

UNIVERSITATEA “BABEȘ – BOLYAI” CLUJ – NAPOCA
FACULTATEA DE MATEMATICĂ SI INFORMATICĂ

Limbajul JavaScript: aplicații în programarea distribuită

Coordonator științific:
Prof. Dr. Florian Mircea Boian

Absolventă:
Avram Sanda - Maria

1999

Cuprins

	Pag
Introducere.....	3
I. Ce este JavaScript?	4
1. Cum poate World Wide Web să prindă viață?.....	5
2. Rolul scripturilor.....	8
3. Extinderea capacităților unei pagini HTML.....	8
4. JavaScript și Java.....	11
II. Descrierea limbajului JavaScript	12
1. Sintaxa limbajului JavaScript.....	12
2. Variabile și valori.....	13
3. Instrucțiuni și operatori.....	17
3.1. Structura instrucțiunilor JavaScript.....	18
3.2. Operatori.....	18
3.3. Comentarii în JavaScript.....	20
4. Structurile de control.....	20
4.1. Instrucțiunea if	20
4.2. Instrucțiunea while	21
4.3. Folosirea instrucțiunii break	22
4.4. Folosirea instrucțiunii continue	22
4.5. Instrucțiunea for	22
5. Funcții și obiecte.....	24
5.1. Funcții.....	24
5.2. Obiecte.....	25
III. Tratarea evenimentelor în JavaScript	26
1. Evenimente și acțiuni.....	26
1.1. Acțiuni și evenimente de navigare.....	26
1.2. Completarea și evenimentele formularelor.....	27
1.3. Acțiuni care nu sunt evenimente.....	29
2. Scrierea de cod în JavaScript.....	29

2.1. Tagul SCRIPT	30
2.1. Declararea handlerelor de tratare a evenimentelor JavaScript..	30
3. Folosirea handler-elor pentru tratarea evenimentelor JavaScript.....	31
IV. Obiecte JavaScript	34
1. Obiecte, proprietăți și metode în JavaScript.....	34
2. Obiecte predefinite.....	43
2.1. Obiecte <i>string</i>	43
2.2. Obiectul <i>Math</i>	45
2.3. Obiectul <i>Date</i>	46
2.4. Funcții predefinite.....	47
3. Obiecte browser și HTML.....	47
3.1. Obiecte browser.....	47
3.2. Obiecte HTML.....	49
V. JavaScript pe server	51
1. Mediul LiveWire.....	51
2. Obiecte și proprietăți <i>Live</i>	51
2.1. Obiectul <i>request</i>	52
2.2. Obiectul <i>client</i>	53
2.3. Obiectul <i>project</i>	53
2.4. Obiectul <i>server</i>	54
3. Comunicarea externă a proceselor: JavaScript și CGI.....	55
4. Fișiere externe și baze de date cu JavaScript.....	58
VI. Aplicații JavaScript	63
1. Pagina Web a universității (http://www.ubbcluj.ro).....	63
2. Pagina Web a studenților (http://www.scs.ubbcluj.ro).....	65
3. Jocul “Lines” (http://www.scs.ubbcluj.ro/divertisment/).....	66
ANEXA.....	73
Bibliografie.....	86

Introducere

Lucrarea de față este structurată în două părți: o parte teoretică și o parte aplicativă în care se vor folosi elemente definite în prima parte.

Partea teoretică este destinată prezentării limbajului JavaScript. În *primul capitol* se face o prezentare pe larg a limbajului: care este utilitatea lui și cum se integrează la nivel de HTML. Tot aici se face o paralelă între Java și JavaScript.

Al doilea capitol începe o prezentare în detaliu a limbajului JavaScript prin descrierea unor concepte de bază ca: sintaxa limbajului, tipurile de date, instrucțiuni, operatori, structuri de control, funcții și obiecte.

În *capitolul al treilea* se prezintă modalitatea prin care se face comunicarea dintre elementele unei pagini Web și JavaScript. Această comunicare se realizează prin intermediul evenimentelor și acțiunilor JavaScript.

Capitolul patru prezintă obiectele JavaScript. Acestea pot fi obiecte definite de utilizator, obiecte predefinite, obiecte legate de browser sau obiecte care se referă la elementele unei pagini Web.

Spre deosebire de primele patru capitole care se referă doar la posibilitatea limbajului JavaScript de a dezvolta clientul, adică pagina Web la nivel de browser, *capitolul cinci* se referă la posibilitatea acestuia de a dezvolta aplicații pentru server. Acest lucru se poate realiza folosind pachetul de utilitare LiveWire.

Partea practică este cuprinsă în *capitolul șase* al acestei lucrări și constă în prezentarea următoarelor aplicații scrise folosind limbajul scriptural JavaScript: pagina Web a universității, pagina Web a studenților și un joc. Cea mai consistentă aplicație este jocul numit “Lines”, care are ca scop evidențierea rolului important pe care îl joacă JavaScript în realizarea unei pagini de Web interactive în care răspunsul la interacțiunile utilizatorului se realizează la nivel de browser, fără a fi nevoie de trimiterea unei cereri către serverul Web.

Textul sursă al paginilor Web în care se folosește JavaScript se află în *anexa lucrării*.

I. Ce este JavaScript?

Cu opt ani în urmă Internet-ul era un domeniu folosit doar de oameni de știință și programatori, iar World Wide Web era doar o idee încă neconturată în mintea unor cercetători. Astăzi interesul pentru Internet cât și pentru World Wide Web a cunoscut o creștere explozivă. Paginile Web sunt create într-un ritm surprinzător. Principala provocare a dezvoltării paginilor de Web este aceea că deși este foarte ușor să faci o pagină Web este însă foarte greu să realizezi o pagină atractivă.

HTML, limbajul de marcare care descrie formatul unei pagini Web este foarte ușor de învățat și nu necesită nici o cunoștință în programare. HTML a suportat câteva înnoiri în ideea acoperirii și extinderii cerințelor creatorilor de pagini Web. Totuși există limite la nivelul HTML.

Limbajul de programare Java a fost introdus pentru a extinde dezvoltarea paginilor Web prin introducerea unui set de utilitare, însă mult mai complex decât HTML. Java este ușor de învățat, însă ca majoritatea limbajelor de programare nu e ușor de stăpânit. JavaScript depășește această barieră.

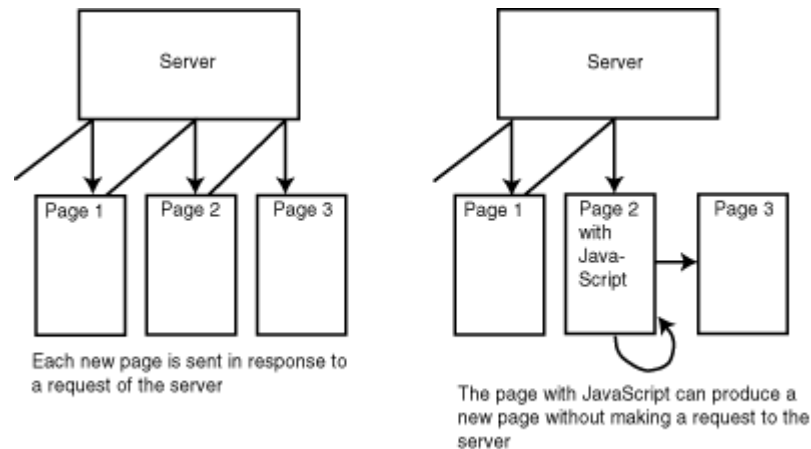


Fig 1.1 Trecerea la o altă pagină fără a face o cerere serverului

Cu JavaScript noi elemente dinamice vor trece dincolo de simpla navigare de la o pagină la alta. Utilizatorii nu vor citi numai pagini ci vor și interacționa cu ele. Paginile

vor prinde viața pentru fiecare utilizator. Orice interacțiune va avea un răspuns rapid deoarece nu se va solicita implicarea serverului ci rezolvarea acesteia se va realiza la nivel de browser (fig 1.1).

Aceste interacțiuni pot schimba o pagină într-o aplicație. Câteva butoane, un textbox și un simplu cod pot alcătui un calculator. Se poate construi de asemenea un editor sau un joc.

JavaScript este un limbaj scriptual compact, orientat obiect folosit în scopul dezvoltării aplicațiilor Internet client și server. Netscape începând cu versiunea 2.0 și Internet Explorer începând cu versiunea 4.0 știu să interpreteze instrucțiuni JavaScript incluse direct într-o pagină HTML, LiveWire oferind posibilitatea creării unor aplicații server similare cu programele CGI (Common Gateway Interface).

Într-o aplicație client pentru browser, instrucțiunile JavaScript incluse într-o pagină HTML pot recunoaște și răspunde la evenimente ca: click-ul cu mouse-ul, introducerea datelor într-un formular sau navigarea într-o pagină.

De exemplu se pot scrie funcții JavaScript care verifică dacă utilizatorul a introdus date valide în câmpurile unui formular. Fără a fi nevoie de o transmitere prin rețea, o pagină HTML care include JavaScript poate interpreta textul introdus și să avertizeze utilizatorul printr-o căsuță de dialog dacă textul introdus nu este valid. Se mai poate folosi JavaScript și pentru executarea unor acțiuni (redarea unui fișier audio, executarea unui applet, comunicarea cu un plug-in) ca răspuns la deschiderea unei pagini existente.

Cum poate World Wide Web să prindă viață?

În construirea unei pagini Web se urmărește prezentarea unei informații. Pentru ca respectiva pagină să fie vizitată, forma care îmbracă conținutul trebuie să fie cât mai atractivă.

Cu JavaScript, paginile pot prinde viață. De obicei, paginile răspund unei cereri în urma unui singur click. Multe alte elemente interactive sunt acum disponibile pentru a trezi interesul vizitatorilor. Acesta nu vor mai fi simpli cititori, ei vor interacționa cu paginile.

Formulare cu informatii

Formularele sunt binevenite în orice pagină Web. Câmpurile de text se pot schimba dinamic ca răspuns la o cererea utilizatorului:

- *Un calculator.* Pe lângă calculele aritmetice simple se pot efectua conversii în hexazecimal, si multe alte calcule.
- *Afisarea orei.* Se poate realiza o simplă cutie de text în care să se afișeze ora curentă, sau se poate afișa ora în fiecare colt al lumii si diferenta de timp dintre diverse regiuni. Se poate realiza un cronometru de numărătoare inversă.
- *Validări.* Intrările pot fi validate înainte de a fi trimise serverului.
- *Divertisment.* Oricine vrea să se distreze chiar si în timp ce învață. Informatia poate fi prezentată sub forma unui joc interactiv sau a unei glume.

Posibilitatea unei optiuni

Toate elemntele care apar într-o fereastră a browserului sunt disponibile limbajului JavaScript. Unele dintre aceste elemente pot fi modificate, altele pot contribui la crearea unui alt document.

- *Schimbarea culorilor.* Utilizatorul are posibilitatea de a-si alege singur culoarea fondului si a textului.
- *Schimbarea linkurilor.* Linkurile pot fi modificate astfel încât să corespundă preferintelor utilizatorului.
- *Reformatarea paginii.* Deoarece JavaScript are acces la toate elementele, reformatarea unui document nu este ceva prea complicat de realizat.

Ferestre multiple si frame-uri

Netscape introduce frame-uri si JavaScript începând cu versiunea 2.0. Microsoft Internet Explorer 3.0 suportă de asemenea JavaScript, frame-uri si ferestre multiple.

- **Cutiile de dialog alert, confirm si prompt.** JavaScript are propriile ei cutii de dialog care pot fi folosite. Attentionarea utilizatorului asupra unui lucru (alert), confirmarea sau anularea unei actiuni (confirm) sau introducerea unui text (prompt).
- **Controlul ferestrelor.** Pe lângă ferestrele de dialog se mai pot crea propriile controale cu ferestre obisnuite. Modificările aduse asupra butoanelor, textbox-urilor sau icoanelor din noua fereastră de control vor fi răspuns pentru fereastra inițială.
- **Ferestre de navigare.** Putem avea două sau trei ferestre de navigare deschise simultan. În timp de utilizatorul navigheaza într-o fereastră, celelalte se încarcă.

Interactiunea cu alte obiecte “Live”

Sun Microsystems si Netscape Communications au introdus JavaScript. Încă alte 28 de companii de calculatoare printre care AOL, AT&T, Borland, Digital Equipment Corporation, Hewlett-Packard Corporation si Oracle Corporation sprijină JavaScript. Microsoft suportă JavaScript începând cu Internet Explorer 3.0. Aceste companii suportă JavaScript pentru că este un limbaj obiectual standard. Câteva companii vor introduce produse care incorporează JavaScript. Acest lucru va permite o mai mare interactiune. Următoarele produse suportă JavaScript:

- **Netscape's LiveWire.** LiveWire este mediu de dezvoltare vizual pentru crearea aplicatiilor Internet. Acest produs nou aduce îmbunătățiri la server incluzând JavaScript. Acelasi limbaj care este folosit la nivel de browser pentru a face paginile să prindă viață poate fi folosit pentru a răspunde unor cereri la nivel de server. În locul scrierii unor scripturi CGI în Perl, C sau orice alt limbaj, se foloseste JavaScript.
- **Plug-in-uri.** Plug-in-urile sunt programe care lucrează cu Netscape's Navigator. Aceste aplicatii pot fi produse cu editoare ca Adobe, MacroMedia, Borland si altele. Aceste programe permit interpretarea altor formate de fisiere, prezentări multimedia si functii specializate.

Numărul de plug-in-uri pentru browser este de asteptat să crească foarte mult. Cum aceste aplicatii devin din ce în ce mai sofisticate, se va dori să se folosească

JavaScript ca și limbajul lor pentru scripturi. În plus, multe dintre aceste editoare vor permite ca, la nivel de browser, JavaScript să interacționeze cu plug-in-urile lor. Obiectele pentru aceste programe vor fi disponibile din JavaScript.

Cu obiectele Live, se pot crea continuturi pline de multimedia în cadrul site-urilor Internet, permitând utilizatorilor să vadă împreună conținutul cu plug-in-uri din Adobe Acrobat, Apple QuickTime și Macromedia Shockwave for Director într-o fereastră client fără lansarea unui utilitar extern.

Rolul scripturilor

Nu există o definiție definitivă asupra limbajelor scriptuale. Câteodată termenul este folosit pentru a face o diferență față de limbajele compilabile. Unele limbaje ca C sau C++ pot fi folosite atât pentru scripturi cât și pentru aplicații. Termenul de script este folosit pentru că limbajele respective reacționează la, controlează sau tratează o serie de evenimente.

Scopul celor mai multe limbaje scriptuale este de a extinde noi capacități ale aplicațiilor. Un creator de scripturi lucrează cu componente finale relevând proprietăți high-level cum ar fi “*visible*” și “*color*”; ia și setează proprietăți pentru a obține efectul dorit.

JavaScript oferă creatorilor de pagini Web un nou nivel de complexitate fără a cere acestora să devină programatori. JavaScript aduce capacități dinamice și puternice paginilor de Web, însă JavaScript este mult mai greu de învățat decât HTML. JavaScript poate fi folosit în rezolvarea unor probleme obișnuite cum ar fi validarea valorilor introduse într-un formular, dar poate fi folosit pentru a crea un conținut interesant, cu un aspect vizual atrăgător, lucru care nu poate fi realizat cu HTML.

Extinderea capacităților unei pagini HTML

Ca și celelalte limbaje scriptuale care extind capacitățile aplicațiilor cu care acestea lucrează, JavaScript extinde paginile Web standard dincolo de utilitatea lor uzuală.

Cu o pagină Web standard poti obtine multe informatii dând click pe un hiperlink pentru ca serverul să-ti trimită o altă pagină. O pagină mai atractivă ar fi aceea în care completezi un formular care va fi trimis serverului si astepti un răspuns. În ambele cazuri trebuie să astepti ca serverul să trimită un nou fisier. Aceste informatii se află aproape întotdeauna într-o altă pagină chiar dacă este vorba de un fisier multimedia, un clip audio sau unul animat.

Paginile dezvoltate cu JavaScript sunt pagini HTML în care se află si cod JavaScript. JavaScript poate analiza informatia introdusă fără a fi nevoie să se astepte vreun răspuns de la server. Aceste informatii pot veni odată cu fisierul curent însă într-o formă ascunsă.

Paginile dezvoltate JavaScript vor face ca aceste informatii “ascunse” să devină vizibile prin actualizarea continutului formularului sau prin generarea unui întreg document nou.

Javascript lucrează cu browserul prin înglobarea codului direct într-o pagină HTML. Netscape a introdus un nou tag generic numit **SCRIPT** pentru a recunoaste limbajele scriptuale. Pentru a informa browserul că codul e scris în JavaScript trebuie adăugat la tagul **SCRIPT** atributul **LANGUAGE="JavaScript"**. Majoritatea codului JavaScript se află între aceste taguri, după cum se poate observa în următorul exemplu:

```
<SCRIPT LANGUAGE="JavaScript">  
  a = "Hello!"  
  //...setarea unei variabile numită 'a' la valoarea "Hello!"  
</SCRIPT>
```

Între tagurile **SCRIPT** se pot scrie două tipuri de cod: *instructiuni* sau *functii*. Instructiunile sunt executate de către browser în momentul în care sunt încărcate. De exemplu, obiectele sunt initializate în cadrul unor instructiuni. Functiile sunt blocuri de cod care sunt executate numai de către alt cod sau eveniment. Majoritatea programelor vor folosi atât direct instructiuni cât si functii.

Multe dintre tagurile HTML deja existente au attribute care suportă JavaScript. De exemplu, toate elementele unui formular pot fi identificate printr-un nume cu ajutorul

atributului **NAME**. Folosirea atributului **NAME** pentru a identifica obiectele vor simplifica scrierea codului si corectarea acestuia.

O altă modificare adusă la HTML este recunoasterea evenimentelor ca click-ul mouse-ului, modificarea continutului unui textbox, încărcarea si descărcarea unei pagini. Prin aceste evenimente documentul recunoaste interactiunea utilizatorului. Ele sunt folosite pentru a declansa actiunile JavaScript.

Codul JavaScript care este declansat de un eveniment poate fi simplu sau complex. În actiuni simple, întregul cod JavaScript poate fi plasat în elementul eveniment.

```
<FORM>
```

```
<P>Faceti click în interiorul textbox-ului pentru a vedea diferenta.
```

```
<INPUT TYPE="text" NAME="exemplu" onChange = "exemplu.value = a">
```

```
</FORM>
```

Ca orice limbaj de programare, JavaScript lucrează cu informatii. Fiind un limbaj obiectual are *metode* si *functii* care actionează asupra informatiilor din obiecte. În afară de obiectele **string**, **math** si **date**, JavaScript este limitat să opereze pe obiectele browserului si pe alte obiecte atasate acestuia. Aceste obiecte pot include plug-in-uri, aplete Java si elemente ActiveX. Acestea permit crearea de noi documente, modificarea unor formulare existente si construirea unor aplicatii.

JavaScript lucrează cu obiectele browserului, însă majoritatea codului manipulează elemente HTML. Limitarea există datorită faptului că nu sunt operatii noi pentru care să ofere capacități multimedia (sunet sau grafică). Pentru a adăuga acest tip de elemente, este nevoie de o extindere a capacităților browserului cu plug-in-uri, aplete Java sau alte aplicatii externe. Aceste programe pot sau nu pot să pună obiectele lor disponibile pentru JavaScript.

JavaScript si Java

Limbajul JavaScript mosteneste limbajul Java, cu exceptia puternicei tipizării. JavaScript suportă majoritatea sintaxelor expresiilor si a structurilor de flux de control Java. În contrast cu sistemul de clase *compile-time* Java, JavaScript suportă un sistem *run-time* bazat pe un număr mic de tipuri de date: numeric, Boolean, string.

Programele Java consistă exclusiv din clase si metodele lor. Cerintele Java pentru declararea claselor, scrierea metodelor, siguranta tipului fac programarea mult mai complexă decât JavaScript.

JavaScript se apropie mai mult de limbajale scriptuale ca HyperTalk si dBASE care oferă instrumente de programare unei mai largi audiente datorită sintaxei simple specializate pe funcționalități *built-in* si necesitatea minimă de creare de obiecte.

Următorul tabel cuprinde asemănările si deosebirile dintre Java si JavaScript:

JavaScript	Java
Interpretat (nu compilat) de client	Compilat pe server înainte de a fi executat de client
Orientat obiect. Există obiecte predefinite, se pot extinde obiecte dar nu există clase sau mostenire.	Orientat obiect. Appletele sunt clase de obiecte cu mostenire.
Codul este inclus în HTML.	Apletele sunt accesate din pagini HTML însă se află separat.
Nu se declară tipul variabilelor (<i>loose typing</i>)	Tipul variabilelor trebuie declarat (<i>strong typing</i>)
Legare dinamică. Referirea obiectelor se verifică la rulare.	Legare statică. Referirea obiectelor trebuie să existe la compilare.
Sigur. Nu poate scrie pe hard disk.	Sigur. Nu poate scrie pe hard disk.

II Descrierea limbajului JavaScript

Există în ziua de azi multe limbaje de programare. Multe dintre ele sunt asociate unor aplicatii specifice: Ada este regăsit în aplicatiile militare iar Lisp este asociat aplicatiilor de inteligență artificială. Unele limbaje, de exemplu HTML, limbaj utilizat pentru a descrie paginile World Wide Web, au o organizare structurală bine definită, însă au foarte puține elemente dintr-un program structurat traditional.

Încercarea de a înțelege un nou limbaj de programare nu este importantă numai pentru stăpânirea sintaxei, însă este necesară și pentru a putea descoperi cum poate fi folosit mai bine limbajul pentru atingerea unor scopuri propuse.

În primul capitol "*Ce este JavaScript?*" s-a încercat sublinierea câtorva scopuri principale ale limbajului JavaScript cât și diferențele față de limbajul mult mai structurat: Java. Acest capitol descrie limbajul JavaScript din ambele perspective. Este dată o descriere a sintaxei și câteva concepte de bază în ceea ce privește structura unui program JavaScript.

Sintaxa limbajului JavaScript

JavaScript urmează modelul bazat pe acțiuni al World Wide Web [2]. Elementele unei pagini Web ca: butoane, checkbox-uri pot declanșa acțiuni sau evenimente. Când are loc un astfel de eveniment, se va executa o parte de cod scrisă în JavaScript, de obicei o funcție JavaScript. Această funcție poate să facă anumite calcule, să examineze sau să modifice conținutul paginii Web, sau să facă alte operații pentru a răspunde acelui eveniment.

Elementele unui program JavaScript pot fi împărțite în cinci categorii [2]:

1. Variabile și valorile lor;
2. Expresii care lucrează cu aceste valori;
3. Structuri de control, care dirijează ordinea de efectuare a instrucțiunilor;
4. Funcții, care sunt compuse din blocuri de instrucțiuni;

5. Obiecte si tablouri, care sunt modalități de grupare a datelor înrudite.

Aceste categorii se regăsesc si în cadrul altor limbaje de programare.

Multe elemente familiare cum ar fi declararea tipului variabilelor (int, String, Real) lipsesc sau au fost substantial simplificate. JavaScript cuprinde o serie de constructii obiectuale care, de asemenea, simplifică organizarea programului. În acest fel JavaScript are puterea expresivă a limbajelor ca C sau Java, dar e nevoie de mai putine reguli de tinut minte.

Variable si valori

Principala diferetă dintre JavaScript si alte limbaje de programare este că nu are tipuri explicite de date. Orice variabilă JavaScript poate fi interpretată diferit în contexte diferite.

Toate variabilele JavaScript sunt declarate folosind cuvântul rezervat *var*. Variabilele pot fi initializate în sensul că la declarare ele pot primi o valoare. Pot fi declarate pe aceeasi linie mai multe variabile, fiind separate prin virgulă:

```
var x = 7  
var y,z = "19"  
var lk = "lucky"
```

Se poate considera că *x* este întreg, *z*, *lk* sunt stringuri, iar variabila *y* poate fi orice. De fapt, valoarea fiecărei variabile depinde de contextul în care este folosită. Acest context este stabilit în functie de ordinea în care apar variabilele. Astfel, expresiile:

```
5 + x  
lk + z
```

sunt evaluate cu 12 si "lucky19". Este de asemenea posibilă formarea următoarelor expresii:

$$lk + x$$
$$x + z$$

care sunt evaluate cu *“lucky7”*, respectiv 26. În prima expresie, *x* a fost interpretat ca string, în timp ce în a doua *z* a fost interpretat ca întreg.

Aceste două exemple ilustrează două lucruri importante despre JavaScript. În primul rând, JavaScript nu are tipuri de date explicite, însă are tipuri de date implicite. În al doilea rând, JavaScript are o serie de reguli de conversie care îi permit să decidă, în funcție de context, cum să trateze valorile variabilelor. Contextul este stabilit prin citirea expresiei de la stânga la dreapta.

Ce se va întâmpla în cazul în care încercăm $x + lk$? Variabila *x* este cea care apare prima în stânga, deci valoarea expresiei ar trebui să fie un număr. JavaScript va încerca să interpreteze pe *lk* ca un număr, însă *“lucky”* nu poate fi interpretat ca un număr. JavaScript dă eroare în cazul în care va trebui să evalueze o expresie de acest fel.

Ca ultim exemplu vom considera o expresie în care să apară și variabila *y* care nu este inițializată cu nici o valoare.

$$x = z + y$$

În majoritatea celorlalte limbaje de programare, rezultatul folosirii unei variabile care nu a fost inițializată este întotdeauna gresit. Datorită faptului că *y* nu a fost inițializată, nu există nici o posibilitate ca expresia să fie evaluată.

Tipuri de date implicite în JavaScript

Există cinci tipuri importante de date în JavaScript:

- Numeric (exemplu: -5, 8, 6.89);
- String (exemplu: *“JavaScript”*);
- Logic (valorile **true** sau **false**);
- Element JavaScript *“non-atomic”*, cum ar fi o funcție sau un obiect;
- Valoarea **null**.

Varibile si nume de variabile

Este foarte important să se facă distincție între variabile și valorile lor. Instrucțiunea $x = 10$ conține două componente: variabila x și literalul 10 . Un literal înseamnă orice valoare care este referită direct. O variabilă este doar o abstracție care denumeste o valoare. Faptul că lui x i s-a atribuit o valoare numerică nu ne împiedică ca mai târziu, în program să facem: $x = \text{"string"}$, însă acest lucru poate conduce la confuzii.

În JavaScript, numele variabilelor trebuie să înceapă cu o literă sau cu caracterul de subliniere `'_'`. Se va face deosebirea între litere mari și litere mici, astfel: `varib`, `Varib`, `VARIAB` și `_VARIAB` vor fi identificatori JavaScript care reprezintă variabile distincte.

Valori numerice

În JavaScript există două tipuri numerice: întregi și numere în virgulă flotantă. Regulile de tratare a acestor tipuri sunt foarte asemănătoare celor din C, C++ sau Java. Întregul poate fi specificat în baza 10 (zecimal), în baza 8 (octal) sau în baza 16 (hexazecimal). Cele trei forme se deosebesc prin primele două cifre de la începutul numărului:

- 1-9 urmat de alte cifre reprezintă un întreg zecimal;
- 0 urmat de cifre între 0 și 7 este un întreg octal;
- 0x sau 0X urmat de 0-9, a-f sau A-F este un întreg hexazecimal.

Toate dintre cele trei forme pot începe cu semnele `'+'` sau `'-'`. Minimul sau maximul unui întreg trebuie să încapă ca reprezentare pe 32 biți.

Numerele în virgulă flotantă pot fi scrise folosind punctul zecimal `'.'` sau folosind notația E. Aceste numere trebuie să contină punctul zecimal sau un exponent care poate începe cu `'e'` sau `'E'`. Astfel, 0.0, -1.5e67 și 7.8909 sunt exemple de numere în virgulă flotantă.

Stringuri

În JavaScript stringurile pot fi specificate folosind apostroafe ('string') sau folosind ghilimele ("string"). Nu se admit construcții de forma "string' sau 'string". Există posibilitatea de a încuiba stringuri prin alternarea apostroafelor cu ghilimelele sau invers. De exemplu:

"Limbajul pe care îl vom folosi este 'JavaScript'."

Ca și în C și Java stringurile din JavaScript pot conține combinații de caractere speciale, numite *escape sequences*, pentru a distinge anumite caractere speciale. În cazul acestor caractere speciale se suportă regulile din C.

Se suportă următoarele secvențe:

\t	Tab
\r	Line feed
\n	Return
\f	Form feed (vertical tab)
\b	Backspace

Stringurile "" și '' reprezintă siruri de lungime zero. În JavaScript, acesta este sirul cu cea mai mică lungime, iar cea mai mare lungime pentru un sir, de obicei, depinde de implementare.

Valorile logice TRUE și FALSE

Valorile logice sau booleene **TRUE** și **FALSE** sunt folosite de obicei în expresii conditionale. Rezultatul unei expresii conditionale este o valoare booleană.

JavaScript convertește aceste valori logice în **1** pentru **TRUE** și **0** pentru **FALSE**. De asemenea, valorile nenule vor fi interpretate ca **TRUE**. Multe limbaje de programare folosesc aceeași convenție.

Valoarea null

Valoarea **null** are un rol special în limbajul JavaScript. Se folosește pentru a initializa variabilele care nu au valori initiale. În cazul folosirii acestor variabile în expresii care necesită o valoare, valoarea **null** a acestora va fi convertită astfel: **0** pentru conversie la întreg, **""** pentru conversie la string și **FALSE** pentru conversie la boolean.

Conversii de tip

Se fac conversii de tip în cazul în care într-o aceeași expresie apar variabile de tipuri diferite, astfel în expresia:

$$VarString + VarNum\bar{a}r$$

Numărul va fi convertit la string, iar **+** va fi operator de concatenare a două siruri. În cazul în care avem:

$$VarNum\bar{a}r + VarString$$

stringul va fi convertit la număr și va fi adunat cu *VarNumăr*. Dacă *VarString* nu poate fi convertit la număr expresia va fi eronată. Acestea reprezintă conceptele de conversie în JavaScript.

Se recomandă folosirea conversiilor explicite în locul conversiilor implicite.[2] Conversiile explicite vor fi dezvoltate în *capitolul 5 "Obiecte predefinite JavaScript"* în secțiunea *"Obiectul String"*.

Instructiuni si Operatori

Unitatea de lucru în JavaScript, ca în majoritatea limbajelor de programare, este *instructiunea*. Programele JavaScript sunt colecții de instructiuni, organizate în funcții, care lucrează cu variabile și cu elemente HTML.

Structura instructiunilor JavaScript

O instructiune de forma:

$$y = x + 5$$

contine trei p rti: rezultatul y , operetorul $=$ si expresia $x + 5$. Rezultatul se afl   ntotdeauna  n partea st ng  si se numeste *lvalue*. Acest rezultat trebuie s  fie ceva care s  poat  fi modificat. Nu se poate folosi instructiunea: $null = x + 5$.

Operatorul $=$ este operatorul de atribuire. El determin  evaluarea expresiei din dreapta iar valoarea rezultat  este atribuit  variabilei din st nga.  n cadrul expresiei $x + 5$ se foloseste un alt operator: $+$ care este numit *operator supra nc rcat* datorit  faptului c   n cazul  n care x este un num r va realiza sum  de numere iar  n cazul  n care x este sir, va face concatenare de siruri.

O instructiune JavaScript poate fi terminat  cu caracterul $;$. Pe o linie de program se pot pune mai multe instructiuni separate de acest separator.

Operatori

Operatorii folositi  n JavaScript sunt foarte asem n tori cu cei din C, C++ si Java. Ei pot fi clasificati  n urm toarele grupe [2]:

- operatori aritmetici
- operatori logici
- operatori pe biti
- operatori de atribuire si agregare

Operatori aritmetici

+	Adunare, Concatenare de siruri
-	Sc�dere, Negatie unar�
*	�nmultire
/	�mp�rtire
%	Modul
++	Incrementare
--	Decrementare

Operatori logici

==, !=	Egalitate, Inegalitate
<, <=, >=, >	Comparare aritmetică și de siruri
!	NOT (negare logică)
&&,	AND, OR (și, sau logic)
?	Selectie condițională
,	Concatenare logică

Operatori pe biti

&,	AND pe biti, OR pe biti
^	OR eXclusiv pe biti (XOR)
~	NOT pe biti
<<, >>, >>>	Siftare la stânga, la dreapta, la dreapta fără semn

Operatori de atribuire

=	Atribuire
Op=	Agregare (+, -, *, /, %, &, , ^, ~, <<, >>, >>>)

Din punctul de vedere al ordinii de evaluare a expresiilor există două soluții de abordare: cunoașterea tabelului de precedență al operatorilor sau folosirea parantezelor.

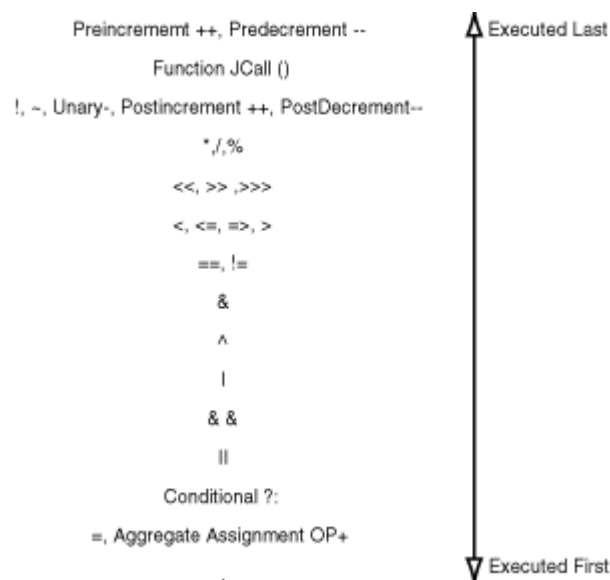


Fig 2.1 Tabelul de precedență al operatorilor

Comentarii în JavaScript

Orice cod profesional trebuie să contină comentarii care să indice scopul și logica fiecărei secțiuni mari de cod. JavaScript oferă două tipuri de comentarii: unul preluat din C și comentariul linie preluat din C++ și Java.

Comentariile preluate din C sunt folosite pentru blocuri de cod sau funcții. Un comentariu C începe cu `/*` și se termină cu `*/`.

Comentariile C++ sunt folosite pentru descrieri scurte care să încapă într-o singură linie. Un astfel de comentariu începe cu `//`.

Ambele stiluri de comentarii pot fi folosite împreună în același program, însă ele nu trebuie încuibate pentru a nu crea confuzii.

Nu se pot folosi comentariile HTML (`<!--` și `-->`).

Structuri de control

În JavaScript există trei tipuri de structuri de control [6]:

- **if**
- **while**
- **for**

Aceste structuri de control sunt foarte asemănătoare între ele. Toate încep cu un cuvânt rezervat (**while**, **for**, **if**) și fiecare manipulează un bloc de instrucțiuni JavaScript. Acest bloc este cuprins între două acolade (`{ ...blocul de instrucțiuni... }`) și poate să contină oricâte instrucțiuni vrea sau poate să nu contină nici o instrucțiune. Un bloc de instrucțiuni se execută compact, se execută tot blocul sau nu se execută nici o instrucțiune din bloc. Datorită faptului că blocurile de instrucțiuni se comportă ca o singură instrucțiune se pot încuiba blocuri în alte blocuri de instrucțiuni.

Instrucțiunea if

Instrucțiunea **if** simplă cuprinde o expresie condițională numită *test* și un bloc de instrucțiuni care este executat numai dacă expresia condițională este evaluată la **TRUE**.

```

if ( instr_cond ) {
    zero sau mai multe instructiuni
}

```

Blocul de cod se mai numeste *ifblock*. În locul expresiei conditionale se mai pot folosi valorile **0** pentru **FALSE** si numere nenule pentru **TRUE**.

În cazul structurii de control **if...else** există un bloc de cod care se execută dacă testul este adevărat si un alt bloc care se excută dacă testul este fals. O astfel de instructiune are următorul format:

```

if (instr_cond) {
    ifblock-ul de instructiuni
} else {
    elseblock-ul de instructiuni
}

```

Instructiunea while

Instructiunea **while** are următoarea formă:

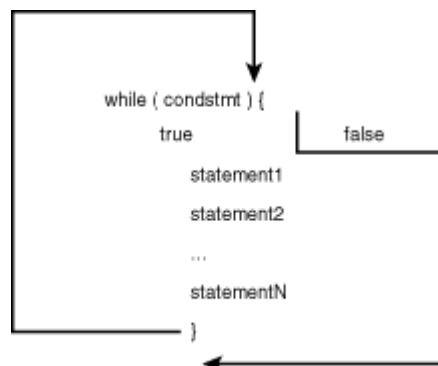
```

while (instr_cond) {
    zero sau mai multe instructiuni
}

```

Expresia condițională *instr_cond* este evaluată. Dacă rezultatul evaluării este **TRUE** atunci se vor executa instructiunile dintre acolade si se va trece la următorul pas **while**. Se evaluează din nou *instr_cond* si procesul continuă până când *instr_cond* devine fals sau se fortează terminarea ciclului. Fiecare parcurgere a blocului de cod se numeste *iteratie*.

Fig 2.2 Structura de control **while** execută conditional un bloc de instructiuni



Folosirea instructiunii **break**

Există o a treia modalitate de terminare a unui ciclu *while*. Dacă instrucțiunea **break** se găsește într-un bloc *while*, ciclul este forțat să se termine imediat. Nici o altă instrucțiune din blocul de cod din ciclu *while* nu va mai fi executată și nici *instr_cond* nu va fi retestată.

Folosirea instructiunii **continue**

O altă instrucțiune specială care poate fi folosită într-un ciclu *while* este instrucțiunea **continue**. Instrucțiunea **continue** se folosește pentru a forța trecerea la un pas următor al ciclului *while* fără executarea instrucțiunilor dintre ea și sfârșitul blocului de cod.

Instructiunea **for**

Instrucțiunea **for** este cea mai puternică și mai complexă dintre cele trei construcții de control al fluxului în JavaScript. Scopul principal al acestei instrucțiuni este de a executa un bloc de instrucțiuni pentru o anumită variabilă care ia, pe rând, toate valorile dintr-un interval. Instrucțiunea **for** are următorul format:

```
for ( instr_init; instr_cond; instr_pas ) {  
    forblock  
}
```

O clauză **for** are trei părți despărțite prin separatorul ‘;’. Instrucțiunea *instr_init* reprezintă initializarea variabilei de ciclare. Această instrucțiune se execută exact odată, și anume, la primul pas al instrucțiunii **for**. Testul condițional, *instr_cond* are aceeași funcție ca și în cazul instrucțiunii **while**; se va testa la fiecare ciclu. Instrucțiunea *instr_pas* se execută la sfârșitul fiecărui ciclu, ca și cum această instrucțiune ar fi plasată imediat după ultima instrucțiune din *forblock*. Această instrucțiune se folosește pentru a actualiza valoarea variabilei de ciclare care a fost inițializată în cadrul instrucțiunii *instr_init*.

Instrucțiunea **for** mai are câteva elemente speciale care nu se regăsesc la instrucțiunea **while**. Primul element îl constituie faptul că în porțiunea de initializare a variabilei de ciclu se pot face și alte initializări ale altor variabile. Un alt element este coexistența atât în secțiunea de inițializare cât și în cea de actualizare a variabilei de ciclu a mai multor instrucțiuni separate prin virgulă.

Pentru obiectele și tablourile din JavaScript există o altă formă de instrucțiune **for**: instrucțiunea **for...in**, însă această formă se va discuta în capitolul 4, “Obiecte JavaScript”.

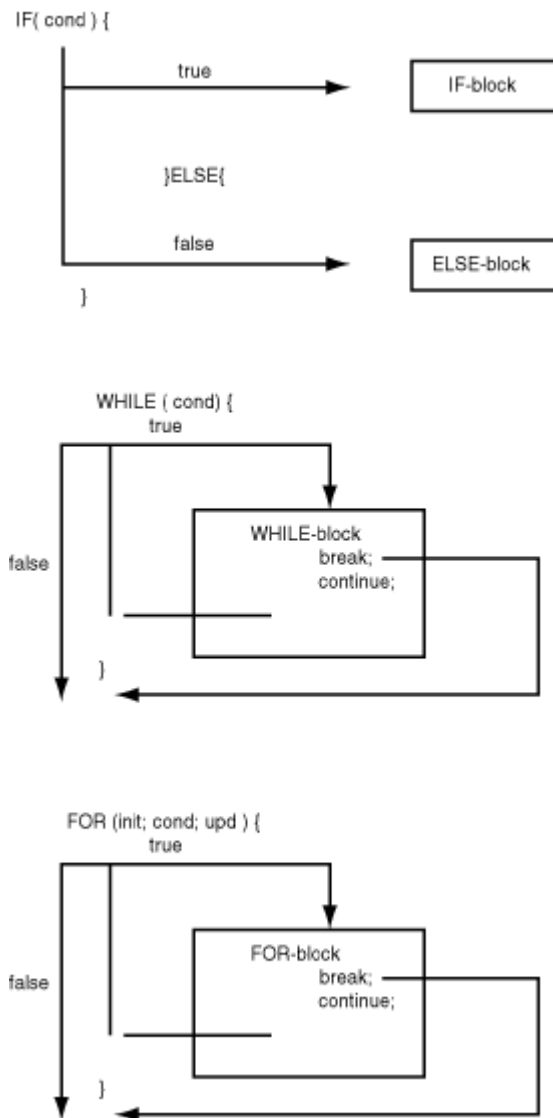


Fig 2.3 Structurile de control determină firul execuției unui cod JavaScript

Funcții și obiecte

Instrucțiunile de bază, expresiile și operatorii sunt ceea ce informaticienii numesc *primitive*. Primitivele sunt elementele din care se construiesc elementele mai complexe ale unui program. Structurile de control **for**, **while** și **if** reprezintă următorul nivel de complexitate în JavaScript. Fiecare dintre aceste structuri de control lucrează cu blocuri de cod a căror execuție este controlată de diverse teste conditionale și alte clauze.

Funcțiile și obiectele reprezintă un nivel și mai complex de organizare în JavaScript.

Funcții

O **funcție** este un bloc de cod căruia i se atribuie un nume. Când numele respectiv este menționat se va apela funcția, ceea ce înseamnă că blocul de instrucțiuni va fi executat. Funcțiile pot fi apelate cu unele valori care se numesc parametri, și pot fi folosite în corpul funcției.

Funcțiile au două scopuri principale. Primul este acela că o **funcție** este un instrument în sensul că permite folosirea aceluiași cod fără ca acesta să fie multiplicat. Al doilea scop este acela că, în JavaScript, funcțiile sunt folosite pentru a lega acțiuni care se efectuează asupra unei pagini Web de cod JavaScript. Click cu mouse-ul, apăsarea unui buton, selectarea unui text și alte asemenea acțiuni pot apela funcții JavaScript.

Sintaxa unei funcții JavaScript este următoarea:

```
function Nume_funcție ( listă_de_parametri ) {  
    corpul_funcției  
}
```

Numele unei funcții trebuie să fie unic și să nu fie un cuvânt rezervat JavaScript. Lista de parametri este o listă de valori separate prin virgulă și care vor fi folosite în interiorul funcției. Această listă poate fi vidă. Corpul funcției este o mulțime de instrucțiuni care constituie de fapt funcția.

Obiecte

Funcțiile sunt folosite în scopul obținerii unei metode uniforme de organizare a codului. **Obiectele** servesc același scop pentru date. Până la acest moment singurele date pe care le-am văzut sunt variabilele declarate cu *var*. Fiecare dintre acestea poate avea la un moment dat o singură valoare. **Obiectele** oferă posibilitatea de reținere a mai multor valori astfel încât un grup de date înrudite pot fi asociate cu alt grup.

Ceea ce în JavaScript se numește obiect, în alte limbaje de programare se numește structură de date sau clasă. Ca și în cazul funcțiilor, există două aspecte legate de obiectele JavaScript: crearea și folosirea lor.

Un obiect JavaScript are două tipuri de părți componente: *proprietăți* și *membri*. Să presupunem că avem un obiect numit *orar*, care este folosit pentru stabilirea cursurilor. Un obiect *curs* poate avea proprietăți care specifică *data*, *ora* la care începe cursul și *numele profesorului* care predă acel curs. Mai poate exista și un câmp de *descriere* a respectivului curs. Deci ne putem imagina că obiectul *orar* va avea următoarele proprietăți:

- *ziua*
- *luna*
- *ora*
- *nume_profesor*
- *descriere_curs*

Fiecare dintre proprietățile obiectului *orar* se vor referi folosind operatorul punct ‘.’. Astfel, *orar.ora* se va referi la proprietatea *ora* a obiectului *orar*.

Orice obiect poate conține orice fel de variabile. Pot conține și metode, funcții care realizează un anumit lucru. Obiectele pot conține chiar și alte obiecte, astfel încât se poate crea o structură ierarhică de date.

III. Tratarea evenimentelor în JavaScript

Deși paginile Web sunt greu de dezvoltat, ele sunt foarte ușor de utilizat. O pagină Web simplă poate conține doar: text, grafice sau imagini și sunete de fond. De obicei, vizitarea unei pagini Web nu oferă utilizatorului posibilitatea de a interacționa cu aceasta. Singura interacțiune apare atunci când utilizatorul selectează un link sau o imagine mapată.

Introducerea formularelor a făcut ca interacțiunea utilizatorilor cu paginile Web să crească. Un formular poate conține elemente de introducere de date, câmpuri de text, butoane, checkbox-uri și opțiuni de selectare. Pentru validarea datelor introduse, întreg formularul trebuie transmis serverului.

Unul dintre scopurile principale ale JavaScript este de a face majoritatea calculelor și validărilor direct în browser-ul utilizatorului. Acest scop este atins prin tratarea evenimentelor. Tratarea evenimentelor se face, în JavaScript, prin instrucțiuni (de obicei funcții) care sunt apelate atunci când se întâmplă ceva.

Evenimente și acțiuni

Pentru a înțelege care dintre acțiunile browser-ului corespund evenimentelor JavaScript, este important să recunoaștem acele acțiuni care determină schimbări în afișarea unei pagini Web. Există două tipuri de astfel de acțiuni: navigarea și interacțiunea cu un formular HTML.

Acțiuni și evenimente de navigare

În timpul navigării se disting următoarele acțiuni:

- Selectarea unui hiperlink;
- Avansarea sau întoarcerea în cadrul listei “history”;
- Deschiderea unui nou URL (în aceeași sau altă fereastră);
- Părăsirea browser-ului.

În majoritatea cazurilor de mai sus, pagina curentă se *descarcă*. În primele trei dintre cazurile de mai sus o nouă pagină se va *încărca*. Aceste două evenimente, de încărcare și descărcare a unei pagini, pot fi tratate de JavaScript, adică se poate scrie cod care să se apeleze fie la încărcarea, fie la descărcarea unei pagini HTML.

Alte două evenimente apropiate de navigare sunt:

- mouseover
- statechange

În momentul în care mouse-ul se mișcă deasupra unui hiperlink se generează un eveniment *mouseover*. Evenimentul *statechange* este propus de Microsoft în ideea indică stadiul de desfășurare a unei operații care tocmai are loc.

Completarea și evenimentele formularelor

O altă modalitate de interacțiune cu paginile Web este formularul. De aceea, orice element care permite o astfel de interacțiune este asociat cu unul sau mai multe evenimente JavaScript.

Elementele unui formular HTML sunt următoarele:

- butoane
- câmpuri pentru text
- liste de selecție

Butoanele pot fi:

- butoane simple
- checkbox-uri
- butoane radio
- butoane “submit”
- butoane “reset”

Un buton se poate specifica în HTML prin tagul `<INPUT TYPE = “”>`, unde tipul este *button* pentru butoane simple, și respectiv *checkbox*, *radio*, *submit*, *reset* pentru

celelalte. Lucrul care este comun acestor butoane este acțiunea “*click*” care se efectuează asupra lor. Deoarece aceasta este o acțiune foarte întâlnită, JavaScript definește evenimentul *click* ca și un eveniment al formularelor HTML. Când se trimite un formular se generează evenimentul *submit*. Acest eveniment se realizează la trimiterea unui formular și nu la apăsarea butonului submit care o generează.

HTML Elements \ Events	Events								
	Blur	Click	Change	Focus	Load	Mouseover	Select	Submit	Unload
Button		X							
Checkbox		X			X				X
Document								X	
Form						X			
Link		X							
Radio		X							
Reset		X							
Selection	X		X	X					
Submit		X							
Text	X		X	X			X		
Textarea	X		X	X			X		

Fig 3.1 Evenimentele JavaScript diferă în funcție de interacțiunile utilizatorului asupra paginii Web

Elementele de text sunt în număr de trei:

- câmpuri de text
- câmpuri de text multi-line
- câmpuri de parolă

Aceste elemente se introduc folosind același tag ca și în cazul butoanelor cu tipul *text* și *password* în cazul câmpului de text și a câmpului de parolă; iar cu tagul

TEXTAREA în cazul câmpurilor de text multi-line. Asupra acestor elemente se pot efectua mai multe operații. Se poate face *click*, se poate introduce text, se poate edita text, se poate selecta text sau se poate decide dacă se va trece mai departe. Evenimentele JavaScript care se generează ca răspuns la aceste acțiuni sunt următoarele: *change*, *select*, *focus*, *blur*. Aceste evenimente nu sunt asociate câmpului de parolă. Evenimentele de *change* și *select* au loc când un text a fost schimbat sau selectat. Evenimentul *focus* are loc când se acceptă introducerea de text de la tastatură. Evenimentul *blur* are loc când obiectul de text nu mai are *focus*.

Elementele selectie din cadrul formularelor sunt definite cu tagul **SELECT**; iar opțiunile lor sunt enumerate folosind tagul **OPTION**. Acestea lucrează aproape ca și elementele de text. Ele generează evenimentele *focus*, *blur* și *change*. Paradoxal, evenimentele de selectie nu generează evenimentul *select*.

Actiuni care nu sunt evenimente

Nu toate acțiunile sunt considerate evenimente JavaScript. Derularea unei ferestre, citirea newsgroup-urilor, răspunsul la mail sunt acțiuni care însă nu sunt evenimente. Folosirea butoanelor **Back**, **Forward** sau **Home** din toolbar-ul browserului nu sunt evenimente JavaScript, însă ele devin evenimente JavaScript prin faptul că determină încărcarea unei pagini în timp ce pagina curentă va fi descărcată. Prin urmare, dacă o acțiune afectează pagina curentă, atunci ea este asociată cu unul sau mai multe evenimente JavaScript.

Scrierea de cod în JavaScript

Până acum am vorbit de limbajul JavaScript, însă nu stim cum se poate integra acesta într-un fișier HTML și cum se pot atasa funcțiile JavaScript de tratarea a evenimentelor de elementele HTML.

Tagul SCRIPT

Orice pagină Web este construită din instrucțiuni HTML care împart pagina în două părți: <HEAD> și <BODY>. Directivele HTML din secțiunea <HEAD> dau informații despre pagină, în timp ce în secțiunea <BODY> se realizează descrierea efectivă a paginii. Oricare dintre cele două secțiuni poate conține cod JavaScript. Este recomandat ca definirea funcțiilor de tratare a evenimentelor să se facă în secțiunea <HEAD>, înainte de definirea oricărui element din secțiunea <BODY>. [2]

Codul JavaScript este introdus de tagul **SCRIPT**. Tot ceea ce se găsește între acest tag și tagul corespunzător **/SCRIPT** se consideră a fi un script. Sintaxa acestui tag este următoarea.

<SCRIPT LANGUAGE="JavaScript" [SRC="URL"]>

Atributul SRC specifică URL-ul fișierului care conține cod JavaScript. Acest fișier trebuie să se termine cu “.js”. Dacă atributul SRC apare, atunci <SCRIPT> poate fi terminat direct cu </SCRIPT>, altfel între cele două taguri se așteaptă să existe cod.

Declararea handler-elor de tratare a evenimentelor JavaScript

Declararea handler-elor de tratare a evenimentelor arată exact ca și atributele HTML obișnuite. Orice atribut care începe cu particula “**on**” urmată de numele unui eveniment este o astfel de declarație. Astfel, *onClick* este atributul care va folosi pentru a declara handler-ul de tratare a evenimentului *click*. O declarație completă a unui handler de tratare a unui eveniment arată astfel:

onEvent="javascriptcode"

Folosirea handler-elor pentru tratarea evenimentelor JavaScript

Evenimentele JavaScript pot avea loc la orice nivel din structura arborescentă de nivelele ale unui document. În același timp, orice element din această structură arborescentă de nivele poate genera unul sau mai multe evenimente.

Handler-e pentru tratarea evenimentelor din nivelul DOCUMENT

În cadrul tagului BODY se realizează descrierea conținutului unei pagini HTML. Acest tag poate conține două declarații de handler-e de tratare a evenimentelor **Load** și **Unload**.

<BODY onLoad="loadfunc()" onUnload="unloadfunc()">

Atributul *onLoad="loadfunc()"* declară un handler JavaScript care va trata evenimentul **Load**. Acest eveniment este generat după ce tot conținutul paginii, adică tot ce se află între tagurile <BODY> și </BODY>, a fost citit, dar înainte de a fi afișat. Handler-ul pentru evenimentul **onLoad** poate fi folosit pentru a afișa un “SplashScreen” care să conțină numele companiei, produsul, informații de copyright; sau poate fi folosit pentru lansarea unei cutii de dialog conținând nume de utilizator și parolă pentru accesarea selectivă a unor pagini de Web.

Atributul *onUnload="unloadfunc()"* declară un handler JavaScript care va trata evenimentul invocat atunci când o pagină se descarcă. Acest lucru se întâmplă atunci când utilizatorul execută o acțiune care face să se treacă la o nouă pagină. Evenimentul **onUnload** are loc doar dacă trecerea la o nouă pagină se face în aceeași fereastră a browser-ului. Acest eveniment poate fi folosit, de exemplu, în cazul în care utilizatorul a completat un formular dar a uitat să apese butonul de “submit”, prin avertizarea lui și în funcție de dorința utilizatorului formularul poate fi trimis către server sau nu.

Mai există un handler de tratare a unui eveniment la nivel de document, deși acesta nu este asociat cu tagul BODY. Orice link HTML poate declara un handler de tratare a unui eveniment **mouseover**. Acest eveniment are loc când cursorul mouse-ului se află peste un text sau o imagine care este hyperlink către o altă pagină. Se pot realiza

astfel efecte vizuale sau realizarea unor acțiuni speciale înainte ca utilizatorul să poată accesa acel hiperlink.

Handler-ul pentru tratarea evenimentului SUBMIT dintr-un tag FORM

Tagul **FORM** este folosit pentru a defini începutul unui formular HTML. Include atributele **METHOD** pentru a specifica modalitatea de transmitere a formularului, **ACTION** pentru a specifica acțiunea care să aibă loc la transmiterea respectivului formular. Acest tag poate conține un singur atribut pentru handler-ul evenimentului **onSubmit**. Sintaxa pentru un tag **FORM** este următoarea:

```
<FORM NAME="formname" ... onSubmit="submitHandler()">
```

Handler-ul pentru tratarea evenimentului **onSubmit** este activat când conținutul formularului este pe cale de a fi transmis. Aceasta este o acțiune care se aplică întregului formular. Un efect echivalent îl are tratarea evenimentului **onClick** pentru butonul de “submit”. Utilitatea handler-ului de tratare a evenimentului **onSubmit** este cea de validare a conținutului formularului respectiv.

Handler-e pentru tratarea evenimentelor elementelor dintr-un formular

Aproape toate elementele unui formular au unul sau mai multe handler-e de tratare a evenimentelor. Tipul acestor handler-e pentru un anumit element depinde de tipul elementului respectiv. Butoanele pot genera evenimente **click**, elementele de text și selecție pot genera evenimente **focus**, **blur**, **select** și **change**. Confuzia care se poate face aici este cea că listele de selecție nu au evenimentul **select** deoarece ele nu oferă posibilitatea de a introduce text.

Există două excepții importante de la regula că toate elementele unui formular pot avea handler-e. Prima excepție se referă la elementele ascunse, cele de forma: **<INPUT TYPE="hidden">**. Atâta timp cât aceste elemente nu pot fi văzute utilizatorul nu poate interacționa cu ele, deci nu pot genera evenimente. A doua excepție se referă la elementele **OPTION** din cadrul listelor de selecție **SELECT**. Aceste elemente nu pot genera evenimente, orice eveniment care apare se referă la întreaga listă de selecție.

Evenimente *click* pentru butoane. Toate tipurile de butoane din cadrul unui formular HTML poate avea handler-ul de tratare a evenimentului **click** prin adăugarea unui atribut **onClick** în declararea lor cu tagul **<INPUT>**. Butoanele reactionează diferit la evenimentul **click**. Butoanele simple, de tipul “button”, “reset” sau “submit” transmit semnalul că au fost apăstate. Checkbox-urile si butoanele radio se selectează sau se deselectează. Un grup de butoane radio indică indicele butonului selectat.

O problemă curentă a formularelor HTML este cea a conflictelor între opțiuni. Utilizatorii sunt deseori puși să aleagă una din mai multe opțiuni. Unele combinații de opțiuni poate fi invalidă sau chiar dubioasă. În HTML-ul standard nu există nici o altă modalitate de validare a acestor opțiuni, decât cea de a transmite formularul către server unde se va face respectiva validare. Evenimentele JavaScript sunt ideale pentru astfel de validări.

Evenimentele pentru câmpurile de text și selecție. Câmpurile de text HTML declarate prin tagul **<INPUT>** cu atributul **TYPE=“text”** pot declara handler-e de tratarea a evenimentelor **focus**, **blur**, **change** sau **select**. Elementul HTML introdus prin tagul **TEXTAREA** poate avea, de asemenea, aceste evenimente. Listele de selecție create cu **<SELECT>** pot genera, de asemenea, toate aceste evenimente cu excepția evenimentului **select**.

Aceste evenimente pot fi folosite pentru a obține un control foarte strict asupra conținutului unui element de text sau a unei liste de selecție. Cele mai multe aplicații folosesc evenimentele **change** sau **blur** pentru a asigura un conținut valid a unui câmp de text.

IV. Obiecte JavaScript

Idea programării orientate-obiect nu este una nouă. Ea datează de acum 30 de ani. Astăzi, programarea orientată-obiect este considerată un concept stabil, care trebuie să facă parte din toate limbajele moderne de programare.

Într-o încercare de a defini programarea orientată-obiect cât mai simplu, putem zice că ea este un stil de programare în care concepte înrudite sunt grupate împreună. Astfel, dacă avem cinci date și trei funcții care lucrează cu aceste date, toate aceste date și funcții vor fi grupate împreună, formând un *obiect*.

Obiecte, proprietăți și metode în JavaScript

Obiectele conțin elemente înrudite, grupate într-un singur loc, ceea ce le face mult mai simple de utilizat sau cel puțin mult mai logice. În JavaScript elementele grupate într-un obiect se numesc *proprietăți*. Obiectele JavaScript conțin pe lângă date și funcții care lucrează cu aceste date. Aceste funcții se numesc *metode* ale unui obiect.

Definirea unui obiect: Instrucțiunea *new*

În JavaScript crearea unui obiect se face prin definirea unui tip special de funcție. Spre exemplu, dacă avem un obiect “casă”, crearea lui va fi următoarea:

```
function casa( cam, stl, ac, agr ) {      // definirea obiectului “casa”
    this.camere = cam;                  // numărul de camere (întreg)
    this.stil = stl;                    // stilul , e.g. Colonial, Tudor, Ranch (string)
    this.anconstr = ac;                 // anul acnstructiei, întreg
    this.aregaraj = agr;                // are garaj? (boolean)
}
```

Să observăm câteva lucruri. Primul lucru este că numele funcției este numele obiectului. Al doilea lucru pe care îl observăm este că funcția nu întoarce nimic. Explicatia acestui lucru ar fi faptul că crearea unui obiect se face prin modificarea lui *this*, deci nu trebuie returnat nimic.

O instanțiere a obiectului “casa” va avea forma:

```
var casamea = new casa( 10, "Colonial", 1989, true );
```

Instanta *casamea* va fi tratată ca orice altă variabilă. Ea trebuie declarată folosind *var*. Referirea la proprietățile acestui obiect se va face folosind operatorul punct (.).

Obiecte si tablouri

Un tablou este o colecție indexată de elemente de același tip.

În C sau Java: *int A[10]* reprezintă definirea unui tablou de 10 întregi. Referirea la întregii din acest tablou se face prin indexare de la *A[0]* la *A[9]*. Aceste două limbaje folosesc *zero-based indexing*, ceea ce înseamnă că primul element din tablou se află în locația 0 iar ultimul element are indicele numărul de elemente din tablou minus unu. Alte limbaje de programare au *one-based indexing*, ceea ce înseamnă că elementele sunt indexate de la 1 la numărul de elemente din tablou. JavaScript se încadrează în categoria limbajelor care folosesc *zero-based indexing*.

În JavaScript tablourile și obiectele sunt două fete ale aceluiași concept. Orice obiect este un tablou al valorilor proprietăților sale și orice tablou este un obiect. De exemplu, instanța *casamea* este un tablou care conține următoarele elemente:

```
casamea [0] = 10;           // camere  
casamea [1] = "Colonial";   // stil  
casamea [2] = 1989;         // anconstr  
casamea [3] = true;         // aregaraj
```

JavaScript propune si o a treia solutie hibrid între referirea cu punct (.) si modalitatea de indexare folosită la tablouri ([]). Proprietățile obiectelor pot fi referite nu numai ca si un tablou indexat ci si ca un tablou cu elemente cu nume. Acest tip de tablou este cunoscut sub numele de *associative array*.

```
casamea ["camere"] = 10;  
casamea ["stil"] = "Colonial";  
casamea ["anconstr"] = 1989;  
casamea ["aregaraj"] = true;
```

Extindere dinamică si tablouri cu lungime variabilă

Există posibilitatea de a *extinde dinamic* o instantă prin simpla adăugarea a unei proprietăți. Considerând o instantă a obiectului "casa":

```
var casamea = new casa( 10, "Colonial", 1989, true ),
```

vom putea adăuga, spre exemplu, proprietatea *aregradina* prin simpla atribuire a unei valori:

```
casamea.aregradina = true;
```

Prin această instructiune s-a adăugat o nouă proprietate la sfârșitul tabloului *casamea*. Deci, proprietatea *aregradina* va fi elementul de indice 4 al tabloului: *casamea[4]*. Extensia dinamică se aplică doar instantei. Alte instante ale aceleiasi clase nu vor fi afectate si nici obiectul în sine nu va fi modificat.

O situatie în care extensia dinamică este foarte folositoare este cea a tablourilor cu lungime variabilă. Datorită faptului că proprietățile obiectelor sunt considerate elemente ale unui tablou care pot fi referite prin indexare numerică, este usor să se scrie functii care să construiască tablouri de lungime si continut variabil. Spre exemplu, definirea unui tablou de stringuri se va face astfel:

```
function StringArr( cate, initstr) { // "câte" stringuri
    this.length = howmany;
    for( var i = 1; i <= cate; i++ ) {
        this[i] = initstr;          // valoarea inițială este "initstr"
    }
}
```

Dacă vom apela funcția astfel:

```
mystringarr = new stringarr( 100, "ceva" );
```

se va crea o instanță cu *101* proprietăți. Prima proprietate, de index *0*, va fi proprietatea *length* a tabloului. Următoarele *100* proprietăți, cu indici de la *1* la *100*, vor fi initializate cu valoarea "ceva".

Dacă se va dori adăugarea unor noi elemente la acest tablou, vom avea:

```
mystringarr[101] = "altceva";
mystringarr[102] = "este";
mystringarr.length += 2;
```

Ultima instrucțiune este cea prin care se actualizează lungimea tabloului.

Instrucțiunea *for...in*

În *capitolul 2* s-a prezentat instrucțiunea **for**. Există o altă variantă a acestei instrucțiuni care poate fi folosită pentru a executa acțiuni iterative asupra proprietăților unui obiect. Această instrucțiune are următoarea formă:

```
for ( varname in objname ) {
    forbody
```

```
}
```

Variabila *varname* va lua pe rând numele proprietăților obiectului *objname*. Un exemplu al folosirii acestei instrucțiuni ar fi acela al afisării proprietăților unui obiect oarecare:

```
function showany(anyobj) {    // afisează proprietățile unei instante sau obiect
    for( var iter in anyobj ){ // ia pe rând toate proprietățile
        document.write("<BR>Proprietatea " + iter + " este " + anyobj[iter]);
    }
    document.write("<BR>");
}
```

Metode

Unul dintre aspectele cele mai puternice ale programării orientate-obiect în JavaScript este acela al posibilității creării unor obiecte cu proprietăți functionale. Aceste proprietăți functionale se numesc **metode**.

Am văzut cum este folosit cuvântul rezervat *this* în crearea unui obiect. El este folosit și în referirea metodelor la *obiectul curent*. Un exemplu de utilizare a acestuia este următorul:

```
/*
    Această funcție crează un obiect "casa" cu metoda "arata"
*/
function casa(cam, stl, ac, agr) {
    this.length = 5;        // patru proprietăți și proprietatea length
    this.camere = cam;      // camere; proprietatea [1]
    this.stil = stl;        // stil; proprietatea [2]
    this.anconstr = ac;     // anul construcției; proprietatea [3]
    this.aregaraj = agr;    // are garaj?; proprietatea [4]
```

```

        this.arata = aratacasa;      // metoda aratacasa; proprietatea [5]
    }
    /*
    Această funcție este metoda "arata" a obiectului "casa"
    */
    function aratacasa() {           // observatie: fără argumente!
        var npropr = this.length;    // lungimea tabloului de proprietăți nu
        include "arata"
        for ( var iter = 1; iter < npropr; iter++) { //iteratia
            document.write("<BR>Proprietatea "+iter+"este"+this[iter]);
        }
        document.write("<BR>");
    }

```

Metoda arata va fi apelată astfel:

```
casamea.arata();
```

si va avea acelasi efect ca si o simplă apelare a functiei care definește metoda:

```
aratacasa( casamea );
```

Obiecte încuibate

Proprietățile obiectelor sunt cantități fără tip. Ele pot fi variabile obișnuite de orice tip implicit. Putem considera că fiecare obiect reprezintă un nou tip de date si fiecare instanță o nouă variabilă. Bazându-ne pe faptul că JavaScript nu are tipuri de date, putem spune că un obiect poate contine alte obiecte? Răspunsul este DA.

Vom crea un nou obiect care se numește *desc* si care are proprietățile *length*, *lățime*, *înălțime*, *culoare* si o metodă care să calculeze volumul.

```

/*
    Lungimea, lățimea si înălțimea vor fi date în metri. Culoarea va fi un string.
*/
function desc( ln, la, ina, col) {    // descrierea unui obiect
    this.length = 5;        // patru proprietăți si proprietatea length
    this.lung = ln;         // lungimea; proprietatea [1]
    this.lat = la;          // lățimea; proprietatea [2]
    this.inalt = ina;       // înălțimea; proprietatea [3]
    this.culoare = col;     // culoarea; proprietatea [4]
    this.volum = volum;    // metode de calculare a volumului
}
/*

```

Metoda de calcularea a volumului.

```

*/
function volum ( ) {
    var mylun, mylat, myina;

    mylun = this.lung;
    mylat = this.lat;
    myina = this.inalt;
    return( mylun * mylat * myina );    // returnez volumul
}

```

Vom putea acum adăuga obiectul nou creat ca proprietate a obiectului *casa*.

```

/*
    Această funcție crează un obiect casa cu o metoda "arata" si un subobiect "desc"
*/
function casa( cam, stl, ac, arg, descri ) {

```

```
this.length = 5;           // patru proprietăți si proprietatea length  
this.camere = cam;         // camere; proprietatea [1]  
this.stil = stil;          // stil; proprietatea [2]  
this.anconstr = ac;        // anul constructiei; proprietatea [3]  
this.aregaraj = arg;       // are garaj?; proprietatea [4]  
this.descr = descri;       // instantă de descriere; proprietatea [5]  
this.arata = aratacasa;    // metoda aratacasa; proprietatea [6]  
  
}
```

Instructiunea **with**

Scopul acestei instructiuni este acela de a permite să se facă mai multe referiri la un acelasi obiect (sau instantă) fără ca numele obiectului să fie repetat. Formatul instructiunii este:

```
with ( objname ) {  
    statements  
}
```

unde *objname* este numele obiectului sau instantei respective. În interiorul blocului de instructiuni **with** referirea la proprietățile obiectului *objname* vor fi interpretate ca si cum ar fi prefixate cu *objname* urmat de operatorul punct (.).

Functii cu număr variabil de argumente

Am văzut că funcțiile sunt folosite pentru a construi obiecte. Obiectele, tabourile indexate si tablourile asociative sunt cam acelasi lucru. Asemănarea dintre aceste concepte merge mai departe. Funcțiile JavaScript au ele însele proprietăți. Toate funcțiile JavaScript au cel puțin următoarele două proprietăți:

- **caller**
- **arguments**

Proprietatea **caller** reprezintă numele cu care se apelează funcția. Proprietatea **arguments** este un tablou care conține toate argumentele care nu sunt în lista de argumente ale funcției. Această a doua proprietate ne permite să scriem funcții care au un număr variabil de argumente. Argumentele din lista de argumente ale unei funcții sunt obligatorii, în timp ce argumentele din proprietatea **arguments** sunt optionale.

```

/*
....Demonstrarea caracterului obligatoriu si optional al argumentelor unei functii.
*/
function addem( str1 ) {           // un argument obligatoriu
    var nopt = addem.arguments.length; // numărul argumentelor
    var sum = 0;                   // suma argumentelor optionale
    var strres;                    // stringul rezultat
    for( var i = 1; i < nopt; i++ ) { // iterarea argumentelor optionale
        sum += addem.arguments[i]; // însumarea lor
    }
    strres = "Salut " + str1 + ", suma este " + sum;
    return(strres);
}

```

Pentru a vedea cum funcționează presupunem că această funcție este apelată astfel:

```
var str = addem( "vouă", 1, 3, 5, 7 );
```

Argumentul obligatoriu "vouă" se va asigura parametrului *str1* al funcției *addem*. Lista completă de argumente este asignată tabloului de lungime variabilă *addem.arguments*. Acest tablou are o proprietate *length* = 5 care indică câte argumente are funcția: unul obligatoriu și patru optionale. Valoarea returnată va fi stringul:

"Salut vouă, suma este 16."

Un exemplu în care atât argumentul obligatoriu *str1* și argumentele optionale fac parte din lista de argumente este următorul:

```
var str = addem( "nouă" );
```

care va returna valoarea: *"Salut nouă, suma este 0."*

Obiecte predefinite

Obiectele JavaScript pot fi împărțite în următoarele trei categorii:

- **obiecte predefinite**
- **obiecte HTML**
- **obiectele browserului**

Obiectele predefinite includ obiectele **string**, **Date** și **Math**. Sunt definite ca fiind obiecte predefinite deoarece nu au nici o legătură cu paginile Web.

Obiectele HTML, însă, sunt asociate direct cu elementele unei pagini Web. Orice **link**, **anchor**, **formular** sau **element al unui formular** este obiect HTML. Organizarea ierarhică a elementelor care apar într-o pagină Web se reflectă aproape identic în ierarhia obiectelor încuibate HTML.

Obiectele browserului se află în vârful ierarhiei obiectelor JavaScript. Obiectele din această categorie se referă la **window** (fereastra curentă), **history** (lista paginilor vizitate) și **location** (URL-ul paginii curente).

Obiecte *string*

Orice variabilă care este *string* este un obiect **string**. Obiectele *string* au o proprietate, *length*, și mai multe metode. Această proprietate conține lungimea sirului respectiv. Metodele stringurilor se împart în trei categorii: care manipulează conținutul

stringurilor, care manipulează formatul stringurilor și metode care convertesc stringurile în elemente HTML.

Metode care se referă la conținutul sirurilor.

charAt(idx)	returnează caracterul care se află la indicele “idx” în sir;
indexOf(chr)	returnează primul indice la care se află caracterul “chr” în sir;
lastIndexOf(chr)	returnează ultimul indice la care se află caracterul “chr” în sir;
substring(fromidx, toidx)	returnează subsirul cuprins între indicii “fromidx” și “toidx”;
toLowerCase()	convertesc conținutul sirului la un sir care conține doar litere mici;
toUpperCase()	convertesc conținutul sirului la un sir care conține doar litere mari;

Metode care se referă la formatul sirurilor.

big()	scriu conținutul sirului cu litere mai mari decât cele obișnuite;
blink()	scriu conținutul sirului cu litere care pâlpâie;
bold()	scriu conținutul sirului cu litere îngrosate;
fixed()	scrie sirului cu litere care au aceeași lățime (font fixat);
fontcolor(colr)	scrie sirului cu culoarea dată de “colr”;
fontsize(sz)	scrie sirului cu litere de dimensiunea indicată de “sz”;
italics()	scrie sirului cu litere înclinate;
small()	scrie sirului cu litere mai mici decât cele obișnuite;
strike()	scrie sirului cu litere tăiate la mijloc;
sub()	subscrie sirului;
sup()	superscrie sirului;

Metode HTML.

anchor(namestr)	crează un “anchor” HTML care are numele “namestr”;
link(hrefstr)	crează un “hyperlink” care va duce spre “hrefstr”;

Obiectul **Math**

Obiectul **Math** este folosit pentru calcule matematice. Este un *obiect static*, adică nu se schimbă. Spre deosebire de obiectele **string**, care au instance dar nu au un obiect explicit, obiectul **Math** se are doar pe el însuși neavând nici o instanță.

Obiectul **Math** are următoarele proprietăți:

E	constanta lui Napier, aproximativ 2.71828;
LN10	logaritm natural din 10, aproximativ 2.30259;
LN2	logaritm natural din 2, aproximativ 0.69315;
PI	numărul PI, aproximativ 3.141592653589793;
SQRT1_2	rădăcină pătrată din 1/2, aproximativ 0.7071;
SQRT2	rădăcină pătrată din 2, aproximativ 1.4142;

Toate metode obiectului **Math** lucrează cu numere în virgulă flotantă. Acestea sunt:

abs(num)	modulul numărului "num", $\text{num} \in [-1, 1]$;
acos(num)	arccos din numărul "num", $\text{num} \in [-1, 1]$;
asin(num)	arcsin din numărul "num", $\text{num} \in [-1, 1]$;
atan(num)	arctan din numărul "num", $\text{num} \in [-1, 1]$;
ceil(num)	cel mai mic întreg mai mare sau egal cu numărul "num";
cos(ang)	cos din numărul "num", $\text{num} \in [-1, 1]$;
exp(num)	E la puterea dată de numărul "num";
floor(num)	cel mai mare întreg mai mic sau egal cu numărul "num";
log(num)	logaritm din numărul "num";
max(num1, num2)	maximul dintre numerele "num1" și "num2";
min(num1, num2)	minimul dintre numerele "num1" și "num2";
pow(num1, num2)	ridică "num1" la puterea "num2";
random()	generează un număr aleator între 0 și 1;
round(num)	cel mai apropiat întreg de numărul "num";
sin(ang)	sin din numărul "num", $\text{num} \in [-1, 1]$;
sqrt(num)	rădăcina pătrată a numărului "num", 0 când $\text{num} < 0$;
tan(ang)	tan din numărul "num", $\text{num} \in [-1, 1]$;

Obiectul *Date*

Obiectul **Date** simplifică și automatizează o serie de conversii dintre reprezentarea folosită de oameni a datei și cea internă. Obiectul JavaScript **Date** urmează standardul UNIX de memorare a datei și a orei, ca număr de milisecunde din 1 Ianuarie 1970. Acest obiect nu are proprietăți, însă are multe metode. Înainte de a folosi obiectul **Date** trebuie, mai întâi să înțelegem cum se poate instanția. Există trei metode de bază pentru crearea unei instanțe **Date**:

new Date()	crează o instanță cu data și ora curentă;
new Date(datestring)	converteste sirul "datestring" într-o instanță <i>Date</i> ;
new Date(yr, mon, day)	transformă cei trei întregi: "yr", "mon" și "day" în instanță <i>Date</i> ;

Metodele obiectului **Date** sunt următoarele:

getDate()	returnează un număr între 1 și 31 corespunzător zilei din lună;
getDay()	returnează un număr între 0 și 6 corespunzător zilei din săptămână;
getHours()	returnează ora, 0 când instanța la care se aplică metoda nu are oră;
getMinutes()	returnează minutul, 0 când instanța la care se aplică metoda nu are minute;
getMonth()	returnează un număr între 0 și 11 corespunzător lunii;
getSeconds()	returnează secunda, 0 când instanța la care se aplică metoda nu are secunde;
getTime()	returnează numărul de milisecunde din 1 Ianuarie 1970;
getTimeZoneOffset()	returnează deplasamentul de timp local;
getYear()	returnează anul;
setDate()	setează data;
setHours()	setează ora;
setMinutes()	setează minutele;
setMonth()	setează luna;
setSeconds()	setează secunde;
setTime()	setează timpul;
setYear()	setează anul;

Pe lângă metodele **set** și **get**, obiectul **Date** are și metode de conversie ale unei instanțe **Date** în string și două metode statice de trecere de la stringuri la dată.

toGMTString()	string reprezentând data si ora relativă la GMT (Greenwich Mean Time);
toLocaleString()	string reprezentând data si ora relativă la conventia curentă;
toString()	converteste instanta <i>Date</i> în string;
parse(datestr)	converteste stringul "datestr" în timpul local;
UTC(datestr)	converteste stringul "datestr" în timpul UTC (Universal Coordinated Time);

Functii predefinite

Există un set de functii JavaScript care nu sunt metode si nu se apelează niciodată folosind operatorul punct (.). Acestea sunt:

escape(str)	converteste caracterele speciale din "str" în formă explicită;
eval(str)	evaluatează expresia conținută de stringul "str";
parseFloat(str)	returnează numărul în virgulă flotantă care se află în "str";
parseInt(str, radix)	returnează întregul scris în baza "radix" care se află în "str";
unescape(str)	converteste caracterele speciale din "str" în formă implicită;

Obiecte browser si HTML

Modelul obiectual JavaScript, alcătuit dintr-o multime de obiecte predefinite, metode si functii, ne oferă tot ceea ce așteptăm de la un limbaj de programare modern. Ne oferă structuri de control, încapsulare, operatii matematice, samd. Datorită faptului că JavaScript a fost conceput pentru a functiona în World Wide Web trebuie să existe o legătură între el si conținutul unei pagini HTML. Această legătură este oferită de o multime mare de obiecte ale browser-ului si obiecte HTML.

Obiecte browser

Principalele obiecte browser sunt:

- **window**
- **document**
- **location**
- **history**

Obiectul window

Obiectul **window** se află în vârful ierarhiei obiectelor JavaScript. Orice fereastră a browserului care este deschisă corespunde unui obiect **window**. Toate celelalte obiecte sunt copii ale obiectelor **window**. Deci, fiecare fereastră este asociată unei pagini Web iar structura HTML a acestei pagini se reflectă în obiectul **document** al ferestrei. Fiecare fereastră corespunde unui URL iar URL-ul se reflectă în obiectul **location**. Fiecare fereastră are o listă a paginilor care au fost deschise în respectiva fereastră, ceea ce se regăsește prin intermediul proprietăților obiectului **history**.

Obiectele **window** au următoarele metode:

alert(msgstr)	deschide o fereastră de avertizare cu mesajul “msgstr”;
close()	închide fereastra curentă;
confirm(msgstr)	deschide o fereastră de confirmare cu mesajul “msgstr”;
open(urlstr, wname)	deschide o fereastră cu numele “wname” si URL “urlstr”;
prompt(msgstr)	deschide o fereastră de input cu mesajul “msgstr”;

Obiectul document

Fiecare fereastră este asociată unui obiect **document**. Obiectul **document** are proprietăți pentru fiecare **anchor**, **link**, **formular** sau **element al formularelor** care poate apărea într-o pagină Web. Are, de asemenea, proprietăți pentru titlu, pentru culoarea de foreground, pentru culoarea de background, pentru culorile link-urilor si pentru alte atribute ale unei pagini. Metodele obiectului **document** sunt următoarele:

clear()	sterge o fereastră document;
close()	închide buferul de iesire (output);
open()	deschide buferul de iesire (output);
write(str)	afisează într-o pagină conținutul stringului “str”;
writeln(str)	afisează într-o pagină conținutul stringului “str” si trece la rând nou;

Obiectele history si location

Obiectul **history** este folosit pentru a face referire la lista URL-urilor vizitate anterior. Acest obiect are proprietatea *length* care indică câte URL-uri cuprinde lista si următoarele trei metode:

back()	afisează pagina anterioară celei curente în lista de URL-uri;
forward()	afisează pagina următoare celei curente în lista de URL-uri;
go(where)	afisează pagina indicată de “where” în lista de URL-uri;

Obiectul **location** descrie URL-ul unui document. Printre proprietățile lui se regăsesc și cele care reprezintă componentele unui URL, cum ar fi *protocolul*, *pathname-ul* și *portul*. Are și metoda **toString()** care convertește obiectul **location** la un string.

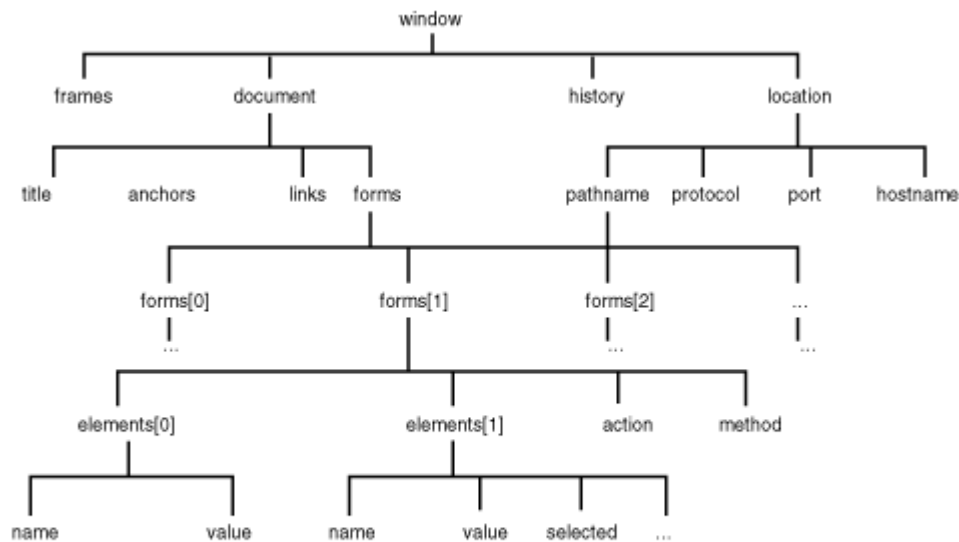


Fig 4.1 Obiectele JavaScript browser și HTML referă toate elementele unei pagini Web

Obiecte HTML

Elementele HTML se reflectă în ierarhia de obiecte JavaScript, de aceea ele pot fi accesate folosind următoarele proprietăți:

- **anchors**
- **forms**
- **links**

Aceste proprietăți ale obiectului **document** sunt tablouri care contin fiecare element HTML care este *anchor*, *formular* sau *link* într-o pagină. Indexarea acestor tablouri se face în ordinea în care respectiv elemente HTML apar în pagină.

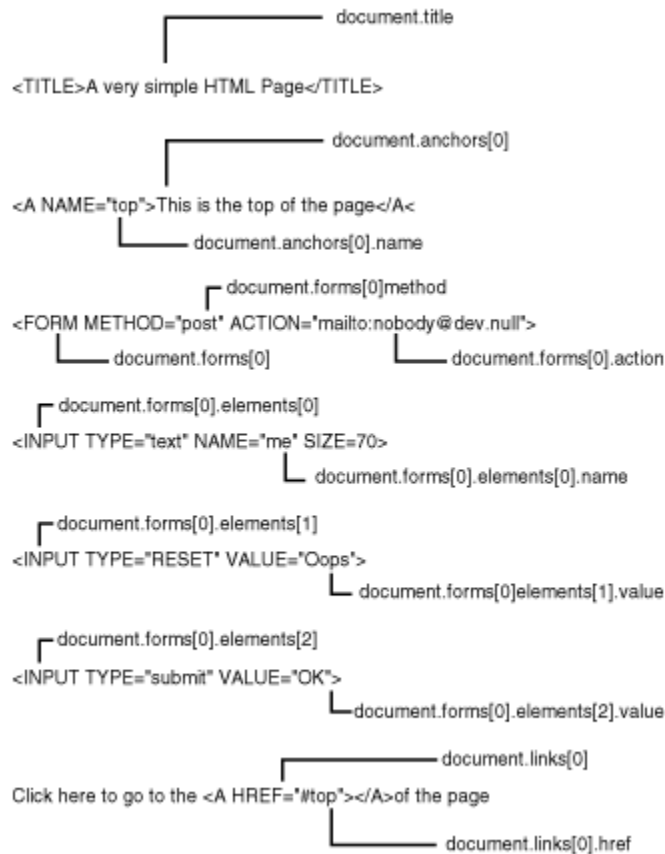


Fig 4.2 Anchor-urile, link-urile, formularele si elementele formulelor sunt obiecte JavaScript

V. JavaScript pe server

Până la acest punct al lucrării am vorbit doar de capacitățile JavaScript de a dezvolta partea de client. Acest capitol arăta cum poate JavaScript deveni parte integrantă dintr-un server.

JavaScript poate extinde posibilitățile unui server. Acest lucru se poate realiza folosind mediul Netscape numit *LiveWire*. Acesta nu este un server, însă lucrează cu serverele Netscape pentru a crea aplicații care să facă paginile Web să arate mai vii.

Mediul LiveWire

LiveWire este un mediu vizual, integrat, utilizat pentru crearea aplicațiilor client-server. Aceste aplicații pot fi folosite pentru a lărgi sau a restrânge accesul unor anumite persoane într-o rețea de tip intranet.

LiveWire este un ansamblu care constă din:

- *Netscape Navigator Gold*;
- *LiveWire Site Manager*;
- *LiveWire Server Extensions*;
- *LiveWire Server Front Panel*;
- *JavaScript*.

LiveWire rulează pe sistemele de operare UNIX sau Windows NT.

Obiecte și proprietăți “Live”

LiveWire are patru obiecte predefinite: **request**, **client**, **roject** și **server**.

Aceste obiecte sunt înrudite astfel: fiecare **server** poate rula mai multe **proiecte**; orice **proiect** poate avea mai mulți **clienți** care să-l folosească simultan; majoritatea **clienților** cer mai multe pagini (**request**).

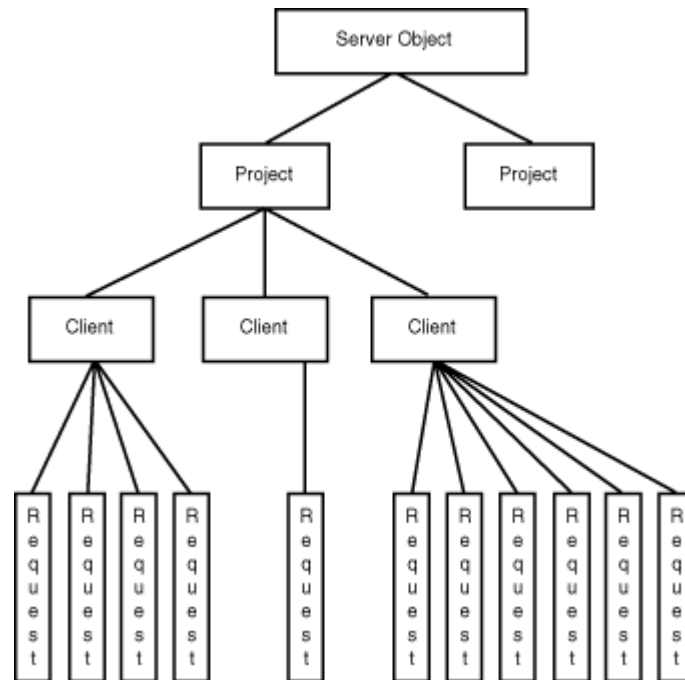


Fig 5.1 Descrierea ierarhică a obiectelor: **server**, **project**, **client** si **request**

Obiectul *request*

De fiecare dată când un browser vrea mai multe informații trimite o cerere (**request**) serverului.

Obiectul **request** are patru **proprietăți**:

agent	linie de text care conține tipul browser-ului clientului;
ip	ip-ul clientului;
method	metoda HTTP asociată cererii (<i>get</i> , <i>post</i> sau <i>head</i>);
protocol	text standard care indică versiunea HTTP a clientului (poate fi HTTP/1.0).

Alte proprietăți ale obiectului request: O cerere (**request**) poate include și alte informații. Cererea unui browser poate fi făcută în urma unui submit al unui formular HTML. Informațiile din formular devin proprietăți **request**. În proprietățile obiectului **request** se pot stoca informații, însă aceste informații au deseori o viață scurtă.

Obiectul *client*

De fiecare dată când un client accesează aplicația server se crează un nou obiect **client**. Nu există proprietăți predefinite ale acestui obiect. Dacă se dorește o anumită informație despre client, atunci se va crea o proprietate relativă la acea informație.

Proprietăți: O proprietate foarte folosită a clientului este **IN**(identification number). Se poate realiza o "abonare" la server prin completarea unui formular de către fiecare nou client, în urma căreia fiecărui client i se asociază un **IN**.

Expirarea unui obiect client: Datorită faptului că obiectele client sunt create automat trebuie să existe un mecanism de stergere a acestora.

LiveWire sterge automat toți clienții care nu au nici o proprietate. Timpul implicit de expirare este de 10 minute de inactivitate. Dacă un client nu trimite o cerere timp de 10 minute de la cererea anterioară, obiectul expiră. Acest timp de expirare se poate seta folosind sintaxa: *client.expiration(seconds)*

Un alt mod de control manual al expirării obiectelor **client** este *destroy*. Dacă nu se mai dorește folosirea unui anumit obiect **client**, atunci se poate folosi instrucțiunea: *client.destroy()*

Stocarea informațiilor dintre sesiuni în cookie-uri: O altă modalitate de a reține informații despre client este folosirea cookie-urilor. Browser-ul trebuie să suporte protocolul Netscape *cookie*. Dacă îl suportă, serverul trimite clientului o informație sub forma unei perechi: nume/valoare.

Obiectul *project*

Fiecare aplicație care porneste crează un obiect **project**. Acesta este o dată globală pentru întreaga aplicație. Fiecare client care accesează o aceeași aplicație împart un obiect **project**.

Proprietăți: Nu are proprietăți predefinite. Dacă se dorește reținerea unor informații cu privire la proiect, atunci se vor crea proprietăți corespunzătoare.

Blocarea obiectelor project: Orice mediu multi-user trebuie să aibă cazuri de acces simultan. Un fișier de pe server trebuie blocat atât timp cât este folosit. Deblocarea fișierului se face când nu se mai lucrează cu el. ceilalți trebuie să aștepte până ce fișierul este liber. Fără blocare datele pot fi corupte.

Un obiect project se blochează astfel: *project.lock();*

Deblocarea se face prin: *project.unlock();*

Obiectul *server*

Datele globale pentru întregul server se află în obiecte **server**. Aceste obiecte pot fi împărțite între aplicații.

Proprietățile acestui obiect sunt:

agent	linie de text care conține numele și versiunea serverului. (ex: Netscape-Commerce/1.12)
protocol	text standard care indică versiunea HTTP utilizată de client (ex: HTTP/1.0)

Adăugarea proprietăților: Ca în cazul majorității obiectelor JavaScript, obiectului **server** îi pot fi adăugate noi proprietăți. Spre exemplu, dacă se dorește adăugarea orei când serverul a fost ultima dată accesat vom avea următoarea succesiune de instrucțiuni:

```
today = new Date()
server.accesstime = today.getTime()
```

Pentru a fi folosit, un obiect **server**, asemenea unuia **project**, trebuie blocat. Se va folosi aceeași procedură de blocare ca și în cazul obiectului **project**.

Comunicarea externă a proceselor: JavaScript si CGI

Scripturile **CGI** (Common Gateway Interface) sunt adesea folosite în scrierea paginilor Web. Cu ajutorul acestora se pot crea countere, e-mailere si multe alte lucruri.

LiveWire poate înlocui programarea CGI. În locul apelării unor programe externe, softul serverului poate rula aplicatii care să fie integrate în el.

Aplicatiile sunt dezvoltate folosind trei utilitare. Aplicatiile se vor construi utilizând **LiveWire's Site Manager**. Sursele fișierelor pentru aplicatii sunt scrise folosind acelasi utilitar cu care se pot construi pagini JavaScript. Limbajul în care sunt scrise aceste aplicatii este **JavaScript**.

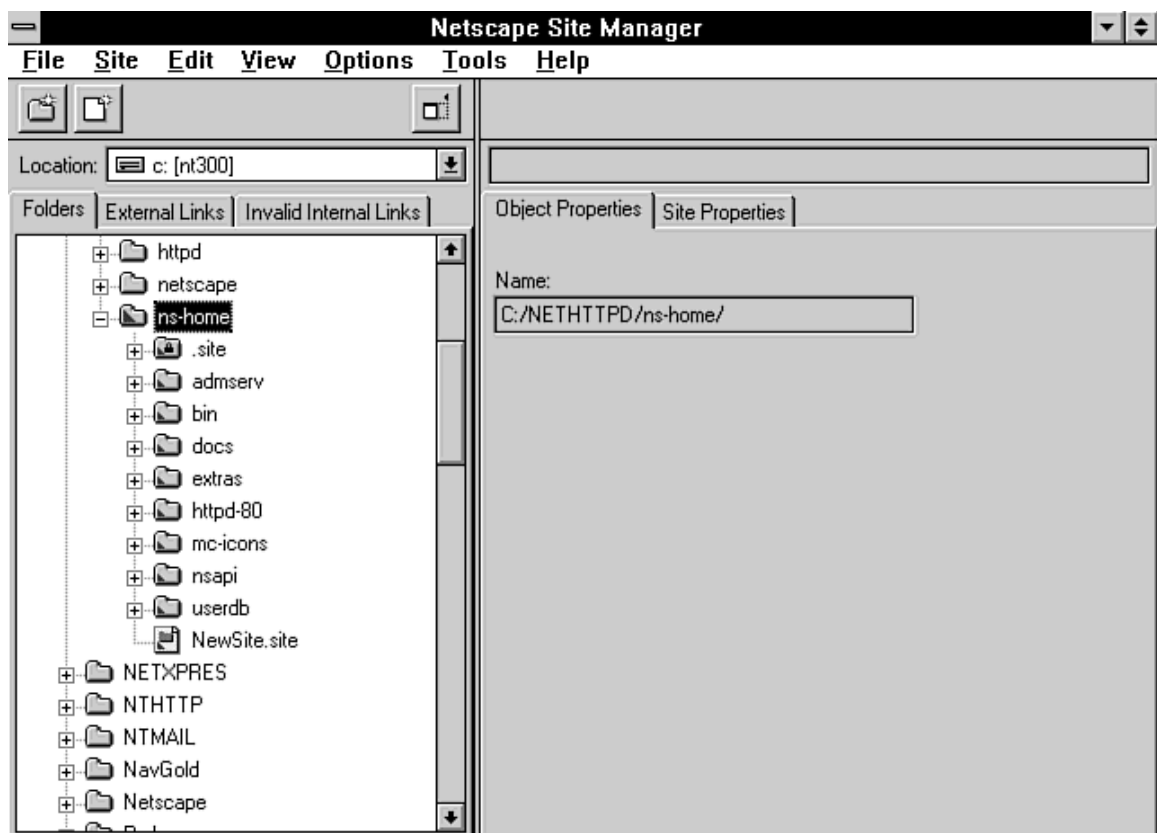


Fig 5.2 Netscape Site Manager

Pasi în construirea unei aplicații LiveWire

1. Crearea fisierului sursă.
2. Utilizând interfața grafică a **Site Manager**-ului, fisierele trebuie aduse sub gestiunea paginii.
3. Crearea aplicației prin compilarea fisierului în *.web.
4. Instalarea aplicației folosind **LiveWire Application Manager**.
5. Dacă se reconstruiește aplicația trebuie repornită folosind **Application Manager**.
6. Rularea aplicației prin încărcarea oricărei pagini cu browser-ul.



Fig 5.3 LiveWire Application Manager

Navigarea într-o aplicație LiveWire

Pentru accesarea unei aplicații LiveWire este nevoie ca browser-ul să încarce o pagină Web. Browser-ul poate încărca orice pagină din cadrul aplicației. Serverul vede această cerere ca pe oricare alta dar o tratează diferit.

URL-ul are următoarea formă:

<http://server.domain/application/page.html>

În acest caz:

- *domain* este domeniul Internet;
- *server* este numele serverului HTTP;
- *application* este numele definit la instalarea aplicației cu **Application Manager**;
- *page.html* este numele oricărei pagini din aplicație.

Fiecare aplicație are o pagină implicită. Dacă *page.html* lipsește, se va încărca aceasta pagină.

Fisierele sursă LiveWire

Pentru a construi o aplicație LiveWire trebuie scrise unul sau mai multe fișiere sursă. Se pot construi următoarele trei tipuri de fișiere sursă:

- *standard HTML*: fișiere cu extensia .htm sau .html;
- *HTML cu JavaScript*: codul JavaScript este plasat între tagurile SERVER sau apostroafe. Tot fișiere .htm sau .html;
- *funcții JavaScript*: fișiere cu extensia .js care conțin numai cod JavaScript.

Utilizarea tagului SERVER

Tagul **SERVER** conține cod JavaScript care execută o instrucțiune sau produce HTML cu ajutorul funcției *write*. Când se folosește acest tag rezultatul este trimis ca răspuns la cerere.

Codul sursă care conține tagul **SERVER** se află în siguranță, pe server. Acest lucru diferă față de paginile HTML unde codul sursă poate fi văzut cu browser-ul.

```
<P>Cererea a venit de la adresa IP:  
<SERVER> write(request.ip) </SERVER>
```

Utilizarea apostroafelor

O metodă mai scurtă de a pune cod JavaScript într-un document HTML este utilizarea apostrofului ('). Utilizând apostroful, se generează automat HTML fără să se mai folosească funcția *write*.

```
<P> Cererea a venit de la adresa IP: 'request.ip' </P>
```

Schimbul de informații dintre Browser și Server

Între server și browser trebuie să existe un schimb de informații. Clientul (*browserul*) trimite răspunsul user-ului către server iar serverul trimite date înapoi către browser.

Răspunsul user-ului este trimis serverului analog modului în care sunt trimise datele dintr-un formular. Odată aceste date ajunse la server acesta le pune în obiectul **request**.

De obicei serverul întoarce, ca răspuns a cererii browserului, o pagină statică. În LiveWire răspunsul este tot o pagină, însă aceasta are un conținut variat.

Fisiere externe și baze de date cu JavaScript

În scrierea unor aplicații este nevoie de scrierea și citirea unor date din fișiere. *LiveWire* oferă obiectul **file** care permite aplicațiilor să scrie în sistemul de fișiere al serverului. Ca o măsură de securitate, JavaScript la nivel de browser nu permite salvarea datelor în sistemul de fișiere.

LiveWire Pro suportă baze de date SQL (Structured Query Language), Informix, Oracle, Sybase si Microsoft.

Crearea unui obiect *File*

Operarea cu fisiere în JavaScript se face prin intermediul obiectelor. Pentru a folosi un fisier trebuie creat un obiect **file**.

```
fileObjectName = new File("path")
```

Parametrul *"path"* reprezintă calea spre fisier relativă la directorul aplicației.

Deschiderea unui fisier

După crearea obiectului **file** trebuie ca fisierul să fie deschis. Deschiderea unui fisier se face folosind metoda **open**.

```
result=fileObjectName.open ("mode")
```

Variabila *result* este **true** dacă fisierul s-a deschis cu succes, altfel este **false**. Dacă fisierul este deja deschis operația de redeschidere a lui va eșua. Obiectul **file** original va rămâne deschis și nemodificat. Deoarece unei aplicații JavaScript îi pot fi trimise două cereri simultane, trebuie luată în considerare această situație.

Un fisier poate fi deschis pentru citire și/sau pentru scriere. Modurile în care se poate deschide un fisier sunt:

r	citire, dacă fisierul nu există returnează <i>false</i> .
w	scriere, dacă fisierul nu există se va crea un fisier nou.
a	adăugare (scrie la sfârșitul fisierului), dacă fisierul nu există se va crea un fisier nou.
r+	citire și scriere; scrierea și citirea se va face la începutul fisierului.
w+	citire și scriere, dacă fisierul nu există se va crea un fisier nou.
a+	scrierea și citirea se va face la sfârșitul fisierului, dacă el nu există se va crea un fisier nou.

Toate aceste metode se referă la fișiere text. Prin adăugarea unui **b** la oricare dintre modelele mai sus amintite, **LiveWire** oferă posibilitatea de salvare a fișierelor în format *binar*.

Pentru afișarea numelui fișierului se poate folosi metoda **write** în formatul următor:

write (fileObjectName)

Pozitionarea în cadrul unui fișier

Pentru lucrul cu date păstrate într-un fișier este nevoie de posibilitatea de a putea accesa date din orice parte a fișierului. Obiectul **file** permite citirea poziției curente, schimbarea poziției curente sau verificarea dacă ne aflăm la sfârșitul fișierului.

La deschiderea unui fișier, poziția curentă depinde de modul în care a fost deschis fișierul respectiv. Poate fi la începutul sau sfârșitul fișierului.

Poziția curentă poate fi modificată folosind metoda **getPosition**. Primul octet din fișier este 0 și orice poziție este un număr pozitiv. Eroarea este indicată prin returnarea numărului *-1*. Sintaxa metodei este:

x = fileObj.getPosition()

Metoda **setPosition** este folosită pentru a schimba poziția curentă. Poziția poate fi schimbată relativ la începutul fișierului, la sfârșitul acestuia sau la poziția curentă. Aceasta se numește *referință* și reprezintă un parametrul optional la metodei. *Referința* implicită este cea relativă la începutul fișierului. Sintaxa acestei metode este:

fileObj.setPosition(position [,reference])

unde este o valoare numerică- **0** relativ la începutul fișierului, **1** relativ la poziția curentă și **2** relativ la sfârșitul fișierului. Metoda returnează **true** în caz de succes și **false** altfel.

Există situații în care este nevoie să testăm dacă ne aflăm la sfârșitul fisierului. Acest lucru se realizează folosind metoda **eof**. Această metodă returnează **true** după prima metodă care încearcă să citească după sfârșitul fisierului.

fileObj.eof()

Scrierea datelor în fișiere

Obiectul **file** oferă trei metode de scriere a datelor în fișiere. Aceste metode permit scrierea de stringuri sau scrierea a câte unui singur octet. Fiecare dintre aceste metode returnează **true** dacă operația s-a încheiat cu succes sau **false** în caz contrar. Sintaxa lor este:

fileObj.write(string)

fileObj.writeln(string)

fileObj.writeByte(number)

Pentru a crește eficiența trimiterii datelor către fișier, acestea sunt stocate mai întâi într-un bufer. Acest bufer intern stochează date până ce se umple, până ce fișierul se închide sau până ce buferul este golit. Golirea buferului înseamnă că programul forțează datele din bufer să fie scrise în fișier.

Pentru a ne asigura că datele au fost salvate în fișier se poate forța golirea buferului prin metoda **flush**:

fileObj.flush()

Citirea datelor dintr-un fișier

Ca și în cazul citirii, există trei metode de citire dintr-un fișier. Se poate citi un anumit număr de octeți, o linie întreagă sau un singur octet. Fiecare metodă returnează **true** în cazul în care citirea s-a efectuat cu succes și **false** în caz contrar. Sintaxa lor este:

```
fileObj.read(count)
fileObj.readln()
fileObj.readByte()
```

Convertirea datelor

Datele pot fi stocate în cadrul unui fisier ca si text ASCII sau în format binar. Obiectul **file** oferă două metode de convertire a datelor dintr-un format în altul. Ambele metode sunt statice, deci nu este nevoie de un obiect.

stringToByte	converteste primul caracter dintr-un string în număr;
byteToString	converteste un număr într-un string de un caracter.

Sintaxa acestor metode este:

```
File.byteToString(number)
File.StringToByte(string)
```

Verificarea erorilor si obtinerea altor informatii

Există o serie de alte metode ale obiectului **file** care oferă informatii cu privire la un fisier. Aceste metode sunt:

exists	returnează true dacă un fisier există, false altfel;
getLength	returnează numărul de octeti sau caractere din fisier, -1 la eroare;
error	pentru prevenirea situatiilor de eroare dintr-un fisier;
clearError	pentru înlăturarea situatiilor de eroare dintr-un fisier.

VI. Aplicații JavaScript

Crearea unor pagini Web statice și plate este deja un lucru de domeniul trecutului. Cum numărul paginilor Web tinde să ajungă până la zeci de milioane, cele mai vizitate vor fi acelea care captează atenția privitorilor și le stârnesc interesul. Creatorii de pagini Web se îndreaptă acum spre tot mai multe instrumente care să creeze pagini cu un aspect vizual cât mai atrăgător.

JavaScript poate fi folosit pentru a realiza câteva efecte vizuale.

Pagina Web a Universității (<http://www.ubbcluj.ro>)



În această pagină efectul vizual constă în faptul că la mișcarea mouse-ului peste o imagine aceasta se schimbă. Realizarea acestui lucru s-a făcut prin implementarea următorului cod:

```
<script language="JavaScript">
function image() {
```

```
        this[0]="UBB";
        this[1]="Faculties";
        this[2]="Multicultural";
        this[3]="Staff";
        this[4]="Research";
        this[5]="Students";
        this[6]="Intranet";
        this[7]="Services";
        this[8]="Pictures";
    }

    var i=new image();

    function change(nr, on) {
        var name=i[nr];
        if(on) {
            document.forms[0].details.value=t[nr];
            if(nr==0);
            if(nr==1) document.Faculties.src="./images/Faculties2.jpg";
            if(nr==2) document.Multicultural.src="./images/Multicultural2.jpg";
            if(nr==3) document.Staff.src="./images/Staff2.jpg";
            if(nr==4) document.Research.src="./images/Research2.jpg";
            if(nr==5) document.Students.src="./images/Students2.jpg";
            if(nr==6) document.Intranet.src="./images/Intranet2.jpg";
            if(nr==7) document.Services.src="./images/Services2.jpg";
            if(nr==8) document.Pictures.src="./images/Pictures2.jpg";
        }
        else {
            document.forms[0].details.value=t[0];
            if(nr==0);
            if(nr==1) document.Faculties.src="./images/Faculties0.jpg";
```

```

        if(nr==2) document.Multicultural.src="/images/Multicultural0.jpg";
        if(nr==3) document.Staff.src="/images/Staff0.jpg";
        if(nr==4) document.Research.src="/images/Research0.jpg";
        if(nr==5) document.Students.src="/images/Students0.jpg";
        if(nr==6) document.Intranet.src="/images/Intranet0.jpg";
        if(nr==7) document.Services.src="/images/Services0.jpg";
        if(nr==8) document.Pictures.src="/images/Pictures0.jpg";
    }
}
</script>

```

Apelarea acestor functii de schimbarea a unei imagini se face din HTML folosind sintaxa:

```

<A HREF="/Faculties/index.html"
    onMouseOver="change(1,1);"
    onMouseOut ="change(1,0);">
    
</A>

```

Pagina Web a studentilor (<http://www.scs.ubbcluj.ro>)

În această pagină pe lângă efectul vizual întâlnit în pagina precedentă mai apare si un ceas care se actualizează automat la fiecare secundă. Codul sursă pentru un astfel de ceas este:

```

<SCRIPT LANGUAGE="JavaScript">
<!--
function clockTime()
{
    currentTime = new Date();

```

```

document.clock_form.clock.value = " "+currentTime;
document.clock_form.clock.blur();
setTimeout("clockTime()", 1000);
}
clockTime();
// -->
</SCRIPT>

```



Jocul “Lines”

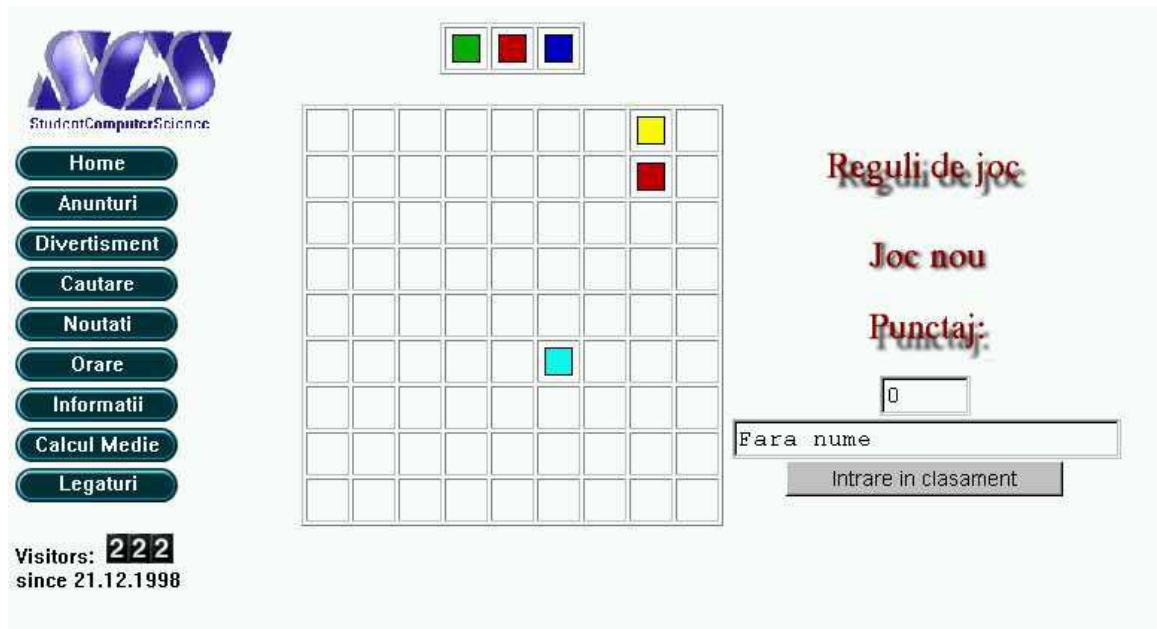
Jocului “Lines” este un exemplu de aplicatie JavaScript în care interactiunea utilizatorului are loc la nivel de browser fără a fi nevoie de o implicare a serverului.

Acest joc are următoarele reguli: avem o tabla de 9x9 si piese de 7 culori:



Initial tabla de joc va fi goală si vor apărea trei bile de culori aleatoare si pe pozitii aleatoare. Aceste piese trebuie mutate astfel încât să formeze linii de câte 5 sau mai multe piese de o aceeași culoare. Liniile astfel formate vor dispărea de pe tablă iar punctajul va

crește în funcție de câte piese a conținut linia respectivă. Jocul se va încheia atunci când pe tablă nu va mai fi nici o poziție liberă.



De fiecare dată trebuie mutată o singură piesă. După o astfel de mutare pe tablă vor mai apărea în mod aleator pe pozițiile libere trei piese de culori aleatoare.

Piesele trebuie astfel mutate încât să se formeze linii (pe linie, coloana sau diagonală) de câte 5 sau mai multe piese de o aceeași culoare. Punctajul care se acordă pentru construirea unei linii este de:

- 10 puncte pentru linie de 5 piese;
- 12 puncte pentru linie de 6 piese;
- 18 puncte pentru linie de 7 piese;
- 28 puncte pentru linie de 8 piese;
- 52 puncte pentru linie de 9 piese.

O altă facilitățile a jocului este păstrarea unui clasament. În funcție de punctajul obținut, se formează un clasament a 10 dintre cei mai buni jucători.



Clasament

Home	310	Cocosatul
Anunturi	302	Sanda
Divertisment	300	Vlad
Cautare	282	Imparateasa rosie
Noutati	258	Vlad cool%3C%2FI%3E
Orare	234	%7E%21%40%23%24%25%5E%26*%28%29_%2B%22%3A
Informatii	226	%5C%3CI%5C%3EVlad cool%5C%3C%5C%2FI%5C%3E
Calcul Medie	224	noname
Legaturi		

Visitors: **222**
since 21.12.1998

[Inapoi la joc](#)

Un jucător poate intra în clasament dacă are un punctaj mai mare decât punctajele deja existente în clasament. Intrarea în clasament se face apăsând butonul “*Intrare în clasament*”. Se va afișa *numele* și *punctajul* obținut de jucător sub forma:



SALUT Alexandra

Ai obținut scorul de **40** puncte!

Felicitari!

[Inapoi la joc](#)
[Afișare clasament](#)

Visitors: **222**
since 21.12.1998

Această facilitate putea fi implementată în JavaScript la nivel de server, însă acest lucru nu s-a putut realiza datorită faptului că nu dispunem încă de pachetul LiveWire. Implementarea pentru această parte a fost făcută sub formă de shell-uri CGI.

O altă extensie a jocului este prezentarea *regulilor de joc*



Reguli de joc

Acest joc, la o prima vedere, aduce foarte mult cu jocul cunoscut sub numele de **Lines**, însă are câteva "capcane". Cine le descopera este invitat sa trimita un mail pe adresa: savram@cs.ubbcluj.ro. Acest joc se joaca in felul urmator:
 Avem o tabla de 9x9 si piese de 7 culori:

De fiecare data trebuie mutata o singura piesa. Dupa o astfel de mutare pe tabla vor mai aparea in mod aleator pe pozitiiile libere trei piese de culori aleatoare.

Piese trebuie astfel mutate incat sa se formeze linii (pe linie, coloana sau diagonala) de cate 5 sau mai multe piese de o aceeași culoare. Daca se formeaza o astfel de linie, piesele din respectiva linie vor disparea iar punctajul va creste cu:

- 10 puncte pentru 5 piese;
- 12 puncte pentru 6 piese;
- 18 puncte pentru 7 piese;
- 28 puncte pentru 8 piese;
- 52 puncte pentru 9 piese.

Jocul se va termina in momentul in care pe tabla nu vor mai fi pozitii libere.

[Inapoi la joc](#)

Implementarea algoritmilor necesari jocului a fost realizată în JavaScript si se află în fisierul *lines.js* (vezi anexa). Legătura dintre acest cod JavaScript si HTML s-a realizat prin intermediul elementelor din pagina *index.html*.

Funcțiile si variabilele folosite

A) Funcții de definire a unor obiecte

color();	-definește un obiect-tablou care contine cele 7 culori folosite plus culoarea albă care reprezintă pozitia liberă.
tabel();	-definește un obiect-tablou care contine 81=9x9 elemente.
linie();	-definește un obiect-tablou de maxim 9 elemente.

B) Funcții folosite în inițierea jocului

start();	-este funcția de începere a unui joc nou;
-----------------	---

start_game();	-resetează punctajului si începe jocul prin apelarea procedurilor pentru generare de piese de culori aleatoare si asezarea lor pe pozitii aleatoare;
rndColor();	generează trei piese de culori aleatoare si le aseză în tabelul de “ <i>preview</i> ” (tabelul de deasupra tablei de joc).
pune(c1, c2, c3);	-pune cele trei culori (c1 , c2 si c3) pe tabla de joc în trei pozitii aleatoare

C) Functii folosite pentru mutarea unei piese si eliminarea liniilor

moveXY(xx);	-determină, în functie de interactiunile utilizatorului, de pe ce pozitie x0 pe ce pozitie x se va muta o piesă. xx poate fi x sau x0 .
muta();	-mută o piesă de pe pozitia x0 pe pozitia x în cazul în care între cele două pozitii este drum, altfel dă mesaj de eroare.
miscare();	-determină dacă între cele două pozitii, x0 si x , există drum;
cale();	-determină drumul dintre cele două pozitii x0 si x ;
elim();	-în functie de pozitia finală x a piesei se determină dacă nu s-a format o linie (pe linie, coloană, diagonala principală sau diagonala secundară) de 5 sau mai multe piese de aceeași culoare pentru a fi eliminată;
init(l);	-iniază vectorul l în care se păsrează o pozitiiile unei linii cu piese de aceeași culoare;
verif(xx, culoare, v);	-verifică relativ la pozitia xx de pe tabla de joc dacă am o linie de 5 sau mai multe piese de culoarea culoare pe directia indicată de v . Sterge linia si măreste punctajul.

D) Functii auxiliare

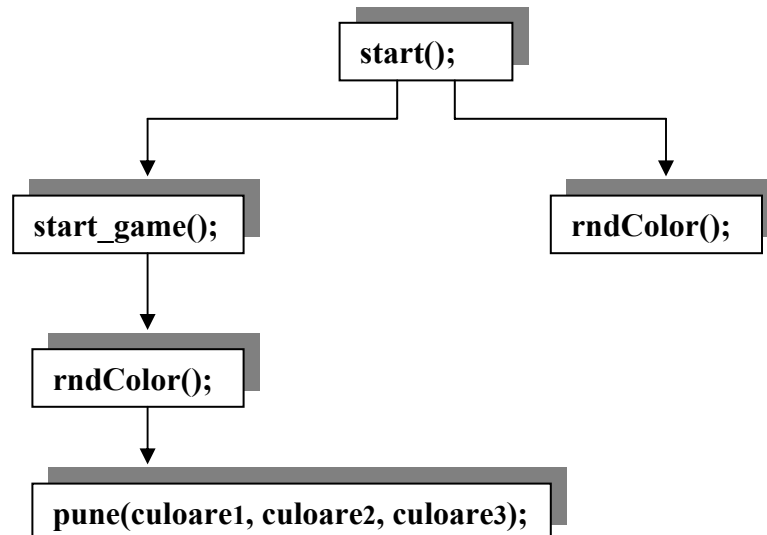
afis(tt);	-afisează în format numeric starea tablei de joc tt la un moment dat;
------------------	--

(functie folosită în scop de depanare)

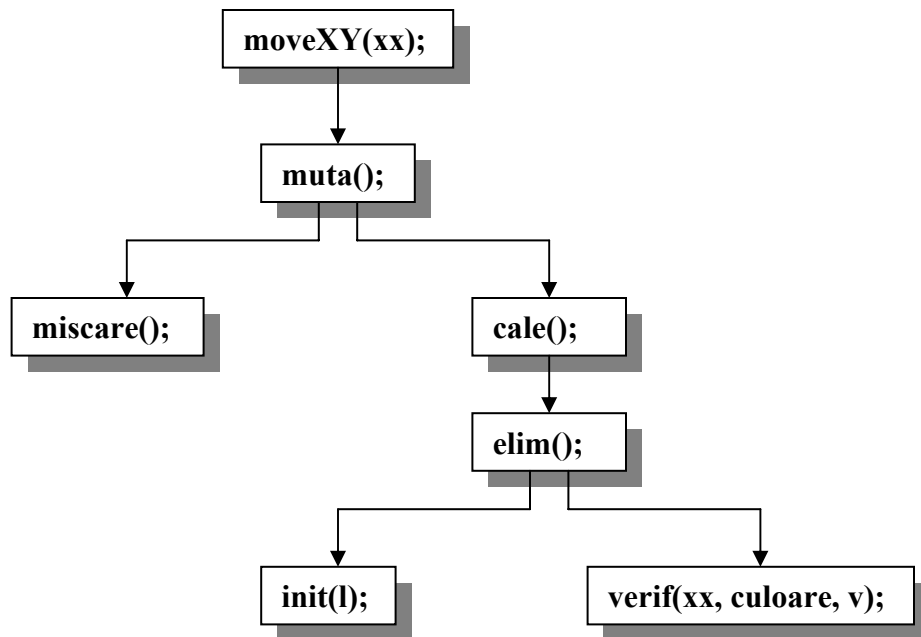
lafis(II);

-afisează starea liniei **II** cu pozițiile elementelor de aceeași culoare de pe o linie; (functie folosită în scop de depanare)

Legătura dintre aceste funcții este redată în următoarele diagrame de apelare ale funcțiilor:



(a) Funcțiile de inițiere a unui joc



(b) Funcții de mutare a pieselor pe tabla de joc și de stergere a liniilor de 5 sau mai multe piese

Principalele variabile folosite sunt următoarele:

var t=new tabel(0);	-tablou în care se păstrează tabla de joc;
var m=new tabel(-1);	-matrice temporară folosită la căutarea drumului;
var drum=new tabel(0);	-drum parcurs de o piesă;
var ind;	-ține minte săritura în căutarea drumului;
var punctaj=0;	-se păstrează punctajul;
var x0, x;	-punct de plecare/sorire al bilei;
var first, next;	-dacă poziția selectată de utilizator este poziția de plecare sau cea de sosire;
var l=new linie();	-se păstrează pozițiile unei linii cu piese de aceeași culoare;
var n;	-nr bile pe linie;
var am_linie;	-indică dacă am făcut o linie;
var c=new color;	-vector de culori.

Codul sursă al fișierelor care compun această aplicație se găsesc în anexa acestei lucrări.

ANEXĂ

Lines.js – fișier care conține codul JavaScript pentru joc

```
<!-- part of the HEAD section -->
//////////////////// COLOR //////////////////////
function color(){
    this[0]="blank";
    this[1]="red";
    this[2]="blue";
    this[3]="yellow";
    this[4]="green";
    this[5]="magenta";
    this[6]="maroon";
    this[7]="cyan";
}

var i;                ////////////////// variabila auxiliara

//////////////////// TABEL //////////////////////
function tabel(v){    ////////////////// obiectul tabel
    for(i=0; i<=80; i++) this[i]=v;
    this.len=81;
}

//////////////////// LINIE //////////////////////
function linie(){    ////////////////// obiectul linie
    for(i=0; i<=9; i++) this[i]=0;
}

var t=new tabel(0);    ////////// tablou in care se pastreaza tabla de joc
var m=new tabel(-1);   ////////// matrice temporara
var drum=new tabel(0);  ////////// drum parcurs de o bila
var ind;                ////////// tine minte saritura in cautarea drumului
var punctaj=0;

//////////////////// AFIS //////////////////////
function afis(tt) {
    var text=" ";
    for(i=0; i<=80; i++) {
        if((i%9)==0) text=text+"\n ";
        text=text+tt[i]+" ";
    }
    alert(text);
    return true;
}

//////////////////// LAFIS //////////////////////
function lafis(ll) {
    var text=" ";
```

```

        for(i=0; i<=9; i++) text=text+ll[i]+" ";
        alert(text);
        return true;
    }

//-----//

var x0, x;        // punct de plecare/sorire al bilei
var first, next;

var l=new linie(); // linie
var n;            //nr bile pe linie

var am_linie;    //daca am facut o linie

////////// INIT //////////
////////// initializez cu 0 liniile
function init(ll){
    for(i=0; i<=9; i++) ll[i]=0;
    return true;
}

////////// VERIF //////////
////////// verific daca am 5 in linie
function verif(xx, culoare, v){
    n=0;
    init(l);
    ////////////in sus
    var xx0=xx;
    while((xx0>=0) && (((xx0+v)%9))||((xx0%9)&&(v==9)) &&
((xx0+v+1)%9))||(((xx0+1)%9)&&(v==9))) {
        if(t[xx0]==culoare){ l[n]=xx0; n++;}
        else break;
        xx0-=v;
    }
    ////////////in jos
    xx0=xx+v;
    while((xx0<=80) && ((xx0-v)%9)||((xx0%9)&&(v==9)) && ((xx0-
v+1)%9)||(((xx0+1)%9)&&(v==9))) {
        if(t[xx0]==culoare){ l[n]=xx0; n++;}
        else break;
        xx0+=v;
    }

    //////////// stergerea liniei curente
    if(n>=5){
        am_linie=true;
        for(i=0; i<n; i++){
            document.images[l[i]+3].src=c[0]+".gif";
            t[l[i]]=0;
        }
    }

    //////////// punctaj

```

```
        if(n==5) punctaj+=10;
        if(n==6) punctaj+=12;
        if(n==7) punctaj+=18;
        if(n==8) punctaj+=28;
        if(n==9) punctaj+=52;

        document.forms[0].scor.value="" + punctaj;
        t.len+=n;
    }

    first=false;
    next=false;
    x=0; x0=0;
    return true;
}

////////// ELIM //////////
////////// elimin mai mult de 5 bile pe linie
function elim(xx){
    var culoare=t[xx];
    ///alert("Caut de culoare"+culoare);
    am_linie=false;

    ////////// verific linie verticala
    verif(xx, culoare, 9);

    ////////// verific linie orizontala
    verif(xx, culoare, 1);

    ////////// verific diagonala principala
    verif(xx, culoare, 10);

    ////////// verific diagonala secundara
    verif(xx, culoare, 8);

    if((t.len>0) && (!am_linie)) start_game();
    return true;
}

////////// CALE //////////
////////// gaseste drumul
function cale(){
    drum[indice]=x;
    for(i=indice-1; i>=0; i--){
        ///stanga
        if(((drum[i+1]%9)!=0) && (m[drum[i+1]-1]==i)) drum[i]=drum[i+1]-1;
        ///dreapta
        if(((drum[i+1]%9)!=8) && (m[drum[i+1]+1]==i)) drum[i]=drum[i+1]+1;
        ///sus
        if((Math.floor(drum[i+1]/9)!=0) && (m[(Math.floor(drum[i+1]/9)-1)*9+(drum[i+1]%9)]==i))
            drum[i]=(Math.floor(drum[i+1]/9)-1)*9+(drum[i+1]%9);
        ///jos
        if((Math.floor(drum[i+1]/9)!=8) &&
            (m[(Math.floor(drum[i+1]/9)+1)*9+(drum[i+1]%9)]==i))
```

```

        drum[i]=(Math.floor(drum[i+1]/9)+1)*9+(drum[i+1]%9);
    }

    for(i=0; i<indice; i++){
        document.images[drum[i]+3].src=c[0]+".gif";
        document.images[drum[i+1]+3].src=c[t[x0]]+".gif";
    }
    t[x]=t[x0];
    t[x0]=0;

    elim(x);
    return true;
}

//////////////////// MISCARE //////////////////////
//////////////////// drumul parcurs de o bila (daca exista)
function miscare(){
    var gata=true;
    indice=0;
    m[x0]=0;
    if(x0!=x){
        gata=false;
        while((!gata)&&(indice<=81)){
            for(i=0;i<=80;i++){
                if(m[i]==indice){
                    if(((i%9)!=0) && (t[i-1]==0) && (m[i-1]==-1)) m[i-1]=indice+1;
                    if(((i%9)!=8) && (t[i+1]==0) && (m[i+1]==-1)) m[i+1]=indice+1;
                    if(((Math.floor(i/9))!=0)&&(t[(Math.floor(i/9)-
1)*9+(i%9)]==0)&&(m[(Math.floor(i/9)-1)*9+(i%9)]==-1))
                        m[(Math.floor(i/9)-1)*9+(i%9)]=indice+1;
                    if(((Math.floor(i/9))!=8)&&(t[(Math.floor(i/9)+1)*9+(i%9)]==0)&&(
m[(Math.floor(i/9)+1)*9+(i%9)]==-1))
                        m[(Math.floor(i/9)+1)*9+(i%9)]=indice+1;
                }
            }///for
            indice++;
            if(m[x]==indice) gata=true;
        }
        return gata;
    }
}

//////////////////// MUTA //////////////////////
function muta() {
    for(i=0; i<=80; i++) m[i]=-1;
    if (miscare()) cale();
    else {alert("Nu este drum!"); first=false; next=false;}
    return true;
}

//////////////////// MOVEXY //////////////////////
//////////////////// mutarea unei bile de pe pozitia X0 pe pozitia X
function moveXY(xx) {

```

```

    if(t[xx]!=0) {x0=xx; first=true;}
    else if (first) {x=xx; next=true;}
    if(next) muta();

}

//-----//

var c=new color;          //culoarea
var nr1, nr2, nr3;
var culoare1, culoare2, culoare3;

var n, n1, n2, n3;

////////// PUNE //////////
////////// asezarea pe tabla a trei culori aleatoare
function pune(c1, c2, c3) {
    if(t.len<3) alert("Felicitari!\n Ati obtinut
"+document.forms[0].scor.value + " puncte.");
    else{
        n1=Math.round(Math.random()*100)%81;
        n2=Math.round(Math.random()*100)%81;
        n3=Math.round(Math.random()*100)%81;

        while(true) {
            if(t[n1-1]==0) {t[n1-1]=nr1; break;}
            else n1=(n1+1)%81;
        }
        document.images[n1+2].src=c1;
        verif(n1-1);

        while(true) {
            if(t[n2-1]==0) {t[n2-1]=nr2; break;}
            else n2=(n2+1)%81;
        }
        document.images[n2+2].src=c2;
        verif(n2-1);

        while(true) {
            if(t[n3-1]==0) {t[n3-1]=nr3; break;}
            else n3=(n3+1)%81;
        }
        document.images[n3+2].src=c3;
        verif(n3-1);

        t.len-=3;
    }
    return true;
}

////////// RNDCOLOR //////////
////////// generare aleatoare a trei culori
function rndColor() {
    nr1=(Math.round(Math.random()*10)%7)+1;

```

```

culoare1=c[nr1]+".gif";
document.r1.src=culoare1;
nr2=(Math.round(Math.random()*10)%7)+1;
culoare2=c[nr2]+".gif";
document.r2.src=culoare2;
nr3=(Math.round(Math.random()*10)%7)+1;
culoare3=c[nr3]+".gif";
document.r3.src=culoare3;
return true;
}

////////// START_GAME //////////
function start_game() {
    document.forms[0].scor.value="" + punctaj;
    first=false;
    next=false;
    pune(culoare1, culoare2, culoare3);
    rndColor();
    return true;
}

////////// START //////////
function start() {
    var nr;
    rndColor();
    start_game();
    return true;
}

//-----//
<!-- part of the BODY section -->

```

index.html – pagina care contine jocul si foloseste codul scris în fisierul *lines.js*

```

<html>
<head>
<script language="JavaScript" src="lines.js"></script>
<script language="JavaScript">

function reguli() {
    window.open('reguli.html', 'Regulile
jocului', 'toolbar=no, location=no, directories=no, status=no, menubar=no, sc
rollbars=no, width=400, height=500');
}

</script>
</head>

<body onLoad="start()" bgcolor=white>

<table align=center>
<tr><td>

```

```
<center>
<table border=1 rows=1 cols=3>
<tr>
<td width=25></td>
<td width=25></td>
<td width=25></td>
</tr>
</table>
</center>
<br>
<center>
<table border=1 rows=9 cols=9>
<tr>
<td width=25><a href="javascript:moveXY(0)"
    onClick="Click(0);"></a></td>
<td width=25><a href="javascript:moveXY(1)"
    onClick="Click(1);"></a></td>
<td width=25><a href="javascript:moveXY(2)"
    onClick="Click(2);"></a></td>
<td width=25><a href="javascript:moveXY(3)"
    onClick="Click(3);"></a></td>
<td width=25><a href="javascript:moveXY(4)"
    onClick="Click(4);"></a></td>
<td width=25><a href="javascript:moveXY(5)"
    onClick="Click(5);"></a></td>
<td width=25><a href="javascript:moveXY(6)"
    onClick="Click(6);"></a></td>
<td width=25><a href="javascript:moveXY(7)"
    onClick="Click(7);"></a></td>
<td width=25><a href="javascript:moveXY(8)"
    onClick="Click(8);"></a></td>
</tr>
<tr>
<td width=25><a href="javascript:moveXY(9)"></a></td>
<td width=25><a href="javascript:moveXY(10)"></a></td>
<td width=25><a href="javascript:moveXY(11)"></a></td>
<td width=25><a href="javascript:moveXY(12)"></a></td>
<td width=25><a href="javascript:moveXY(13)"></a></td>
<td width=25><a href="javascript:moveXY(14)"></a></td>
<td width=25><a href="javascript:moveXY(15)"></a></td>
<td width=25><a href="javascript:moveXY(16)"></a></td>
<td width=25><a href="javascript:moveXY(17)"></a></td>
</tr>
<tr>
<td width=25><a href="javascript:moveXY(18)"></a></td>
<td width=25><a href="javascript:moveXY(19)"></a></td>
```



```

<td width=25><a href="javascript:moveXY(71)"></a></td>
</tr>
<tr>
<td width=25><a href="javascript:moveXY(72)"></a></td>
<td width=25><a href="javascript:moveXY(73)"></a></td>
<td width=25><a href="javascript:moveXY(74)"></a></td>
<td width=25><a href="javascript:moveXY(75)"></a></td>
<td width=25><a href="javascript:moveXY(76)"></a></td>
<td width=25><a href="javascript:moveXY(77)"></a></td>
<td width=25><a href="javascript:moveXY(78)"></a></td>
<td width=25><a href="javascript:moveXY(79)"></a></td>
<td width=25><a href="javascript:moveXY(80)"></a></td>
</tr>
</table>
</center>
</td>
<td valign=bottom>
<center>
<form action="../../../cgi-bin/insert.cgi" method="post">
<a href="javascript:reguli();"
onMouseOver="document.reguli.src='./images/reguli_on.gif';"
onMouseOut="document.reguli.src='./images/reguli_off.gif';">
</a>
<br>
<a href="javascript:window.location.reload();"
onMouseOver="document.nou.src='./images/nou_on.gif';"
onMouseOut="document.nou.src='./images/nou_off.gif';">
</a>
<br>
<a href="../../../cgi-bin/h.cgi"
onMouseOver="document.punctaj.src='./images/punctaj_on.gif';"
onMouseOut="document.punctaj.src='./images/punctaj_off.gif';">
</a>
<br>

<input type="text" SIZE="5" MAXLENGTH="0" name="scor">
<br>
<input type="text" size="25" name="nume" value="Fara nume"
MAXLENGTH="45">
<br>
<input type="submit" value="Intrare in clasament">
</form>
</center>
</td></tr>
</table>

</body>

```

</html>

reguli.html- pagina care contine regulile jocului

```
<html>
<head>
<title>Regulile jocului</title>
</head>
<body>
<font color="red">
<center><h1>Reguli de joc</h1></center>
</font>
Acest joc, la o prima vedere, aduce forte mult cu jocul cunoscut sub
numele
de <font color=blue><b>Lines</b></font>, insa are cateva "capcane".
Cine le descopera este invitat sa trimita un mail pe adresa:
<a href="mailto:savram@cs.ubbcluj.ro">savram@cs.ubbcluj.ro</a>.
<br>
Acest joc se joaca in felul urmator:
<br>
Avem o tabla de 9x9 si piese de 7 culori:
<br>
<center>
<table>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</table>
</center>
<br>
De fiecare data trebuie mutata o singura piesa. Dupa o astfel de mutare
pe
tabla vor mai aparea in mod aleator pe pozitiile libere trei piese de
culori
aleatoare.
<br><br>
Piese trebuie astfel mutate incat sa se formeze linii (pe linie,
coloana
sau diagonala) de cate 5 sau mai multe piese de o aceeaasi culoare. Daca
se
formeaza o astfel de linie, piesele din respectiva linie vor disparea
iar
punctajul va creste cu:
<center>
<table>
<tr><td>- 10 puncte pentru 5 piese;</td></tr>
<tr><td>- 12 puncte pentru 6 piese;</td></tr>
<tr><td>- 18 puncte pentru 7 piese;</td></tr>
<tr><td>- 28 puncte pentru 8 piese;</td></tr>
```

```
<tr><td>- 52 puncte pentru 9 piese.</td></tr>
</table>
</center>
<br>
Jocul se va termina in momentul in care pe tabla nu vor mai fi pozitii
libere.
<br>
<center>
<form>
<input type="button" value="Close"
onClick="javascript:window.close();">
</form>
</center>
</body>
</html>
```

h.cgi- shell-ul CGI care afisează clasamentul

```
#!/bin/sh
echo -e "Content-type: text/html\n"
echo "<html>"
echo "<head>"
echo "<title>Clasamentul celor mai buni jucatori</title>"
echo "</head>"
echo "<body bgcolor=white>"
echo "<br>"
echo "<h1><font color=blue><center>Clasament</center></font></h1>"
echo "<br>"
echo "<center>"
echo "<table><tr><td>"
echo "<pre><b>"
less -N da
echo "</b></pre>"
echo "</td></tr></table>"
echo "<form>"
echo "<input type='button' value='Inapoi la joc' onClick='javascript:
parent.main.open(\"../divertisment/index.html\", \"main\");>"
echo "</form>"
echo "</center>"
echo "</body>"
echo "</html>"
```

insert.cgi – shell-ul CGI care introduce un jucător în clasament

```
#!/bin/sh
echo -e "Content-type: text/html\n"
read intrare
echo $intrare | sed 's/\&/=/g' | awk -F= '{print $2" "$4" <br>";}' |
sed 's/+/ /g' > nu
cat da > temp
cat nu >> temp
sort -nr +0 temp|head -n 10 > da
echo "<html>"
```


Bibliografie

- [1] **Stephen Feather, Luke Cassady-Dorion**, *JavaScript by Example*, 1996
- [2] **Mark C. Reynolds**, *Using JavaScript*, 1996
- [3] **Arman Danesh**, *Teach JavaScript in 7 days*, 1996
- [4] **Dumitru Radoiu**, *HTML. Publicatii Web*, 1996
- [5] **Andrew S. Tanenbaum**, *Retele de calculatoare*, 1997
- [6] **Jasons J. Manger**, *JavaScript Essentials*, 1996
- [7] <http://www/wwwhelp/jsdocs/index.html>
- [8] <http://www.c2.org/~andreww/javascript/>
- [9] <http://www.netscape.com>
- [10] <http://www.mpc.com>