea sintaxă:

```
var nume = new Number(valoare);
```

Obs: Dacă parametrul **valoare** nu poate fi convertit într-un număr, va fi returnată valoarea **NaN** (Not-a-Number).

Proprietăţile obiectului `Number`

Proprietate	Descriere
`constructor`	Returnează funcţia care a creat prototipul obiectului
`MAX_VALUE`	Returnează cel mai mare număr posibil în JavaScript
`MIN_VALUE`	Returnează cel mai mic număr posibil în JavaScript
`NEGATIVE_INFINITY`	Reprezintă infinitul negativ (returnat la depăşire)
`POSITIVE_INFINITY`	Reprezintă infinitul pozitiv (returnat la depăşire)
`prototype`	Permite adăugarea de proprietăţi şi metode

Metodele obiectului `Number`

Metodă	Descriere
`toExponential(x)`	Converteşte numărul într-o notaţie exponenţială
`toFixed(x)`	Formează un număr cu **x** cifre după virgulă
`toPrecision(x)`	Formează un număr cu lungimea **x**

`toString()`	Converteşte un obiect **Number** în şir de caractere. Dacă metoda are parametru, acesta precizează baza în care este reprezentat numărul convertit în şir.
`valueOf()`	Returnează valoarea primară a obiectului

Exemple

Exemplul 1
Ilustrează cum se afişează cel mai mare număr din JavaScript.

```
<html>
<body>

<script type="text/javascript">
document.write(Number.MAX_VALUE);
</script>

</body>
</html>
```

Exemplul 2
Ilustrează cum se stabileşte numărul de zecimale afişate.

```
<html>
<body>

<script type="text/javascript">

var num = new Number(27.2547);
document.write(num.toFixed()+"<br />");
document.write(num.toFixed(1)+"<br />");
document.write(num.toFixed(3)+"<br />");
document.write(num.toFixed(10));

</script>

</body>
</html>
```

Exemplul 3
Ilustrează cum se stabileşte precizia numărului afişat (numărul total de cifre afişate).

```
<html>
```

```
<body>

<script type="text/javascript">

var num = new Number(31.1593);
document.write(num.toPrecision()+"<br />");
document.write(num.toPrecision(2)+"<br />");
document.write(num.toPrecision(3)+"<br />");
document.write(num.toPrecision(10));

</script>

</body>
</html>
```

Exemplul 4
Ilustrează cum se converteşte în şir un număr, folosind diferite baze de numerație.

```
<html>
<body>

<script type="text/javascript">

var num=new Number(31);
document.write(num.toString()+"<br />");
//numarul este reprezentat in baza 10 (implicit)
document.write(num.toString(2)+"<br />");
//numarul este reprezentat in baza 2
document.write(num.toString(8)+"<br />");
//numarul este reprezentat in baza 8
document.write(num.toString(16)+"<br />");
//numarul este reprezentat in baza 16

</script>

</body>
</html>
```

27. Obiectul `Navigator`

Obiectul **`Navigator`** conține informații despre browserul vizitatorului.

Aproape orice exemplu din acest curs funcţionează în browserele care recunosc JavaScript. Totuşi, unele exemple nu funcţionează în anumite browsere, în special în cele vechi. De aceea, uneori este foarte util să determinaţi browserul utilizat de vizitator pentru a-i putea furniza informaţiile potrivite. Cea mai bună metodă este să vă proiectaţi paginile web să se comporte diferit, în funcţie de browserul vizitatorului. Obiectul Navigator poate fi utilizat în acest scop, deoarece conţine informaţii despre numele şi versiunea browserului vizitatorului.

Navigator Object Properties

Proprietate	Descriere
`appCodeName`	Returnează codul browserului
`appName`	Returnează numele browserului
`appVersion`	Returnează informaţii despre versiunea browserului
`cookieEnabled`	Determină dacă are cookies activate
`platform`	Returnează pentru ce platformă este compilat browserul

Exemple

Exemplul 1
Variabila "**browser**" din exemplul următor memorează numele browserului. Proprietatea **`appVersion`** returnează un şir de caractere care conţine mult mai multe informaţii, nu numai numărul versiunii. Deoarece ne interesează numai versiunea, pentru a o extrage din şir, este utilizată o funcţie numită **`parseFloat()`**, care extrage din şir şi returnează prima secvenţă care arată ca un număr zecimal.

```
<html>
<body>

<script type="text/javascript">
var browser=navigator.appName;
var b_ver=navigator.appVersion;
var versiune=parseFloat(b_ver);

document.write("Numele browserului: "+ browser);
document.write("<br />");
document.write("Versiunea browserului: "+ versiune);
</script>
</body>
</html>
```

Exemplul 2
Ilustrează cum pot fi afişate diferite mesaje, în funcție de tipul şi versiunea browserului.

```
<html>
<head>
<script type="text/javascript">
function detectBrowser()
{
var browser=navigator.appName;
var b_ver=navigator.appVersion;
var versiune=parseFloat(b_ver);
if ((browser=="Netscape"||browser=="Microsoft
Internet Explorer")
  && (versiune>=4))
{
  alert("Browserul dvs. este destul de bun!");
  }
else
  {
  alert("Este timpul sa va upgradati browserul!");
  }
}
</script>
</head>

<body onload="detectBrowser()">
</body>
</html>
```

Exemplul 3
Ilustrează cum pot fi afişate mai multe detalii despre browserul vizitatorului.

```
<html>
<body>
<script type="text/javascript">
document.write("<p>Browser: ");
document.write(navigator.appName + "</p>");

document.write("<p>Versiune: ");
document.write(navigator.appVersion + "</p>");

document.write("<p>Cod: ");
document.write(navigator.appCodeName + "</p>");
```

```
document.write("<p>Platforma: ");
document.write(navigator.platform + "</p>");

document.write("<p>Cookies activate: ");
document.write(navigator.cookieEnabled + "</p>");

</script>
</body>
</html>
```

28. Cookies

Un **cookie** este o variabilă păstrată în calculatorul vizitatorului. De fiecare dată când calculatorul respectiv cere browserului o pagină, el va trimite şi cookie-ul respectiv. Cu JavaScript, puteţi crea şi extrage cookies.
Exemple de cookies:

- Numele utilizatorului – Prima dată când utilizatorul vă vizitează pagina trebuie să-şi completeze numele. Numele este stocat într-un cookie. Următoarea dată când vizitatorul ajunge la pagina dvs., puteţi să-l întâmpinaţi cu un mesaj de genul "Bine ai venit.....!" Numele este recuperat dintr-un cookie.
- Parolă – Prima dată utilizatorul vă vizitează pagina trebuie să completeze o parolă. Parola este memorată într-un cookie. Data viitoare când vizitatorul ajunge în pagină, parola poate fi recuperată dintr-un cookie.
- Data calendaristică – Prima dată când utilizatorul vă vizitează pagina, data curentă este memorată într-un cookie. Data viitoare când utilizatorul vă vizitează pagina, puteţi să afişaţi un mesaj de genul "Ultima dvs. vizita a fost in data de ....." Această dată este recuperată dintr-un cookie.

Crearea şi memorarea unui cookie

În acest exemplu vom crea un cookie care memorează numele vizitatorului, apoi vom folosi numele memorat în variabila cookie pentru a afişa un mesaj de bun venit. Prima dată vom crea o funcţie care memorează numele vizitatorului într-o variabilă cookie:

```
function setCookie(c_name,value,expiredays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+ "=" +escape(value)+
```

```
((expiredays==null) ? "" :
";expires="+exdate.toGMTString());
}
```

Parametrii funcției reprezintă numele şi valoarea cookie-ului şi numărul de zile până când acesta expiră. Funcția converteşte numărul de zile într-o dată validă şi apoi adaugă numărul de zile după care va expira cookie-ul va expira. Apoi, numele şi valoarea cookie-ului şi data expirării sunt memorate într-un obiect **document.cookie**.

În continuare, vom crea o funcție care verifică dacă cookie-ul a fost setat:

```
function getCookie(c_name)
{
if (document.cookie.length>0)
  {
  c_start=document.cookie.indexOf(c_name + "=");
  if (c_start!=-1)
    {
    c_start=c_start + c_name.length+1;
    c_end=document.cookie.indexOf(";",c_start);
    if (c_end==-1) c_end=document.cookie.length;
         return unescape(document.cookie.substring
                                    (c_start,c_end));
    }
  }
return "";
}
```

Funcția verifică mai întâi dacă în obiectul document.cookie este memorat vre-un cookie. În caz afirmativ, verificăm dacă este memorat cookie-ul dorit. Dacă cookie-ul este găsit, îi returnăm valoarea, în caz contrar returnăm un şir vid.
În cele din urmă, vom crea funcția care afişează un mesaj de bun venit dacă cookie-ul este setat şi o casetă prompt care cere numele vizitatorului, în caz contrar:

```
function checkCookie()
{
username=getCookie('username');
if (username!=null && username!="")
  {
  alert('Welcome again '+username+'!');
  }
else
  {
  username=prompt('Please enter your name:',"");
  if (username!=null && username!="")
```

```
{
    setCookie('username',username,365);
    }
}
}
```

Programul complet este:

```
<html>
<head>
<script type="text/javascript">
function getCookie(c_name)
{
if (document.cookie.length>0)
  {
  c_start=document.cookie.indexOf(c_name + "=");
  if (c_start!=-1)
    {
    c_start=c_start + c_name.length+1 ;
    c_end=document.cookie.indexOf(";",c_start);
    if (c_end==-1) c_end=document.cookie.length
    return
unescape(document.cookie.substring(c_start,c_end));
    }
  }
return ""
}

function setCookie(c_name,value,expiredays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+                                    "="
+escape(value)+((expiredays==null)    ?    ""    :    ";
expires="+exdate.toGMTString());
}

function checkCookie()
{
username=getCookie('username');
if (username!=null && username!="")
  {
  alert('Bine ai revenit '+username+'!');
  }
else
  {
```

```
  username=prompt('Va rog sa va introduceti
 numele:',"");
  if (username!=null && username!="")
    {
    setCookie('username',username,365);
    }
  }
}
</script>
</head>
<body onLoad="checkCookie()">
</body>
</html>
```

Exemplul prezentat execută funcția **checkCookie()** când pagina se încarcă.

29. Validarea formularelor

JavaScript poate fi utilizat pentru a valida formularele HTML înainte de a fi trimise către server.
Datele verificate uzual cu JavaScript pot fi:

- au rămas câmpuri obligatorii necompletate?
- adresa de e-mail este validă?
- data calendaristică este validă?
- s-a introdus text într-un câmp numeric?

Câmpuri obligatorii

Funcția următoare verifică dacă un câmp obligatoriu a rămas necompletat. În caz afirmativ, o casetă de alertare afișează un mesaj şi funcția returnează valoarea **false**. Dacă câmpul a fost completat, funcția returnează valoarea **true** şi data este considerată validă:

```
function valideaza_obligatoriu(camp,txt)
{
with (camp)
  {
  if (value==null||value=="")
    {
    alert(txt);return false;
    }
else
    {
```

```
return true;
    }
  }
}
```

Scriptul complet, cu formularul HTML, ar putea arăta în felul următor:

```
<html>
<head>
<script type="text/javascript">
function valideaza_obligatoriu(camp,txt)
{
with (camp)
  {
  if (value==null||value=="")
    {
    alert(txt);return false;
    }
  else
    {
    return true;
    }
  }
}

function valideaza_formular(formular)
{
with (formular)
  {
  if (valideaza_obligatoriu(email,"Campul Email este
obligatoriu!")==false)
  {email.focus();return false;}
  }
}
</script>
</head>

<body>
<form action="submit.htm" onsubmit="return
valideaza_formular(this)" method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Trimite">
</form>
</body>
```

```
</html>
```

Validarea adresei de e-mail

Funcția următoare verifică dacă câmpul respectă sintaxa generală a unei adrese de e-mail. Asta înseamnă că date respectivă trebuie să conțină cel puțin caracterul @ şi un punct. De asemenea, @ nu poate fi primul caracter din şir, iar ultimul punct trebuie să fie cel puțin la un caracter distanță de @:

```
function valideaza_email(camp,txt)
{
with (camp)
  {
  apos=value.indexOf("@");
  dotpos=value.lastIndexOf(".");
  if (apos<1||dotpos-apos<2)
    {alert(txt);return false;}
  else {return true;}
  }
}
```

Scriptul complet, cu formularul HTML, ar putea arăta în felul următor:

```
<html>
<head>
<script type="text/javascript">
function valideaza_email(camp,txt)
{
with (camp)
  {
  apos=value.indexOf("@");
  dotpos=value.lastIndexOf(".");
  if (apos<1||dotpos-apos<2)
    {alert(txt);return false;}
  else {return true;}
  }
}

function valideaza_formular(formular)
{
with (formular)
  {
  if (valideaza_email(email,"Adresa e-mail nu este
valida!")==false)
    {email.focus();return false;}
```

```
}
}
</script>
</head>

<body>
<form action="submit.htm" onsubmit="return
valideaza_formular(this);" method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Trimite">
</form>
</body>

</html>
```

30. Animație

Puteți folosi JavaScript pentru a crea imagini animate. Secretul este să lăsați scriptul să afişeze imagini diferite pentru evenimente diferite.
În exemplul următor vom adăuga o imagine care se va comporta ca un link în pagina web. Vom adăuga apoi un eveniment **onMouseOver** şi un eveniment **onMouseOut** care vor apela două funcții JavaScript ce vor comuta între două imagini.
Codul HTML arată astfel:

```
<a href="http://www.google.com" target="_blank">
<img border="0" alt="Vizitati Google!" src="img2.gif"
id="m1"
onmouseOver="mouseOver()" onmouseOut="mouseOut()"
/></a>
```

Observați că imaginea a primit un id, pentru ca JavaScript să poată referi imaginea în diferite puncte din script. Evenimentul **onMouseOver** va spune browserului că, în momentul în care mouse-ul trece peste imagine, trebuie apelată o funcție care să schimbe imaginea. Evenimentul **onMouseOut** va spune browserului că, atunci când mouse-ul se îndepărtează de imagine, trebuie apelată o funcție care va afişa din nou imaginea inițială.
Codul celor două funcții este:

```
<script type="text/javascript">
function mouseOver()
{
document.getElementById("m1").src ="img1.gif";
```

```
}
function mouseOut()
{
document.getElementById("m1").src ="img2.gif";
}
</script>
```

Funcția **mouseOver()** va determina afişarea imaginii "**img1.gif**", iar funcția **mouseOut()** va determina afişarea imaginii "**img2.gif**". Efectul de animație este mai vizibil dacă cele două imagini sunt foarte asemănătoare, diferind spre exemplu prin culoare.

Codul întregului program este:

```
<html>
<head>
<script type="text/javascript">
function mouseOver()
{
document.getElementById("m1").src ="img1.gif";
}
function mouseOut()
{
document.getElementById("m1").src ="img2.gif";
}
</script>
</head>

<body>
<a href="http://www.google.com" target="_blank">
<img border="0" alt="Vizitati Google!" src="img2.gif"
id="b1"
onmouseover="mouseOver()" onmouseout="mouseOut()"
/></a>
</body>
</html>
```

31. Imagini mapate

O imagine mapată (**image-map**) este o imagine care are zone ce pot fi acționate cu mouse-ul. În mod normal, fiecare zonă are un hiperlink asociat.

În tagurile **<area>** din imaginea mapată pot fi adăugate evenimente care apelează funcții JavaScript. Tagul **<area>** suportă evenimentele **onClick**, **onDblClick**, **onMouseDown**, **onMouseUp**, **onMouseOver**,

onMouseMove, **onMouseOut**, **onKeyPress**, **onKeyDown**, **onKeyUp**, **onFocus** şi **onBlur**.

Exemplul următor ilustrează utilizarea unei imagini mapate într-un program HTML:

```
<html>
<head>
<script type="text/javascript">
function writeText(txt)
{
document.getElementById("desc").innerHTML=txt;
}
</script>
</head>

<body>
<img src="planets.gif" width="145" height="126"
alt="Planets" usemap="#planetmap" />

<map name="planetmap">
<area shape ="rect" coords ="0,0,82,126"
onMouseOver="writeText('Soarele si planetele gigante
gazoase, ca Jupiter, sunt cu certitudine cele mai
mari corpuri din sistemul nostru solar.')"
href ="sun.htm" target ="_blank" alt="Sun" />

<area shape ="circle" coords ="90,58,3"
onMouseOver="writeText('Planeta Mercur este foarte
greu de studiat de pe Pamant datorita apropierii ei
de Soare.')"
href ="mercur.htm" target ="_blank" alt="Mercur" />

<area shape ="circle" coords ="124,58,8"
onMouseOver="writeText('Pana in anii '60, Venus a
fost adesea considerata sora geamana a Pamantului
pentru ca este cea mai apropiata de noi, si cele doua
planete au multe caracteristici comune.')"
href ="venus.htm" target ="_blank" alt="Venus" />
</map>

<p id="desc"></p>
</body>
</html>
```

32. Programarea evenimentelor

Codurile JavaScript pot fi execute la intervale de timp programate. Programarea evenimentelor JavaScript se realizează uşor cu ajutorul următoarelor două metode:

- **`setTimeout()`** – execută un cod cândva în viitor
- **`clearTimeout()`** – anulează programările realizate cu **`setTimeout()`**

Obs: Ambele metode aparţin obiectului Window din HTML DOM.

Metoda `setTimeout()`

Sintaxă:

```
var t=setTimeout("declaraţie javascript",
                                    milliseconds);
```

Metoda **`setTimeout()`** returnează o valoare care este memorată în variabila **t** declarată mai sus. Dacă doriţi să anulaţi programarea, o puteţi face folosind variabila asociată. Primul argument al metodei este un şir de caractere care conţine o declaraţie JavaScript care poate fi, de exemplu, un apel de funcţie sau instrucţiunea de afişare a unei casete de mesaj. Al doilea parametru precizează numărul de milisecunde (începând de acum) după care va fi executat primul parametru.

Obs: O secundă are 1000 de milisecunde.

Exemplul 1

În exemplul următor, când butonul este apăsat, o casetă de alertare va fi afişată după 7 secunde.

```
<html>
<head>
<script type="text/javascript">
function mesaj()
{
var t=setTimeout("alert('Caseta afisata dupa 7
secunde!')",7000);
}
</script>
</head>

<body>
<form>
```

```
<input type="button" value="Afiseaza mesajul!"
onClick="mesaj()" />
</form>
</body>
</html>
```

Exemplul 2

Pentru o repeta la infinit o secvență de cod, trebuie să scriem o funcție care se autoapelează. În exemplul următor, când butonul este apăsat, un câmp de intrare dintr-un formular va începe să numere, plecând de la zero, secundele scurse, fără să se oprească. A fost inclusă şi o funcție care verifică dacă numărătorul funcționează deja, pentru a nu crea un alt numărător dacă butonul este apăsat de mai multe ori.

```
<html>
<head>
<script type="text/javascript">
var c=0;
var t;
var pornit=0;

function numara()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("numara()",1000);
}

function verifica()
{
if (!pornit)
  {
  pornit=1;
  numara();
  }
}
</script>
</head>

<body>
<form>
<input type="button" value="Incepe numararea!"
onClick="verifica()">
<input type="text" id="txt" />
```

```
</form>
</body>
</html>
```

Metoda `clearTimeout()`

Sintaxă:

```
clearTimeout(variabila_setTimeout)
```

În exemplul următor utilizăm numărătorul infinit din exemplul următor şi adăugăm o funcție care va opri numărătorul la apăsarea unui buton:

```
<html>
<head>
<script type="text/javascript">
var c=0;
var t;
var pornit=0;

function numara()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("numara()",1000);
}

function verifica()
{
if (!pornit)
  {
  pornit=1;
  numara();
  }
}

function stop()
{
clearTimeout(t);
pornit=0;
}
</script>
</head>
<body>
<form>
<input type="button" value="Incepe numararea!"
```

```
onClick="verifica()">
<input type="text" id="txt">
<input type="button" value="Opreste numararea!"
onClick="stop()">
</form>
</body>
</html>
```

Exemplu

Acest exemplu ilustrează crearea unui ceas cu ajutorul evenimentelor programate.

```
<html>
<head>
<script type="text/javascript">
function numara()
{
var azi=new Date();
var h=azi.getHours();
var m=azi.getMinutes();
var s=azi.getSeconds();
// adauga un zero in fata numerelor <10
m=verifica(m);
s=verifica(s);
document.getElementById('txt').innerHTML=h+":"+m+":"+
s;
t=setTimeout('numara()',500);
}

function verifica(i)
{
if (i<10)
  {
  i="0" + i;
  }
return i;
}
</script>
</head>

<body onload="numara()">
<div id="txt"></div>
</body>
</html>
```

33. Crearea obiectelor proprii

Obiectele sunt utile pentru a organiza informaţia. În completarea obiectelor predefinite în JavaScript, cum ar fi String, Date, Array, etc., vă puteţi crea propriile obiecte.
Un obiect este doar un tip particular de date, cu o colecţie de proprietăţi şi metode. Spre exemplu, o persoană este un obiect înzestrat cu proprietăţi cum ar fi: nume, vârstă, înălţime, greutate, culoarea ochilor etc. Proprietăţile au anumite valori, care diferă de la o persoană la alta. Metodele sunt acţiuni care pot fi realizate cu un obiect. Pentru obiectul persoana, acestea ar putea fi: mananca(), munceste(), doarme, scrie(), citeste() etc.

Proprietăţile
Pentru a accesa o proprietate a unui obiect sintaxa este:

```
nume_obiect.nume_proprietate
```

Puteţi adăuga proprietăţi unui obiect prin simpla atribuire de valori. Dacă presupunem că obiectul persoana există deja, putem să-i adăugăm proprietăţi prin atribuiri, ca în exemplul următor:

```
persoana.nume="Popescu";
persoana.prenume="Andrei";
persoana.varsta=25;
persoana.ochi="verzi";

document.write(persoana.nume);
```

Codul de mai sus va afişa rezultatul: **Popescu**

Metodele
Pentru a accesa o metodă a unui obiect sintaxa este:

```
Nume_obiect.nume_metoda()
```

Obs: Dacă metoda are parametri, ei vor fi scrişi între paranteze.

Un obiect poate fi creat în trei moduri:

1. Crearea directă a unui obiect
Secvenţa următoare de cod creează un obiect şi îi adaugă proprietăţi:

```
persoana=new Object();
persoana.nume="Popescu";
persoana.prenume="Andrei";
```

```
persoana.varsta=25;
persoana.ochi="verzi";
```

Adăugare unei metode se face simplu, ca în exemplul următor:

```
persoana.scrie=scrie;
```

2 Crearea şablonului unui obiect

Şablonul defineşte structura unui obiect:

```
function persoana(nume,prenume,varsta,ochi)
{
this.nume=nume;
this.prenume=prenume;
this.varsta=varsta;
this.ochi=ochi;
}
```

Observaţi că şablonul este doar o funcţie. În interiorul funcţiei trebuie să faceţi atribuiri pentru **this.nume_proprietate**. Construcţia "this" se referă la instanţa curentă a obiectului.

După ce aţi construit şablonul obiectului, puteţi crea noi instanţe după modelul următor:

```
tata=new persoana("Marcu","Ion",40,"verzi");
mama=new persoana("Marcu","Maria",38,"negri");
```

Adăugarea de metode obiectului persoana se realizează tot în interiorul şablonului:

```
function persoana(nume,prenume,varsta,ochi)
{
this.nume=nume;
this.prenume=prenume;
this.varsta=varsta;
this.ochi=ochi;

this.numenou=numenou;
}
```

Observaţi că metodele sunt funcţii ataşate obiectului. Acum va trebui scrisă funcţia numenou():

```
function numenou(str)
{
this.nume=str;
}
```

Puteţi folosi metoda astfel: **mama.numenou(”Georgescu”);**

34. Proprietăți şi metode globale

Aceste proprietăți şi metode pot fi folosite pentru orice variabile, din acest motiv se numesc globale.

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

Exemplul 1
Ilustrează utilizarea proprietății **NaN**:

```
<html>
<body>

<script type="text/javascript">

var luna=13;

if (luna < 1 || luna > 12)
{
luna = luna.NaN;
}

document.write(luna);

</script>

</body>
</html>
```

Exemplul 2
Ilustrează utilizarea proprietății **undefined**:

```
<html>
<body>

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

```
var t1="";
var t2;

if (t1==undefined)
{
document.write("Variabila t1 nu este definita");
}
if (t2==undefined)
{
document.write("Variabila t2 nu este definita");
}

</script>

</body>
</html>
```

Funcție	Descriere
`escape()`	Codează caracterele speciale dintr-un şir de caractere astfel încât şirul devine portabil în rețea către orice calculator care suportă codurile ASCII
`eval()`	Evaluează un şir de caractere şi, dacă şirul conține o secvență de cod JavaScript, execută secvența
`isFinite()`	Determină dacă valoarea este un număr valid, finit
`isNaN()`	Determină dacă valoarea este un număr invalid
`Number()`	Converteşte valoarea unui obiect în număr
`parseFloat()`	Converteşte un şir într-un număr zecimal
`parseInt()`	Converteşte un şir într-un număr întreg
`String()`	Converteşte valoarea unui obiect într-un şir
`unescape()`	Decodează un şir codat

Exemplul 1
Ilustrează utilizarea metodei **`eval()`**:

```
<html>
<body>

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

```
eval("x=10;y=20;document.write(x*y)");
document.write("<br />" + eval("2+2"));
document.write("<br />" + eval(x+17));

</script>

</body>
</html>
```

Exemplul 2
Ilustrează utilizarea metodei **`Number()`**:

```
<html>
<body>

<script type="text/javascript">
var t1= new Boolean(true);
var t2= new Boolean(false);
var t3= new Date();
var t4= new String("921");
var t5= new String("193 469");

document.write(Number(t1)+ "<br />");
document.write(Number(t2)+ "<br />");
document.write(Number(t3)+ "<br />");
document.write(Number(t4)+ "<br />");
document.write(Number(t5)+ "<br />");
</script>

</body>
</html>
```

Exemplul 3
Ilustrează utilizarea metodei **`parseFloat()`** pentru a extrage valoarea dintr-un şir ca număr zecimal:

```
<html>
<body>

<script type="text/javascript">
document.write(parseFloat("10") + "<br />");
document.write(parseFloat("10.00") + "<br />");
document.write(parseFloat("10.33") + "<br />");
document.write(parseFloat("34 45 66") + "<br />");
document.write(parseFloat("   60   ") + "<br />");
```

```
document.write(parseFloat("40 de ani") + "<br />");
document.write(parseFloat("Ea are 40 de ani") + "<br
/>");
</script>

</body>
</html>
```

Obs. Metoda verifică dacă primul caracter din şir poate apare în scrierea unei valori zecimale şi, în caz afirmativ continuă construirea acestui număr până la întâlnirea primului caracter care nu poate apare în scrierea unui număr. Programul anterior va afişa valorile:

10
10
10.33
34
60
40
NaN

Exemplul 4

Ilustrează utilizarea metodei **`parseInt()`** pentru a extrage valoarea dintr-un şir ca număr întreg:

```
<html>
<body>

<script type="text/javascript">
document.write(parseInt("10") + "<br />") ;
document.write(parseInt("10.33") + "<br />");
document.write(parseInt("34 45 66") + "<br />");
document.write(parseInt(" 60 ") + "<br />");
document.write(parseInt("40 de ani") + "<br />");
document.write(parseInt("Ea are 40 de ani") + "<br
/>");
document.write("<br />");
document.write(parseInt("10",10)+ "<br />");
document.write(parseInt("010")+ "<br />");
document.write(parseInt("10",8)+ "<br />");
document.write(parseInt("0x10")+ "<br />");
document.write(parseInt("10",16)+ "<br />");
</script>

</body>
</html>
```

Obs. Dacă numărul începe cu 0 va fi interpretat ca fiind scris în baza 8, iar dacă începe cu 0x ca fiind scris în baza 16. Baza poate fi specificată şi prin adăugarea celui de-al doilea parametru în metodă. Conversia se încheie la întâlnirea primului caracter din şir care nu poate apare în scrierea unui număr întreg. Programul anterior afişează valorile:

```
10
10
34
60
40
NaN

10
8
8
16
16
```

35. Obiectele browserului

Obiectul `window`

Acest obiect reprezintă o fereastră deschisă în browser. Dacă conține cadre (tagurile **`<frame>`** sau **`<iframe>`**), browserul creează un obiect window pentru documentul HTML, şi câte un obiect window pentru fiecare cadru.

Proprietățile obiectului `window`

Proprietate	Descriere
`closed`	Returnează o valoare booleană care indică dacă fereastra a fost închisă sau nu
`defaultStatus`	Setează sau returnează textul implicit din bara de stare a ferestrei
`document`	Returnează obiectul **Document** al ferestrei
`frames`	Returnează un tablou cu toate cadrele din fereastra curentă
`history`	Returnează obiectul **History** al ferestrei
`innerHeight`	Setează sau returnează înălțimea interioară a zonei de conținut a ferestrei
`innerWidth`	Setează sau returnează lățimea interioară a zonei de conținut a ferestrei

length	Returnează numărul de cadre (inclusiv cele inline) din fereastră
location	Returnează obiectul **Location** al ferestrei
name	Setează sau returnează numele ferestrei
opener	Returnează referința care a creat fereastra
outerHeight	Setează sau returnează înălțimea exterioară a ferestrei (inclusiv toolbars/scrollbars)
outerWidth	Setează sau returnează lățimea exterioară a ferestrei (inclusiv toolbars/scrollbars)
pageXOffset	Returnează numărul de pixeli cu care documentul curent a fost derulat orizontal, în raport cu colțul stânga sus al ferestrei
pageYOffset	Returnează numărul de pixeli cu care documentul curent a fost derulat vertical, în raport cu colțul stânga sus al ferestrei
parent	Returnează fereastra părinte a ferestrei curente
screenLeft	Returnează coordonata **x** a ferestrei, relativ la ecran
screenTop	Returnează coordonata **y** a ferestrei, relativ la ecran
screenX	Returnează coordonata **x** a ferestrei, relativ la ecran
screenY	Returnează coordonata **y** a ferestrei, relativ la ecran
self	Returnează fereastra curentă
status	Setează textul din bara de stare a ferestrei
top	Returnează cea mai din vârf fereastră deschisă în browser

Metodele obiectului `window`

Metodă	Descriere
alert()	Afişează o casetă de alertare care conține un mesaj şi un buton OK
blur()	Îndepărtează focus-ul de la fereastra curentă
clearInterval()	Resetează timer-ul setat cu **setInterval()**
clearTimeout()	Resetează timer-ul setat cu **setTimeout()**
close()	Închide fereastra curentă
confirm()	Afişează o casetă de dialog care conține un mesaj şi butoanele OK şi Cancel

createPopup()	Creează o fereastră pop-up
focus()	Fixează focus-ul pe fereastra curentă
moveBy()	Mută fereastra, relativ la poziția ei curentă
moveTo()	Mută fereastra într-o nouă poziție
open()	Deschide o nouă fereastră în browser
print()	Tipăreşte conținutul ferestrei curente
prompt()	Afişează o casetă de dialog care cere utilizatorului să introducă anumite informații
resizeBy()	Redimensionează fereastra la dimensiunea specificată în pixeli
resizeTo()	Redimensionează fereastra la înălțimea şi lățimea specificate
scrollBy()	Derulează conținutul ferestrei cu numărul specificat de pixeli
scrollTo()	Derulează conținutul ferestrei până la coordonatele specificate
setInterval()	Apelează o funcție sau evaluează o expresie la intervale specificate de timp (în milisecunde)
setTimeout()	Apelează o funcție sau evaluează o expresie dup un număr specificat de milisecunde

Exemplul 1
Ilustrează utilizarea metodelor **open()** şi **focus()**.

```
<html>
<head>
<script type="text/javascript">
function deschide()
{
myWindow=window.open('','','width=200,height=100');
myWindow.document.write("<p>Aceasta este o fereastra
creata cu metoda open()</p>");
myWindow.focus();
}
</script>
</head>
<body>

<input type="button" value="Deschide fereastra"
onclick="deschide()" />
```

```
</body>
</html>
```

Exemplul 2
În acest exemplu, funcția **clock()** este apelată la fiecare 1000 milisecunde şi actualizează ceasul afişat. Ceasul poate fi oprit prin apăsarea unui buton.

```
<html>
<body>

<input type="text" id="clock" />
<script language=javascript>
var int=self.setInterval("clock()",1000);
function clock()
  {
  var d=new Date();
  var t=d.toLocaleTimeString();
  document.getElementById("clock").value=t;
  }
</script>
</form>
<button
onclick="int=window.clearInterval(int)">Stop</button>

</body>
</html>
```

Exemplul 3
Ilustrează mutarea ferestrei curente cu 250 pixeli relativ la poziția ei curentă.

```
<html>
<head>
<script type="text/javascript">
function deschide()
{
myWindow=window.open('','','width=200,height=100');
myWindow.document.write("<p>Aceasta este o fereastra
deschisa cu open()</p>");
}

function muta()
{
myWindow.moveBy(250,250);
```

```
myWindow.focus();
}
</script>
</head>
<body>

<input type="button" value="Deschide fereastra"
onclick="deschide()" />
<br /><br />
<input type="button" value="Muta fereastra"
onclick="muta()" />

</body>
</html>
```

Exemplul 4
Ilustrează redimensionează redimensionarea ferestrei curente.

```
<html>
<head>
<script type="text/javascript">
function redimensioneaza()
{
top.resizeTo(500,300);
}
</script>
</head>

<body>
<form>
<input type="button" onclick="redimensioneaza()"
value="Redimensioneaza fereastra" />
</form>
</body>

</html>
```

Exemplul 5
Ilustrează utilizarea metodei **blur()** pentru a trimite o fereastră în background.

```
<html>
<head>
<script type="text/javascript">
function deschide()
```

```
{
myWindow=window.open('','','width=200,height=100');
myWindow.document.write("<p>Aceasta fereastra este
deschisa cu open()</p>");
myWindow.blur();
}
</script>
</head>
<body>

<input type="button" value="Deschide fereastra"
onclick="deschide()" />

</body>
</html>
```

Obiectul `screen`

Acest obiect conține informații despre ecranul vizitatorului.

Proprietățile obiectului `screen`

Proprietate	Descriere
`availHeight`	Returnează înălțimea ecranului (fără Taskbar)
`availWidth`	Returnează lățimea ecranului (fără Taskbar)
`colorDepth`	Returnează numărul de biți din paleta de culori folosită pentru afișarea imaginilor
`height`	Returnează înălțimea totală a ecranului
`pixelDepth`	Returnează rezoluția culorii ecranului (în biți/pixel)
`width`	Returnează lățimea totală a ecranului

Exemplu următor ilustrează utilizarea tuturor proprietăților obiectului **screen** pentru a obține informații despre ecranul vizitatorului:

```
<html>
<body>

<h3>Ecranul dumneavoastra are proprietatile:</h3>

<script type="text/javascript">
document.write("Latime/Inaltime totala: ");
document.write(screen.width + "*" + screen.height);
document.write("<br />");
```

```
document.write("Latime/Inaltime disponibila: ");
document.write(screen.availWidth + "*" +
screen.availHeight);
document.write("<br />");
document.write("Numarul de biti ai culorii: ");
document.write(screen.colorDepth);
document.write("<br />");
document.write("Rezoluția culorii: ");
document.write(screen.pixelDepth);
</script>

</body>
</html>
```

Obiectul `history`

Acest obiect conține URL-urile vizitate de utilizator (într-o fereastră de browser). Obiectul **`history`** face parte din obiectul window şi este accesat prin proprietatea **`window.history`**.

Proprietățile obiectului `history`

Proprietate	Descriere
`length`	Returnează numărul de URL-uri din lista **history**

Metodele obiectului `history`

Metodă	Descriere
`back()`	Încarcă URL-ul anterior din lista **history**
`forward()`	Încarcă URL-ul următor din lista **history**
`go()`	Încarcă un anumit URL din lista **history**

Obiectul `location`

Obiectul location conține informații despre url-ul curent..
Obiectul location este parte a obiectului window şi este accesat prin intermediul proprietății window.location.

Proprietățile obiectului `location`

Proprietate	Descriere
`hash`	Returnează porțiunea de ancoră din URL
`host`	Returnează hostname-ul şi port-ul URL-ului

hostname	Returnează hostname-ul URL-ului
href	Returnează întregul URL
pathname	Returnează numele căii URL-ului
port	Returnează numărul de port pe care serverul îl utilizează pentru URL
protocol	Returnează protocolul URL-ului
search	Returnează porțiunea query din URL

Metodele obiectului `location`

Metodă	Descriere
assign()	Încarcă un nou document
reload()	Reîncarcă documentul curent
replace()	Înlocuieşte documentul curent cu un alt document

Exemplu

Ilustrează utilizarea metodei **`assign()`**.

```
<html>
<head>
<script type="text/javascript">
function nou()
  {
  window.location.assign("http://www.google.com")
  }
</script>
</head>
<body>

<input type="button" value="Incarca noul document"
onclick="nou()" />

</body>
</html>
```

Bibliografie

Jim Keogh, JavaScript fara mistere - ghid pentru autodidacti, Editura Rosetti Educational, 2005

Tom Negrino, Dori Smith, JavaScript pentru World Wide Web, Editura Corint, 2004

Richard Wagner, JavaScript, Editura Teora

Diana Elena Diaconu, Pagini web cu JavaScript, Editura Edusoft, 2006

http://www.w3schools.com/

http://www.howtocreate.co.uk/tutorials/javascript/