**REVISTA SCOLII GIMNAZIALE BALCESTI**

**Nr. 35 2021**



**COLECTIVUL DE REDACŢIE**

**COORDONARE:**

**FONDATOR BOGDAN CONSTANTIN**

**REDACTOR- ŞEF: DOBRE ROXANA , BUJOR MARIA MIHAELA**

**CONSULTANT: COJOCARU MIHAELA , CEPOI DELIA , RADOI CARMEN**

**TEHNOREDACTARE:OPREA RADU, IENCUT CRISTINA**

**CORECTOR:COJAN GEORGIANA**

**PUBLICARE REVISTA:BOGDAN CONSTANTIN**

**Adresa redactiei:**

BALCESTI , COMUNA BENGESTI CIOCADIA

**Fiecare autor îşi asumă responsabilitatea pentru conţinutul textului publicat.**

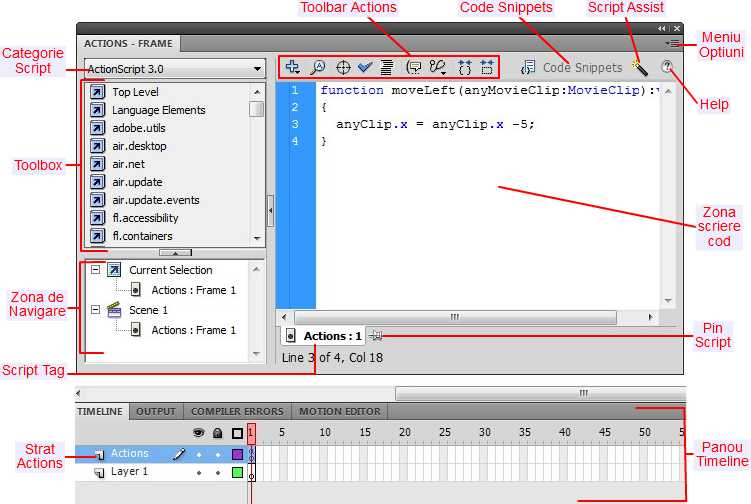
# ActionScript

# PROF DR BOGDAN CONSTANTIN

BUJOR MARIA MIHAELA

**Limbajul de programare** pentru Flash este **ActionScript**.  
Cu **ActionScript** se scriu instructiuni prin care pot fi controlate elementele din prezentarea Flash. De exemplu, se pot adauga instructiuni la butoane prin care sa fie pornita si oprita o animatie, sa se schimbe culoarea si dimensiunile unui obiect; sau se pot crea obiecte, text si animatii in prezentarea Flash, folosind cod ActionScript.  
Cu ActionScript se pot crea programe care ruleaza pe internet si in pagini web (de ex. jocuri), sau pentru iPhone, telefoane mobile ori Desktop.  
Este un limbaj de scripting asemanator cu JavaScript; de exemplu, in ambele limbaje variabilele se declara utilizand comanda "var", sirurile se unesc folosind caracterul "+". Daca stiti sa lucrati cu JavaScript, veti intelege si invata mai usor ActionScript.  
**ActionScript 3** a fost introdus de Adobe in 2006 cu "Adobe Flex 2.0".  
    - Este un limbaj de programare orientat pe obiecte (*OOP - Object Oriented Programming*) destul de avansat si puternic, incluzand unele din cele mai noi concepte de programare, utilizand Clase, obiecte, metode si evenimente.  
  
Pentru a crea o prezentare Flash cu ActionScript 3, se deschide un document nou alegand "ActionScript 3.0" de la **Create New** din fereastra de inceput a programului Adobe Flash.  
Codul ActionScript se adauga la cadrele din Timeline sau intr-un fisier extern (cu extensia ".as").  
- *Spre deosebire de ActionScript 2.0, in aceasta versiune mai noua, codul nu se adauga direct in Movie Clips sau alte obiecte*.  
Pentru inceput va fi aratata medoda de adaugare a lui direct in document, la cadre din Timeline.  
Dupa ce se deschide un document Flash nou, este indicat sa fie mai intai creat un Strat (Layer) special pentru coduri ActionScript (din meniul **Insert -> Timeline -> Layer**). Se poate folosi in orice Cadru din Timeline, dar fiind pe un Layer separat e mai usor de structurat; iar pentru un control mai bun, sa i-se dea un nume special acelui Strat, intuitiv (prin dublu-click pe numele lui), de ex. "*Actions*" (mai ales pt. cazuri cand se va lucra cu mai multe Straturi).  
Codul ActionScript se scrie intr-un panou special, ca cel din imaginea urmatoare, care are un set complet de optiuni, descrise mai jos. Acest panou se poate deschide din meniul **Window -> Acions** (dupa ce a fost selectat un Frame din Stratul "Actions"), sau prin click-dreapta pe cadrul din acel Strat, la care va fi apicat codul, si se alege optiunea **Actions**.

**- Panou pt. ActionScript / Strat Actions in Timeline**

  
- In partea de jos a imaginii este panoul Timeline, cu stratul "Actions" creat pt. adaugare ordonata a codului ActionScript la cadrele din el. Aici, cadrul selectat este primul din acest Layer.  
- Iata ce fac optiunile din panoul "ACTIONS - FRAME":

* **Categorie Script** - Se alege tipul de script /versiune folosit la cod.
* **Toolbox** - Ofera o lista cu toate actiunile instalate in program, la versiunea de script aleasa, organizate pe categorii.
* **Zona de navigare** - Aici sunt organizate toate scripturile din document, si pot fi gasite dupa "**Strat : Numar cadru**".
* **Script Tag** - Indica ce script (din ce "Cadru : Frame") este editat.
* **Pin Script** - Adauga in acea linie un "Tab" la scriptul curent, pt. a naviga mai usor din unu in altu cand se lucreaza cu scripturi in mai multe cadre.
* **Zona scriere cod** - Este locul unde se scrie codul ActionScript.
* **Help** - Deschide pagina online de Ajutor de la Adobe, pentru elementul din cod selectat.
* **Meniu Optiuni** - Meniul cu optiuni ale panoului, pentru control, format, si altele.
* **Script Assist** - Cand aceasta optiune este activata, codul in script se adauga controlat, din zona "Toolbox". Script Assist "supravegheaza" sa se scrie cod fara erori.
* **Code Snippets** - Deschide un panou care ofera segmente de cod gata facute pentru cele mai utilizate actiuni, organizate pe categorii.
* **Toolbar Actions** - Bara de instrumente a panoului, cu optiunile prezentate mai jos:  
  
  + **Show/Hide Toolbox** - Ascunde si afiseaza Tollbox.
  + **Remove Comment** - Sterge caracterele specifice pt. comentarii; de la cel selectat.
  + **Apply Line Comment** - Adauga caracterele "//" (pentru creare linie comentariu) la inceputul randurilor selectate.
  + **Apply Block Comment** - Transforma liniile selectate intr-un "bloc de comentariu".
  + **Expand All** - Permite extinderea tuturor codurilor care au fost restranse.
  + **Collapse Selection** - Restrange randurile de cod selectate.
  + **Collapse between braces** - Restrange codul dintre acolade.
  + **Debug Options** - Adauga /sterge "Breakpoints" (puncte de pauza), unde intentionat se face o pauza a executiei scriptului la o anumita linie din cod , pentru examinarea lui.
  + **Show code hint** - Afiseaza indicii pentru auto-completare cod.
  + **Auto Format** - Aranjeaza scriptul pentru o mai usoara citire a lui, aplicand spatii si culori. Optiunile pt. format pot fi setate din meniul **Edit -> Preferences -> ActionScript**.
  + **Check syntax** - Verifica scriptul, cautand erori de sintaxa, si afiseaza detalii intr-un panou "Compiler Errors".
  + **Insert target path** - Ajuta la alegerea obiectelor la care face referire o anumita functie din cod.
  + **Find** - Ofera posibilitatea de cautare si inlocuire cuvinte din codul scriptului.
  + **Add script items** - Ofera acces la elemente din limbajul ActionScript, precum cele din Toolbox.

- Acest panou, cu toate optiunile lui, este folosit pentru adaugarea de scripturi in documentul Flash, dupa cum se poate invata din lectiile urmatoare.  
Esential la oricare limbaj de programare este scrierea corecta a codului, respectand syntaxa limbajului. Flash ofera indicii destul de bune in cazul unor erori, pentru depistarea cat mai exacta a lor.  
    - Important de avut in vedere este faptul ca ActionScript este un limbaj cu sintaxa case-sensitive (face diferenta intre majuscule si litere mici), astfel, "Buton" e diferit de "buton".  
  
- Lectiile de ActionScript din acest curs sunt indicate mai mult celor care cunosc cel putin la nivel de incepatori bazele unui **limbaj de programare** (C++, C#, PHP sau JavaScript), deoarece elementele de baza ale oricarui limbaj de programare: variabile, operatori, instructiuni conditionale si ciclice sunt prezentate mai pe scurt, cu putine explicatii.

**Primul script ActionScript 3**

Inainte de a trece la prezentarea elementelor limbajului **ActionScript 3**, iata un script simplu ca sa vedeti inca de la inceput cat de usor se pot crea diferite obiecte (text, figuri geometrice) intr-o prezentare Flash, scriind direct cod ActionScript in document.  
Efectuati urmatorii pasi:

**1.** Deschideti un document Flash nou, **ActionScript 3.0**  
**2.** Click-dreapta pe primul cadru din Timeline si alegeti **Actions**  
**3.** In fereastra care se deschide pentru scriere ActionScript, adaugati urmatorul cod:

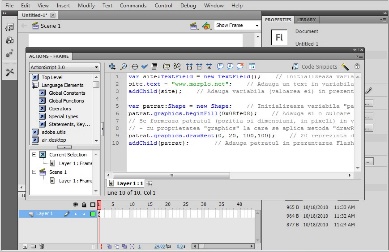
var site:TextField = new TextField(); // Initializeaza variabila "site", avand tipul "TextField" (camp pt. text)

site.text = "www.marplo.net"; // Adauga un text in variabila "site"

addChild(site); // Adauga variabila (valoarea ei) in prezentarea Flash

**4.** Apasati "**Ctrl+Enter**" pentru a vedea rezultatul, prezentarea Flash obtinuta.

- Acest **script ActionScript** afiseaza in player-ul Flash textul "*www.marplo.net*". Destul de simplu :).  
  
Se pot crea multe alte elemente in prezentarea Flash, folosind cod ActionScript: figuri geometrice, miscari de animatie, si altele.  
Iata, de exemplu, sa adaugam un patrat la prezentarea de mai sus, sub text. *Explicatiile necesare, privind rolul fiecarei linii de cod sunt scrise in script*.

**5.** Se continua de la exemplul de sus. In acelasi panou pt. scriere scripturi ActionScript adaugati codul de mai jos, sub cel deja existent (dupa cum vedeti in imaginea urmatoare).  
  
  - *Daca ati inchis deja panoul, dati iar click-dreapta pe primul Cadru in timeline si apoi****Actions***.

var patrat:Shape = new Shape; // Initializeaza variabila "patrat", avand tipul "Shape" (Forma)

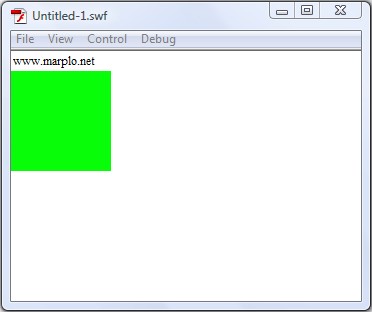
patrat.graphics.beginFill(0x08fe08); // Adauga si o culoare la "patrat", cu proprietatea "graphics" si metoda "beginFill(0xRRGGBB)

// Se formeaza patratul (pozitia si dimensiuni, in pixeli) in variabila "patrat"

// - cu proprietatea "graphics" la care se aplica metoda "drawRect(spatiu x, spatiu y, lungime, inaltime)"

patrat.graphics.drawRect(0, 20, 100,100); // 20 reprezinta distanta in pixeli fata de marginea de sus

addChild(patrat); // Adauga patratul in prezentarea Flash

**6.** Apasati iar "Ctrl+Enter", prezentarea Flash creata va fi ca cea din imaginea urmatoare.  


Asfel, treptat, folosind cod AS3 (ActionScript 3), se pot crea Flash-uri complexe; elementele adaugate si proprietatile lor pot fi controlate prin functii si valori adaugate in cod. De exemplu, pentru a adauga transparenta la patrat, se foloseste metoda "beginFill(culoare, nr\_alpha)" cu doi parametri, unde, "nr\_alpha" este un numar intre 0 si 1 pt. transparenta ( ex.: **beginFill(0x08fe08, 0.5)** ).  
  
Mai multe despre acest limbaj de programare puteti invata din lectiile urmatoare, treptat, incepand cu elementele si actiunile simple.

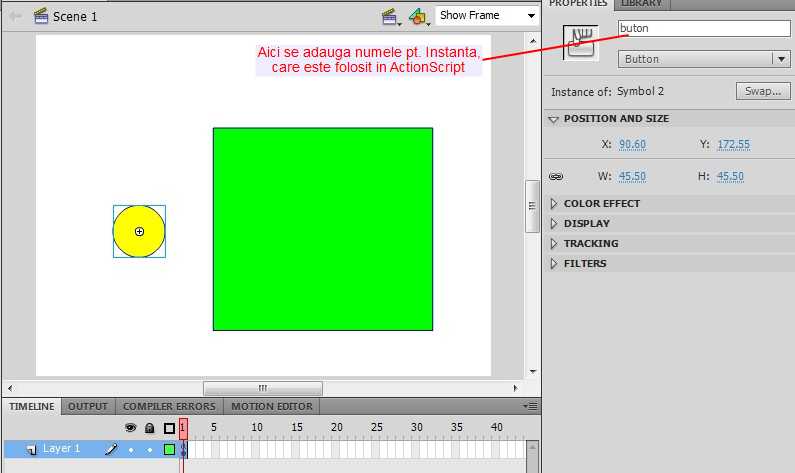
**Intelegere OOP - Object Oriented Programming**

Conceptul **OOP (Object Oriented Programming)** [*Programare orientata pe obiecte*] are la baza impartirea codului programului in obiecte separate, care pot fi si grupate, formand astfel un alt obiect. De exemplu, rotile, cadrul, ghidonul unei biciclete sunt obiecte separate, dar unite formeaza o bicicleta, care si ea poate fi considerata un obiect. La fel si in programarea ActionScript in Flash, textul, figura geometrica, symbolul movie clip, etc., sunt considerate obiecte.  
• **Obiectele** sunt definite folosind doi identificatori primari: **proprietati** si **metode**  
• Proprietatile unui obiect ii definesc caracteristicile acestuia, iar metodele sunt functii (actiuni) aplicate obiectului printr-o anumita proprietate.  
De exemplu, obiectul poate fi o "casa", iar "culoare", "ferestre", "usi" pot fi considerate proprietati, si "deschidere", "inchidere" sunt metode. Aplicarea unei metode la un obiect printr-o proprietate s-ar putea traduce prin "*deschidere fereastra la casa*".  
  
• In programare, "metoda()" este o functie care face parte din definitia unei clase, de aceea se foloseste cu paranteze rotunde.  
• Legatura intre obiecte, proprietati si metode se face prin caracterul punct (.).  
• Liniile de cod trebuie sa se termine cu un caracter punct-si-virgula (;).  
Astfel, exemplul dat anterior (*deschidere fereastra la casa*) s-ar scrie "*casa.fereastra.deschidere();*".  
  
Caracteristicile unui obiect se schimba prin proprietatile si metodele pe care le poate avea.  
Atribuirea (setarea) unei valori la proprietatea unui obiect se face comform sintaxei:  
          **obiect.proprietate = valoare;**  
Aplicarea unei metode se face dupa sintaxa:  
          **obiect.proprietate.metoda();**  
De exemplu:  
          ***patrat.width = 100;***  
- Obiectul este "patrat", proprietatea lui e "width" iar "100" este valoarea care i-se da.  
          ***patrat.graphics.beginFill(0x08fe08);***  
- Metoda "beginFill()" este aplicata obiectului "patrat" prin proprietatea acestuia "graphics", acestea fiind unite prin caracter punct (.).  
- "metoda()" fiind o functie, poate avea unul sau mai multi parametri, care se adauga intre parantezele rotunde.  
  
Unele metode pot fi aplicate direct unui obiect, fara intermediul vreunei proprietati  
    De ex.: **myClip.gotoAndStop(9);**  
- "gotoAndStop(9)" muta cadru curent din animatia "myClip" la cadrul 9 si opreste.  
  
• Pe langa obiecte, proprietati si metode; in OOP sunt si elemente cunoscute ca "**events**" (evenimente). Acestea sunt actiuni care declanjeaza executia unei functii.  
De exemplu, apasarea unui buton, sau trecerea cu mouse-ul deasupra unui obiect, sunt evenimente prin care se poate porni executia unei functii.

Pe scurt, in OOP (Programarea Orientata pe Obiecte) se folosesc urmatoarele elemente:

* **Classes (*clase*)** - Este reprezentarea unui obiect, unde sunt definuite proprietatile si metodele lui.
* **Instances (*instante*)** - Instanta unui obiect, denumirea lui, la care se pot aplica proprietati si metode.
* **Properties (*proprietati*)** - Caracteristici care definesc un obiect.
* **Methods (*metode*)** - Functii (actiuni) care se aplica obiectelor.
* **Events (*evenimente*)** - Actiuni ce declanseaza executia unei functii, efectuarea unei comenzi.  
  Evenimentele pot fi cele ale mouse-ului (click-ul sau miscarea cursorului), ale tastaturii (apasarea unui buton), ale Cadrelor (cand Playhead ajunge la un anumit cadru) sau "event loads" ("evenimente de incarcare", care se raporteaza la progresul de incarcare a fisierelor externe).

Iata un exemplu practic, in care vor fi folosite elementele prezentate in acest tutorial. Prin apasarea unui buton se va schimba lungimea si pozitia unui patrat.  
- *Observati cum sunt folosite elementele OOP: obiect, proprietate, eveniment; si cum se face legatura intre obiecte din Scena si script.*

**1.** Deschideti un document Flash nou (ActionScript 3.0) si desenati in scena un patrat.  
**2** Transformati patratul in Symbol **Movie Clip** (din meniul **Modify -> Convert to Symbol**, la ***Type*** puneti **Movie Clip**), apoi dati instantei acestui Movie Clip (patratul din Scena) numele **patrat**.  
  - Numele unei instante se da (se scrie) in panoul Properties, in casuta din partea de sus, unde este "***<Instance Name>***".  
Numele dat astfel unui obiect din Scena este unealta prin care se poate face legatura dintre el si codul ActionScript. Acel nume va reprezenta in script obiectul respectiv.  
**3** Langa patrat desenati si un cerc (dupa cum se vede in imaginea de mai jos) pe care-l transformati in **Buton** (din meniul **Modify -> Convert to Symbol**, la ***Type*** alegeti**Button**), apoi dati-i ca nume de instanta **buton** (*ca la patrat, in casuta din Properties apartinand instantei cerc*).  
  
**4.** Dati click-dreapta pe primul Cadru din Timeline (la care sunt de fapt create cele 2 obiecte din Scena) si alegeti **Actions**.  
**5.** In fereastra care se deschide (pt. scriere cod ActionScript) adaugati urmatorul cod AS3:

// Comanda pt. apelarea functiei "setWidth()" cand e apasat (CLICK) obiectul "buton"

buton.addEventListener(MouseEvent.CLICK, setWidth);

// Se defineste functia "setWidth()"

function setWidth(evt)

{

patrat.width = 115; // Da o valoare proprietatii "width" (lungime) a obiectului "patrat"

patrat.y = 100; // Da o valoare proprietatii "y" (distanta Y, fata de marginea de sus) a obiectului "patrat"

}

**6.** Apasati "**Ctrl+Enter**" pentru a vedea rezultatul, prezentarea Flash obtinuta.  
Va apare ceva asemanator cu prezentarea urmatoare (apasati click pe buton, cercul galben):

- Comanda "addEventListener()" este o functie din ActionScript 3 pentru lucru cu evenimente. Aceasta primeste ca parametrii evenimentul (aici "MouseEvent.CLICK") si functia care va fi apelata cand este produs acel eveniment (aici functia "setWidth()"). Conecteaza evenimentul cu functia , si este adaugata la obiectul "buton" (prin caracterul punct "."), care reprezinta cercul din scena (instanta la care s-a dat numele "buton").  
- "MouseEvent.CLICK" este o constanta ce precizeaza exact evenimentul la care se apeleaza functia.  
- Functia "setWidth()" executa codul scris intre acoladele ei ( **{}** ) cand este apelata. Aici sunt doua linii de cod.  
          - "patrat.width" - defineste (schimba) valoarea proprietatii "width" (lungime) a obiectului "patrat" (acesta fiind instanta din scena la care s-a dat numele "patrat").  
          - "patrat.y" - defineste (schimba) valoarea proprietatii "y" (distanta Y, fata de marginea de sus) a obiectului "patrat".  
Despre functii veti putea invata mai multe in lectiile urmatoare; aici e folosita doar pentru intelegerea utilizarii evenimentelor.

**Variabile, Comentarii si trace**

**1. Variabile**

**Variabilele** sunt elemente din limbaj in care sunt stocate date. Ca analogie, **variabila** este precum o cutie in care poate fi adaugata o piesa pentru a fi transportata dintr-un loc in altul; importanta este piesa adaugata, nu cutia in sine, singura. Asa si la **variabile**, importante sunt datele stocate in ele, care, prin numele variabilei pot fi folosite in diferite locatii din script.  
Precum o cutie poate fi de mai multe feluri, in functie de lucrul pe care-l transporta, asa si variabilele sunt de mai multe tipuri, in functie de datele pe care le contin. Astfel, sunt variabile de tip "String" (contin date de tip sir, text), de tip "Number" (contin doar numere), variabile de tip "Array" (stocheaza mai multe valori intr-o singura variabila), si altele.  
• Sintaxa generala pentru a crea (defini) o variabila in ActionScript 3 este urmatoarea:  
          **var nume:Tip = valoare;**  
sau  
          **var nume:Tip;**  
            **nume = valoare;**  
- "var" este cuvantul special rezervat din limbaj pentru a declarara variabile.  
- "nume" este numele dat variabilei. Poate fi orice cuvant, care sa nu inceapa cu un numar sau alt caracter special, sa nu contina spatii si sa fie diferit de cuvintele rezervate ale limbajului (precum "var", "function", si altele, ce pot fi identificate prin faptul ca se coloreaza in albastru).  
- "valoare" reprezinta valoarea (data) stocata in variabila.  
- La "Tip" se specifica tipul de date ce-l va contine variabila. Specificarea tpiului poate fi omisa, nu e obligatoriu sa fie adaugat "Tip" (se poate scrie si **var nume = valoare;**), dar este indicat deoarece va reduce cantitatea de memorie pe care Flash va trebui sa o foloseasca pt. a recunoaste tipul datelor din variabila. Acesta poate fi de mai multe feluri, dar principalele tipuri generice sunt:

* **String** - folosit pt. date ce contin orice text sau sir de caractere. Acestea sunt scrise intre ghilimele (simple sau duble).
* **Number** - pentru numere, orice tip de numere, intregi sau cu zecimale (numerele se adauga libere, fara ghilimele).
* **int** - doar pt. numere intregi.
* **uint** - pt. numere intregi pozitive.
* **Boolean** - pt. declaratiile logice **true** sau **false**.
* **Array** - folosit pt. a stoca mai multe valori, adaugate intre o pereche de paranteze patrate.
* **Date** - pt. valori care indica o data in timp.
* **RegExp** - folosit pt. valori cu expresii regulate
* **null** - pt. variabile care nu au vreo valoare atribuita, nule.
* **void** - acest tip e folosit daca o functie nu returneaza nici o valoare.
* **\*** - folosit daca variabila nu are un tip specific.
* **undefined** - indica variabile fara un tip specific si care nu au fost initializate.

• Important - ActionScript fiind un limbaj case-sensitive (cu diferenta intre majuscule si litere mici), trebuie sa se respecte exact sintaxa, deoarece, de ex. "String" e diferit de "string".  
De exemplu, pentru a declara o variabila ce contine un text (un sir), se defineste astfel:  
          **var un\_sir:String = 'Cursuri programare web www.marplo.net';**  
sau pentru numere intregi:  
          **var nr:Number = 78;**  
iar un Array, pentru mai multe valori in acelasi nume de variabila:  
          **var un\_arr:Array = ["cursuri web www.marplo.net", "tutoriale", 78];**

**2. Comentarii**

**Comentariile**, desi nu sunt importante la script, fiind ignorate la executia scriptului, sunt importante pentru subiectul uman.  
Programatorul poate adauga **comentarii**, explicatii in script, la diferite linii de cod, ca sa includa anumite indicii si explicatii in legatura cu scriptul respectiv, despre functiile si codul din el; usurand astfel intelegerea acestuia de catre alte persoane.  
• Comentariile se adauga:  
    - folosind doua slash-uri (//) la inceputul lor, cand sunt scrise pe o singura linie.  
            ***// Comentariu pe o linie***  
    - intre **/\*** si **\*/** cand sunt scrise pe mai multe randuri.  
            ***/\* Comentariu pe mai multe  
              linii, doua, trei, sau  
              oricate sunt nevoie \*/***

**3. trace()**

**trace()** este un tip special de functie a limbajului ActionScript. E folosita in general pentru "debug" (depanare /verificare si evaluare cod).  
**trace()** nu are nici un rol in prezentarea Flash creata, dar cand se testeaza lucrarea facuta (cu "Ctrl+Enter") afiseaza in panoul "**Output**" informatii despre argumentul (variabila, expresie de cod) adaugata intre paranteze. Pentru cine cunoaste PHP, este oarecum asemanatoare cu "var\_dump()".  
Sintaxa ei este urmatoarea:  
          **trace(param);**  
- "param" poate fi o variabila sau o expresie mai complexa.

- Iata un exemplu in care sunt folosite cele explicate in aceasta lectie (*variabile, comentarii, trace*).

**1.** Deschideti un document Flash nou, **ActionScript 3.0**  
**2.** Click-dreapta pe primul cadru din Timeline si alegeti **Actions**  
**3.** In fereastra care se deschide pentru scriere ActionScript, adaugati urmatorul cod:

/\* Comentariu pe mai multe linii

- Exemplu ActionScript cu variabile, comentarii si trace()

- Site: http://www.marplo.net/ \*/

// Se declara 2 variabile (String si Number)

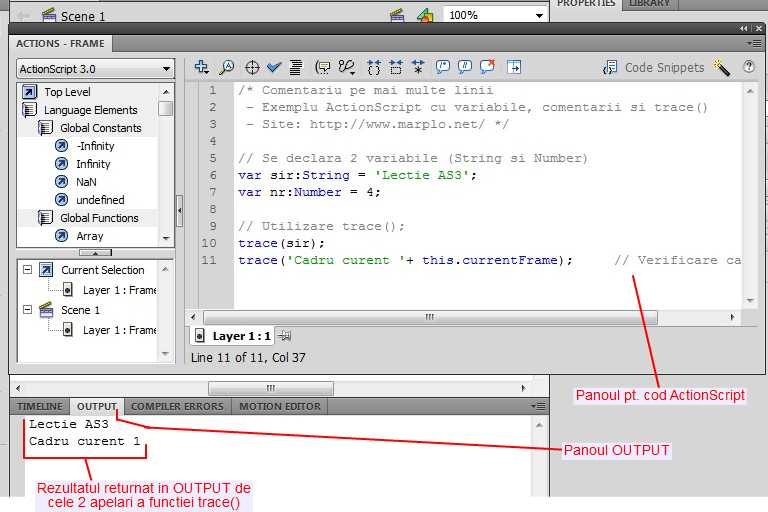
var sir:String = 'Lectie AS3';

var nr:Number = 4;

// Utilizare trace();

trace(sir);

trace('Cadru curent '+ this.currentFrame); // Verificare cadru curent

- Observati ca se poate adauga comentariu si intr-un rand cu cod, dar dupa el (dupa caracterul (;)).  
- A doua apelare "trace()" contine ca parametru o expresie mai compleza. Operatorul "+" (de unire) uneste la sirul "*Cadru curent*" valoarea data de formula "*this.currentFrame*" (returneaza cadrul curent din Timeline la care e adaugat scriptul).  
**4.** Apasati "**Ctrl+Enter**". Playerul Flash va apare gol, il puteti inchide; important la acest exemplu e panoul **OUTPUT** in care este afisat rezultatul dat de functia "***trace()***", dupa cum se vede in imaginea urmatoare.  


- Primul "trace()" returneaza valoarea variabilei "sir", iar al doilea, valoarea expresiei din parantezele lui.

**Constante si Operatori**

**1. Constante**

**Constantele** sunt ca o variabila cu valoare fixa. O **constanta** dupa ce a fost definita, pastreaza aceeasi valoare pe tot parcursul programului.  
Sintaxa generala pentru a crea o constanta este urmatoarea:  
          **const NUME:Tip = valoare;**  
- "const" este cuvantul special rezervat din limbaj pentru a declarara constante.  
- "NUME" este numele constantei. Poate fi orice cuvant, care sa nu inceapa cu un numar sau alt caracter special, sa nu contina spatii si sa fie diferit de cuvintele rezervate ale limbajului (precum "var", "const", "function", si altele, ce pot fi identificate prin faptul ca se coloreaza in albastru). Numele poate fi scris si cu litere mici, dar pentru a face diferenta intre variabile obisnuite si **constante**, programatorii cu experienta le scriu cu majuscule.  
- La "Tip" se specifica tipul de data atribuita (*o parte din ele, cele generale, sunt prezentate in lectia anterioara*). Specificarea tpiului poate fi omisa, nu e obligatoriu sa fie adaugat "Tip" (se poate scrie si **const NUME = valoare;**), dar este indicat deoarece va reduce cantitatea de memorie pe care Flash va trebui sa o foloseasca pt. a recunoaste tipul datelor din constanta.  
- "valoare" reprezinta valoarea atribuita constantei, si care ramane fixa.  
De exemplu, pentru a declara o constanta cu numele unui frate si alta cu varsta lui, se poate scrie un cod astfel:

const FRATE:String = 'Victor';

const FRATE\_ANI:Number= 33;

// Pentru "debug" (verificare in Output) se poate folosi "trace()"

trace(FRATE+ ' - '+ FRATE\_ANI); // Va afisa in panoul Output: *Victor - 33*

**2. Operatori**

**Operatorii** sunt simboluri (sau cuvinte-cheie) prin care se manipuleaza, combina si modifica datele din script. De exemplu, cei mai cunoscuti operatori sunt: + (adunare), - (scadere) si = (atribuire valoare).  
Operatorii pot fi folositi pentru a efectua operatii cu mai multe tipuri de date (numere, siruri, variabile, constante).  
Operatorii pot fi impartiti in mai multe grupe:

**- Operatori aritmetici**

* **+**   - adunare
  + 2 + 3 , var1 + var2
* **-**   - scadere
  + 2 - 3 , var1 - var2
* **\***   - inmultire
  + 2 \* 3 , var1 \* var2
* **/**   - impartire
  + 2 / 3 , var1 / var2
* **%**   - modulo (returneaza restul impartirii)
  + 8 % 3 (returneaza 2) , var1 % var2

- Daca sunt mai multi operatori aritmetici in aceeasi expresie, se respecta regulile de executie din aritmetica (*intai inmultire /impartire sau modulo, apoi adunare sau scadere. Daca sunt si paranteze, inainte de toate se executa codul din ele*).  
- Caracterul "+" e folosit si ca operator de concatenare (unire) pentru siruri.  
          De ex.: *var curs:String = "Tutoriale Flash " + "- www.marplo.net";       // Tutoriale Flash - www.marplo.net*

**- Operatori aritmetici compusi**

* **++**   - incrementare (adauga o unitate)
  + var1++; (echivalent cu *var1 = var1 + 1;*)
* **--**   - decrementare (scade o unitate)
  + var1--; (echivalent cu *var1 = var1 - 1;*)
* **+=**
  + var1 += nr; (echivalent cu *var1 = var1 + nr;* // "nr" poate fi un numar sau o variabila)
* **-=**
  + var1 -= nr; (echivalent cu *var1 = var1 - nr;* // "nr" poate fi un numar sau o variabila)
* **\*=**
  + var1 \*= nr; (echivalent cu *var1 = var1 \* nr;* // "nr" poate fi un numar, variabila)
* **/=**
  + var1 /= nr; (echivalent cu *var1 = var1 / nr;* // "nr" poate fi un numar, variabila)
* **%=**
  + var1 %= nr; (echivalent cu *var1 = var1 % nr;* // "nr" poate fi un numar sau o variabila)

**- Operatori relationali si de egalitate**

* **>**   - mai mare
  + 3 > 2 , var1 > var2
* **<**   - mai mic
  + 2 < 3 , var1 < var2
* **>=**   - mai mare sau egal
  + 3 >= 2 , var1 >= var2
* **<=**   - mai mic sau egal
  + 2 <= 3 , var1 <= var2
* **==**   - egalitate
  + 8 == 5+3 , var1 == var2
* **!=**   - inegalitate
  + 7 != 5+3 , var1 != var2
* **===**   - egalitate stricta, atat ca tip de date cat si la valoare
  + 8 === 5+3 , var1 === var2
* **!==**   - inegalitate stricta (la tip de data sau la valoare)
  + 8 !== "8" (*primul e de tip numar intreg, al doilea e sir*), var1 !== var2

- caracterul egal, simplu (=) este un operator de atribuire, atribuie o valoare unei variabile (ex.: *var un\_nume = ceva*).

**- Operatori logici**

* **&&**   - SI
* **||**   - SAU

- Acestia sunt cel mai des folositi cu instructiuni conditionale: "if()", "else if()".  
  
Iata un exemplu cu elementele prezentate in aceasta lectie (constante si cativa operatori).

**1.** Deschideti un document Flash nou, **ActionScript 3.0**  
**2.** Click-dreapta pe primul cadru din Timeline si alegeti **Actions**  
**3.** In fereastra care se deschide pentru scriere ActionScript, adaugati urmatorul cod:

const FORMA:String = 'patrat'; // Definire constanta

// Definire 2 variabile

var latura:Number = 8;

var masura:String = ' pixeli';

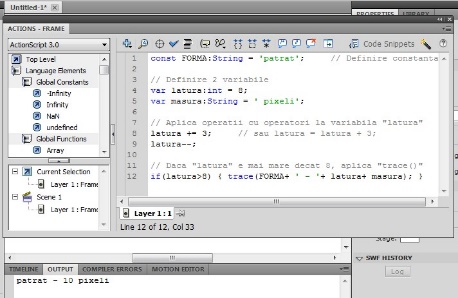
// Aplica operatii cu operatori la variabila "latura"

latura += 3; // sau latura = latura + 3;

latura--;

// Daca "latura" e mai mare decat 8, aplica "trace()"

if(latura>8) { trace(FORMA+ ' - '+ latura+ masura); }

**4.** Apasati "**Ctrl+Enter**". Player-ul Flash va apare gol, il puteti inchide; important la acest exemplu e panoul **OUTPUT** in care este afisat rezultatul dat de functia "***trace()***", dupa cum se vede in imaginea urmatoare.  


- Dupa ce au fost definite (declarate) constanta si variabilele, expresia "*latura += 3;*" adauga 3 unitati la valoarea variabilei "latura", iar "*latura--*" ii scade o unitate.  
- Observati ca dupa ce variabila e declarata cu "var", pentru operatii cu ea si modificarea ei nu se mai foloseste "var", ci doar numele variabilei.  
*Daca se incearca modificarea constantei "FORMA" (care trebuie sa ramana fixa), de ex. "FORMA = 'ceva';"*, in Output va apare mesaj de eroare.  
- Expresia "*if(latura>8)*" verifica daca latura e mai mare decat 8, si daca este mai mare, executa codul "*trace(FORMA+ ' - '+ latura+ masura);*". Acesta va returna in Output un sir format prin concatenarea (unirea) cu operatorul "+" a valorilor constantei si variabilelor, dupa cum sunt adaugate in paranteza.  
- Rezultatul din Output va fi sirul "*patrat - 10 pixeli*".  
*Cand valorile variabilelor sunt numere, operatiorul "+" executa adunare, dar daca vreuna din variabile e un sir, operatorul "+" face unirea (alaturarea) lor intr-un sir.*

**Instructiuni conditionale: if, else, switch**

**Instructiunile conditionale** determina programele sa testeze diferite conditii si in functie de acestea sa decida executia anumitor comenzi.  
Aceste **instructiuni conditionale** sunt:

* **if()** - executa comenzile dorite cand o conditie scrisa in paranteze este adevarata.
* **if() ... else** - executa anumite comenzi cand o conditie este adevarata si alte comenzi cand aceasta este falsa.
* **switch** - selecteaza care comanda va fi executata.

**1. Instructiunea "if"**

"**if()**" (*daca*) este cea mai simpla instructiune conditionala. Forma generala a acestei instructiuni este urmatoarea :

* **if (*conditie*) {  
      // Codul care va fi executat daca este Adevarata conditia  
  }**

- Unde 'conditie' poate fi orice expresie alcatuita cu operatori rationali, de egalitate si logici.  
          Ex.:

latura += 3; // sau latura = latura + 3;

latura--;

// Daca "latura" e mai mare decat 8, aplica "trace()"

if(latura>8) { trace(FORMA+ ' - '+ latura+ masura); }

- Daca rezultatul conditiei este TRUE se executa codul dintre acolade, in caz contrar, cand conditia returneaza FALSE, se trece peste acest cod.

**2. Instructiunea "if ... else"**

In exemplu anterior am vazut ca daca este adevarata conditia din parantezele instructiunii "if()", se executa codul dintre acoladele ei, iar in caz contrar nu se intampla nimic.  
Folosind combinatia "**if() ... else**" (*daca ... altfel*) putem stabili comenzi care sa fie executate si cand conditia instructiunii "if()" este FALSE.  
Forma generala a instructiuni "if() ... else" este urmatoarea :

* **if (*conditie*) {  
      // codul care va fi executat daca este Adevarata conditia  
  }  
  else {  
      // codul ce va fi executat daca conditia este falsa  
  }**

- Unde 'conditie' poate fi orice expresie logica.  
Daca rezultatul conditiei este TRUE se executa codul dintre primele acolade, care apartin de "if()", in caz contrar, cand conditia returneaza FALSE, sunt executate comenzile din acoladele de la "else".  
De exemplu:

var ziua:String = 'o\_zi';

if(ziua=='duminica') {

trace('Stau acasa');

}

else {

trace('Merg la lucru');

}

- Daca "ziua" are valoarea "*duminica*" va afisa in Output sirul "*Stau acasa*", altfel, se executa codul din acoladele lui "else" (afiseaza in Output "*Merg la lucru*").

**3 Formula "else if()"**

Cu "if() ... else" sunt posibile executia doar a doua optiuni, cea de la "if()" sau de la "else". Dar sunt stuatii in care avem mai multe optiuni, in acest caz se foloseste formula "**else if()**" (*altfel daca*).  
Cu aceasta se pot crea si alte optiuni suplimentare intre cele doua. Sintaxa generala pentru folosirea acestei formule este:

* **if (*conditie 1*) {  
      // codul care va fi executat daca este Adevarata conditia 1  
  }  
  else if (*conditie 2*) {  
      // codul ce va fi executat daca prima conditie este Falsa si este Adevarata conditia 2  
  }  
  else if (*conditie 3*) {  
      // codul care va fi executat daca primele doua conditii sunt False si este Adevarata conditia 3  
  }  
  // ...   
  else {  
      // codul executat daca toate conditiile sunt False  
  }**

- Pot fi adaugate oricate optiuni "else if".  
De exemplu, in urmatorul cod,

var ziua:String = 'o\_zi';

if(ziua=='duminica') {

trace('Stau acasa');

}

else if(ziua=='luni') {

trace('Merg la cursuri');

}

else if(ziua=='sambata') {

trace('Merg la bunici');

}

else {

trace('Merg la lucru');

}

- In functie de valoarea lui "ziua", va fi afisat in Output sirul din optiunea cu "trace()" corespunzaoare. Daca "ziua" nu are vreuna din valorile: *duminica, luni, sambata*, se executa codul din acoladele lui "else".

**4. Instructiunea switch**

Aceasta instructiune e folosita pentru a compara o valoare cu altele dintr-o lista, si in functie de acea valoare se executa codul asociat ei in lista "**switch**".  
Sintaxa generala a instructiuni "switch" este urmatoarea.

* **switch (expresie) {  
  case valoare1:  
      cod executat daca expresie = valoare1  
      break;  
  case valoare2:  
      cod executat daca expresie = valoare2  
      break;  
  case valoare3:  
      cod executat pt. expresie = valoare3  
      break;  
  default :  
      cod executat daca expresie e diferit de valoare1, valoare2 sau valoare3  
  }**

- Prima data este evaluata expresia scrisa intre paranteze rotunde, la "switch()", apoi valoarea expresiei este comparata pe rand cu fiecare valoare determinata de "case". Daca se gaseste o identitate se executa codul asociat acelui "case". Apoi se iese din instructiunea "switch".  
Daca, parcurgand fiecare "case", nu se gaseste o egalitate, se executa codul de la "default".  
Prin folosirea lui "break" se opreste parcurgerea corpului instructiunii atunci cand s-a gasit o valoare egala cu 'expresie' si se iese din "switch".  
- Instructiunea "switch" poate inlocui un sir de conditii cu "else if".  
De exemplu:

var ziua:String = 'o\_zi';

switch (ziua) {

case 'duminica':

trace('Stau acasa');

break;

case 'luni':

trace('Merg la cursuri');

break;

case 'sambata':

trace('Merg la bunici');

break;

default:

trace('Merg la lucru');

}

**5. Operatori logici si instructiunile conditionale**

Operatorii logici sunt "**&&**" si "**||**". Rolul lor este asemanator cu cel al operatorilor relationali si de egalitate, in sensul ca si acestia compara valoarea a doi operanzi.  
Operatorii logici compara operatii, expresii de cod, iar rezultatul lor este unul din valorile TRUE (*Adevarat*) sau FALSE (*Fals*).

* **exp1 && exp2**   - Returneaza True daca si "exp1" si "exp2" sunt Adevarate; altfel, returneaza False.
* **exp1 || exp2**   - Returneaza True daca oricare din "exp1" sau "exp2" e Adevarat; altfel, daca amandoua sunt false, returneaza False.
* **exp1 && exp2 || exp3**   - Returneaza True daca expresia (exp1 && exp2) sau "exp2" e Adevarata; altfel, returneaza False.

De exemplu:

* **3>2 && 5<=8**   True (*ambele adevarate*)
* **3>2 || 5>=8**   True (*una din expresii, prima e adevarata*)
* **3<2 && 5>8 || 4<9**   True (*a treia expresie, dupa || (SAU), e adevarata*)

Datorita rezultatului True sau False pe care-l dau acesti operatori logici, ei sunt folositi cu instructiuni conditionale, contribuind la efectuarea unor conditii mai complexe in parantezele instructiunii "if()" si "else if()".  
Iata un exemplu:

var ziua:String = 'luni';

var nr:Number = 8;

if(ziua=='duminica' && nr==8) {

trace('Stau acasa');

}

else if(ziua=='luni' || nr>9) {

trace('Merg la cursuri');

}

else {

trace('Merg la lucru');

}

- In Output va fi afisat "*Merg la cursuri*" deoarece conditia "*(ziua=='duminica' && nr==8)*" returneaza False [nu sunt amandoua expresii adevarate], iar "*(ziua=='luni' || nr>9)*" returneaza True [cu "||" e suficient una sa fie adevarata].

**6. Operatorul conditional "? :"**

O alta metoda de a executa un cod in functie daca o expresie este Adevarata sau Falsa e **operatorul "? :"** (*intrebare doua puncte*).  
Acest operator conditional, desi arata diferit de ceilalti, este o forma prescurtata a instructiunii "if() else". Sintaxa generala de folosire a lui este:

* **expresie-conditionala ? daca-TRUE : daca-FALSE;**

- Operatorul conditional evalueaza expresia conditionala. Daca expresia are valoarea TRUE, operatorul conditional returneaza valoarea de la "**daca-TRUE**"; in caz contrar, returneaza valoarea de la "**daca-FALSE**".  
Iata un exemplu:

var a:Number = 9.3;

var b:Number = 8;

// Se defineste "c", in functie de "a>b"

var c:String = (a > b) ? 'valoare pt. True' : 'val. pt. False'

trace(c); // valoare pt. True

/\* Echivalent cu:

if(a>b) { var c:String = 'valoare pt. True'; }

else { var c:String = 'val. pt. False'; }

\*/

- In acest cod se defineste o variabila "c" in functie de relatia dintre alte doua variabile (a si b). Daca "a" este mai mare decat "b", "c" va primi valoarea "*valoare pt. True*", in caz contrar, "c" va avea "*val. pt. False*".  
- Pentru verificare, functia "trace(c)" afiseaza in Output valoarea pe care "c" a primit-o. Comentariu care este dupa "trace(c)" arata asemanarea cu "if() else".  
  
Pe langa atribuirea unei valori in functie de rezultatul unei expresii, acest operator conditional poate fi utilizat in aceeasi forma si la determinarea apelarii unei anumite functii, dupa rezultatul unei expresii logice. Formula este:

* **(expresie-logica) ? dacaTrue() : dacaFalse();**

- "dacaTrue()" si "dacaFalse()" pot fi considerate doua functii.  
Exemplu:

(var1 == var2) ? oFunctie() : altaFunctie();

- Daca "var1" este egal cu "var2", va fi apelata "oFunctie()", in caz contrar este executata "altaFunctie()".

**Instructiuni repetitive - FOR**

**Instructiunile repetitive** se folosesc atunci cand se doreste efectuarea unei comenzi de mai multe ori.  
**Instructiunea for()** (denumita si *bucla for()*) poate fi conasiderata cea mai des folosita instructiune repetitiva. In ActionScript 3 putem folosi bucla "for()" cu mai multe variante:

* **for** - la fel ca si in alte limbaje de programare, executa codul de un numar specificat de ori  
  **for ... in** - executa cate un set de instructiuni pentru fiecare proprietate dintr-un obiect sau element din Array  
  **for each ... in** - cu aceasta se pot manipula direct proprietatile sau functiile din cadrul unui obiect.

**1. Instructiunea for**

Aceasta are urmatoarea forma generala:

* **var i:Number  
  for (i=nr\_start; i<nr\_end; i++) {  
      cod care va fi executat  
  }**
* Sau :
* **for (var i:Number=nr\_start; i<nr\_end; i++) {  
      cod care va fi executat  
  }**

- "i" este o variabila de initializare a buclei "for()", folosita ca un contoar pentru numarul de repetari. Poate fi utilizata si alta litera sau cuvant, ca la orice variabila.  
*Se poate folosi si direct "****var i = nr\_start;****", dar din motivele precizate in lectia despre variabile, este indicat sa se specifice si tipul.*  
- "**nr\_start**" este un numar initial de la care incep repetarile executiei codului dintre acolade.  
- Conditia "**i<nr\_end**" verifica si asigura ca numarul de repetari sa se incadreaza intr-o anumita valoare, stabilita la "nr\_end". Cat timp rezultatul acestei conditii este TRUE se executa inca o data codul dintre acolade.  
- "**i++**" incrementeaza cu o unitate valoarea la care a ajuns "i", apoi aceasta valoare este verificata din nou de "i<nr\_end", pana cand rezultatul este FALSE. Atunci se opreste executia buclei.  
- *In loc de "i<nr\_end" se poate folosi si alta conditie relationala, de exemplu "i<=nr\_end"*  
Iata un exemplu simplu de utilizare a instructiunii "for":

var total:Number = 1; // Se defineste o variabila

// Se executa o bucla "for()" de 4 ori, care dubleaza mereu "total"

for(var i:Number=0; i<4; i++)

{

total \*= 2;

trace(total);

}

// In Output afiseaza *2 4 8 16*

- Dupa ce e difinita variabila "total", la instructiunea "for()" se atribue unei variabile 'i' valoarea 0, se verifica conditia (i<4) care este adevarata si se executa corpul instructiunii (dintre acolade).  
- Codul dntre acolade dubleaza valoarea lui "total", iar functia "trace(total)" afiseaza in Output valoarea lui, apoi se incrementeaza 'i' cu o unitate (i++), acum 'i' are valoarea 1.  
- Instructiunea "for()" verifica din nou conditia (i<4) si fiindca este adevarata, se executa iar codul dintre acoladele ei, apoi se incrementeaza valoarea lui 'i', ..., si tot asa pana cand 'i' va ajunge sa aibe valoarea 4 care la verificarea conditiei (i<4) returneaza FALSE, moment in care se termina executia instructiunii "for()".  
- Ca rezultat, in Output afiseaza *2 4 8 16*.  
  
• Adesea, bucla "for()" este folosita pentru parcurgerea elementelor Array, dupa cum se vede in urmatorul exemplu:

// Se defineste o variabila de tip Array

var arr:Array = ['site', 'cursuri', 'tutoriale'];

// Se parcurge Array-ul, de la 0 (primul element)

// pana la numarul de elemente din Array (aflat cu "arr.length")

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

{

// Afiseaza in Output fiecare element parcurs,

// aflat cu numarul de ordine "i" (care e incrementat la fiecare executie a buclei)

trace(arr[i]);

}

// In Output afiseaza "site cursuri tutoriale"

- Variabila de tip Array va fi prezentata amanuntit in lectiile urmatoare. Explicatiile necesare pentru obiectul acestei lectii sunt cele din cod.

*In interiorul instructiunii****for****(ca si la instructiunea "if"), intre acolade pot fi introduse si alte instructiuni "for", sau alte instructiuni conditionale, acest lucru se numeste****imbricarea instructiunilor****.*

**2. Instructiunea for ... in**

Pentru utilizarea acestei instructiuni trebuie sa aveti cunostinte despre obiectele ActionScript asa ca va trebui sa reveniti asupra ei dupa ce le veti invata - in lectiile urmatoare .  
Cu "**for ... in**" se poate afla numele fiecarui element dintr-un obiect sau cheie din Array si se executa cate un set de instructiuni pentru fiecare element sau proprietate parcursa. Daca obiectul nu are nici o proprietate nu se desfasoara nici o repetare.  
Instructiunea "**for ... in**" are urmatoarea forma generala:

* **for (*var nume\_el:String* in *Obiect*) {  
      Instructiuni  
  }**

- Unde "nume\_ el" este o variabila de tip sir. Pentru fiecare repetare a executiei instructiunii, la "nume\_el" i-se atribuie ca valoare numele elementului sau proprietatii curente parcurse, din "Obiect" (care poate fi si un Array).  
- Executia buclei se face pana cand sunt parcurse toate.  
Iata un exemplu practic (*explicatiile necesare sunt in comentariile din cod*).

// Se defineste un obiect

var obj:Object = {'site':'www.marplo.net', 'gen':'cursuri', 'web':'tutoriale'};

// Cu "for ... in" se parcurge Obiectul

for(var elm:String in obj)

{

// Se aplica "trace()", care, pentru fiecare element curent parcurs

// returneaza numele si valoarea lui

trace(elm+ ' - '+ obj[elm]);

// obj[elm] returneaza valoarea elementului din "obj" care are numele dat de "elm"

}

// In Output va afisa: "gen - cursuri site - www.marplo.net web - tutoriale"

**3. Instructiunea for each ... in**

Aceasta este o alta versiune a lui "for ... in", noua in ActionScript 3.  
Sintaxa generala este:

* **for each (*var element:Object* in *Obiect*) {  
      Instructiuni  
  }**

- Desi pare asemanatoare cu standardul "for ... in", principala diferenta la "**for each ... in**" este faptul ca prin aceasta se pot manipula direct proprietatile sau functiile din cadrul unui obiect.  
- Aici "element" este proprietatea din "obiect", cu tot ce contine ea, nu doar numele ei.  
Pentru a cunoaste mai bine diferenta, iata exemplul anterior, dar folosind "for each ... in".

// Se defineste un obiect

var obj:Object = {'site':'www.marplo.net', 'gen':'cursuri', 'web':'tutoriale'};

// Cu "for each ... in" se parcurge Obiectul

for each (var elm:Object in obj)

{

// Se aplica "trace()", care, pentru fiecare element curent parcurs returneaza valoarea lui

trace(elm);

}

// In Output va afisa: "tutoriale cursuri www.marplo.net"

- Diferenta se observa la "trace(elm)", aici "elm" a returnat continutul (valoarea) elementului, pe cand la "for ... in" returna numele lui.

**Instructiuni repetitive - WHILE**

Aceasta instructiune are doua variante: WHILE simplu si DO ... WHILE.

**1. While**

**Instructiunea while** (denumita si "*bucla while*") este asemanatoare cu "for()", la fel ca aceasta, si **while()** repeta executia unui cod (din corpul buclei) cat timp o expresie este adevarata.  
Diferenta fata de "for()" este aceea ca "**while()**" are un singur parametru intre parantezele rotunde. Sintaxa generala este:

* **while (conditie) {   
      codul care va fi executat   
  }**

- Codul dintre acolade va fi executat in continuu, pana "**conditie**" returneaza False sau este oprit intentionat.  
*Atentie, daca "conditie" ramane mereu True, repetarea executiei buclei se va face la infinit, pana cand eventual Flash va genera eroare si opreste tot scriptul. De aceea, in cadrul corpului buclei (intre acolade) se adauga in general o declaratie care modifica elementele de la "conditie".*  
Iata un exemplu de cod simplu cu "while()".

var total:Number = 1; // Se defineste o variabila de lucru

// Se defineste si o variabila care va fi folosita in "conditie"

// De asemenea, va avea si rol do contor, pt. controlul numarului de repetari

var i:Number=0;

// Se executa o bucla "while()", care dubleaza mereu "total"

while(i<4)

{

total \*= 2;

trace(total);

i++; // Incrementeaza variabila contor cu o unitate

}

// In Output afiseaza *2 4 8 16*

- Pentru controlul numarului de repetari, se declara variabila 'i' dandu-i valoarea 0.  
- Instructiunea "while" verifica conditia (aici "i<4") care este adevarata si permite executarea codului dintre acolade, care dubleaza valoarea lui "total", iar functia "trace(total)" afiseaza in Output valoarea lui, apoi se incrementeaza 'i' cu o unitate (i++), acum 'i' are valoarea 1.  
- Instructiunea "while()" verifica din nou conditia (i<4) si fiindca este adevarata, se executa iar codul dintre acoladele ei, apoi se incrementeaza valoarea lui 'i', ..., si tot asa pana cand 'i' va ajunge la valoarea 4 care la verificarea conditiei (i<4) returneaza FALSE, moment in care se termina executia instructiunii "while()".  
- Ca rezultat, in Output afiseaza *2 4 8 16*.

**2. Do ... While**

Asemanatoare in mare parte cu instructiunea "while", instructiunea "**do ... while**" intai executa codul din corpul instructiunii (care e dupa DO), apoi se verifica conditia. Procesul se repeta pana cant conditia returnbeaza FALSE.  
Diferenta fata de "while()" simplu este faptul ca la "**do ... while**" corpul functiei este executat cel putin o data, chiar daca conditia nu este adevarata.  
Forma generala a acestei instructiuni este:

* **do {   
      codul care va fi executat   
      }   
  while (conditie);**

Iata un exemplu:

var total:Number = 1; // Se defineste o variabila de lucru

// Se defineste si o variabila care va fi folosita in "conditie"

// De asemenea, va avea si rol do contor, pt. controlul numarului de repetari

var i:Number = 1;

// Se executa o bucla "do ... while", care dubleaza mereu "total"

do

{

total \*= 2;

trace('total este '+ total);

i++; // Incrementeaza variabila contor cu o unitate

} while(i<0);

// In Output afiseaza *total este 2*

- Aceasta functie afiseaza in Output "*total este 2*".  
- Observati ca desi conditia (x<0) este de la inceput Falsa ('i' fiind definit cu valoarea 1), codul dintre acolade este totusi executat o singura data.

**Utilizare Break si Continue**

Instructiunile "for" si "while" sunt adesea folosite si cu alte instructiuni "complementare" care pot fi executate impreuna cu acestea, anume **break** si **continue**.  
Cand este folosita o instructiune repetitiva, aceasta continua sa se repete pana cand conditia returneaza FALSE. Aceasta executie poate fi schimbata cu ajutorul instructiunilor **break** si **continue**, care dau posibilitatea intreruperii ciclului sau iesirea din acesta inainte de a se ajunge la sfarsit (la ultima acolada).

**1. Break**

**Break** - intrerupe definitiv executarea buclei.  
Exemplu cu "for()" si "break":

var total:Number = 1; // Se defineste o variabila

// Se executa o bucla "for()" de 4 ori, care dubleaza mereu "total"

for(var i:Number=0; i<4; i++)

{

total \*= 2;

if(total==8) break; // Opreste executia completa cand total=8

trace('i='+ i+ ' - total='+ total);

}

// In Output afiseaza "*i=0 - total=2 i=1 - total=4*"

- Expresia "*if(total==8) break;*" verifica daca "total" are valoarea 8, si cand egalitatea este confirmata, se geneereaza executia instructiunii "break", care va determina oprirea in acel moment a executiei codului care este dupa ea in acolade, implicit si oprirea altor repetari a buclei "for()", chiar daca conditia dintre paranteze e inca asdevarata.  
- Cu acest cod din finctia "trace()" (*trace('i = '+ ' - total = '+ total);*) se poate cunoaste ce valoare are "i" si "total" dupa fiecare executie a buclei "for()".  
- In Output afiseaza "*i=0 - total=2 i=1 - total=4*". Desi "total" ajunge la valoarea 8 cand se opreste complet executia instructiunii "for()", deoarece "trace()" este dupa "break", nu mai e afisata si aceasta valoare.

**2. Continue**

**continue** - intrerupe executia repetarii curente, inainte de a fi executate celelalte operatii de dupa ea din corpul (acoladele) buclei, apoi se verifica din nou conditia si se continua cu executarea ciclului.  
Studiati si urmatorul exemplu, in care este folosit "continue" cu "while()".

var total:Number = 1; // Se defineste o variabila de lucru

// Se defineste si o variabila care va fi folosita in "conditie"

// De asemenea, va avea si rol do contor, pt. controlul numarului de repetari

var i:Number=0;

// Se executa o bucla "while()", care dubleaza mereu "total"

while(i<4)

{

total \*= 2;

if(total==4 || total==8) continue; // Sare peste executia codului urmator cand "total" are valoarea 4 sau 8

trace('i='+ i+ ' - total='+ total);

i++; // Incrementeaza variabila contor cu o unitate

}

In Output afiseaza:

i=0 - total=2  
i=1 - total=16  
i=2 - total=32  
i=3 - total=64

- Observati ca atunci cand "total" are valoarea 4 sau 8, "*if(total==4 || total==8)*, "prin instructiunea "*continue*" se intrerupe executia codului care urmeaza dupa aceasta (aici "*trace('i='+ i+ ' - total='+ total);*" si "*i++;*") din repetarea curenta, dar se continua cu verificare conditiei instructiunii "while" si executia ciclului pana cand "i" va ajunge la 4.

**Functii - Creare si Utilizare**

O **functie** este o bucata de cod, sau un set de instructiuni care sunt executate doar atunci cand respectiva functie este apelata.  
*Functiile fac lucruri sa se intample doar atunci cand se vrea asta.*  
O alta utilitate a **Functiilor** este aceea ca ajuta la divizarea mai multor sarcini pe care trebuie sa le faca un program.  
O functie poate contine mai multe instructiuni si comenzi care ulterior pot fi utilizate usor si de mai multe ori prin apelarea functiei respective.  
Daca un program necesita multe linii de cod, folosind functiile putem imparti codul in parti mai mici pe care le putem utiliza separat acolo unde este nevoie.  
Pot fi doua grupe generale de **functii**:

* **predefinite** - care sunt incluse direct in limbajul de programare (cum e de exemplu "trace(), stop(), play(), ...").
* **create de programator**
  + care returneaza o valoare
  + care nu returneaza nici o valoare

**1. Creare (definire) Functii**

O functie se creaza utilizand cuvantul special **function** (functioneaza la fel ca "var" pt. variabile), urmat de numele ei si corpul functiei.  
Sintaxa pentru a crea o functie are 2 variante:

- Forma simpla, generala

* **function numeFunctie(param1, param2, ...) {   
      Codul care va fi executat  
  }**

- Forma complexa (sau completa)

* **function numeFunctie(param1:tip1, param2:tip2, ...):Tip {   
      Codul care va fi executat  
  }**

- "numeFunctie" este numele functiei care e folosit pentru apelarea ei. Respecta aceleasi reguli ca si la numele de creare a variabilelor.  
- "param1, param2, ..." sunt parametri ei (variabile) folosite de functie si a caror valoare este transmisa si preluata cand e apelata. Parametrii nu sunt obligatorii, o functie poate fi definita si fara parametri, dar se pastreaza parantezele rotunde. Sau pot fi si valori fixe, ex.: "*(param1, "un\_sir", 8)*".  
- "tip1", "tip2", etc., de la forma complexa, definesc tipul de variabila a parametrului respectiv (de ex.: *param:String*, "param" este parametrul, "String" reprezinta tipul de date pe care-l poate contine).  
- "Tip", de la forma complexa (cel dupa paranteze rotunde), defineste tipul de data pe care-l va returna functia (precum "tipul" de date al unei variabile). Cand functia nu are o declaratie "*return*" se foloseste "void" ca tip general.  
*Oricare din "tip1", "tip2", "Tip", pot fi omise.*  
  
Iata un exemplu cu o functie simpla.

// Se declara o functie, cu numele "salut"

function salut(mesaj:String)

{

trace(mesaj);

}

- Cand functia este apelata, dupa cum puteti vedea in continuare, se executa codul dintre acoladele ei.

**2. Apelare Functii**

Dupa ce a fost creata o functie, pentru a fi utilizata ea trebuie apelata.  
Apelarea unei functii se face utilizand numele ei.

• O functie fara parametru se apeleaza astfel:  
                **numeFunctie();**  
• O functie care contine parametri (denumita si "**modulara**") se apeleaza in felul urmator:  
                **numeFunctie(arg1, arg2, ...);**  
- unde "arg1, arg2, ..." sunt argumentele (variabile, valori, siruri) transmise functiei (conform cu numarul si ordinea parametrilor ale ei), care in general sunt folosite in codul din corpul functiei.  
  
*In locul unde este apelata o functie va fi executat codul din ea.*  
De exemplu, iata cum poate fi apelata functia declarata anterior:

// Se declara o functie, cu numele "salut"

function salut(mesaj:String)

{

trace(mesaj);

}

salut("Bine ai venit"); // Se apeleaza functia, cu argumentul "Bine ai venit"

- La apelare, sirul "*Bine ai venit*" e transmis ca argument, functia il preia si-l retine in "mesaj" (functioneaza ca si o variabila), care e transmis la codul dintre acoladele ei; aici la "trace(mesaj)", care va afisa in Output "*Bine ai venit*".

*O functie poate fi apelata de mai multe ori. Acesta fiind principalul avantaj, ca, desi codul ei este scris o singura data, poate fi utilizat de oricate ori e necesar.  
- Cum ar fi la exemplu anterior:****salut("Bine ai venit");******salut("Drum bun");***

**3. Utilizare return**

Pentru ca o functie sa returneze special o valoare, un rezultat, foloseste instructiunea **return**. Aceasta specifica valoarea pe care o returneaza functia cand este apelata.  
Iata un exemplu din care puteti intelege modul de aplicare a instructiunii **return**:

// Se declara o functie, cu numele "suma" si 2 parametri: "a", "b"

function suma(a:Number, b:Number)

{

// Se adauga suma lui 'a' si 'b' intr-o variabila 'c'

var c:Number = a+b;

return c; // Functia returneaza valoarea lui 'c'

}

- La apelare, de ex.: ***suma(7, 8);*** va returna valoarea lui "*c = 7+8*" (15).

***return****se poate utiliza si in functii fara parametru.*

• Valoare returnata de o functie, care foloseste "return", poate fi atribuita direct unei variabile; dupa cum este prezentat in exemplul urmator.

// Se declara o functie, cu numele "suma" si 2 parametri: "a", "b"

function suma(a:Number, b:Number)

{

// Se adauga suma lui 'a' si 'b' intr-o variabila 'c'

var c:Number = a+b;

return c; // Functia returneaza valoarea lui 'c'

}

// Se apeleaza functia "suma" ca valoare a unei variabile "d"

var d:Number = suma(7.5, 8);

trace(d); // Verificare rezultat stocat in "d" (15.5)

- In Output va apare "15.5", valoarea variabilei "d" care este cea returnata de functia apelata.

**Definire tip date returnate**

Cand se defineste un tip de data pentru o variabila, constanta sau parametru, cu doua-puncte (:) dupa numele elementului, indica programului Flash ce tip de informatie ar trebui stocata in acel element. La fel se poate proceda si cu functiile care folosesc declaratia "return".  
Caracterul doua-puncte (:) si tipul se scriu dupa parantezele rotunde ale functiei, forma generala fiind:

* **function numeFunctie(param1, param2, ...):Tip {  
      Codul care va fi executat  
      return valoare;  
  }**

- "Tip" poate fi orice tip de data folosit pt. variabile (String, Number, int, void, \*, etc.) - De exempu:

* - Daca se doreste ca tipul de data returnat sa fie sir, se specifica **String**
* - Daca nu se stie sigur tipul datei ce va fi returnata, se foloseste caracterul asterix **\*** (reprezinta tip nedefinit)
* - **void** se adauga la functii care nu folosesc comanda "return".  
  *Daca se foloseste "void" la functii cu "return", va genera eroare.*

Iata exemplul anterior, de data aceasta cu specificarea tipului de data returnat de functie.

// Se declara o functie, cu numele "suma" si 2 parametri: "a", "b"

function suma(a:Number, b:Number)**:Number**

{

// Se adauga suma lui 'a' si 'b' intr-o variabila 'c'

var c:Number = a+b;

return c; // Functia returneaza valoarea lui 'c'

}

// Se apeleaza functia "suma" ca valoare a unei variabile "d"

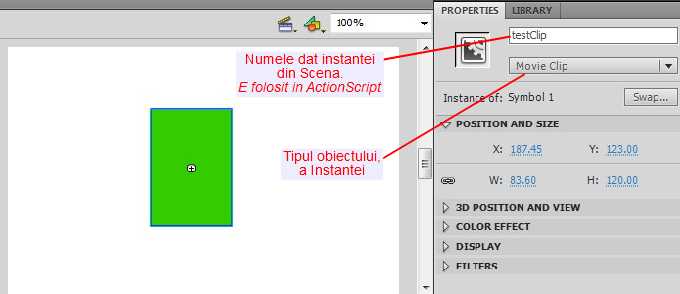
var d:Number = suma(7.5, 8);

trace(d); // Verificare rezultat stocat in "d" (15)

- "**:Number**" dupa parantezele rotunde seteaza (si forteaza) ca tipul de data returnat sa fie "Integer" (numar natural fara virgula).  
- Se observa ca desii, comform argumentelor de la apelare, "c = 7.5+8" da valoarea 15.5, functia returneaza variabilei "d" valoarea 15, (partea din numar fara virgula, dupa tipul "int" specificat).

**Functii - Utilizare avansata**

In aceasta lectie sunt prezentate lucruri mai **avansate** legate de utilizarea functiilor, ca o a doua parte a lectiei precedente.  
Deoarece scopul utilizarii Flash e acela de a afisa intr-un player sau pagina web efecte vizuale si animatii, in exemplele prezentate in aceasta lectie e folosita instanta unui obiect Movie Clip din Scena (*metoda mai atragatoare decat testarea in Output, cu "trace()"*).  
Inainte de a trece la studierea si testarea exemplelor, deschideti un document Flash nou, desenati in Scena un dreptunghi pe care-l transformati in Movie Clip (din meniul **Modify -> Convert to Symbol**, iar la ***Type*** alegeti **Movie Clip**). In panoul Properties dati numele de instanta **testClip** dreptunghiului din Scena (in caseta unde apare "***<Instance Name>***, dupa cum este in imaginea urmatoare").



**1. Setare parametri cu valoare fixa**

In ActionScript 3 se pot seta parametrii unei **functii** cu valoare fixa, similar cu definirea unei variabile.  
Acest lucru face ca sa nu mai fie necesar adaugarea unui argument pentru acel parametru la apelare, parametrul fix va fi transmis automat codului dintre acolade.  
Iata un exemplu (*codul va fi aplicat in documentul creat mai sus, cu Instanta "testClip", de tip MovieClip*).  
- Click-dreapta pe primul Cadru din Timeline si alegeti **Actions**, se deschide fereastra pentru ActionScript, in care adaugati urmatorul cod:

// Se creaza o functie cu doi parametri, al doilea cu valoare fixa

function testF(dy:Number, rotatie:Number = 45):void

{

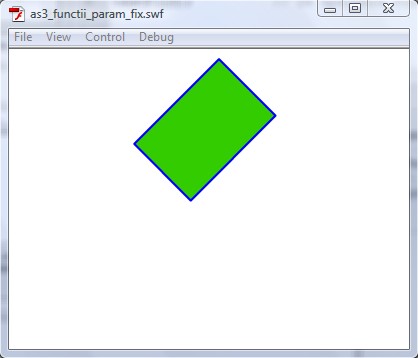
// Se aplica valorile parametrului la proprietatile "y" si "rotation" a Instantei din Scena

testClip.y = dy;

testClip.rotation = rotatie;

}

testF(10); // Se apeleaza functia

- Cu "testClip.y" se defineste distanta Y, fata de latura de sus, iar "rotation" este o proprietate pentru setare (adaugare) rotire.  
Testati rezultatul apasand "Ctrl+Enter", dreptunghiul din scena va fi mutat la distanta "y=10" si rotit 45 grade, dupa cum e scris in codul functiei (precum se vede in imaginea urmatoare).  
  
- Observati ca la apelare "testF(10)" e transmis doar un singur argument (10), pentru "dy", desi functia are 2 parametri, al doilea "rotatie" are deja valoare si e transmisa automat la codul din ea.  
- Daca se specifica la apelarea functiei un argument si pentru parametrul fix, valoarea argumentului transmis va inlocui pe cea setata la parametru. Incercati de ex.: "***testF(10, 8);***", 8 va inlocui valoarea 45 de la "rotatie".

*Important - La o functie cu mai multi parametri (fixi si variabili), cei fixi trebuie scrisi mereu la urma, dupa cei variabili.  
- De ex.:  
            function testF(dy:Number, rotatie:Number = 45, dx:Number)  
nu e valida deoarece dupa "rotatie" (care e cu valoare fixa) urmeaza "dx" (care e variabil).*

**2. Utilizare (... parametru)**

In ActionScript 3 e adaugata o noua facilitate pentru functii, denumita "*rest parameters*" (*resturi de parametri*).  
Acest "rest parametrs" e un simbol trei-puncte (...) ce reprezinta un numar variabil de parametri ca un Array, urmat de un nume pentru acel Array.  
Aceasta permite apelarea functiei cu oricati parametri se doreste, fiind toti stocati intr-un Array.  
- *Despre Array vor fi prezentate mai multe explicatii in lectiile urmatoare.*  
Iata un exemplu (*codul va fi aplicat in documentul creat la inceput, cu Instanta "testClip", de tip MovieClip*).  
- Click-dreapta pe primul Cadru din Timeline si alegeti **Actions**, se deschide fereastra pentru ActionScript, in care adaugati urmatorul cod (daca exista deja alt cod AS3, stergeti-l):

// Se creaza o functie cu un "rest parameters"

function testF2(... numere):void

{

// Se defineste o variabila in care se va adauga suma numerelor trimise ca argumente

var latura:Number = 0;

// Se parcurge cu "for each ... in" fiecare element din Array "numere"

// La fiecare parcurgere se adauga valoarea elementului curent in "latura"

for each (var num:Number in numere)

{

latura += num;

}

// Se defineste lungimea si inaltimea instantei dreptunghi din Scena "testClip"

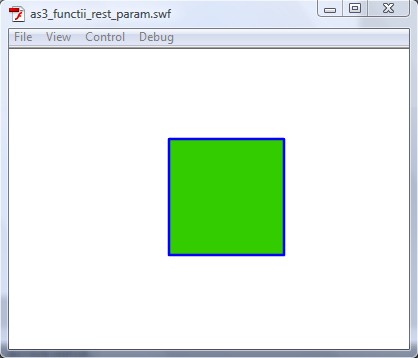
// cu valoarea variabilei "latura", obtinuta din suuma argumentelor

testClip.width = latura;

testClip.height = latura;

}

testF2(8, 20, 40, 50); // Se apeleaza functia, cu mai multe numere ca argumente

- Vaorile transmise functiei la apelare sunt continute in Array-ul denumit "numere".  
- "width" si "height" sunt proprietati pentru lungime si inaltime.  
Testati rezultatul apasand "Ctrl+Enter". Figura din scena va deveni patrat (precum se vede in imaginea de mai jos), cu latura egala cu suma argumentelor transmise (*8+20+40+50 = 118*).  


*Pe langa symbolul "rest parameters" (...) pot fi adauati si parametri normali. La apelare si executia functiei toti parametri (inclusiv cei normali) vor fi stocati in Array-ul pt. "rest parameters".*

**3. Definire functie direct in variabila**

O alta metoda de a crea o functie este aceea de a fi definita direct intr-o variabila de tip "Function".  
Sintaxa generala este:

* **var numeFunctie:Function = function (param1, param2, ...) {  
      Codul functiei  
  }**

- Aceasta forma a functiei se apeleaza normal, cu numele variabilei si argumentele intre paranteze rotunde, **numeFunctie(arg1, arg2, ...);**  
- Diferentele fata de crearea standard a unei functii sunt urmatoarele:

* O functie definita standard poate fi apelata in cod si inainte de locul unde e creata, dar aceasta, tip-variabila, poate fi apelata doar dupa declararea ei.
* Functia normala exista in memorie ca parte a obiectului ce o contine (in cazul OOP), dar aceasta, tip-variabila, este independenta in memorie, ca orice variabila, poate fi stearsa si modificata.

Studiati si testati singuri urmatorul exemplu (in documentul Flash creat la inceput, cu instanta "testClip").

// Se creaza o functie tip-variabila, denumita "vrF", cu 3 parametri

var vrF:Function = function (obj:MovieClip, sx:Number, sy:Number):\*

{

// "obj" reprezinta instanta de tip MivieClip a unui obiect din Scena, transmis ca parametru

// la "obj" se definesc /modifica proprietatile "scaleX" si "scaleY"

obj.scaleX = sx;

obj.scaleY = sy;

}

// Se apeleaza functia, transmitand ca argument pt. "obj" numele instantei din Scena

vrF(testClip, 2.5, 1.4);

- Proprietatile "scaleX" si "scaleY" multiplica lungimea si inaltimea obiectului (aici "obj") de atatea ori cat este valoare lor.

*Indicat este totusi utilizarea formei standard.*

**4. Variabile si functii**

Variabilele care sunt declarate in afara unei functii, inaintea ei, pot fi utilizate si in codul functiei.  
Variabilele care sunt create in interiorul functie pot fi folosite doar in cadrul ei, in afara acelei functii ele nu exista. Acestea se numesc "variabile locale".  
  
Studiati urmatorul exemplu:

var ext\_vr:Number = 18; // Variabila definita in afara oricarei functii

// Creare functie care foloseste variabila externa "ext\_vr", din afara ei

function oFunctie():Number

{

// Se creaza o variabila in interiorul functiei

var in\_vr:Number = 7;

// Se adauga la aceasta si variabila externa

in\_vr += ext\_vr;

return in\_vr; // Returneaza valoarea finala a lui "in\_vr"

}

trace(oFunctie()); // Se afiseaza in Output valoarea returnata de functie (25)

// Testeazaa fisarea valorii variabilei "in\_vr" create in interiorul functiei (daca se sterg "//")

// trace(in\_vr);

- Observati ca la apelare, functia a returnat 25, deoarece a folosit si valoarea lui "ext\_vr", chiar daca e creata in afara ei.  
- Dar daca se sterg caracerelr "//" de la "trace(in\_vr);" afiseaza eroare deoarece variabila "in\_vr" nu exista in afara functiei.

**5. Functii recursive**

**Functiile recursive** sunt functii care se auto-apeleaza singure in interiorul propriului cod.  
O metoda buna de a demonstra capacitatea recursiva a functiilor este rezolvarea unei ecuatii factoriale, care e produsul numerelor naturale, pozitive, de la 1 pana la un anumit numar precizat (1\*2\*3\*4\*...\*n).  
In exemplul urmator avem o functie recursiva "factorial()" care afla factorialul unui numar precizat "n" (aici 8).

// Se defineste o functie recursiva

function factorial(n:uint):uint

{

// Daca n>0, inmulteste 'n' cu auto-apelarea functiei

if(n > 0)

{

return (n \* factorial(n-1));

}

else { return 1; } // Returneaza direct 1 cand 'n' nu e mai mare ca 0

}

// Se adauga /apeleaza functia intr-o variabila;

var fact:Number= factorial(8);

trace(fact); // Afiseaza in Output valoarea lui "fact" (40320)

- Functia verifica intai daca "n" este mai mare decat 0, apoi in caz afirmativ, "n" se inmulteste cu rezultatul returnat de auto-apelarea functiei cu argumentul "n-1". Astfel, inainte de a returna vreo valoare, functia s-a auto-apelat iar, dar cu argumentul mai mic (n-1), si tot asa, pana cand "n" ajunge 0, aparitia cea mai imbricata a functiei se incheie. Flash incheie fiecare functie imbricata pana ce ajunge la apelarea initiala a functiei "factorial", apoi intoarce rezultatul final. - In Output va afisa valoarea 40320 (1\*2\*3\*4\*5\*6\*7\*8)

**Obiectul String - Siruri**

Termenul de **sir** (*string*) desemneaza orice cuvant, text, fraze, sau un singur caracter, puse intre ghilimele (simple sau duble).  
**Sirurile** pot contine numere, caractere speciale, spatii; in general orice caracter de la tastatura.  
ActionScript 3 contine proprietati si metode (functii) speciale pentru lucru cu **siruri**, prin care se pot face diferite operatii de manipulare a sirurilor, cautare in ele, conversie.  
Dupa cum totul in ActionScript 3 e vazut ca **obiect**, inclusiv sirurile, cand este definit un text, de ex.: "*Site www.marplo.net*", se creaza o instanta la clasa **String**.

**1. Ghilimele si secvente escape**

O variabila care stocheaza in ea date de tip **String** (sir) se poate defini cu oricare din urmatoarele forme:  
                **var nume\_var:String = "sirul";**  
                **var nume\_var:String = new String("Sirul");**  
                **var nume\_var:String = String("Sirul");**  
- Ghilimelele (pot fi simple sau duble) indica faptul ca textul din ele trebuie interpretat ca tip **String** (sir).  
- De obicei se folosesc ghilimele duble, dar la fel e de preferat si simple, mai ales cand in textul respectiv sunt adaugate ghilimele duble ca parte din text. Si invers, daca textul respectiv contine ghilimele simple, e de preferat sa fie incadrat intre ghilimele duble.  
      De exemplu:  
                var sir1:String = "Tutoriale 'Flash' www.marplo.net";  
                var sir2:String = 'Tutoriale "Flash" www.marplo.net';  
• Ca reguli generale:  
          - Tipul de ghilimele care incepe sirul trebuie sa fie folosit si la sfarsitul lui.  
          - Daca in text sunt ghilimele de acelasi tip cu cele folosite la inceputul si sfarsitul sirului, cele din text trebuie precedate de caracterul backslash (\) "escape". Astfel programul recunoaste ca acele ghilimele sunt parte din text si nu pt. sintaxa de creare a sirului.  
      De exemplu:  
                var sir1:String = "Tutoriale \"Flash\" www.marplo.net";  
                var sir2:String = 'Tutoriale \'Flash\' www.marplo.net';  
• **Secventele escape** sunt grupuri de caractere formate din backslash si alt caracter, care astfel unite indeplinesc un rol special.  
      **Secventele escape** folosite sunt:

* **\b** - caracter backspace
* **\f** - avanseaza o pagina (rar folosit)
* **\n** - linie noua. Adauga un rand nou
* **\r** - la fel ca '\n', adauga un rand nou, dar aduce cursoru inapoi la inceputul liniei
* **\t** - caracter Tab
* **\unnnn** - adauga un caracter ce are un cod Unicode format din 4 numere hexadecimal. De ex., codul "\u0278" adauga caracterul 'ɸ'
* **\xnn** - adauga un caracter format din 2 numere hexadecimal, codul ASCII. De ex., codul "/x25" adauga caracterul '%'
* **\'** - ghilimele simple (')
* **\"** - ghilimele duble (")
* **\\** - backslash (\)

- Unirea mai multor siruri intr-un singur sir se face cu caracterul '+'  
        Exemplu:

var sir1:String = ' Jocuri ';

var sir2:String = 'Anime';

var sir3:String = 'Cursuri'+ sir1+ sir2;

trace(sir3); // Cursuri Jocuri Anime

**2. Proprietati si Metode obiect String**

**Obiectul String** are o singura proprietate, si anume **length**. Aceasta returneaza numarul de caractere din sir.  
      Exemplu:

* *var sir:String = "Un sir";  
  trace(sir.length);             // 6*

Metode ale obiectului String sunt:

* **charAt(nr)** - Returneaza caracterul din sir care are indexul de ordine specificat la 'nr' (primul are index 0).  
  + *var sir:String = "Un sir";  
    trace(sir.charAt(3));             // s*
* **charCodeAt(nr)** - Codul numeric Unicode al caracterului cu numarul de ordine (index) 'nr' (primul caracter are index 0).  
  + *var sir:String = "Un sir";  
    trace(sir.charCodeAt(3));             // 115*
* **concat('sir1', 'sir2', ...)** - Adauga argumentele 'sir1', 'sir2', ... sub forma de sir la sfarsitul sirului.  
  + *var sir:String = "Test ";  
    trace(sir.concat('sir1', ' sir2'));             // Test sir1 sir2*
* **fromCharCode(cod1, cod2)** - Returneaza un sir compus din caracterele echivalente codurilor Unicode 'cod1', 'cod2', ....  
  + *trace(String.fromCharCode(115, 118));             // sv*
* **indexOf('sub\_sir', nr)** - Returneaza prima pozitie a lui 'sub\_sir' intr-un sir, incepand cautarea de la index 'nr' ('nr' e optional, daca nu e adaugat, se considera 0). Daca 'sub\_sir' ne se afla in sirul la care e aplicata, returneaza -1.  
  + *var sir:String = "Tutoriale";  
    trace(sir.indexOf('tor'));             // 2*
* **lastIndexOf('sub\_sir', nr)** - Similar cu "indexOf()", dar face cautarea de la dreapta la stanga. Returneaza ultima pozitie a lui 'sub\_sir' intr-un sir, incepand cautarea, spre stanga, de la index 'nr' ('nr' e optional, daca nu e augat, cautarea incepe de la ultimul caracter).  
  + *var sir:String = "Tutoriale, cursuri online";  
    trace(sir.lastIndexOf('ri'));             // 16*
* **match('sablon')** - Returneaza un Array cu toate partile din sir care se potrivesc sablonului specificat la "sablon", acesta poate fi un sir, o expresie regulata RegExp sau un obiect (care va fi convertit in sir).  
  + *var sir:String = "Tutoriale, tutor, cursuri";  
    var exp:RegExp = /tu[a-z]+/gi;  
    trace(sir.match(exp));             // Tutoriale,tutor*
* **replace('sablon', 'repl')** - Inlocuieste "sablon" (sir, RegExp) gasit in sir cu "repl".  
  + *var sir:String = "Tutoriale, tutor, cursuri";  
    var exp:RegExp = /tu[a-z]+/gi;  
    trace(sir.replace(exp, 'Lectii'));             // Lectii, Lectii, cursuri*
* **search('sablon')** - Returneaza prima pozitie a lui "sablon" (sir, RegExp) gasita in sir. Daca "sablon" nu e gasit, returneaza -1.  
  + *var sir:String = "Tutoriale, tutor, cursuri";  
    var exp:RegExp = /r[is]/i;  
    trace(sir.search(exp));             // 4*
* **slice(startIndex, endIndex)** - Returneaza sub-sirul care incepe de la caracterul cu index "startIndex" (inclusiv) pana la caracterul cu index "endIndex" (fara el). Similara e si functia **substring(startIndex, endIndex)**. Diferenta e ca "slice()" poate folosi si valori negative, dar "substring()" nu.  
  + *var sir:String = "Curs gratuit online";  
    trace(sir.slice(5, 12));             // gratuit*
* **split('sep')** - Transforma sirul in Array, cu elemente obtinute din sub-sirurile delimitate de "sep".  
  + *var sir:String = "Curs-gratuit-online";  
    trace(sir.split('-'));             // Curs,gratuit,online*
* **substr(startIndex, lun)** - Obtine sub-sirul care incepe de la caracterul cu idex 'startIndex' pana la lungimea data la 'lun'.  
  + *var sir:String = "http://www.marplo.net";  
    trace(sir.substr(11, 6));             // marplo*
* **toLowerCase()** - Returneaza o copie a sirului, cu toate caracterele mici; fara sa afecteze sirul original.  
  + *var sir:String = "Tutoriale, Cursuri ONLINE";  
    trace(sir.toLowerCase());             // tutoriale, cursuri online*
* **toUpperCase()** - Returneaza o copie a sirului, cu toate caracterele mari (majuscule); fara sa afecteze sirul original.  
  + *var sir:String = "Tutoriale, Cursuri ONLINE";  
    trace(sir.toUpperCase());             // TUTORIALE, CURSURI ONLINE*
* **toString()** - Transforma in sir obiectul la care e aplicata.  
  + *var nr:Number = 7;  
    var sir:String = '8';  
    trace(nr.toString()+ sir);             // 78     (sau String(nr)+ sir )*
* **Number('sir')** - Transforma "sir" in Numar.  
  + *var nr:Number = 7;  
    var sir:String = '8';  
    trace(nr+ Number(sir));             // 15*

**Obiect Number si Math - Numere**

**Numerele** sunt orice **numar** care nu e adaugat intre ghilimele (*daca e scris intre ghilimele, e considerat String (sir)*).  
**Numerele** pot fi de mai multe feluri:  
          - Pozitive ( 8, 11, 12.5 )  
          - Negative ( -8, -11, -12.5 )  
          - Intregi ( -40, -2, 4, 78 ) - Denumite "integer"  
          - Cu virgula, sau fractionare ( -4.5, 7.8 ) - Denumite "float". *In programare se foloseste punctul (.) intre partea intreaga si zecimale.*  
          - Hexazecimale ( 0x7d8 reprezinta numarul 2008 )  
          - Exponentiale - Folosite pt. valori numerice mari (7.8e+8 = 780 000 000, adica 7.8x108).

• Ca si celelalte elemente de baza in ActionScript (siruri, Array), si numerele sunt considerate **obiecte**, deoarece au metode proprii pentru lucru cu ele.  
O variabila care stocheaza **numere** se poate defini cu oricare din urmatoarele forme:  
                **var nume\_var:Tipnr = nr;**  
                **var nume\_var:Tipnr = new Tipnr(nr);**  
                **var nume\_var:Tipnr = Tipnr(nr);**  
- **nr** este numarul adaugat in variabila "nume\_var".  
- **Tipnr** reprezinta clasa din ActionScript pentru tipul de numere care sunt stocate in variabila, poate fi de 3 feluri:

* **Number** - pentru orice tip de numere, intregi, cu zecimale, hexazecimale sau exponentiale.
* **int** - pt. numere intregi (fara zecimale, pozitive sau negative).
* **uint** - pt. numere intregi pozitive (fara zecimale, mai mari decat 0).

- *De obicei se foloseste prima varianta*.  
  
• Daca variabilei de tip "Number" nu i-se atribuie o valoare (un numar), ci este doar simplu declarata (**var nume\_var:Number;**), valoarea va fi cea "defaul" folosita in AS3: **NaN** (*Not a Number*). Dar i-se poate atribui dupa ce a fost declarata.  
Pentru a verifica daca o variabila are valoarea NaN se foloseste metoda "**isNaN(nume\_var)**" (aceasta returneazaa **true** daca e NaN, sau**false**).  
      Exemplu:

var nr:Number;

trace(nr); // NaN

trace(isNaN(nr)); // true

nr = 78;

trace(nr); // 78

*Cum pt. tipul****Number****valoarea default este****NaN****, pentru tipurile****int****si****uint****valoarea default este 0.  
- "var nr:Number =****.nr****" si "var nr:Number =****0.nr****" reprezinta acelasi numar.  
- Pentru a defini o culoarea in ActionScript 3 se foloseste expresia****0xFFRRGGBB****[F - Alpha (transparenta), R - Red (rosu), G - Green (verde), B - Blue (albastru)], aceasta e un "32-bit integer" si poate fi stocat in tipul "uint".  
                Ex.     var culoare:Number= 0xafe8edfe;*

**1. Metode si Constante obiect Number**

**Obiectul Number** nu are proprietati specifice, are cateva metode (functii) si constante. Operatiile cu numere se fac cu operatorii aritmetici si relationali (*vedeti lectia*[*Constante si Operatori*](http://www.marplo.net/flash/constante-operatori_as)) si cu obiectul Math (*e prezentat mai jos*).

**- Metode obiect Number**

* **toExponential(ze)** - Returneaza un sir cu numarul in notatie exponentiala. "ze" indica numarul de zecimale afisate in notatia exponentiala returnata  
  + *var nr:Number = 123456789;  
    trace(nr.toExponential(2));             // 1.23e+8*
* **toFixed(ze)** - Returneaza un sir cu numarul fixat la cate zecimale indica "ze".  
  + *var nr:Number = 12.3456789;  
    trace(nr.toFixed(3));             // 12.346*
* **toPrecision(ze)** - Returneaza un sir cu numarul in notatia exponentiala sau normala. "ze" reprezinta numarul total de intregi si zecimale.  
  + *var nr1:Number = 12.3456789;  
    trace(nr1.toPrecision(4));             // 12.35  
      
    var nr2:Number = 123400000;  
    trace(nr2.toPrecision(4));             // 1.234e+8*
* **toString()** - Transforma numarul in sir.  
  + *var nr:Number = 7;  
    var sir:String = '8';  
    trace(nr.toString()+ sir);             // 78     (sau String(nr)+ sir )*
* **Number('sir')** - Transforma "sir" in Numar.  
  + *var nr:Number = 7;  
    var sir:String = '8';  
    trace(nr+ Number(sir));             // 15*

**- Constante obiect Number**

* **MAX\_VALUE** - Numarul cel mai mare care poate fi folosit.  
  + *trace(int.MAX\_VALUE);             // 2147483647  
    trace(uint.MAX\_VALUE);             // 4294967295  
    trace(Number.MAX\_VALUE);             // 1.79769313486231e+308*
* **MIN\_VALUE** - Numarul cel mai mic care poate fi folosit.  
  + *trace(int.MIN\_VALUE);             // -2147483648  
    trace(uint.MIN\_VALUE);             // 0  
    trace(Number.MIN\_VALUE);             // 4.9406564584124654e-324*
* **NEGATIVE\_INFINITY** - Reprezinta Infinitul Negativ si cu ea se pot verifica valorile "**-Infinity**" (rezultate de exemplu in urma impartirii la 0).  
  + *var nr:Number = -7/0;  
    trace();             // -Infinity  
    trace(nr == Number.NEGATIVE\_INFINITY);             // true*
* **POSITIVE\_INFINITY** - Reprezinta Infinitul Pozitiv si cu ea se pot verifica valorile "**Infinity**" (rezultate de exemplu in urma impartirii la 0).  
  + *var nr:Number = 8/0;  
    trace();             // Infinity  
    trace(nr == Number.POSITIVE\_INFINITY);             // true*

**2. Obiectul Math**

**Obiectul Math** (sau *clasa Math*) contine metode si constante prin care se pot efectua operatii matematice mai avansate, cum ar fi: ridicarea la putere, obtinerea unui numar aleator, logaritmi, sau trigonometrice (cu sinus, cosinus, etc.).  
• Sintaxa generala pentru folosirea metodelor **Math** este:  
                **Math.metoda(parametru)**  
      Exemple:

var nr:Number = Math.pow(3, 4); // Ridica 3 la puterea 4

trace(nr); // 81

var nr2:Number = Math.random();

trace(nr2); // 0.32177373580634594

**- Metode obiect Math pt. operatii aritmetice**

* **pow(a, b)** - Returneaza valoarea lui 'a' ridicat la puterea 'b'.
* **exp(nr)** - Returneaza 'e' la puterea 'nr' (e = 2.718281828 [*valoarea bazei logaritmului natural*]).
* **floor(nr)** - Returneaza valoarea rotunjita a lui 'nr' la anteriorul (mai mic) intreg.
* **ceil(nr)** - Valoarea rotunjita a lui 'nr' la urmatorul (mai mare) intreg.
* **round(nr)** - Valoarea rotunjita a lui numar la intregul cel mai apropiat.
* **max(a, b, c, ...)** - Cel mai mare numar dintre cele date la parametri.
* **min(a, b, c, ...)** - Cel mai mic numar dintre cele date la parametri.
* **sqrt(nr)** - Radical din 'nr'.
* **abs(nr)** - Valoarea absoluta (cu +) a lui 'nr'.
* **log(nr)** - Returneaza logaritmul natural al lui 'nr'.
* **random()** - Un numar aleator intre 0 si 1 (poate fi 0, dar nu si 1).

**- Metode obiect Math pt. operatii trigonometrice**

* **sin(rad)** - Sinusul unghiului specificat la 'ran', in radiani.
* **cos(rad)** - Cosinusul unghiului specificat la 'ran', in radiani.
* **tan(rad)** - Tangenta unghiului specificat la 'ran', in radiani.
* **asin(rad)** - Returneaza arcsinus-ul, unghiul in radiani, a numarului 'ran' (in radiani).
* **acos(rad)** - Returneaza arccosinus-ul, unghiul in radiani, a numarului 'ran' (in radiani).
* **atan(rad)** - Returneaza arctangenta, unghiul in radiani, a numarului 'ran' (in radiani).
* **atan2(y, x)** - Returneaza unghiul in radiani, trasat din origine la punctul de intersectie y/x.

- Pentru a afla valoarea in radiani, cand se cunosc gradele, se foloseste urmatoarea formula:  
                **val\_radiani = val\_grade \* Math.PI/180;**  
      Exemplu:

var grade:Number = 45;

var radiani:Number = grade \* Math.PI/180;

trace(radiani); // 0.7853981633974483

- Pentru a afla gradele, cand se cunosc radianii, se foloseste urmatoarea formula:  
                **val\_grade = val\_radiani \* 180/Math.PI;**  
      Exemplu:

var radiani:Number = 0.7854;

var grade:Number = radiani \* 180/Math.PI;

trace(grade); // 45

• Constantele sunt un numar fix. Sintaxa generala pentru folosirea constantelor **Math** este:  
                **Math.CONSTANTA**  
      Exemple:

trace(Math.PI); // 3.141592653589793

trace(Math.E); // 2.71828182845905

**- Constante Math**

* **E** - Valoarea bazei logaritmului natural ( 2.71828182845905 ).
* **LN10** - Logaritm natural din 10 ( 2.302585092994046 ).
* **LN2** - Logaritm natural din 2 ( 0.6931471805599453 ).
* **LOG10E** - Logaritm in baza 10 din E ( 0.4342944819032518 ).
* **LOG2E** - Logaritm in baza 2 din E ( 1.442695040888963387 ).
* **PI** - Constanta matematica PI, pentru cerc ( 3.141592653589793 ).
* **SQRT1\_2** - Radical din 1/2 [0.5] ( 0.7071067811865476 ).
* **SQRT2** - Radical din 2 ( 1.4142135623730951 ).

# Data si Timp - Obiect Date

Pentru a lucra cu **data** zilei si perioade de **timp** in ActionScript 3 se foloseste obiectul (sau clasa) **Date**.  
Ca si la celelalte tipuri de clase, pentru a folosi obiectul Date trebuie creata o instanta a lui (cu operatorul **new**), dupa care pot fi utilizate proprietatile si metodele clasei Date la acea instanta.  
Sintaxa generala de creare a unei instante Date este:  
                **var nume\_ins:Date = new Date();**  
- Cu aceasta formula se memoreaza data curenta intr-o variabila "nume\_ins", dupa cum se observa in rezultatul returnat de "trace()" din exemplul urmator.

var dat:Date = new Date();

trace(dat); // Tue Nov 9 09:53:59 GMT+0200 2010

• Data reprezinta orice punct in timp. Se pot aplica la crearea instantei Date o serie de valori ca parametri: An, Luna, Zi a lunii, Ora, Minut, Secunda si Milisecunde; pentru a utiliza o anumita data din timp. Sintaxa generala este:  
                **var nume\_ins:Date = new Date(an, luna, zi, ora, minut, secunda, milisecunde);**  
- Trebuie inclusi cel putin 2 parametri, cel putin Anul si Luna (ceilalti vor avea valoarea implicita 0)  
- Prima luna din an (Ianuarie) are valoarea 0, iar ultima luna (Decembrie) are valoarea 11. Orele sunt de la 0 la 23 (24 e ora 0).  
*Data 0 e considerata data Unix (cand a aparut) 1-Ianuarie-1970.*  
Exemplu:

var dat:Date = new Date(2010, 0, 15);

trace(dat); // Fri Jan 15 00:00:00 GMT+0200 2010

var dat2:Date = new Date(2010, 11, 15, 18, 22, 32, 88);

trace(dat2); // Wed Dec 15 18:22:32 GMT+0200 2010

• Un alt mod de a prelua un anumit moment din timp este adaugarea la parametru a unui sir reprezentand acea data, dupa cum e in exemplele de mai jos. Sirul trebuie sa contina cel putin Anul, Luna si Ziua.

var dat1:Date = new Date('1/15/2010'); // Luna/Zi/An

trace(dat1); // Fri Jan 15 00:00:00 GMT+0200 2010

var dat2:Date = new Date('Sun Jan 28 2007');

trace(dat2); // Sun Jan 28 00:00:00 GMT+0200 2007

var dat3:Date = new Date('15 Oct 1976');

trace(dat3); // Fri Oct 15 00:00:00 GMT+0300 1976

var dat4:Date = new Date('Sun Jan 28 2007 14:16:12');

trace(dat4); // Sun Jan 28 15:16:12 GMT+0200 2007

- *Daca sirul adaugat nu poate fi recunoscut si transferat ca data, obiectul Date va returna****NaN***  
• Pentru a verifica daca o data este valida se poate folosi formula:  
                **isNaN(nume\_ins.valueOf())**  
- Aceasta verifica daca valoarea lui "nume\_ins" e un numar. Returneaza **true** daca NU e numar, altfel, **false**.  
Exemplu:

var dat:Date = new Date('Ieri');

trace(dat); // Invalid Date

trace(isNaN(dat.valueOf())); // true

*Zona de timp folosita poate fi timpul local, apare ca GMT; sau Timpul Universal, abreviat UTC (este timpul civil al meridianului 0, determinat prin masurare astronomica).*

**Proprietati si metode obiect Date**

Dupa ce a fost creata o **Instanta Date**, se pot accesa si modifica datele din ea folosind proprietatile si metodele obiectului.  
De exemplu, daca se doreste obtinerea orei si numarul zilei curente (care e de la 1 la 31) se pot utiliza instructiunile ca in urmatorul exemplu:

// Seteaza o instanta la obiectul Date

// in care va fi stocata data si timpul curent

var dat:Date = new Date();

// Preia ziua

var ziua:Number = dat.date; // sau **dat.getDate();**

// Preia ora

var ora:Number = dat.getHours(); // sau **dat.hours;**

// Afisaza in Output ziua si ora

trace(ziua); // 9

trace(ora); // 11

- Modificarea unei date, a anumitor parti din ea (an, luna, ora, ...), se face dupa cum puteti vedea in acest exemplu:

// Seteaza o instanta la obiectul Date

// va stoca data si timpul din ziua de: Duminica, 28-Ianuarie-2007, ora 14:16:12

var dat:Date = new Date('Sun Jan 28 2007 14:16:12');

// Seteaza o noua valoare pt. zi

dat.date = 11; // sau **dat.setDate(11);**

// Seteaza o noua valoare pt. ora

dat.setHours(8); // sau **dat.hours = 8;**

// Verifica in Output data obtinuta

trace(dat); // Thu Jan 11 08:16:12 GMT+0200 2007

- S-a modificat numarul zilei la valoarea 11 (initial 28) si al orei 8 (initial 14). Implicit a modificat si ziua din saptamana (vedeti in rezultatul dat de "trace").  
- Observati cum sunt aplicate proprietatile si metodele obiectului.

**Proprietati obiect Date**

* **date** - ziua din luna (intre 1 si 31)
* **dateUTC** - ziua din luna (intre 1 si 31), in acord cu timpul universal (UTC)
* **day** - numarul zilei din saptamana (0 pt. Duminica, 1 pt. Luni, ...)
* **dayUTC** - numarul zilei din saptamana (0 pt. Duminica, 1 pt. Luni, ...), in acord cu timpul universal (UTC)
* **fullYear** - anul intreg (format din 4 caractere)
* **fullYearUTC** - anul intreg (format din 4 caractere), in acord cu timpul universal
* **hours** - ora (un numar intre 0 si 23)
* **hoursUTC** - ora (un numar intre 0 si 23), timp UTC
* **milliseconds** - milisecunde (un numar intre 0 si 999)
* **millisecondsUTC** - milisecunde (un numar intre 0 si 999), in acord cu timpul universal
* **minutes** - minute (un numar intre 0 si 59)
* **minutesUTC** - minute (un numar intre 0 si 59), in acord cu timpul universal
* **month** - numarul lunii din an (0 pt. Ianuarie, 1 pt. Februarie, ..., 11 pt. Decembrie)
* **monthUTC** - numarul lunii din an (0 pt. Ianuarie, 1 pt. Februarie, ..., 11 pt. Decembrie), timp UTC
* **seconds** - secunde (un numar intre 0 si 59)
* **secondsUTC** - secunde (un numar intre 0 si 59), in acord cu timpul universal
* **time** - numarul de milisecunde de la 1-Ianuarie-1970
* **timezoneOffset** - diferenta in minute dintre timpul universal (UTC) si timpul local (al calculatorului unde ruleaza prezentarea)

**Metode obiect Date**

* **getDate()** - returneaza numarul zilei din luna (un numar intre 1 si 31)
* **getDay()** - returneaza numarul zilei din saptamana (0 pt. Duminica, 1 pt. Luni, ...)
* **getFullYear()** - returneaza anul intreg (format din 4 caractere)
* **getHours()** - obtine ora (un numar intre 0 si 23)
* **getMilliseconds()** - obtine milisecundele (un numar intre 0 si 999)
* **getMinutes()** - obtine minutele (un numar intre 0 si 59)
* **getMonth()** - returneaza numarul lunii din an (0 pt. Ianuarie, 1 pt. Februarie, ..., 11 pt. Decembrie)
* **getSeconds()** - returneaza secundele (un numar intre 0 si 59)
* **getTime()** - obtine numarul de milisecunde de la 1-Ianuarie-1970
* **getTimezoneOffset()** - obtine diferenta in minute dintre timpul universal (UTC) si timpul local (al calculatorului unde ruleaza prezentarea)
* **getUTCDate()** - returneaza numarul zilei din luna (un numar intre 1 si 31), in acord cu timpul universal
* **getUTCDay()** - returneaza numarul zilei din saptamana (0 pt. Duminica, 1 pt. Luni, ...), in acord cu timpul universal
* **getUTCFullYear()** - returneaza anul intreg (format din 4 caractere), in acord cu timpul universal
* **getUTCHours()** - obtine ora (un numar intre 0 si 23), dupa timpul universal
* **getUTCMilliseconds()** - obtine milisecundele (un numar intre 0 si 999), dupa timpul universal
* **getUTCMinutes()** - obtine minutele (un numar intre 0 si 59), dupa timpul universal
* **getUTCMonth()** - returneaza numarul lunii din an (0 pt. Ianuarie, 1 pt. Februarie, ..., 11 pt. Decembrie), timp UTC
* **getUTCSeconds()** - returneaza secundele (un numar intre 0 si 59), dupa timpul universal
* **parse('data')** - converteste sirul de la 'data' intrun numar ce reprezinta milisecundele trecute de la 1-Ianuarie-1970 pana la acea data
* **setDate(nr\_zi)** - seteaza numarul zilei (intre 1 si 31) si returneaza noul timp in milisecunde
* **setFullYear(an, luna, zi)** - seteaza anul, luna si ziua, daca sunt mentionate (acestea fiind optionale) si returneaza noul timp in milisecunde
* **setHours(ore, minute, secunde, milisecunde)** - defineste ora (ultimii 3 parametri sunt optionali), returneaza noul timp in milisecunde
* **setMilliseconds(ms)** - seteaza milisecundele cu valoarea de la 'ms' (un nr. intre 0 si 999) si returneaza noul timp in milisecunde
* **setMinutes(minute, secunde, milisecunde)** - seteaza minutul (ultimii 2 parametri sunt optionali) si returneaza noul timp in milisecunde
* **setMonth(luna, ziua)** - seteaza luna ('ziua' e optional) si returneaza noul timp in milisecunde
* **setSeconds(secunde, milisecunde)** - defineste secundele ('milisecunde' e optional) si returneaza noul timp in milisecunde
* **setTime(ms)** - seteaza data in milisecunde, de la 1-Ianuarie-1970, cu cea specificata la 'ms'
* **setUTCDate(nr\_zi)** - seteaza numarul zilei, la zona de timp UTC (intre 1 si 31) si returneaza noul timp in milisecunde (timpul UTC)
* **setUTCFullYear(an, luna, zi)** - seteaza anul, luna si ziua (la zona de timp UTC), daca sunt mentionate (acestea fiind optionale) si returneaza noul timp in milisecunde
* **setUTCHours(ore, minute, secunde, milisecunde)** - defineste ora, la zona de timp UTC (ultimii 3 parametri sunt optionali), returneaza noul timp in milisecunde
* **setUTCMilliseconds(ms)** - seteaza milisecundele, la timpul UTC, cu valoarea de la 'ms' (un nr. intre 0 si 999) si returneaza noul timp in milisecunde
* **setUTCMinutes(minute, secunde, milisecunde)** - seteaza minutul, la timpul UTC (ultimii 2 parametri sunt optionali) si returneaza noul timp in milisecunde
* **setUTCMonth(luna, ziua)** - seteaza luna, la zona de timp UTC ('ziua' e optional) si returneaza noul timp in milisecunde
* **setUTCSeconds(secunde, milisecunde)** - defineste secundele, timp UTC ('milisecunde' e optional) si returneaza noul timp in milisecunde
* **toDateString()** - returneaza un sir reprezentand ziua si data, fara ora si zona de timp
* **toLocaleDateString()** - la fel ca si "toDateString()"
* **toLocaleString()** - returneaza un sir cu data si timpul zilei (ora cu AM, PM), fara zona de timp
* **toLocaleTimeString()** - returneaza un sir cu timpul zilei (ora cu AM, PM), fara data zilei, an sau zona de timp
* **toString()** - returneaza un sir cu ziua, data, ora si zona de timp
* **toTimeString()** - returneaza un sir cu ora si zona de timp, fara ziua si data
* **toUTCString()** - returneaza un sir cu ziua, data si ora UTC

*Expresia****flash.utils.getTimer()****returneaza numarul de milisecunde de cand Flash Player-ul e pornit.*

**Array**

**Array** (denumit si *matrice*) poate fi considerat si obiect si variabila.  
Se foloseste pentru a stoca mai multe valori ordonate intr-un singur nume de variabila. Pe scurt "***o lista de valori***".  
Fiecare valoare stocata devine un element al matricei, si are asociat un "numar index" (indice sau cheie). Cu ajutorul acestei chei se poate face referire la oricare element din Array.  
In ActionScript sunt folosite 2 moduri de a crea Array:  
• 1. Cu valorile intre o pereche de paranteze patrate [] (*gen JSON*)  
                **var numeAr:Array = [val1, val2, ...];**  
• 2. Cu operatorul **new**  
                **var numeAr:Array = new Array(val1, val2, ...);**  
- La ambele metode, "numeAr" este numele Array-ului (prin care se poate lucra cu elementele lui), iar "val1, val2, ..." sunt valorile adaugate si stocate in "numeAr". Aceste valori (elemente) pot fi de orice tip: variabile, siruri, numere sau chiar alte matrici; separate prin virgula (,).  
De exemplu:  
                **var mAr:Array = ['un sir', o\_variabila, 7.5, 88];**  
      sau  
                **var mAr:Array = new Array('un sir', o\_variabila, 7.5, 88);**

*Pentru a crea un Array gol, fara vreun element, se poate folosi una din expresiile:****var mAr:Array = [];****sau****var mAr:Array = new Array();***

**1. Utilizare elemente din Array**

Dupa ce a fost creat un Array, scopul lui este sa fie accesate elementele din el.  
- Elementele adaugate in Array sunt retinute in ordine numerica, incepand de la indice (index sau cheie) 0. Astfel, primul element are indice 0, al doilea 1, si tot asa.  
Pentru a accesa oricare element din matrice, se scrie numele ei si indicele intre paranteze patrate, **numeAr[index]**  
De exemplu:

var mAr:Array = ['site', 'www.marplo.net', 78];

trace(mAr[1]); // Afiseaza in Output: **www.marplo.net**

- Pentru a folosi primul element, se pune indice 0, ***trace(mAr[0]);*** va returna "site".  
  
Elementele din Array astfel accesate (cu "numeAr[index]"), pot fi utilizate ca o variabila, in operatii sau alte variabile; iar ca valoare pentru "index" se poate folosi si o variabila.  
Studiati urmatorul exemplu:

var mAr:Array = ['site', 'www.marplo.net', 78];

// Se creaza o variabila "i" care va fi folosita pt. indice

var i:Number= 2;

// Se creaza o variabila a carei valoare va fi un numar adunat cu

// elementul din Array-ul "mAr", cu index "i" (aici 2)

var test = 4 + mAr[i];

trace(test); // Verifica valoarea lui "test", in Output: **82**

- "i" avand valoarea 2, "mAr[i]" va returna al treilea element din matricea "mAr" (78), iar variabila "var test = 4 + mAr[i];" va avea valoarea (82 , "4+78") care e afisata in Output.

**2. Modificare, adaugare elemente in Array**

Pe langa accesarea elementelor dintr-un Array, acestea pot fi si modificate (valoarea lor) sau adaugate altele la cele existente.  
Procedeul este asemanator ca la variabile, se precizeaza elementul respectiv (numele matricei si intre paranteze patrate indicile de ordine a lui) si cu semnul egal i-se atribue o valoare.

Studiati urmatorul exemplu:

var mAr:Array = ['site', 'www.marplo.net', 78];

// Se modifica valoarea primului element

mAr[0] = 'tutoriale';

// Se verifica cu "trace(mAr)" valorile din Array

trace(mAr); // tutoriale,www.marplo.net,78

// Se adauga un nou element in "mAr" (al patrulea, inice 3)

mAr[3] = 'ActionScript';

// Se verifica iar cu "trace(mAr)" valorile din Array

trace(mAr); // tutoriale,www.marplo.net,78,ActionScript

*Daca se aplica "trace()" la un Array, va afisa in Output valoarea tuturor elementelor din acel Array.*  
- Acest script va afisa in Output:

tutoriale,www.marplo.net,78  
tutoriale,www.marplo.net,78,ActionScript

- Observati diferenta fata de valorile initiale. Valoarea primului element, "site" a devenit "tutoriale", apoi s-a adaugat inca un element "ActionScript".

*Despre cum se sterg elemente dintr-un Array, vedeti pagina*[*Array - Lucru cu Functii*](http://www.marplo.net/flash/array-lucru-functii_as)*.*

**3. Numar elemente din Array**

In lucrul cu Array este necesara cunoasterea si a numarului de elemente din el. Acesta se poate afla cu proprietatea **length**, folosind sintaxa:  
                **numeAr.length**  
- Daca la un Array care are de exemplu 3 elemente (cu indici 0, 1, 2) se adauga inca un element, dar cu indice mai mare decat cel urmator, de exemplu index 5; acel Array va avea 6 elemente. In locul indicilor lipsa pana la cel adaugat (3 si 4) va crea automat elemente goale.  
Studiati urmatorul exemplu:

var mAr:Array = ['site', 'www.marplo.net', 78];

// Se adauga intr-o variabila numarul de elemente din "mAr"

var nr\_el = mAr.length;

// Se verifica cu "trace(nr\_el)" numarul de elemente

trace(nr\_el); // 3

// Se adauga un nou element in "mAr" (cu inice 5)

mAr[5] = 'ActionScript';

// Se actualizeaza numarul de elemente din "mAr" in "nr\_el" si se verifica iar cu "trace(nr\_el)"

nr\_el = mAr.length;

trace(nr\_el); // 6

// Se verifica si elementele continute in "mAr"

trace(mAr); // site,www.marplo.net,78,,,ActionScript

- Observati ca, desi Array-ul "mAr" are la inceput 3 elemente si se mai adauga doar unul "ActionScript", expresia "mAr.height" returneaza 6 elemente deoarece ultimul a fost adaugat cu inde 5 si astfel sunt completate cu elemente goale cele lipsa (cu indicii: 3 si 4), dupa cum se vede in rezultatul afisat de al treilea "trace(mAr)".  
- In Output afiseaza:

3  
6  
site,www.marplo.net,78,,,ActionScript

*Proprietatea "****length****" poate fi utilizata si pentru a seta unui Array un anumit numar de elemente, adaugad elemente goale (fara valoare).  
De exemplu:****var mAr:Array = ['site', 'www.marplo.net', 78];  
                mAr.length = 10;*** *- Va face ca "mAr" sa aibe 10 elemente, ultimile 7 goale.*

**Array - Avansati**

In aceasta lectie vor fi prezentate metode mai **avansate** de lucru cu **Array**.

**1. Parcurgere Array**

Pentru a parcurge un Array se folosesc oricare din instructiunile repetitive: **for, while** sau **for each ... in** (*vedeti lectiile despre Instructiuni Repetitive*).  
Parcurgerea unui Array da posibilitatea aplicarii unei actiuni fiecarui element din el.  
- Iata cate un exemplu cu fiecare.

**Cu for()**

var mAr:Array = [2, 6, 8];

// Se parcurge Array "mAr" cu "for()"

for(var i:Number=0; i<mAr.length; i++)

{

// Se dubleaza valoarea elementului curent

mAr[i] \*= 2;

}

// Verificare valori ale elementelor din "mAr" dupa parcurgere

trace(mAr); // 4,12,16

- "var i:Number=0;" seteaza indicele de la care incepe parcurgerea (se poate da si alta valoare in afara de 0, de la care element sa inceapa).  
- "i<mAr.length;" face ca parcurgerea sa fie realizata pana la ultimul element.  
- "i++" incrementeaza indicele cu o unitate, pt. fiecare parcurgere.  
- In cadrul buclei "for()" (intre acolade) pot fi scrise orice instructiuni si operatii pt. elementul curent parcurs; acesta putand fi accesat cu indicele de ordine "i".

**Cu while()**

var mAr:Array = [2, 6, 8];

var i:Number = 0; // Se defineste indicele de ordine pt. parcurgere

// Se parcurge Array "mAr" cu "while()"

while(i<mAr.length)

{

// Se dubleaza valoarea elementului curent

mAr[i] \*= 2;

i++; // Incrementeaza indicele ca sa treaca la urmatorul element

}

// Verificare valori ale elementelor din "mAr" dupa parcurgere

trace(mAr); // 4,12,16

**Cu for each ... in**

var mAr:Array = [2, 6, 8];

// Se parcurge Array "mAr" cu "for each ... in"

for each(var elm:Number in mAr)

{

// Se dubleaza valoarea elementului curent, care e stocat in "elm"

elm \*= 2;

// Pot fi executate comenzi cu "elm"

}

// Verificare valori ale elementelor din "mAr" dupa parcurgere

trace(mAr); // 2,4,8

- Observati ca, fata de celelalte instructiuni repetitive, "for each ... in" doar preia si foloseste intr-o variabila (aici "elm") valoarea elementului curent parcurs din Array (aici "mAr"), dar nu influenteaza elementele din matrice. Dupa cum se vede la verificarea cu "trace(mAr)", valorile din Array sunt neschimbate.

**2. Array multi-dimensional**

Un **Array** poate contine ca element un alt Array. In acest caz avem de-a face cu o matrice multi-dimensionala.  
Sa luam un exemplu practic :

// Se creaza un Array cu 3 elemente (al treile fiind tot un Array)

var mAr:Array = ['tutoriale', 'www.marplo.net', [88, 'flash']];

// Verificare elemente totale din 'mAr', si al treilea

trace(mAr); // tutoriale,www.marplo.net,88,flash

trace(mAr[2]); // 88,flash

- Al treilea element ([88, 'flash']) fiind un Array in cel principal, se mai numeste si "*array imbricat*".

**Accesare elemente din Array-ul imbricat**

Pentru a accesa un element din matricea imbricata se foloseste urmatoarea formula:  
                **numeAr[i1][i2]**  
- Unde, "i1" este indicile de ordine a matricei imbricate in Array-ul principal, iar "i2" este indicele elementului din matricea interna.  
Mai usor de inteles din urmatorul exemplu:

// Se creaza un Array cu 3 elemente (al treile fiind tot un Array)

var mAr:Array = ['tutoriale', 'www.marplo.net', [88, 'flash']];

// Se adauga intr-o variabila al doilea element din Array-ul imbricat

var test:\* = mAr[2][1];

trace(test); // Verificare valoare lui "test" (flash)

**3. Array asociativ**

Un **Array asociativ** este un Array in care indicii elementelor nu sunt numere intregi consecutive, ci siruri (cuvinte). Acestea nu mai sunt create automat, in ordine de catre program, ci trebuie definiti (asociati) special pentru fiecare element.  
Sintaxa generala este:

* **var numeAr:Array = new Array();  
    numeAr['key1'] = val1;  
    numeAr['key2'] = val2;  
        // ...**

- Unde "numeAr" este numele matricei.  
- 'key1', 'key2' sunt indicii (cheile) asociate fiecarui element (pot fi scrisi si intre ghilimele duble, iar daca sunt numere, nu mai se adauga ghilimele).  
- "val1", "val2" sunt valorile elementelor respective (pot fi: siruri, numere, variabile, expresii sau alt Array).  
- Se pot adauga oricate elemente in Array, cu formula **numeAr['key'] = valoare;**  
  
• Accesarea elementelor dintr-un Array asociativ se face la fel ca la cel cu indexi numere consecutive (*denumit si "secvential"*), cu sintaxa: **numeAr['key']**  
• Acest tip de Array (asociativ) este de fapt si **Obiect**, deoarece valorile elementelor din el pot fi accesate si folosind formula: "**numeAr.key**", aici "key" fiind folosita ca o proprietate.  
Iata un exemplu in care sunt aplicate ambele metode de accesare:

// Array asociativ cu 3 elemente

var mAr:Array = new Array();

mAr['site'] = 'www.marplo.net';

mAr['curs'] = 'Adobe Flash';

mAr['tutoriale'] = 'JavaScript';

// Apelare cu metoda tip Array a primului element

trace(mAr['site']); // www.marplo.net

// Apelarea altui element, cu metoda tip Obiect

trace(mAr.curs); // Adobe Flash

*Daca se aplica "****trace(array\_asociativ)****", fara vreo cheie, nu va afisa nimic.*

**4. Parcurgere Array asociativ**

Un Array asociativ se parcurge cu instructiunile "**for ... in**" sau "**for each ... in**".  
Iata un exemplu pentru fiecare.

**- Cu for ... in**

// Array asociativ cu 3 elemente

var mAr:Array = new Array();

mAr['site'] = 'www.marplo.net';

mAr['curs'] = 'Adobe Flash';

mAr['tutoriale'] = 'JavaScript';

// Se parcurge matricea "mAr" cu "for ... in"

for(var key:String in mAr)

{

// Se afiseaza in Output Cheia si valoarea fiecarui element parcurs

trace('Cheie='+ key+ ' - valoare='+ mAr[key]);

// Pot fi executate orice comenzi si operati cu "key" si "mAr[key]"

}

- "key" e o variabila in care programul retine cheia (indexul) elementului curent parcurs (poate fi orice nume de variabila) din Array-ul "mAr".  
In Output afiseaza:

Cheie=site - valoare=www.marplo.net  
Cheie=tutoriale - valoare=JavaScript  
Cheie=curs - valoare=Adobe Flash

**- Cu for each... in**

// Array asociativ cu 3 elemente

var mAr:Array = new Array();

mAr['site'] = 'www.marplo.net';

mAr['curs'] = 'Adobe Flash';

mAr['tutoriale'] = 'JavaScript';

// Se parcurge matricea "mAr" cu "for each ... in"

for each(var elm in mAr)

{

// Se afiseaza in Output valoarea fiecarui element parcurs

trace('valE - '+ elm);

// Pot fi executate orice comenzi si operati cu "elm"

}

- Observati diferenta fata de "for ... in". Cu "for each ... in" nu se obtine cheia elementului; variabila declarata intre paranteze (aici "elm") stocheaza insusi elementul parcurs.  
In Output afiseaza:

valE - Adobe Flash  
valE - www.marplo.net  
valE - JavaScript

**Array - Lucru cu functii**

ActionScript 3 contine numeroase functii pentru lucrul cu **Array**. In aceasta lectie vor fi prezentate cateva din aceste functii si exemple despre ce se poate face cu ele.

**1. Stergere elemente din Array**

In ActionScript 3 sunt mai multe moduri de a sterge unul sau mai multe elemente dintr-un Array, folosind diferite functii. Acestea sunt prezentate direct in exemplele urmatoare (*explicatiile necesare sunt in comentariile din coduri*):

1. Cu operatorul **delete**  
          Sintaxa generala:  
                  **delete numeAr[index]**  
          - "numeAr" este numele Array-ului, iar "index" este indicele sau numele elementului a carui valoare va fi stearsa (***sterge valoarea, nu si elementul***).  
      Exemplu:

var mAr:Array = ['web', 'www.marplo.net', 'cursuri', 8];

delete mAr[1]; // Sterge valoarea elementului al doilea

var nr\_el:Number= mAr.length; // Aflare nr. elemente dupa stergere

// Verificare cu "trace()"

trace(nr\_el+ ' - '+ mAr); // 4 - web,,cursuri,8

- Dupa cum se vede in rezultatul dat de "trace(nr\_el+ ' - '+ mAr);", operatorul **delete** sterge doar valoarea elementului, acesta ramane gol in Array.  
  
2. Cu metoda **pop()**  
          Sintaxa generala:  
                  **numeAr.pop()**  
          - metoda (functia) "pop()" sterge ultimul element din Array-ul "numeAr" (opusul lui "*push()*", care adauga un element la sfarsit), si returneaza valoarea lui.  
      Exemplu:

var mAr:Array = ['web', 'www.marplo.net', 'cursuri', 8];

var del\_ar:\* = mAr.pop(); // Sterge ultimul element, si-i adauga valoarea in "del\_ar"

var nr\_el:Number= mAr.length; // Aflare nr. elemente dupa stergere

// Verificare cu "trace()" ("del\_ar" si elemente din "mAr")

trace(del\_ar); // 8

trace(nr\_el+ ' - '+ mAr); // 3 - web,www.marplo.net,cursuri

3. Cu metoda **shift()**  
          Sintaxa generala:  
                  **numeAr.shift()**  
          - metoda (functia) "shift()" sterge primul element din Array-ul "numeAr" (opusul lui "*unshift()*", care adauga un element la inceput), si returneaza valoarea lui.  
      Exemplu:

var mAr:Array = ['web', 'www.marplo.net', 'cursuri', 8];

var del\_ar:\* = mAr.shift(); // Sterge primul element, si-i adauga valoarea in "del\_ar"

var nr\_el:Number= mAr.length; // Aflare nr. elemente dupa stergere

// Verificare cu "trace()" ("del\_ar" si elemente din "mAr")

trace(del\_ar); // web

trace(nr\_el+ ' - '+ mAr); // 3 - www.marplo.net,cursuri,8

4. Cu metoda **splice()**  
          Sintaxa generala:  
                  **numeAr.splice(startIndex, nrElm)**  
          - metoda (functia) "splice()" sterge din "numeAr" un anumit numar de elemente, specificat la "nrElm", incepand cu cel al carui index e specificat la "startIndex".  
          - "nrElm" este optional, daca nu e adaugat, functia va sterge toate elementele incepand cu cel de la "startIndex" (inclusiv) pana la sfarsit.  
          - Dupa stergere, "splice()", reordoneaza indicii elementelor ramase.  
      Exemplu:

var mAr:Array = ['web', 'www.marplo.net', 'cursuri', 8];

mAr.splice(1, 2); // Sterge doua elemente, incepand cu cel care are index 1

var nr\_el:Number= mAr.length; // Aflare nr. elemente dupa stergere

// Verificare cu "trace()"

trace(nr\_el+ ' - '+ mAr); // 2 - web,8

*Functia "splice()" poate fi utilizata pentru stergere si /sau pentru adaugare elemente in Array. Formula completa fiind:****numeAr.splice(startIndex, nrElm, elm1, elm2, ...)*** *- Unde "elm1, elm2, ..." sunt elementele ce vor fi adaugate dupa efectuarea stergerii.  
- Daca la "nrElm" se da valoarea 0, functia va efectua doar adaugarea elementelor, dupa cel specificat la "startIndex".*

**2. Ordonare Array**

Ordinea elementelor dintr-un Array poate fi schimbata cu anumite functii specifice. Ordinea este importanta mai ales cand matricea este parcursa cu o instructiune repetitiva.

- **reverse()** - inverseaza ordinea elementelor.  
      Exemplu:

var mAr:Array = ['web', 'www.marplo.net', 'cursuri', 8];

mAr.reverse(); // Inverseaza ordinea, ultimul devine primul

// Verificare cu "trace()"

trace(mAr); // 8,cursuri,www.marplo.net,web

- **sort(Array.ARG)** - "Array.ARG" este optional, daca nu e specificat, aranjeaza elementele in ordine alfabetica, case-sensitive (adica, numerele inaintea literelor si "Z" precede "a").  
          - In loc de "ARG" se poate adauga:

* **CASEINSENSITIVE** - ordonarea se face case-insensitiva (fara diferenta intre majuscule si litere mici)
* **DESCENDING** - ordoneaza in ordine inversa alfabetic
* **NUMERIC** - doar pentru cazuri cand elementele sunt numere, altfel, daca sunt siruri, genereaza avertismente
* **RETURNINDEXEDARRAY** - face ca expresia sa returneze indicii, fara a afecta ordinea din Array
* **UNIQUESORT** - cu acest argument se poate verifica daca in Array exista 2 elemente indentice, deoarece in acest caz returneaza 0. Altfel, returneaza elementele in ordinea lor.

*Pot fi aplicate mai multe argumente in acelasi timp, dupa formula:****sort(Array.ARG1 | Array.ARG2)***

      Exemplu:

var mAr:Array = ['Web', 'www.marplo.net', 'cursuri', 8];

mAr.sort(); // Aranjeaza in ordine alfabetica, case-sensitive

// Verificare cu "trace()"

trace(mAr); // 8,Web,cursuri,www.marplo.net

// Modifica ordonarea, folosind ca argument "DESCENDING" (ordonare invers alfabetic)

mAr.sort(Array.DESCENDING);

trace(mAr); // www.marplo.net,cursuri,Web,8

**3. Alte metode (functii) pentru lucru cu Array**

Iata alte cateva metode ActionScript 3 pentru lucru cu Array.

- **join(sep)** - transforma matricea (grupeaza elementele) in sir. Elementele fiind separate prin ceea ce se specifica la "sep" (*precum "implode()" in PHP*).  
      Exemplu:

var mAr:Array = ['web', 'www.marplo.net', 'cursuri', 8];

var sir:String = mAr.join(' - '); // Transforma "mAr" in sir, adaugat in "sir"

// Verificare cu "trace()" valoarea lui "sir"

trace(sir); // web - www.marplo.net - cursuri - 8

- **concat(val1, val2)** - Uneste mai multe Array si elemente intr-un singur Array. "val1", "val2" pot fi variabile simple sau /si Array.  
      Exemplu:

var mAr:Array = ['web', 'www.marplo.net', 'cursuri'];

// Creaza un Array "arr2" format din "mAr" la care adauga cu "concat()" 2 elemente

var arr2:Array = mAr.concat(7, 8);

// Verificare cu "trace()" elementele din "arr2"

trace(arr2); // web,www.marplo.net,cursuri,7,8

var arr3:Array = [9, 10]; // Un alt Array

var finAr:Array = arr2.concat(arr3, 'abc'); // Adauga in "finAr" elementele din "arr2", "arr3" si 'abc'

// Verificare cu "trace()" elementele din "finAr"

trace(finAr); // web,www.marplo.net,cursuri,7,8,9,10,abc

- **push(val1, val2)** - Adauga argumentele "val1", "val2", ... intr-un Array, la sfarsit, in ordine dupa cele deja existente. "val1, "val2", ... pot fi variabile simple sau /si Array.  
      Exemplu:

var mAr:Array = ['web', 'www.marplo.net', 'cursuri'];

mAr.push(7, 8); // Adauga 7 si 8 in "mAr", la urma

// Verificare cu "trace()"

trace(mAr); // web,www.marplo.net,cursuri,7,8

- **unshift(val1, val2)** - Adauga argumentele "val1", "val2", ... intr-un Array, la inceput, in ordine, inaintea celor deja existente. "val1, "val2", ... pot fi variabile simple sau /si Array.  
      Exemplu:

var mAr:Array = ['web', 'www.marplo.net', 'cursuri'];

mAr.unshift(7, 8); // Adauga 7 si 8 in "mAr", la inceput

// Verificare cu "trace()"

trace(mAr); // 7,8,web,www.marplo.net,cursuri

- **slice(startIndex, endIndex)** - Preia un sub-Array cu un numar de elemente dintr-un Array, incepand de la elementul cu index "startIndex" (inclusiv) pana la cel cu indice "endIndex" (dar fara acesta), (daca "startIndex" are valoare negativa, numaratoarea incepe de la sfarsit).  
      Exemplu:

var mAr:Array = ['web', 'www.marplo.net', 'cursuri', 7, 8];

var subAr:Array = mAr.slice(1, 3); // Preia in "subAr" elementele de la index 1 pana la index 3

// Verificare cu "trace(subAr)"

trace(subAr); // www.marplo.net,cursuri

- **map(functie)** - Aplica o functie (specificata la argument) la fiecare element dintr-un Array (*precum "array\_map() din PHP*).  
      Exemplu:

// Definire functie care v-a fi aplicata elementelor unui Array

function oFunctie(elm:\*, i:Number, a:Array):Number

{

return elm\*elm; // Returneaza elementul la puterea 2

}

var mAr:Array = [5, 6, 7, 8];

// Aplica functia "oFunctie" la elementele din "mAr"

// Stocheaza rezultatul (noul Array) in "mAr2"

var mAr2:Array = mAr.map(oFunctie);

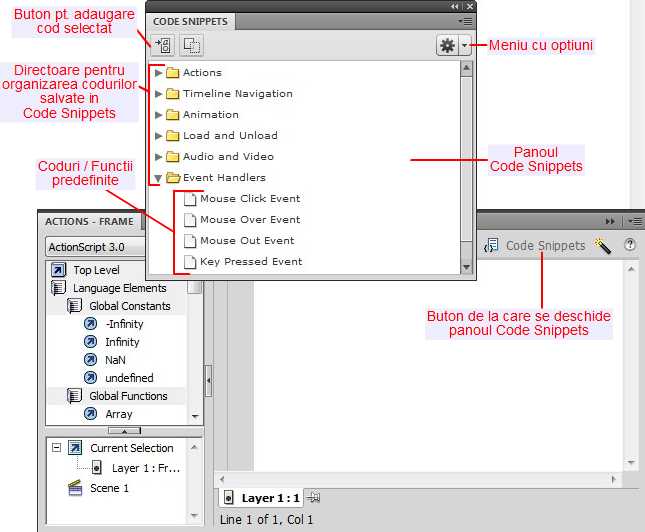
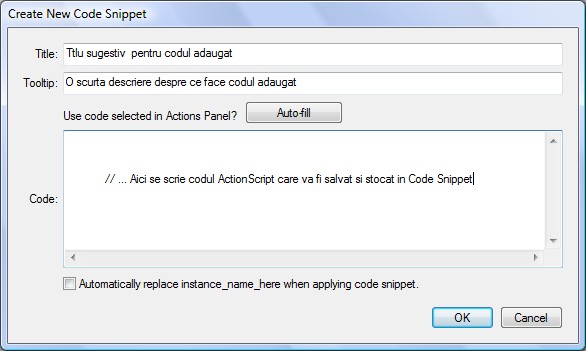
// Verificare cu "trace()"

trace(mAr2); // 25,36,49,64

- Sunt si alte functii pentru lucru cu Array in ActionScript 3; o lista si exemple cu acestea gasiti la pagina oficiala de pe site-ul de suport Adobe: [Functii Array - AS3](http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/Array.html).

**Code Snippets - Adaugare si Creare**

**Code Snippets** sunt bucati de cod predefinite, salvate in panoul "**Code Snippets**, care pot fi refolosite. Acestea sunt organizate pe categorii si functii, precum: "*Animation*", "*Timeline Navigation*", si altele.  
Acest poanou se deschide din meniul **Window -> Code Snippets**, sau de la butonul "*Code Snippets*" din fereastra pentru scriere cod ActionScript (dupa cum se vede in imaginea urmatoare).

  
Flash are cateva functii /coduri salvate in "Code Snippets". De exemplu, pentru a face un Buton sa execute salt la un anumit Cadru, in loc sa fie scris tot codul (cu evenimentul si functia necesara), se poate adauga direct din panoul "Code Snippets", fiind salvat in catalogul "**Timeline Navigation**", cu denumirea "***Click to Go to Frame and Play***". Adaugarea se face prin dublu-click, sau prin selectarea denumirii si click pe butonul "**Add to current Frame**".  
• Flash are adaugate in "Code Snippets" unele din cele mai utilizate functii pentru lucrul cu ActionScript. Ca sa le cunoasteti trebuie sa "rasfoiti" listele cu ele, denumirile prin care sunt salvate explica si rolul lor.  
  
Pe langa aceste functii predefinite incluse de Adobe Flash, puteti adauga si propriile bucati de cod ca sa fie stocate in "Code Snippets"; lucru util cand folositi mai des anumite expresii de cod mai mari, functii sau orice cod.  
• Pentru a crea coduri adaugate in panoul "Code Snippets", se da click pe meniul cu optiuni al panoului (in dreapta-sus) si se alege optiunea**Create New Code Snippet**, se va deschide o fereastra ca cea din imaginea urmatoare.  
  
- La **Title** se scrie numele (cat mai sugestiv) care va apare pentru acel cod adaugat in "Code Snippets".  
- Descrierea care e scrisa la **Tooltip** apare cand mouse-ul va fi pozitionat pe numele de stocare in panou.  
- Codul ActionScript se scrie in zona de text **Code**. Daca e deja o anumita bucata de cod selectata in panoul "Actions-Frame", cu butonul **Auto-fill** se adauga acel cod selectat in zona "Code".  
- Dupa ce se apasa butonul OK, codul respectiv este adaugat cu numele specificat la "Title" intr-un director cu numele "**Custom**" (creat automat de Flash).  
  
Codurile predefinite cat si cele adaugate in "Code Snippets" sunt stocate intr-un fisier XML.  
- Pentru a reedita un Code Snippet, se da click-dreapta pe el si se alege **Edit Code Snippets XML**. Se va deschide fisierul XML cu toate codurile predefinite din panou. In acel fisier trebuie cautat (cu "Ctrl+F") si editat codul dorit. Dupa care se apasa meniul **File -> Save**.  
- Optiunea "**Copy to Clipboard**" copie respectivul cod in memorie, care apoi poate fi adaugat (cu Paste) in "Action-Frame".  
- Cu "**Delete Code Snippet**" se sterge acel cod din panou.  
- Fisierul XML cu toate codurile stocate in "Code Snippets" poate fi exportat cu optiunea "**Export Code Snippets XML**".  
- Cu "**Reset to Default XML**" se reface panoul "Code Snippets" cu codurile initiale predefinite.

**Detectare evenimente in ActionScript 3**

**Evenimentele** (*events*) sunt actiuni realizate in prezentarea Flash de catre utilizator sau de alte elemente din acea prezentare.  
De exemplu, click-ul sau pozitionarea mouse-ului pe un element din Flash, scrierea intr-un camp Text Input, sunt evenimente facute de utilizator; iar, ajungerea la un anumit cadru a unei animatii sau finalizarea unei actiuni sunt evenimente realizate de elemente din prezentare.  
ActionScript 3 contine obiecte si metode speciale pentru detectarea si recunoasterea evenimentelor, in functie de acestea se pot executa anumite instructiuni cand acele evenimente se produc.  
Ca majoritatea lucrurilor in ActionScript 3, evenimentul este considerat obiect, deoarece are proprietati unde sunt stocate informatii despre el si metode pentru lucru cu datele lui.  
• Pe scurt, un eveniment lucreaza astfel: elementul legat de actiunea produsa, denumit "**event target**" (*eveniment tinta*) [un buton apasat, o animatie, etc.) transmite date la script, sub forma unui obiect (denumit "**event object**"). Scriptul trebuie sa contina o functie speciala cunoscuta ca "**event listener**" (*detector de eveniment*), prin care detecteaza evenimentul si preia datele lui pe care apoi le transmite la o functie ce poate efectua diferite comenzi scrise de programator.  
Functia /metoda folosita in ActionScript 3 pentru a detecta si captura evenimente este "**addEventListener()**". *E o metoda din clasa "****EventDispatcher****", si inregistreaza in script detectarea evenimentului specificat in ea.*  
Sintaxa generala pentru utilizarea acestei metode este:

* **event\_target.addEventListener(EventObject.TIP, oFunctie);  
  function oFunctie(ins\_EventObject:EventObject):void  
  {  
      // Codul care va fi executat  
  }**

- "event\_target" este obiectul la care se produce evenimentul (numele de instanta a lui. De ex., a unui buton care va fi apasat).  
- "EventObject" reprezinta categoria de evenimente ce trebuie detectata. Este si obiectul in care sunt stocate informatii despre evenimentul produs (De ex., MouseEvent, TextEvent).  
- "TIP" este o constanta ce reprezinta tipul evenimentului (cum ar fi: CLICK, MOUSE\_OVER, COMPLETE) din categoria specificata.  
            - Formula "event\_object.TIP" precizeaza exact evenimentul ce trebuie recunoscut si preluat. De ex., "*MouseEvent:CLICK*" are ca detectare momentul cand se da click (pe "event\_target"), sau, "*Event.COMPLETE*" detecteaza cand o actiune este finalizata (terminata).  
- "oFunctie" este functia care va fi apelata cand e realizat evenimentul, si va fi executat codul din ea.  
- "ins\_EventObject" este instanta la obiectul "EventObject" prin care pot fi utilizate (procesate) datele transmise in acel obiect.  
  
ActionScript 3 are o multime de evenimente ce pot fi recunescute, cum ar fi: **MouseEvent** (pt. mouse), **KeyboardEvent** (pt. tastatura), **TextEvent** (pt. campuri de text), **ScrollEvent** (pt. Scroll), **IOErrorEvent** (pt. cazuri cand apar erori), si altele. Unele din ele, cele mai utilzate, vor fi prezentate in lectiile urmatoare. Lista cu toate tipurile de evenimente, proprietatile, metodele si constantele lor poate fi gasita la pagina [Evenimente AS3](http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/events/Event.html).  
  
Iata un exemplu practic de utilizare a detectarii unui eveniment. Cu acesta se realizeaza afisarea intr-un camp text din prezentarea Flash a unui continut preluat (incarcat) dintr-un fisier TXT extern.

// Creare "event target"

// O instanta a obiectului URLLoader (pt. incarcare date de la o adresa URL

var urlLoader:URLLoader = new URLLoader( );

// Aplica metoda "load()" (de incarcare) la instanta URLLoader

// La argument creaza o instanta URLRequest pentru specificare adresa fisier

urlLoader.load(new URLRequest("un\_fisier.txt"));

// Setare "event listener", cu eveniment de detectare a finalizarii unei actiuni

// Aplicat la "urlLoader", recunoaste cand incarcarea fisierului s-a terminat

urlLoader.addEventListener(Event.COMPLETE, campTxt);

// Functia care va fi apelata de evenimentul setat

function campTxt(evt:Event):void

{

// Se defineste Instanta de obiect TextField pt. creare camp text

var camp\_txt:TextField = new TextField();

camp\_txt.x = 93; // Distanta fata de marginea din stanga

camp\_txt.y = 75; // Distanta fata de marginea de sus

camp\_txt.width = 125; // Lungimea campului de text

camp\_txt.height = 50; // Inaltimea

camp\_txt.border = true; // Activeaza afisarea unei borduri

// Adauga in campul de text datele incarcate din fisier, stocate in "evt"

camp\_txt.text = evt.target.data;

addChild(camp\_txt); // Adauga campul de text in prezentarea Flash

}

- La inceput trebuie creata o instanta la obiectul "URLLoader", cu aceasta va fi preluat (incarcat in Flash) continutul dintr-un fisier extern. Adresa URL de incarcare trebuie adaugata la aceasta instanta cu obiectul "URLRequest". Aceleiasi instante i-se aplica metoda "addEventListener()", la care se specifica sa recunoasca tipul de eveniment "Event.COMPLETE" (activat la finalizarea unei actiuni, aici cea de incarcare).  
- Cand incarcarea facuta de "urlLoader" se termina, e activat evenimentul inregistrat la acea instanta, care detecteaza finalizarea incarcarii. Atunci preia datele si le transmite la functia "campText", aceasta le retine in variabila (parametru) "evt", creaza un camp de text si cu expresia "evt.target.data" adauga datele preluate in camp si le afiseaza in prezentarea Flash.  
- *Studiati si explicatiile din cod.*  
Daca adaugati acest cod intr-un document Flash nou, faceti prezentarea SWF (cu Export) si in acelasi director creati si "un\_fisier.txt" cu un text in el, va rezulta o prezentare Flash ca aceasta:

*Adresa URL a fisierului, adaugata la "****URLRequest("URL\_fisier")****", trebuie sa fie completa sau relativa la prezentarea Flash, in functie de directotul in care se afla.*

**Stergere detectare eveniment**

Pentru a opri un "detector de eveniment" (event listener) setat cu "addEventListener()" sa primeasca notificari despre evenimentul ineregistrat, trebuie stearsa acea inregistrare, cu metoda "**removeEventListener()**" (apartine clasei "EventDispatcher").  
Formula generala de utilizare a acestei metode este  
                **event\_target.removeEventListener(EventObject.TIP, oFunctie);**  
- "event\_target" reprezinta instanta la care a fost aplicat detectorul de eveniment "addEventListener()" care trebuie sters.  
- "EventObject.TIP" e tipul evenimentului.  
- "oFunctie" e functia care a fost setata sa fie apelata.

*Este indicata stergerea inregistrarii de detectare a unui eveniment, cand nu mai e necesara in program, deoarece ajuta la eliberarea memoriei folosite si a procesorului.  
Evenimentul inregistrat e permanent activ si in legatura cu functia pe care o apeleaza (lucru stocat in memorie) pana cand e sters cu "removeEventListener()".*

Iata in exempul prezentat mai sus cum trebuie alicata metoda

// Creare "event target"

// O instanta a obiectului URLLoader (pt. incarcare date de la o adresa URL

var urlLoader:URLLoader = new URLLoader( );

// Aplica metoda "load()" (de incarcare) la instanta URLLoader

// La argument creaza o instanta URLRequest pentru specificare adresa fisier

urlLoader.load(new URLRequest("un\_fisier.txt"));

// Setare "event listener", cu eveniment de detectare a finalizarii unei actiuni

// Aplicat la "urlLoader", recunoaste cand incarcarea fisierului s-a terminat

urlLoader.addEventListener(Event.COMPLETE, campTxt);

// Functia care va fi apelata de evenimentul setat

function campTxt(evt:Event):void

{

// Se defineste Instanta de obiect TextField pt. creare camp text

var camp\_txt:TextField = new TextField();

camp\_txt.x = 100; // Distanta fata de marginea din stanga

camp\_txt.y = 75; // Distanta fata de marginea de sus

camp\_txt.width = 100; // Lungimea campului de text

camp\_txt.height = 50; // Inaltimea

camp\_txt.border = true; // Activeaza afisarea unei borduri

// Adauga in campul de text datele incarcate din fisier, stocate in "evt"

camp\_txt.text = evt.target.data;

addChild(camp\_txt); // Adauga campul de text in prezentarea Flash

// Dupa ce campul e adagat in prezentare, cu continutul incarcat

// Sterge inregistrarea de detectare a acestui eveniment, pt. a elibera memoria

urlLoader.removeEventListener(Event.COMPLETE, campTxt);

}

- instructiunea "*urlLoader.removeEventListener(Event.COMPLETE, campTxt);*" e adaugata in functia "campTxt()" dupa comanda de afisare a campului de text cu continutul incarcat din fisierul extern. Astfel, aceasta instructiune este executata dupa ce se termina rolul evenimentului de preluare a datelor incarcate si afisarea lor, prin urmare nu mai e necesara pasrtrarea lui in memorie.  
- *Daca aceasta instructiune ar fi scrisa in afara functiei apelate, "campText()", ar avea ca efect stergerea detectarii evenimentului inainte ca funcia sa fi fost apelata si nu va mai fi executata deoarece evenimentul nu mai e detectat.*

**Prioritatea de capturare evenimente**

In mod normal, daca sunt mai multe detectoare de eveniment inregistrate la acelasi obiect (cu tipuri egale sau diferite), ordinea de capturare si declansare a lor e cea in care sunt scrise in script. Dar aceasta ordine poate fi schimbata, folosind un "parametru de prioritate" in metoda "addEventListener()", dupa formula:  
                **event\_target.addEventListener(EventObject.TIP, oFunctie, useCapture, prioritate);**  
- La "useCapture" se pune valoarea **false**  
- "prioritate" e un numar intreg mai mare sau egal decat zero, prin care se determina prioritatea de executie (default are valoarea 0). Detectorul de evenimente cu prioritatea mai mare este declansat inaintea celorlati.  
De exemplu, la urmatoarele 2 evenimente inregistrate:

urlLoader.addEventListener(Event.COMPLETE, oFunctie, false, 0);

urlLoader.addEventListener(Event.COMPLETE, altaFunctie, false, 1);

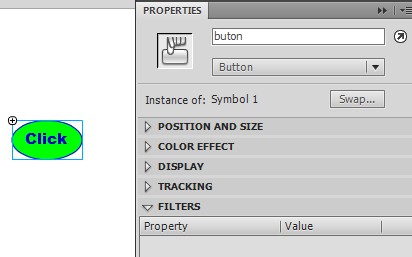
- Este declansat intai cel cu prioritatea 1, care apeleaza "altaFunctie", apoi, se executa cel setat cu prioritatea 0.

**MouseEvent - Evenimente pentru Mouse**

Cursorul mouse-ului e modul cel mai comun prin care utilizatorul interactioneaza cu prezentarea Flash.  
**Evenimentele pentru mouse** sunt diferitele actiuni ce pot fi facute cu mouse-ul, cum e apasarea Click, pozitionarea cursorului deasupra unui obiect sau chiar simpla miscare a lui. Toate aceste actiuni se numesc "**evenimente de mouse**" (*mouse events*) si pot fi detectate de clasa **MouseEvent**.  
Tipul evenimentelor din categoria **MouseEvent** sunt:

* **CLICK** - este declansat cand utilizatorul apasa si elibereaza click-ul (cu butonul din stanga) pe obiectul din prezentare
* **DOUBLE\_CLICK** - declanseaza evenimentul atunci cand e apasat dublu-click pe obiectul inregistrat. Acesta trebuie sa aibe activata /setata proprietatea ***doubleClickEnabled*** cu valoare ***true***
* **MOUSE\_DOWN** - detecteaza momentul cand click-ul e apasat (imediat, fara a fi necesar si eliberarea click-ului)
* **MOUSE\_UP** - executa evenimentul cand apasarea click este eliberata
* **MOUSE\_MOVE** - evenimentul e declansat de fiecare data cand cursorul mouse-ului e miscat pe zona obiectului
* **MOUSE\_OUT** - evenimentul este executat in momentul cand mouse-ul iese din suprafata obiectului sau a oricarui "element copil" din el
* **MOUSE\_OVER** - se declanseaza in momentul cand mouse-ul intra in suprafata obiectului monitorizat sau a oricarui "element copil" din el
* **MOUSE\_WHEEL** - executa evenimentul cand rotita de la mouse este rotita in moment ce cursorul se afla deasupra obiectului
* **ROLL\_OUT** - este similar cu MOUSE-OUT, diferenta fiind ca evenimentul e declansat doar cand mouse-ul paraseste complet zona obiectului, pe cand, la MOUSE\_OUT declansarea se produce si la iesirea din suprafata fiecarui "element copil" care se afla in obiectul monitorizat
* **ROLL\_OVER** - este similar cu MOUSE-OVER, diferenta fiind ca evenimentul e declansat cand mouse-ul intra in zona obiectului, pe cand, la MOUSE\_OVER declansarea se produce la intrarea in suprafata fiecarui "element copil" care se afla in obiectul monitorizat

Iata un exemplu cu MouseEvent pentru CLICK, MOUSE\_MOVE si MOUSE\_DOWN. Va fi creata o prezentare flash cu 2 cadre, in primul se adauga un buton si in al doilea o figura geometrica (o stea). Se adauga cod ActionScript 3 prin care atunci cand e apasat butonul se face salt la cadrul 2, in care e steaua. Cand mouse-ul e miscat pe stea, aceasta se invarte si se micsoreaza, iar daca se apasa click pe ea (MOUSE\_DOWN) face salt la primul cadru, (cu butonul).

**1.** Deschideti un document Flash nou, **ActionScript 3.0**  
**2.** Desenati in Scena un oval (sau un dreptunghi), (*puteti adauga si un text, "Click" pe oval, cu "Text Tool"*), selectati-l si transformati-l in Buton (din ***Modify -> Convert to Symbol*** iar la ***Type*** alegeti ***Button***).  
**3.** Dati numele **buton** instantei butonului din Scena (in panoul Properties, sus, unde e scris "<Instance Name>"). Dupa cum se vede in imaginea urmatoare.  
  
**4.** Click-dreapta pe Cadrul 2 in Timeline si alegeti "**Insert Blank Keyframe**" (se creaza un cadru nou gol). Si selectati instrumentul pentru desenat figuri geometrice "**PolyStar Tool**" (situat la butonul cu dreptunghi).  
**5.** Click pe butonul **Options** din panoul Properties pt. "PolyStar Tool", si in fereastra care se deschide, la ***Style*** alegeti optiunea **star** (ca sa deseneze o stea, puteti modifica numarul de colturi la "***Number of Sides***") si apasati OK.  
**6.** Alegeti o culoare gradient pentru stea (*de la butonul****Fill Color****, cea cu multe culori, pt. un efect al rotirii mai vizibil*) si desenati in Scena o stea.  
**7.** Transformati steaua in Movie Clip (de la meniul **Modify -> Convert to Symbol**, iar la ***Type*** puneti ***Movie Clip*** si apasati OK), apoi, in panoul Properties, in partea de sus, unde e scris "***<Instance Name>***" scrieti numele **stea** petru aceasta instanta (e necesar ca sa fie folosit in codul ActionScript).  
**8.** Creati un nou Layer (din **Insert -> Timeline -> Layer**) care va fi utilizat pt. adaugare coduri ActionScript.  
  - *Puteti sa-i dati un nume specific (de ex. "Actions" sau "Scripts"), prin dublu-click pe numele lui.*  
**9.** Click-dreapta pe Cadru 1 din acest al doilea Layer, alegeti **Actions**, iar in fereastra care se deschide pt. scriere cod ActionScript adaugati codul urmator.

stop(); // Opreste prezentarea la cadrul 1

// Inregistrare eveniment de mouse ( CLICK ) la instanta nuton

buton.addEventListener(MouseEvent.CLICK, gotoF2);

// Functia apelata la declansarea evenimentului inregistrat

function gotoF2(event:MouseEvent):void

{

gotoAndStop(2); // Face salt si oprire la Cadru 2

}

**10.** Click-dreapta pe Cadru 2 din Layer-ul nou creat, alegeti **Actions**, iar in fereastra care se deschide pt. scriere cod ActionScript copiati codul urmator.

// Inregistrare eveniment de mouse ( MOUSE\_MOVE ) la instanta stea

stea.addEventListener(MouseEvent.MOUSE\_MOVE, rotate);

// Inregistrare inca un eveniment de mouse ( MOUSE\_DOWN ) la instanta stea

stea.addEventListener(MouseEvent.MOUSE\_DOWN, gotoF1);

// Functia apelata la declansarea evenimentului MOUSE\_MOVE

function rotate(evt:Event):void

{

// Rotire obiect ce a declansat evenimentul

evt.target.rotationZ -= 2;

// Daca lungimea acestui obiect e mai mare de 80 pixeli

// micsoreaza lungimea si inaltimea cu 5

if(evt.target.width>88) {

evt.target.width -= 5;

evt.target.height -= 5;

}

}

// Functia apelata la declansarea evenimentului MOUSE\_DOWN

function gotoF1(evt2:MouseEvent):void

{

gotoAndStop(1); // Face salt si oprire la Cadru 1

}

  - *Acesta este folosit de Flash cand prezentarea ajunge la cadru 2.*

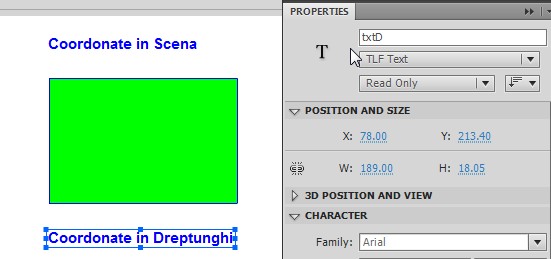
- Prin comanda "stop()" se opreste prezentarea la Cadrul unde e adaugata, ca sa ramana ce e afisat in el; altfel ar fi continuat cu urmatorul Frame.  
- expresia "**evt.target**" face referire la elementul care a initializat evenimentul detectat.  
- "evt.target.rotationZ -= 2;" roteste cu 2 grade in plan Z obiect ce a declansat evenimentul.  
- Acest cod va afisa urmatoarea prezentare Flash. Apasati click pe buton apoi miscati mouse-ul pe steaua care apare si dati click pe ea.

**Coordonate Mouse**

Cu obiectul **MouseEvent** se poate captura /afla si **coordonatele** mouse-ului in prezentarea Flash. Acesta are urmatoarele proprietati pentru aflarea pozitiei cursorului de mouse, in raport cu scena (coltul stanga-sus) si in raport cu elementul ce declanseaza evenimentul.

* **localX** - coordonata pe axa orizontala a mouse-ului in obiectul la care se produce evenimentul
* **localY** - coordonata pe axa verticala a mouse-ului in obiectul la care se produce evenimentul
* **stageX** - coordonata pe axa orizontala, raportata la coltul din stanga-sus a scenei
* **stageY** - coordonata pe axa verticala, raportata la coltul din stanga-sus a scenei

Preluand aceste coordonate, se pot executa diferite instructiuni si comenzi in functie de valoarea lor. Dupa cum puteti vedea in exemplul urmator.  
Efectuati urmatorii pasi.

**1.** Deschideti un document Flash nou, **ActionScript 3.0**  
**2.** Desenati pe mijlocul Scenei un dreptungi si transformati-l in Movie Clip (de la meniul **Modify -> Convert to Symbol**, iar la ***Type*** alegeti ***Movie Clip*** si apasati OK), apoi, in panoul Properties, in partea de sus, unde e scris "***<Instance Name>***" scrieti numele **dreptunghi** (numele de instanta folosit in codul ActionScript).  
**3.** Alegeti instrumentul "Text Tool" si scrieti deasupra dreptunghiului: **Coordonate in Scena**, si dati acestui camp de text numele de instanta **txtS**.  
Scrieti si sub dreptungi textul **Coordonate in Dreptungi**, si dati acestui camp de text numele de instanta **txtD**. Dupa cum se vede in imaginea urmatoare.  
  
**4.** Click-dreapta pe cadrul 1 in Timeline si alegeti **Action**, iar in panoul care se deschide pt. ActionScript adaugati urmatorul cod:

// Inregistrare eveniment de mouse ( MOUSE\_MOVE ) la instanta dreptunghi

dreptunghi.addEventListener(MouseEvent.MOUSE\_MOVE, getCoord);

// Functia apelata

function getCoord(evt:MouseEvent):void

{

// Adauga coordonatele mouse-scena in instanta txtS

txtS.text = 'Coordonate in Scena: '+ evt.stageX+ ', '+ evt.stageY;

// Adauga coordonatele mouse-dreptunghi in instanta txtD

txtD.text = 'Coordonate in Dreptunghi: '+ evt.localX+ ', '+ evt.localY;

// Setare variabila 'culoare' pt. schimbarea culorii

var culoare:ColorTransform = dreptunghi.transform.colorTransform;

// Daca coordonata X locala e mai mare de 90 (pixeli) seteaza culoare galbena

// Altfel, o seteaza verde

if(evt.localX>90) culoare.color = 0xeded01;

else culoare.color = 0x00fe00;

// Aplica schimbarea culoarii la instanta dreptunghi

dreptunghi.transform.colorTransform = culoare;

}

**5.** Apasati Ctrl+Enter ca sa testati rezultatul. Va apare urmatoarea prezentare Flash:.

- Daca miscati mouse-ul pe dreptunghi, in textul de sus va afisa si locatia cursorului in raport cu Scena, si in campul text de jos va afisa coordonatele cursorului in dreptunghi (in raport cu coltul din stanga-sus al lui).  
- Cand mouse-ul ajunge pe dreptunghi la coordonate X (pe orizontala) cu valoare mai mare de 90 pixeli (*in jumatatea dreapta a dreptunghiului*) schimba culoarea acestuia.

- "evt.stageX", "evt.stageY", "evt.localX" si "evt.localY" returneaza coordonatele mouse-ului, in Scena si dreptunghi, stocate in obiectul "evt".  
- Alte explicatii necesare sunt in cod.

**KeyboardEvent - Evenimente pentru Tastatura**

**Evenimentele pentru Tastatura** (*cand e apasat un buton de la tastatura*) sunt detectate cu obiectul **KeyboardEvent**. Acesta are doua tipuri:

* **KEY\_DOWN** - declanseaza evenimentul cand butonul e apasat.
* **KEY\_UP** - declanseaza evenimentul cand apasarea butonului este eliberata.

**KeyboardEvent** are 6 proprietati:

* **charCode** - contine un numar ce reprezinta valoarea de cod a caracterului de la butonul apasat sau eliberat.  
  - De exemplu: pentru 'a' valoarea de cod e 97, pt. 'A' e 65, pt. '3' este 51, iar pentru 'Shift+3' (adica '#') este 35.
* **keyCode** - contine un numar cu valoarea codului butonului apasat. Diferenta fata de "charCode" e aceea ca "charCode" returneaza valoarea 0 pt. tastele care nu reprezinta un caracter (Shift, Ctrl, sagetile, ...), dar "keyCode" contine un numar specific si pentru aceste taste.  
  - De exemplu: pt. 'Shift' returneaza numarul 16, pt. 'sageata-dreapta' numarul 31.
* **keyLocation** - contine un numar ce indica locatia butonului apasat. Folosit pt. taste care apar mai mult de o data pe tastatura, cum sunt Shift, Alt.
* **altKey** - **true** daca tasta 'Alt' e activa, sau **false** daca e inactiva
* **ctrlKey** - **true** daca tasta 'Ctrl' e activa, sau **false** daca e inactiva.
* **shiftKey** - **true** daca 'Shift' e activ, sau **false** daca e inactiv.

- De retinut ca valorile codurilor pt. "charCode" si "keyCode" difera in functie de sistemul de operare al calculatorului. Exemplele de aici sunt pentru Windows.  
- Ca sa fie detectat butonul apasat in prezentarea Flash, se aplica metoda **addEventListener()** (cu **KeyboardEvent**) la Scena (obiectul **stage**).  
  
Iata un exemplu cu "KeyboardEvent" si KEY\_DOWN in care e folosita proprietatea "keyCode", prin care va fi miscat un dreptunghi cu sagetile de la tastatura; iar proprietatea "shiftKey" va fi utilizata pt. a modifica viteza de miscare.

**1.** Deschideti un document Flash nou, **ActionScript 3.0**  
**2.** Desenati un dreptungi si transformati-l in Movie Clip (de la meniul **Modify -> Convert to Symbol**, iar la ***Type*** alegeti ***Movie Clip*** si apasati OK), apoi, in panoul Properties, in partea de sus, unde e scris "***<Instance Name>***" scrieti numele **dreptunghi** (necesar in codul ActionScript).  
**3.** Click-dreapta pe Cadru 1 in Timeline si alegeti **Action**, iar in panoul care se deschide pt. ActionScript adaugati urmatorul cod:

// Inregistrare eveniment pt. detectare apasare buton de la tastatura

// Detectarea se aplica obiectului 'stage' pt. a fi activ in toata prezentarea

stage.addEventListener(KeyboardEvent.KEY\_DOWN, moveF);

// Functia apelata la apasarea tastei

function moveF(key\_evt:KeyboardEvent):void

{

// Setare variabila pt. viteza de miscare, in functie de "shiftKey"

var vi:Number = key\_evt.shiftKey ? 12 : 2;

// Instructiune "switch" cu valori si comenzi pt. tastele sageti

// Codurile pt. sagetile: Stanga, Dreapta, Jos, Sus

switch (key\_evt.keyCode)

{

case 37:

dreptunghi.x -= vi;

break;

case 39:

dreptunghi.x += vi;

break;

case 38:

dreptunghi.y -= vi;

break;

case 40:

dreptunghi.y += vi;

break;

default:

break;

}

}

**4.** Apasati Ctrl+Enter ca sa testati rezultatul. Va apare urmatoarea prezentare Flash:.

- Cand apasati sagetile de la tastatura, dreptunghiul se va misca in directia data de fiecare sageata (*intai sa dati click pe prezentare*).  
- Daca tineti apasata tasta Shift, viteza de miscare va fi mai mare.

- Expresia "var vi:Number = key\_evt.shiftKey ? 12 : 2;" defineste valoarea 12 daca "key\_evt.shiftKey" e TRUE (cand Shift e apasat) sau 2 daca e FALSE (Shift neapasat).  
- Comenzile din instructiunea "switch" modifica valoarea 'x' si 'y' pt. instanta "dreptunghi" (distanta X si Y), cu valoarea variabilei "vi", in functie de sageata apasata de la tastatura.

*Ca sa aflati codul pt. fiecare tasta, puteti folosi urmatorul script AS3:****stage.addEventListener(KeyboardEvent.KEY\_DOWN, keyC);  
                function keyC(key:KeyboardEvent):void { trace(key.keyCode); }*** *- Afiseaza in panoul Output codul tastei apasate.*

**Creare si Adaugare text cu ActionScript 3**

In prima lectie, dupa Introducere, e aratat cum se creaza un script ActionScript, in care e prezentat si un exemplu simplu de adaugare text cu AS3. In aceasta lectie este explicat mai detaliat modul de creare si adaugare text intr-o prezentare Flash cu ActionScript 3.0, precum si cateva proprietati si metode pentru definirea textului.  
- De ce sa creem Text sau alte obiecte cu ActionScript cand acestea pot fi create direct in Scena? De exemplu, pot fi facute prezentari, jocuri care in urma unor anumite actiuni ale utilizatorului sa genereze afisarea unui Text specific; acest lucru se realizeaza cu ActionScript.  
  
• Pentru a crea text cu AS3 se foloseste obiectul /clasa **TextField**, aceasta contine mai multe proprietati si metode pentru definirea textului; cum ar fi, pentru culoare, marime, pozitie, si altele.  
Iata cateva din proprietatile ce pot fi aplicate direct instantei pt. text.

* **x** - Seteaza distanta campului pt. text fata de marginea din stanga
* **y** - Seteaza distanta campului pt. text fata de marginea de sus
* **width** - Defineste lungimea campului pt.afisarea textului
* **height** - Defineste inaltimea campului pt.afisarea textului
* **textColor** - Culoarea textului (in format **0xRRGGBB**, RGB = Red [*rosu*], Green [*verde*], Blue [*albastru*])
* **backgroundColor** - Defineste o culoare pentru fundalul zonei de text
* **background** - **true** sau **false**, specifica daca va fi afisat fundalul definit
* **borderColor** - Defineste o culoare pt. bordura /marginea din jurul textului
* **border** - **true** sau **false**, specifica daca va fi afisata bordura definita
* **autoSize** - Controleaza aliniamentul zonei de text (cat cuprinde textul) in campul pentru text, realizeaza si o auto-dimensionare a zonei textului in campul pt. text.  
  Valorile ce le primeste sunt: "none", "left", "right" si "center".
* **text** - Adauga textul (sirul) in instanta "textField"
* **htmlText** - Adauga un text cu format HTML.  
  Sunt recunoscute tag-urile: ***<a> (cu atributele: "href" si "tatget"), <b>, <br>, <font>, <img> (JPG, GIF, PNG, SWF. Foloseste un "id" in care se specifica numele instantei MovieClip care va contine imaginea), <i>, <li> (fara <ul> sau <ol>), <p>, <span>, <u>; precum si entitatile HTML: &lt;, &gt;, &amp;, &quot;, &apos;***
* **styleSheet** - Ataseaza la text o instanta cu stiluri CSS, creata cu clasa "StyleSheet"
* **multiline** - **true** sau **false**, indica daca in campul pt. text pot fi sau nu mai multe linii (necesar cand se adauga text cu format HTML)
* **wordWrap** - **true** sau **false**, indica daca va avea sau nu "word-wrap" campul de text.

- Lista completa cu proprietatile si metodele ActionScript 3 poate fi gasita la pagina oficiala Adobe Flash: [Proprietati si metode textField](http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/text/TextField.html).  
- Intai se creaza o variabila cu tipul "TextField" si instanta la aceasta clasa, dupa care, la instanta aceasta se pot adauga textul, proprietatile si functiie necesare, iar pentru adaugarea textului in prezentare se foloseste **addChild()**.  
Iata prin exemplu practic cum se folosesc aceste proprietati:

**1.** Deschideti un document Flash nou, **ActionScript 3.0**  
**2.** Click-dreapta pe primul cadru din Timeline si alegeti **Actions**  
**3.** In fereastra care se deschide pentru scriere ActionScript, adaugati urmatorul cod:

// Initializeaza o instanta "TextField" intr-o variabila "txt" de acelasi tip "TextField"

var txt:TextField = new TextField();

// Se aplica proprietati pt. campul de text, la instanta creata

txt.x = 50; // Distanta fata de marginea din stanga

txt.y = 80; // Distanta fata de marginea de sus

txt.width = 200; // Lungimea campului pt. text

txt.textColor = 0x0000da; // Culoarea textului

txt.backgroundColor = 0xededfe; // Seteaza culoare fundal

txt.background = true; // Activeaza afisarea culori de fundal

txt.borderColor = 0x007800; // Seteaza culoare bordura

txt.border = true; // Activeaza afisare bordura

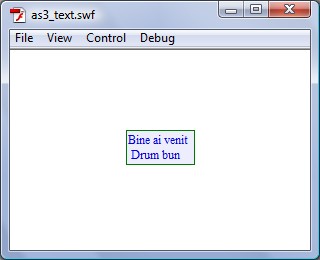
txt.text = "Bine ai venit \n Drum bun"; // Adauga textul

// Alinierea zonei cu text (fundal si bordura) in campul creat

txt.autoSize = "center";

// Aplica "addChild()" la instanta ca sa o adauge in prezentarea Flash

addChild(txt);

  - "\n" adauga o linie noua.  
  - Exista diferenta intre "zona de text" si "campul pt. text". Cand proprietatea "autoSize" are valoare "none", zona de text si campul ei sunt acelasi, dar cand i-se da o alta valoare (left, right, center) zona textului (fundalul si bordura) va cuprinde doar suprafata ocupata de el.  
**4.** Pentru a vedea rezultatul, apasati "Ctrl+Enter", va apare ca-n imaginea urmatoare:.  


• In text se pot folosi si unele tag-uri HTML, precum si stiluri CSS pentru designul textului.  
- Pentru adaugare text in care sa fie recunoscute tag-urile HTML, se foloseste proprietatea "**htmlText**". *Sunt recunoscute doar cateva din tag-urile HTML, cele prezentate mai sus la aceasta proprietate.*  
- Pentru a include si stiluri CSS, se foloseste proprietatea "**styleSheet**", obiectul "**StyleSheet**" si metoda acestuia, "**parseCSS("stilCSS")**", care preia ca argument un sir cu proprietatile CSS. Lista cu proprietatile si metodele clasei "**StyleSheet**" o gasiti la pagina: [Proprietati si metode StyleSheet](http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/text/StyleSheet.html)  
                Proprietatile CSS recunoscute sunt: ***color, display, font-family, font-size, font-style, font-weight, leading, letter-spacing, margin-left, margin-right, text-align, text-decoration, text-indent***.  
  
In continuare puteti testa urmatorul cod AS3, care adauga intr-o prezentare Flash text formatat cu tag-uri HTML si stiluri CSS (*explicatiile sunt in documentatia din script*).

// Initializeaza o instanta "TextField" intr-o variabila "txt" de acelasi tip "TextField"

var txt:TextField = new TextField();

// Definire proprietati pt. campul de text

txt.width = 180;

txt.height = 100;

txt.wordWrap = true;

txt.multiline = true;

// Setare variabila cu stilurile CSS aplicate

var css\_st:String = ".bv{color:#0808fe; font-family:Arial; font-size:16px; font-weight:bold;} .adf{color:#ed0708; font-family:Verdana; font-size:13px; font-style:italic}";

// Defineste instanta la obiectul "StyleSheet"

var styles:StyleSheet = new StyleSheet();

// Aplica metoda "parseCSS" la variabila cu sirul CSS (css\_st)

styles.parseCSS(css\_st);

// Ataseaza la "txt", cu proprietatea "styleSheet" instanta cu stilul CSS din "styles"

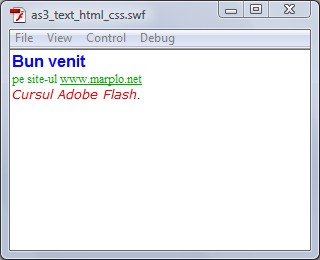
txt.styleSheet = styles;

// Adaugare text cu format HTML si 2 clase pt. CSS ("bv" si "adf")

txt.htmlText = '<span class="bv">Bun venit</span><br><font size="12" color="#00a800">pe site-ul <u><a href="http://www.marplo.net">www.marplo.net</a></u></font><br><span class="adf">Cursul Adobe Flash</span>.';

// Aplica "addChild()" la instanta ca sa o adauge in prezentarea Flash

addChild(txt);

- Acest cod va afisa o prezentare Flash precum cea din imaginea urmatoare:  


**Utilizare TextFormat**

Utilizarea obiectului **TextFormat** este un alt mod de a formata si defini grafic un text in campul "textField".  
Pentru a folosi **TextFormat**, trebuie creata o instanta la acest obiect (cu **new TextFormat**), la care se pot aplica proprietatile si metodele acestuia, pe care le gasiti la pagina: [Proprietati si metode TextFormat](http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/text/TextFormat.html); apoi, instanta acestui obiect se adauga la cea pentru text (creata cu "textField").

- Iata si un exemplu cu "TextFormat":

// Se creaza o instanta TextFormat

var textF:TextFormat = new TextFormat();

// Se aplica proprietati pt. formatul textului care va fi ulterior creat

textF.leftMargin = 55; // Distanta fata de marginea din stanga a campului pt. text

textF.font = "Arial"; // Defineste fontul

textF.color = 0x5678fe; // Culoarea

textF.size = 17; // Marimea textului (in pixeli)

// Initializeaza o instanta "TextField" intr-o variabila "txt"

var txt:TextField = new TextField();

// Definire proprietati pt. campul de text

txt.width = 240;

txt.wordWrap = true;

txt.multiline = true;

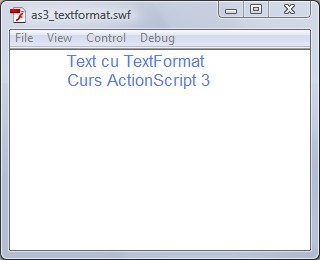
txt.text = "Text cu TextFormat \nCurs ActionScript 3"; // Adauga Textul

// Aplica metoda "setTextFormat()" la campul de text, cu parametru instanta "textF"

txt.setTextFormat(textF);

// Aplica "addChild()" la instanta ca sa o adauge in prezentarea Flash

addChild(txt);

- Acest cod va afisa o prezentare Flash ca cea din imaginea urmatoare:  
  
• Se poate aplica TextFormat doar unei portiuni din text, cu metoda:  
                nameField.setTextFormat(textFormat, caracterStart, caracterEnd);  
- "nameField" este instanta creata cu "**TextField**"  
- "textFormat" este instanta creata cu "**TextFormat**"  
- "caracterStart" e un numar ce reprezinta caracterul din sirul text de la care incepe aplicarea TextFormat  
- "caracterEnd" e un numar ce reprezinta caracterul din sirul text unde se termina aplicarea TextFormat.  
      *De ex., la scriptul anterior, incercati:****txt.setTextFormat(textF, 5, 15);***

*Caracterele folosite pentru text in instanta TextField pot fi restrictionate cu expresii tip REGEX, folosind proprietatea****restrict****.  
Exemple:  
          textField.restrict = "0-9";       // Permite doar numere  
          textField.restrict = "0-9 A-F";       // Permie doar numere si litere mari de la A la F  
          textField.restrict = "0-9 A-F ^ a-z";       // Permie numere si litere mari de la A la F, fara litere mici*

• O alta modalitate de a crea campuri de text in Flash, in care textul sa fie adaugat interactiv cu ActionScript, e combinarea trasarii in Scena a campului pentru text si codul AS3.  
Metoda e simpla, se creaza in Scena campul pt. text (cu "Text Tool"), i-se da un nume de instanta acelui camp, apoi, numele respectiv poate fi utilizat in ActionScript ca si cum ar fi o Instanta de obiect "TextField".  
Avantajul e ca se pot combina optiunile grafice care pot fi aplicate textului cu instrumentele din Scena si interactivitatea data de ActionScript.  
Vedeti tutorialul [Adaugare Text cu ActionScript 3 in camp Text din Scena](http://www.marplo.net/flash/text-actionscript-camp-scena_t).

**Campuri de Text Input**

**Campurile de Text Input** sunt zone in care textul poate fi modificat si adaugat de utilizator.  
Pentru a crea un **camp text input**, se foloseste proprietatea "**type**" a obiectului **TextField**, cu valoarea "**input**" (sau valoarea**TextFieldType.INPUT**).  
Campurile Text Input fiind create cu obiectul "**TextField**", se pot aplica si celelalte proprietati ale acestui obiect, pentru dimensionare, pozitie, formatarea textului si a zonei de text. Cum ar fi, definirea unei borduri, culoare background, setare numar caractere permise sau restrictionare caractere ce pot fi scrise in campul input.

Iata un exemplu cu Text Input (*Explicatiile necesare sunt in cod*):

// Definire Instanta de obiect TextField

var txt\_inp:TextField = new TextField();

// Se aplica proprietati instantei create

txt\_inp.x = 100; // Distanta fata de marginea din stanga

txt\_inp.y = 50; // Distanta fata de marginea de sus

txt\_inp.width = 100; // Lungimea campului de text

txt\_inp.height = 20; // Inaltimea

txt\_inp.type = "input"; // Seteaza campul de text ca Input

txt\_inp.border = true; // Activeaza afisarea unei borduri

txt\_inp.borderColor = 0x0000da; // Defineste culoarea bordurii

// Permite doar adaugarea de numere si litere mici de la 'a' la 'e'

txt\_inp.restrict = "0-9 a-e";

// Setare numar maxim de caractere ce pot si scrise

txt\_inp.maxChars = 28;

txt\_inp.text = 'Nume';

// Adauga Instanta cu "txt\_inp" in prezentarea Flash

addChild(txt\_inp);

- Acest cod va afisa urmatoarea prezentare Flash. Daca testati, in campul de text afisat puteti sterge si adauga alt text, dar, comform setarii "*txt\_inp.restrict = "0-9 a-e";*", utilizatorul poate scrie doar numere si litere de la 'a' la 'e'.

- Pentru ca in campul Input sa poata fi adaugate mai multe randuri de text (cu Enter), trebuie aplicata proprietatea **multiline**, cu valoare **true**.  
- Pentru a face ca zona Input Text sa fie utilizata ca un camp Input pentru parola (caracterele scrise sunt inlocuite cu asterix "\*") se foloseste proprietatea**displayAsPassword** cu valoarea **true**.  
- Ca la orice camp de text creat cu "TextField", acesta poate fi formatat cu proprietatile obiectului "**TextFormat**".

**1. Utilizare TextEvent Input**

In general, rolul unui camp de Text Input e acela de a utiliza textul adaugat de utilizator pentru anumite actiuni.  
Cu obiectul **TextEvent** ataugat la "**addEventListener()**" Flash recunoaste /detecteaza momentul (sau evenimentul) cand utilizatorul scrie ceva in campul Input; poate sa preia caracterul adaugat sau textul din camp, iar in functie de acestea se pot executa diferite instructiuni.  
Sintaxa pentru utilizarea obiectului "TextEvent" este urmatoarea:

* **camp\_txt.addEventListener(TextEvent.TEXT\_INPUT, oFunctie);  
  function oFunctie(insTxtEv:TextEvent):void  
  {  
      // Codul care va fi executat  
      // cand utilizatorul adauga un caracter in "camp\_txt"  
  }**

- "camp\_txt" este instanta "TextField" pt. campul de text Input.  
- "addEventListener()" este o metoda ActionScript 3 de recunoastere /detectare a evenimentelor realizate in prezentarea Flash.  
- "TextEvent.TEXT\_INPUT" este obiectul si cu tipul evenimentului ce trebuie recunoscut si preluat.  
- "oFunctie" este functia care va fi apelata /executata cand utilizatorul scrie in campul Input.  
- "insTxtEv" este instanta la obiectul "TextEvent" prin care pot fi utilizate (procesate) date preluate.  
            - Cu **insTxtEv.text** se preia ultimul caracter introdus de utilizator in zona de text Input.

*Evenimentul nu este expediat la stergerea caracterelor (cu Delete sau Backspace).  
- Pentru a preveni /anula adaugarea in campul Input a caracterului tastat de utilizator, dar evenimentul sa fie insa capturat, se foloseste in functia apelata (aici "txtInp()" metoda****preventDefault()****. Este utila cand, de ex., se doreste afisarea altui cuvant sau caracter in functie de ce e scris.*

Iata un exemplu de cod ActionScript 3 prin care se recunoaste momentul cand sunt scrise caractere intr-un camp text Input, si in functie de numarul de caractere adaugate se modifica culoarea bordurii si cea de fundal. *Studiati si explicatiile din cod.*

// Definire Instanta de obiect TextField

var txt\_inp:TextField = new TextField();

// Se aplica proprietati instantei create

txt\_inp.x = 100; // Distanta fata de marginea din stanga

txt\_inp.y = 50; // Distanta fata de marginea de sus

txt\_inp.width = 100; // Lungimea campului de text

txt\_inp.height = 20; // Inaltimea

txt\_inp.type = "input"; // Seteaza campul de text ca Input

txt\_inp.border = true; // Activeaza afisarea unei borduri

txt\_inp.borderColor = 0x0000da; // Defineste culoarea bordurii

txt\_inp.maxChars = 28; // Numar maxim de caractere permise

txt\_inp.backgroundColor = 0xfefe18; // Defineste culoarea fundal

txt\_inp.background = true; // Activeaza afisare fundal

// Adauga Instanta cu "txt\_inp" in prezentarea Flash

addChild(txt\_inp);

// Se defineste expresia ce recunoaste evenimentul din campul Input

txt\_inp.addEventListener(TextEvent.TEXT\_INPUT, txtInp);

// Functia ce va fi apelata de evenimentul setat

function txtInp(txtEvent:TextEvent):void

{

// Preia numarul de caractere din "txt\_inp"

var nr:Number = txt\_inp.text.length;

// Defineste efectuarea unor comenzi in functie de nr. de caractere

// Schimbare culoare bordura si daca "nr>6" adauga background

if(nr<3) txt\_inp.borderColor = 0xda0000;

else if(nr<5) txt\_inp.borderColor = 0x00da00;

else if(nr>5) txt\_inp.backgroundColor = 0xe8e8fe;

}

- Daca adaugati acest cod intr-un document Flash nou, va rezulta urmatoarea prezentare:

- Vedeti rezultatul adaugand mai multe caractere in campul de text (se va modifica culoarea bordurii si la urma culoarea de fundal).

*Pentru a preveni /anula adaugarea in campul Input a caracterului tastat de utilizator, dar evenimentul sa fie insa capturat, se aplica in functia apelata (aici "txtInp()"), la obiectul cu evenimentul transmis (aici "txtEvent"), metoda****preventDefault()****(de ex.,****txtEvent.preventDefault();****). Este utila cand, de ex., se doreste afisarea altui cuvant sau caracter in functie de ce se scrie.  
- Pentru a adauga mai multe campuri de text input, trebuie creata cate o instanta cu "TextField" pt. fiecare Input.*

**2. Evenimente FocusEvent - FOCUS\_IN, FOCUS\_OUT**

In afara de evenimentul creat la adaugarea de caractere, se pot efectua diferite actiuni si pentru momentele cand utilizatorul da click /intra pe un camp Text Input (*FOCUS\_IN*) sau iese din el (*FOCUS\_OUT*), utilizand evenimentele **FocusEvent** (***FOCUS\_IN*** si ***FOCUS\_OUT***).  
Metoda este similara cu cea prezentata mai sus, pentru "TextEvent"

Studiati urmatorul exemplu si explicatiile din cod. Aici este definita o functie prin care pot fi usor definite mai multe campuri de text input, apoi se scriu expresiile pt. preluarea evenimentelor **FocusEvent** si functiile ce vor fi apelate de acestea.

// Functie pentru creare campuri Text Input

// Preia ca argumente: instanta, distanta Y si text

function setInput(inst, dy, txt):void

{

// Se aplica proprietati instantei primite "inst"

inst.x = 100; // Distanta fata de marginea din stanga

inst.y = dy; // Distanta fata de marginea de sus

inst.width = 100; // Lungimea campului de text

inst.height = 20; // Inaltimea

inst.type = "input"; // Seteaza campul de text ca Input

inst.border = true; // Activeaza afisarea unei borduri

inst.borderColor = 0x0000da; // Defineste culoarea bordurii

inst.maxChars = 28; // Numar maxim de caractere permise

inst.backgroundColor = 0xe8e8e8; // Defineste culoarea fundal

inst.background = true; // Activeaza afisare fundal

inst.text = txt; // Textul initial afisat in Input

// Adauga Instanta in prezentarea Flash

addChild(inst);

}

// Functia pt. FOCUS\_IN

function setFIn(focus:FocusEvent):void

{

// Defineste culoare bordura si background pt. campul tinta (target)

(focus.target as TextField).borderColor = 0x00da00;

(focus.target as TextField).backgroundColor = 0xfefe08;

}

// Functia pt. FOCUS\_OUT

function setFOut(focus:FocusEvent):void

{

// Defineste culoare bordura si background pt. campul tinta (target)

(focus.target as TextField).borderColor = 0xfe0000;

(focus.target as TextField).backgroundColor = 0xe7e8fe;

}

// Definire prima si a doua Instanta de obiect TextField

var txt\_inp1:TextField = new TextField();

var txt\_inp2:TextField = new TextField();

// Setare pt. recunoastere evenimente FocusEvent (FOCUS\_IN si FOCUS\_OUT), pt fiecare instanta

txt\_inp1.addEventListener(FocusEvent.FOCUS\_IN, setFIn);

txt\_inp1.addEventListener(FocusEvent.FOCUS\_OUT, setFOut);

txt\_inp2.addEventListener(FocusEvent.FOCUS\_IN, setFIn);

txt\_inp2.addEventListener(FocusEvent.FOCUS\_OUT, setFOut);

// Apelare functie "setInput" pt. definirea campurilor la fiecare instanta TextField

setInput(txt\_inp1, 50, 'Input 1');

setInput(txt\_inp2, 80, 'Input 2');

- Acest cod afiseaza 2 campuri de Text Input. Cand se da click pe unul din ele (oricare) se schimba culoarea lui de fundal si a bordurii (dupa cum e setat in functia "setFIn()"), iar cand se iese din acel camp se modifica din nou acele culori (dupa cum e setat in functia "setFOut()"). Puteti testa si vedea in prezentarea urmatoare.

**addChild si removeChild**

**addChild()** si **removeChild()** sunt metode din ActionScript prin care se adauga si se sterg elemente vizuale in Scena.

**1. addChild**

**addChild** este o functie ActionScript care adauga elemente vizuale in prezentarea Flash. Sintaxa ei generala este:  
                **addChild(displayObject);**  
- Unde, "displayObject" este obiectul ce trebuie afisat in prezentarea Flash (cum ar fi: un camp de text, o figura geometrica, o imagine, etc.).  
Se poate vedea in urmatorul exemplu cum se aplica. Scriptul din acest exemplu afiseaza trei campuri de text, cu texte care sunt preluate dintr-un Array, si culoare de fundal pt. fiecare camp luate din alt Array. Vedeti si explicatiile din cod.

var texte:Array = ['cursuri', 'tutoriale', 'www.marplo.net']; // Array cu texte pt. afisat

var bcolor:Array = [0xe8e9fe, 0xeded01, 0xfedada]; // Array culori pt. fondul fiecarui text

var lung:Number = 100; // Variabila folosita pt. lungimea campului de text

// Parcurge Array-ul "texte"

for(var i:Number=0; i<texte.length; i++)

{

// Pt. fiecare element parcurs creaza o instanta (un camp) "TextField"

// Seteaza inaltimea campului, lungimea (lung) si distanta 'x' in functie de 'i' (pt. afisare in linie)

// Adauga culoare de fundal (background) din Array-ul "bcolor" si textul din elementul parcurs

var txt:TextField = new TextField();

txt.height = 25;

txt.width = lung;

txt.x = i\*(lung + 8);

txt.backgroundColor = bcolor[i];

txt.background = true;

txt.text = texte[i];

addChild(txt); // Adauga campul in scena

}

- Instructiunea "*for(var i:Number=0; i<texte.length; i++)*" parcurge Array-ul "texte" si pt. fiecare element parcurs creaza un obiect TextField, iar "addChild(txt);" adauga in Scena acest obiect, "txt", cu toate setarile din el (dimensiuni, culoare fundal, text, etc.).  
- Va afisa in prezentarea Flash un rand cu 3 zone de text, precum acesta:

cursuri   tutoriale   www.marplo.net

**2. removeChild**

**removeChild** este o functie ActionScript care sterge elemente vizuale din prezentarea Flash. Sintaxa ei generala este:  
                **removeChild(removeObject);**  
- Unde, "removeObject" este obiectul ce trebuie sters din prezentarea Flash (cum ar fi: un camp de text, o figura geometrica, o imagine, etc.).  
  
Cand aceeasi instanta este adaugata de mai multe ori (precum in exemplu de sus bucla "for()" creaza si adauga aceeasi instanta "txt" de 3 ori), in memorie este retinuta ultima ca fiind acea instanta de obiect. In acest caz, apelul metodei "removeChild(insObj)" va sterge ultimul element cu nume de instanta "insObj", chiar daca au fiost adaugate mai multe, celelalte nu mai exista cu acel nume si da eroare.  
Se poate vedea in urmatorul exemplu (continuare a celui de sus) cum se aplica si efectul obtinut.

**1.** Deschideti un document Flash nou si desenati in Scena o figura geometrica (patrat, cerc) si transformati-o in Buton (de la meniul **Modify -> Convert to Symbol**).  
**2.** Dati butonului din Scena numele de instanta "remove\_bt" (in panoul Properties, la "*<Instance Name>*").  
**3.** Click-dreapta pe Cadru 1 din Timeline, alegeti "Actions", iar in panoul pt. ActionScript adaugati urmatorul cod:

var texte:Array = ['cursuri', 'tutoriale', 'www.marplo.net']; // Array cu texte pt. afisat

var bcolor:Array = [0xe8e9fe, 0xeded01, 0xfedada]; // Array culori pt. fondul fiecarui text

var lung:Number = 100; // Variabila folosita pt. lungimea campului de text

// Parcurge Array-ul "texte"

for(var i:Number=0; i<texte.length; i++)

{

// Pt. fiecare element parcurs creaza o instanta (un camp) "TextField"

// Seteaza inaltimea campului, lungimea (lung) si distanta 'x' in functie de 'i' (pt. afisare in linie)

// Adauga culoare de fundal (background) din Array-ul "bcolor" si textul din elementul parcurs

var txt:TextField = new TextField();

txt.height = 25;

txt.width = lung;

txt.x = i\*(lung + 8);

txt.backgroundColor = bcolor[i];

txt.background = true;

txt.text = texte[i];

addChild(txt); // Adauga campul in scena

}

// Inregistrare eveniment (la butonul din Scena, "remove\_bt") pt. stergere instante din prezentare

remove\_bt.addEventListener(MouseEvent.CLICK, removeIns);

// Functia apelata de eveniment, pt. stergere instanta

function removeIns(evt:MouseEvent):void

{

removeChild(txt); // Sterge elementul cu instanta "txt"

}

  - Acest script adauga in prezentarea Flash trei campuri de text (ca cele din exemplu precedent) deasupra butonului din Scena. Cand se apasa pe buton, acesta activeaza evenimentul "MouseEvent.CLICK" care apeleaza functia "removeIns()".  
  - Comanda "removeChild(txt);" sterge instanta de obiect "txt" din scena.  
**4.** Daca apasati "Ctrl+Enter", rezultatul va fi precum in prezentarea urmatoare.

- Fiind adaugata aceeasi instanta de mai multe ori, ultima o rescrie din memorie pe cea precedenta, astfel, butonul va sterge ultimul camp de text adaugat, dar pe celelalte nu le recunoaste ca fiind instanta "txt" si daca se apasa iar pe buton returneaza eroare.

**3. removeChildAt**

Cand sunt adaugate elemente in scena, inclusiv cu "addChild", acestea sunt retinute in memorie, intr-un Array cu ordinea adaugarii, dupa numarul celor deja adaugate in Scena. De exemplu, daca in scena e doar un buton, acesta are index 0 si urmatoarele sunt indexate ca numaratoare dupa el (1, 2, ...).  
Indexul de adaugare poate fi utilizat pentru stergerea elementelor din prezentare, prin aplicarea lui in metoda:  
                **removeChildAt(index)**  
- Cand un element este sters, Array-ul din memorie cu indexi este actualizat, astfel, elementul care este inregistrat dupa cel sters preia indexul aceluia, urmatoru scade si el cu o unitate, si tot asa. Se intelege mai bine din urmatorul exemplu, care e o noua versiune a celui anterior.  
Lasati butonul "remove\_bt" in scena (din documentul creat pt. exemplu de sus), stergeti codul din panoul pt. ActionScript si adaugati-l pe acesta:

var texte:Array = ['cursuri', 'tutoriale', 'www.marplo.net']; // Array cu texte pt. afisat

var bcolor:Array = [0xe8e9fe, 0xeded01, 0xfedada]; // Array culori pt. fondul fiecarui text

var lung:Number = 100; // Variabila folosita pt. lungimea campului de text

// Parcurge Array-ul "texte"

for(var i:Number=0; i<texte.length; i++)

{

// Pt. fiecare element parcurs creaza o instanta (un camp) "TextField"

// Seteaza inaltimea campului, lungimea (lung) si distanta 'x' in functie de 'i' (pt. afisare in linie)

// Adauga culoare de fundal (background) din Array-ul "bcolor" si textul din elementul parcurs

var txt:TextField = new TextField();

txt.height = 25;

txt.width = lung;

txt.x = i\*(lung + 8);

txt.backgroundColor = bcolor[i];

txt.background = true;

txt.text = texte[i];

addChild(txt); // Adauga campul in scena

}

// Inregistrare eveniment (la butonul din Scena, "remove\_bt") pt. stergere instante din prezentare

remove\_bt.addEventListener(MouseEvent.CLICK, removeIns);

// Functia apelata de eveniment, pt. stergere instanta

function removeIns(evt:MouseEvent):void

{

removeChildAt(2); // Sterge elementul cu index 2

}

- Rezultatul va fi cel din prezentarea urmatoare.  
- Cand e apasat butonul, comanda "removeChildAt(2);" sterge elementul cu index 2. In scena raman doua campuri text (primul si al treilea), dar indexarea din memorie se reactualizeaza, asrfel, campul care avea index 3 va primi indexul 2, fapt demonstrat daca se apasa inca o data butonul de stergere. Conform codului, el sterge mereu elementul cu index 2. Cand in scena nu mai sunt elemente mai multe de 2, care sa ia locul celui sters, apasarea da eroare.

**4. getChildByName**

Pe langa indexul numeric de adaugare a elementelor in Scena, acestea sunt inregistrate si cu un nume unic, care nu se schimba cand un element e sters.  
Fiecarui element adaugat in prezentarea flash i-se atribuie un nume, daca nu e precizat special, va fi adaugat automat de program, cu denumirea "instanceNR" ('NR' este un numar de ordine; "instance12", "instance13", ...).  
Acest nume poate fi definit cu proprietatea **name**, cu sintaxa:  
                **insObj.name**  
- Unde "insObj" este instanta prin care e adaugat obiectul.  
- Aceasta instructiune seteaza, dar si returneaza, numele acelei instante, de exemplu:  
                ***trace(insObj.name);***  
  
Prin atribuirea unui nume se obtine un mod mai eficient de control asupra obiectelor din scena, care pot fi preluate cu functia:  
                **getChildByName("nume")**.  
- "nume" este un sir, cel definit cu "insObj.name".

Iata cum se aplica "insObj.name" si "getChildByName()" in exemplu de mai sus. Se foloseste acelasi document Flash (cu butonul "remove\_bt"), si urmatorul cod ActionScript 3 (*trebuie sters cel care este deja adaugat*).

var texte:Array = ['cursuri', 'tutoriale', 'www.marplo.net']; // Array cu texte pt. afisat

var bcolor:Array = [0xe8e9fe, 0xeded01, 0xfedada]; // Array culori pt. fondul fiecarui text

var lung:Number = 100; // Variabila folosita pt. lungimea campului de text

// Parcurge Array-ul "texte"

for(var i:Number=0; i<texte.length; i++)

{

// Pt. fiecare element parcurs creaza o instanta (un camp) "TextField"

// Seteaza inaltimea campului, lungimea (lung) si distanta 'x' in functie de 'i' (pt. afisare in linie)

// Adauga culoare de fundal (background) din Array-ul "bcolor" si textul din elementul parcurs

var txt:TextField = new TextField();

txt.height = 25;

txt.width = lung;

txt.x = i\*(lung + 8);

txt.backgroundColor = bcolor[i];

txt.background = true;

txt.text = texte[i];

// Atribuie instantei un nume, ca sa fie unic foloseste si valoarea lui 'i'

txt.name = 'camp'+i;

addChild(txt); // Adauga campul in scena

}

// Inregistrare eveniment (la butonul din Scena, "remove\_bt") pt. stergere instante din prezentare

remove\_bt.addEventListener(MouseEvent.CLICK, removeIns);

// Functia apelata de eveniment, pt. stergere instanta

function removeIns(evt:MouseEvent):void

{

removeChild(getChildByName('camp0')); // Sterge elementul cu nume 'camp0'

removeChild(getChildByName('camp1')); // Sterge elementul cu nume 'camp1'

}

- Bucla "for()" din script face ca instructiunea "txt.name = 'camp'+i;" sa dea fiecarei instante "txt" numele de inregistrare: *camp0, camp1, camp2*; in ordinea adaugarii.  
- Cunoscand numele adaugate, se poate folosi metoda "getChildByName()". Observati expresia "removeChild(getChildByName('camp0'));", "removeChild()" sterge elementul preluat de "getChildByName()".  
Rezultatul acestui exempu va fi cel din prezentarea urmatoare.

- Dupa ce sunt sterse campurile de text cu numele de inregistrare "camp0" si "camp1", in scena nu va mai exista alt element cu vreunul din acele nume.  
  
&bull Alte functii utile din aceasta categorie sunt:

* **addChildAt(insObj, nr)** - adauga in scena instanta de obiect "insObj" cu indexul 'nr'
* **getChildIndex(insObj)** - returneaza indexul de adaugare a instantei de obiect "insObj"
* **setChildIndex(insObj, nr)** - schimba indexul (si pozitia) unei instante din scena, cu cel precizat la 'nr'
* **swapChildren(insObj1, insObj2)** - schimba intre ele, in scena, pozitia lui "insObj1" si "insObj2"
* **swapChildrenAt(index1, index2)** - schimba intre ele, in scena, pozitia elementelor cu 'index1' si 'index2'.

**OOP - Clase si Obiecte - Creare Clasa**

Construirea unui program cu **Clase si Obiecte** foloseste o tehnica cunoscuta ca **Programare orientata pe Obiect**, **OOP** (*Object Oriented Programming*).  
O **clasa** este de fapt o structura de date compacta, care contine variabile si functii (numite metode) prin care se pot crea diferite instructiuni legate intre ele in acea structura.  
**Obiectele** sunt elementele din script definite printr-o clasa, acestea efectueaza instructiunile scrise in acea Clasa.  
- De exemplu, pentru a intelege mai bine, prin analogie, o Clasa poate fi asemanata cu planul /schita unei case, iar obiectul este insasi casa, si contine proprietatile (culoare, ferestre, etc.) definite in schita.

*Diferenta intre clasa si obiect este importanta. Daca o clasa poate fi asimilata unui tip de date, un obiect poate fi echivalat cu o variabila sau cu o valoare avand un anumit tip de date. Practic, o clasa este o "fabrica" de obiecte, care produce obiecte cu aceeasi structura, avand proprietati si metode identice.*

• Clasele pot fi utilizate pentru a crea mai multe obiecte care vor contine proprietatile si functiile scrise in ele; fiecare astfel de obiect creat poarta denumirea de **Instanta a clasei**.  
• In OOP (Programarea Orientata pe Obiect) apar termenii "**Encapsulation**" (*Encapsulare*) si "**Polymorphism**" (*polimorfism.*).  
          - Encapsularea reprezinta faptul de a utiliza proprietatile si metodele clasei fara a fi necesara cunoasterea sructurii ei, a codului si instructiunilor din ea.  
          - Polimorfismul e posibilitatea de a utiliza o clasa sau alta pentru a realiza lucrul dorit, avand functii similare. Mai multe obiecte pot efectua operatii similare, in moduri diferite. (*De exemplu, se poate bea apa cu paharul, cu o cana sau cu sticla. Acesta e polimorfism*).

**1. Definire clasa**

ActionScript 3 contine mai multe clase predefinite, cum ar fi "clasa Date" pentru lucru cu data si timp, "clasa String" pentru siruri, si altele; dar pot fi create si clase proprii/ personalizate.  
Definirea unei clase incepe cu un cuvantul cheie "**class**", urmat de numele clasei si corpul ei (cuprins intre acolade). In corpul clasei se definesc proprietatile si metodele ei.  
          - Proprietatile sunt variabile definite in interiorul clasei.  
          - Metodele sunt functii create in interiorul clasei.  
Structura general de alcatuire a unei clase este urmatoarea:

* **class nume\_clasa {  
      var proprietate1;  
      var proprietate2;  
      .................  
      function metoda1() {  
          // Codul functiei  
      }  
      function metoda2() {  
          // Codul functiei  
      }  
  .................  
  }**

Pentru a construi o clasa functionala, care sa poata fi folosita in documentul Flash, sunt necesare si alte lucruri care trebuie adaugate.  
- In primul rand, fiecare clasa trebuie creata in interiorul unui **Package** (*pachet*). **Package** e o structura prin care se pot grupa mai multe clase impreuna. Ajuta la crearea de clase diferite care pot avea acelasi nume.  
- La definirea unei clasei si a elementelor din ea (proprietati, metode) se pot aplica unele atribute speciale prin care se defineste domeniul lor de lucru (de existenta). Acest atribut se adauga inaintea cuvantului de definire (class, var, const si function), si poate fi unul din cuvintele:

* **public** - valabil in tot scriptul
* **private** - valabil doar in interiorul aceleasi clase
* **protected** - valabil doar in acea clasa si sub-clasele ei
* **internal** - valabil in codurile din acelasi package

- Daca nu e adaugat nici un atribut, se considera "private".  
Astfel, formula completa pentru crearea unei clase este:

* **package nume\_package {  
      atribut class nume\_clasa {  
          atribut var proprietate1;  
          atribut var proprietate2;  
          .................  
          atribut function metoda1() {  
              // Codul functiei  
          }  
          atribut function metoda2() {  
              // Codul functiei  
          }  
      .................  
      }  
  }**

- "nume\_package" e optional, pentru incepatori e mai bine sa nu fie adaugat deoarece necesita lucruri suplimentare (explicate mai jos)  
  
• Codul pt. definirea clasei NU se scrie in documentul FLA, ci se creaza intr-un fisier extern, cu extensia "**as**". Din meniul **New** se alege "**ActionScript 3.0 Class**" sau "**ActionScript File**". Se va deschide o mica fereastra in care trebuie scris numele clasei, dupa care se deschide un document pentru scrierea codului clasei. Initial are in structura de inceput declarate corpul "package" si clasa constructor (explicata mai jos), acestea pot fi sterse sau completate cu propriul cod.  
Iata un exemplu de clasa simpla, ce contine o proprietate (denumita "prop"), o metoda (cu numele "metod"), ambele cu atribut "public" ca sa poata fi apelate in afara corpului clasei (adica in orice script) si o constanta numita ASCUNS, cu atribut "protected", ceea ce o face sa poata fi utilizata doar in interiorul clasei si sub-clase ale ei.

**Exemplu TestClas**

// Se creaza package simplu (fara nume)

package {

// Se defineste o clasa, cu atribut public, o proprietate "prop" si o metoda "metod"

public class TestClas

{

public var prop:Number = 7; // Proprietatea publica

protected const ASCUNS:Number = 2; // Constanta "protected"

// Creare metoda, preia un argument de tip Number

public function metod(val:Number):void

{

// Da alta valoare propretatii "prop", un nr. dat de parametru "val" si constanta ASCUNS

this.prop = val + ASCUNS;

}

}

}

- Formula "this.prop" indica proprietatea "prop" din acea clasa (termenul **this** se adauga pentru a indica precis ca se face referire la proprietatea acelei clase, evitand greseli ce pot apare daca in functie sunt folosite si alte variabile sau parametri cu acelasi nume).  
- Documentul cu acesta clasa trebuie salvat (din meniul **File -> Save**) **cu acelasi nume ca al clasei** (adica "TestClas.as"), in directorul in care e salvat si documentul FLA ce foloseste acea clasa (daca "package" e fara vreun nume).  
  
• ***Atentie, nu se creaza mai multe clase in acelasi corp "package". Acesta reprezinta de fapt un director in care sunt adaugate fisierele "\*.as" cu clasele ce apartin aceluiasi grup. In fiecare corp "package" (intre acoladele lui) se poate crea o singura clasa***.  
          - Intrebarea ar fi, cum poate astfel un "package" sa contina mai multe clase? **Package** reprezinta de fapt calea si directorul unde sunt salvate fisiere "\*.as" cu clasele incluse in acel pachet, fiecare fisier contine o singura clasa, iar toate formeaza grupul din pachet. Directorul trebuie sa aiba acelasi nume ca cel dat in cod pentru package; astfel, precum acele fisiere sunt grupate (stocate) in acelasi director, asa clasele definite in ele sunt impachetate in acelasi "package".  
- De exemplu, daca se doreste gruparea a 2 clase (Clasa1 si Clasa2) intr-un pachet denumit "grup", se creaza un director "grup" (in acelasi loc unde e salvat documentul FLA) si in el se salveaza fisierele "\*.as" cu cele 2 clase ("Clasa1.as" in care e definit codul pt Clasa1, si "Clasa2.as" in care e codul pt. Clasa2). Aceste clase sunt grupate in pachetul "grup" si pot fi apelate /folosite in script cu expresia "grup.Clasa1", respectiv "grup.Clasa2".  
• Pentru a folosi o clasa sau un grup de clase dintr-un package cu nume (care sunt salvate intr-un director cu acelasi nume), acestea trebuie importate in document, cu formula:  
                **import nume\_package.nume\_clasa;**  
          sau  
                **import nume\_package.\***  
- *Prima formula importa doar clasa "nume\_clasa" din "nume\_package", a doua formula (cu \*) importa toate clasele din acel pachet (director).*  
• Clasele definite in corpul unui package simplu (fara nume), si care sunt salvate in acelasi director cu documentul Flash, nu e necesar sa fie importate.

**2. Utilizare clasa**

Dupa ce o clasa e creata si salvata in fisierul "nume\_clasa.as", pentru a putea fi utilizata in prezentarea Flash, trebuie creata o instanta de obiect a clasei. Aceasta instanta se declara in scriptul din documentul FLA ca orice obiect folosit in ActionScript 3 (precum Array, Date, etc., care sunt tot clase), cu formula:  
                **var nume\_ins:NumeClasa = new NumeClasa();**  
          sau  
                **var nume\_ins:nume\_package.NumeClasa = new nume\_package.NumeClasa();**       (*daca e intr-un package cu nume*)  
- "nume\_ins" este numele instantei prin care pot fi utilizate proprietatile si metodele acelei clase.

Iata de exemplu cum poate fi utilizata clasa "TestClas" prezentata mai sus.  
Copiati codul de la "Exemplu TestClas" intr-un document "ActionScript 3.0 Class" si salvati-l cu numele "TestClas.as" in acelasi director in care e si documentul FLA. In documentul Flash adaugati urmatorul cod AS3:

// Creaza instanta de obiect la clasa TestClas

var tst:TestClas = new TestClas();

// Verifica valoarea proprietatii "prop"

trace(tst.prop); // 7

// Apeleaza metoda "metod()", care, comform definirii ei in clasa TestClas

// primeste un argument si modifica valoarea lui "prop"

tst.metod(18);

// Verificare "prop"

trace(tst.prop); // 11.747155797667801

- Deoarece documentul FLA si fisierul cu clasa TestClas sunt in acelasi director, Flash importa automat aceasta clasa cand este creata instanta de obiect la ea, in care Flash retine proprietatile si metodele clasei. Prin apelarea acestora se executa instructiunile asociate lor din clasa, dupa cum demonstreaza rezultatul returnat de "trace()".  
- Constanta "ASCUNS" avand atributul "protected", poate fi utilizata doar in codul din corpul clasei (sau sub-clase ale ei), daca se incearca apelarea ei in afara clasei (de ex. in script, *trace(tst.ASCUNS)*), va rezulta eroare.

**3. Metoda constructor**

**Metoda constructor** e un tip special de functie din interiorul Clasei. Trebuie sa aibe acelasi nume cu cel al clasei. Aceasta metoda este mereu "public" chiar daca nu are mentionat special atributul "public", este apelata automat cand e creata o instanta la clasa respectiva.  
O alta difrenta fata de celelalte functii este aceea ca in codul metodei Constructor nu se foloseste instructiunea "return".

Iata o noua versiune a clasei "TestClas", care include si metoda constructor.

// Se creaza package simplu (fara nume)

package {

// Se defineste o clasa, cu atribut public, o proprietate "prop" si o metoda "metod"

public class TestClas

{

public var prop:Number = 7; // Proprietatea publica

protected const ASCUNS:Number = 2; // Constanta "protected"

// Creare metoda constructor

public function TestClas(nr1:Number, nr2:Number)

{

// Retine intr-o variabila media aritmetica a parametrilor 'nr1' si 'nr2'

var medie\_a = (nr1+nr2)/2;

trace(medie\_a); // Returneaza in output valoarea lui "medie\_a"

}

// Creare metoda, preia un argument de tip Number

public function metod(val:Number):void

{

// Da alta valoare propretatii "prop", un nr. dat de parametru "val" si constanta ASCUNS

this.prop = val + ASCUNS;

}

}

}

- Observati ca metoda constructor are acelasi nume ca al clasei. Deoarece preia 2 parametri ('nr1' si 'nr2'), cand se creaza o instanta la aceasta clasa trebuie sa fie adaugate si doua numere ca argumente.  
Daca salvati aceasta clasa si scrieti in Panoul pt. ActionScript din documentul Flash doar urmatoarea linie de cod:

var tst:TestClas = new TestClas(7, 8);

- Va afisa in Output numarul 7.5, fapt ce demonsreaza ca Metoda Constructor este apelata automat si se executa codul din ea cand e creata instanta de obiect.

**4. Metoda Accessor**

Variabilele (sau proprietatile) create intr-o clasa pot avea de la inceput o valoare sau pot fi doar simplu declarate, urmand ca valoarea lor sa fie atribuita prin intermediul unei functii. Aceasta functie e denumita generic **Metoda Accessor**, e la fel ca oricare alta metoda, doar ca scopul ei este de a atribui valori proprietatilor; in rest, se construieste si se apeleaza la fel ca celelalte.  
Ca sa vedeti practic cum functioneaza "metoda accessor", incercati urmatorul exemplu, in care este creata o clasa "testClas2" cu doua proprietati definite simplu, fara valoare, si o functie accessor care da valoare acestor proprietati prin argumentele ei.  
- Deschideti un document nou "ActionScript 3.0 Class", dati ca nume de clasa "testClas2". Stergeti codul initial si copiati urmatorul cod, apoi salvati fisierul cu aceasta clasa, cu numele "testClas2.as", in acelasi director unde este si documentul FLA.

package {

// Construire clasa

public class testClas2 {

// Definire proprietati fara valoare

public var prop1:Number;

public var prop2:String;

// Creare metoda accessor

public function setProp(prop1:Number, prop2:String)

{

// Se atribuie valori proprietatilor (cele cu "this")

this.prop1 = prop1;

this.prop2 = prop2;

}

}

}

- Parametri functiei (aici "setClas") pot avea orice nume, dar pt. a usura intelegerea codului, carui proprietati este destinat fiecare, se foloseste de obicei aceeasi denumire ca a proprietatii.  
- Prin istructiunea "**this**" se specifica exact care este proprietatea, face diferenta intre proprietate si parametru cu acelasi nume.  
  
In Panoul pt. ActionScript din documentul Flash (salvat in acelasi director cu "testClas2") adaugati urmatorul cod:

// Creare instanta obiect la clasa testClas2

var obj4:testClas2 = new testClas2();

// Verificare in Output 'prop1' si 'prop2'

trace(obj4.prop1+"-"+obj4.prop2); // NaN-null

// Apelare metoda accessor

obj4.setProp(8, 'Tutoriale');

// Verificare din nou 'prop1' si 'prop2'

trace(obj4.prop1+"-"+obj4.prop2); // 8-Tutoriale

- "NaN" e returnat de o variabila de tip Number fara valoare, "null" de o variabila de tip String fara valoare.  
- Dupa apelarea metodei "*obj4.setProp(8, 'Tutoriale');*", "prop1" si "prop2" primesc valoarea argumentelor, 8 si 'Tutoriale'.

*Aceasta metoda este utila cand se doreste ca valoarea proprietatilor sa fie atribuita dinamic, in script, in functie de obiectul creat si instructiunile pt. el.*

**OOP - Clase si Obiecte - Folosire Mostenire**

**Mostenirea** este unul din cele mai utile instrumente ale Programarii Orientate pe Obiect - OOP.  
Prin **mostenire** se intelege transmiterea proprietatilor, constanttelor si a functiilor de la o clasa la alta, intr-o structura ierarhica. Prima clasa este clasa de baza, denumita "**parinte**", legata de aceasta se poate crea o sub-clasa, denumita "**copil**"; sub-clasa 'copil' mosteneste proprietatile si metodele clasei 'parinte', le poate folosi in instructiunile din propriul cod si le transmite cand se creaza o instanta de obiect la ea. *Precum in natura copii mostenesc genele parintilor.*  
Iata cum se aplica acest procedeu in programarea ActionScript 3. Vom folosi urmatoarea clasa de baza, "Mover", prin care se pot misca obiecte in prezentarea Flash. Obiectul care trebuie miscat si viteza de miscare pe axa X/Y sunt transferati ca argumente la crearea instantei clasei (*explicatii gasiti in cod*).

// Se creaza package simplu (fara nume)

package {

// Importare clase predefinite ActionSscript 3 folosite in aceasta

import flash.events.Event;

import flash.display.MovieClip;

// Start definire clasa Mover

public class Mover

{

// Setare proprietati

// 'insMC' - pt. instanta MovieCpli, 'xVel' - pt. viteza pe orizontala, 'yVel' - pt. viteza pe verticala

public var insMC:MovieClip;

public var xVel:Number;

public var yVel:Number;

// Creare metoda constructor, preia ca argumente o instanta si 2 valoari numerice

function Mover(inMC:MovieClip, xV:Number, yV:Number)

{

// Atribuie proprietatilor clasei valorie din parametri

this.insMC = inMC;

this.xVel = xV;

this.yVel = yV;

}

// Creare metoda cu atribut "protected" (poate fi utilizata doar in aceasta clasa si sub-clasele ei)

protected function updatePosition(evtObj:Event):void

{

// Incrementeaza distanta 'x' si 'y' a lui "insMC" cu valoarea proprietatilor "xVel" si "yVel"

this.insMC.x += this.xVel;

this.insMC.y += this.yVel;

}

// Definire metoda publica pt. start miscare

public function startMove():void

{

// Aplica detectare Eveniment ENTER\_FRAME la 'insMC'

// si apeleaza metoda protejata, updatePosition

this.insMC.addEventListener(Event.ENTER\_FRAME, this.updatePosition);

}

// Definire metoda publica pt. stop miscare

public function stopMove():void

{

// Sterge detectarea evenimentului setat in "startMove()"

insMC.removeEventListener(Event.ENTER\_FRAME, this.updatePosition);

}

}

}

- Ca sa testati aceasta clasa, efectuati urmatorii pasi:

**1.** Deschideti de la **File -> New** un document nou "**ActionScript 3.0 Class**". In fereastra pentru numele clasei, ce apare la deschidere, scrieti Mover. Stergeti codul care apare initial in document si copiati-l pe cel de sus, apoi salvati-l cu numele "***Mover.as***".  
**2.** Deschideti si un document Flash nou, desenati in Scena o figura geometrica sau orice desen doriti si transformati-l in Movie Clip (de la **Modify - Convert to Symbol**, asigurati-va ca punctul bifat de la **Registration** e cel din mijloc, acesta seteaza locatia centrului de coordonate a lui). Dati instantei din Scena (desenului transformat in MovieClip) un nume, aici e folosit "***sfer***" (in panoul Properties, in casuta de sus, unde e scris "<Insance Name>")  
**3.** Dati click-dreapta pe Cadru 1 din Timeline, alegeti **Actions** si in panoul pt. script ActionScript adaugati urmatorul cod:

// Creare instanta la clasa Mover

var obj:Mover = new Mover(sfer, 2, 3);

obj.startMove(); // Apelare metoda "startMove()"

// Apeleaza metoda "stopMove" dupa 3 secunde (3000 milisecunde)

// "stopMove" sterge inregistrarea evenimentului ENTER\_FRAME, fapt ce opreste miscarea

setTimeout(obj.stopMove, 3000);

  - "setTimeout()" apeleaza o functie /metoda dupa un anumit timp (dat in milisecunde).  
**4.** Salvati documentul in acelasi director unde e salvat si "***Mover.as***", apoi apasati Ctr+Enter. Va apare ceva similar cu prezentarea urmatoare (*click pe imagine*).  


- Aceasta clasa misca obiectul in aceeasi directie pana iese din Scena (sau pana se da o comanda de oprire cu "stopMove()"), dar poate dorim ca unele obiecte sa se miste continuu in cadrul scenei, ricosand din marginile ei. Pentru aceasta, in loc de a crea de la inceput o alta clasa sau de a modifica /strica pe aceasta, definim o sub-clasa copil legata de ea, care va mosteni proprietatile si metodele de miscare. In clasa copil trebuie doar inclusa (importata) clasa parinte si scrise instructiunile noi pt. schimbarea directiei de miscare cand obiecul ajunge in margini.  
  
• Pentru a crea o sub-clasa 'copil' legata de una 'parinte', se foloseste cuvantul **extends** cu urmatoarea sintaxa:

* **package {  
      import clasaParinte;  
      atribut class nume\_clasaCopil extends nume\_clasaParinte {  
          // Codul din clasaCopil  
      }  
  }**

- Cand intr-o clasa trebuie apelate proprietati si /sau metode din alta clasa, aceasta trebuie inclusa (importata) cu instructiunea **import**.   
  
Iata cum se poate crea o sub-clasa copil (denumita aici "MoverChild") legata de cea definita mai sus, "Mover".  
Deschideti un alt document "ActionScript 3.0 Class", dati-i numele de clasa **MoverChild**, stergeti codul initial din el si copiati-l pe urmatorul:

// Se creaza package simplu (fara nume)

package {

// Importare clase a caror proprietati si metode vor fi apelate in aceasta clasa

import flash.display.MovieClip;

import flash.events.Event;

import Mover; // Importare clasa de baza (parinte)

// Incepere definire sub-clasa (copil) a clasei Mover

public class MoverChild extends Mover

{

// Metoda constructor a clasei copil (primeste 3 argumente, un obiect Movie Clip si 2 numere)

public function MoverChild(inMC:MovieClip, xV:Number, yV:Number)

{

// Apeleaza Metoda Constrctor a clasei parinte, cu argumentele primite la parametri

super(inMC,xV,yV);

}

// Functie pentru schimbarea directiei de miscare cand obiectul ajunge la margine

// Cu 4 if-uri pt. verificarea pozitiei obiectului pt. fiecare latura

private function bounceAtBorder():void

{

// Verifica daca obiectul a ajuns in marginea din dreapta

// Scazand din lungimea Scenei (this.insMC.stage.stageWidth) jumatate din lungimea obiectului (this.insMC.width/2)

// (jumatate deoarece se considera obiecul MovieClip respectiv creat cu centrul de inregistrare in mijloc)

if(this.insMC.x > this.insMC.stage.stageWidth-(this.insMC.width/2))

{

// Seteaza distanta 'x' cu valoarea data de lungimea Scenei minus jumate din lungimea obiectului

// Face negativa valoarea proprietatii pt. viteza de miscare orizontala (pt. a schimba directia)

this.insMC.x = this.insMC.stage.stageWidth-(this.insMC.width/2);

xVel \*= -1;

}

// Verifica daca obiectul a ajuns in marginea de jos

// Scazand din inaltimea Scenei (this.insMC.stage.stageHeight) jumatate din inaltimea obiectului (this.insMC.height/2)

if (this.insMC.y > this.insMC.stage.stageHeight-(this.insMC.height/2))

{

// Seteaza distanta 'y' cu valoarea data de inaltimea Scenei minus jumate din inaltimea obiectului

// Face negativa valoarea proprietatii pt. viteza de miscare verticala (pt. a schimba directia)

this.insMC.y = this.insMC.stage.stageHeight-(this.insMC.height/2);

yVel \*= -1;

}

// Verifica daca obiectul a ajuns in marginea din stanga

//(distanta 'x' mai mica decat jumatate din lungimea oiectului)

if (this.insMC.x < this.insMC.width/2)

{

// Seteaza distanta 'x' cu valoarea data de jumatatea lungimii obiectului

// Face negativa valoarea proprietatii pt. viteza de miscare orizontala

this.insMC.x = this.insMC.width/2;

xVel \*= -1;

}

// Verifica daca obiectul a ajuns in marginea de sus

//(distanta 'y' mai mica decat jumatate din inaltimea oiectului)

if (this.insMC.y < this.insMC.height/2)

{

// Seteaza distanta 'y' cu valoarea data de jumatatea inaltimii obiectului

// Face negativa valoarea proprietatii pt. viteza de miscare verticala

this.insMC.y = this.insMC.height/2;

yVel \*= -1;

}

}

// Rescrie metoda "updatePosition()", definita in clasa parinte, Mover

override protected function updatePosition(evtObj:Event):void

{

// Apeleaza /Include codul metodei "updatePosition()" din clasa de baza

super.updatePosition(evtObj);

bounceAtBorder(); // Adauga apelarea metodei "bounceAtBorder()" definita mai sus

}

}

}

- *Despre rolul instructiunilor folosite gasiti explicatii in cod.*  
- Salvati aceasta clasa cu numele "**MoverChild.as**" in acelasi director unde e si "Mover.as".  
- In documentul FLA (din acelasi director) in care e creata instanta Movie Clip "sfer", dati click-dreapta pe primul Frame din Timeline si alegeti "Actions", stergeti daca e ceva adaugat in panoul pt. cod ActionScript si adaugati-l pe urmatoru:

// Creare instanta la clasa MoverChild

var obj2:MoverChild = new MoverChild(sfer, 2, 3);

obj2.startMove(); // Apelare metoda "startMove()"

- Observati ca, desi nu exista metoda "startMove()" in clasa "MoverChild", aceasta este apelata prin instanta acestei clase, si functioneaza deoarece e mostenita de la clasa parinte ca si cum ar fi scrisa si in ea.  
- Daca apasati "Ctrl+Enter", rezultatul va fi miscarea obiectului precum se vede in prezentarea urmatoare.

• Pe langa posibiliatea utilizarii proprietatilor si metodelor din clasa parinte, acestea pot fi si modificate (rescrise) in clasa copil, dupa cum puteti vedea in codul clasei "MoverChild" ca sunt modificate valorile proprietatilor 'xVel', 'yVel' si rescrisa metoda 'updatePosition()'; toate definite initial in clasa parinte "Mover". Aici s-a adaugat inca o instructiune la "updatePosition()", dupa ce a inclus cu "*super.updatePosition(evtObj)*" si codul ei original.  
Rescrierea unei metode se face folosind cuvantul **override** la definirea acelei metode in clasa copil. Sintaxa generala este:  
                **override atribut function numeMetoda() {  
                    // Codul care-l va inlocui pe cel original, din clasa parinte  
                }**  
  - Rescrierea unei metode nu afecteaza cu nimic pe cea originala, modificarile efectuate sunt valabile doar in clasa copil in care se face rescrierea, respectiv, in alte sub-clase ale ei.  
  - Pentru a include si codul initial (din clasa parinte) in functia rescrisa, se foloseste formula **super.numeMetoda();** in corpul functie care face rescrierea (*tehnica folosita si in exemplu de sus*).  
Important:  
          **super()** apeleaza metoda constructor a clasei parinte, aceasta trebuie sa contina un numar de argumente egal cu numarul de parametri ai metodei constructor din clasa parinte. Daca aceasta nu e adaugata in corpul clasei copil, va fi automat inclusa de program, dar fara argumente. Prin urmare, daca metoda constructor a clasei de baza contine parametri, trebuie adaugat "**super(parametri)**" in sub-clasa copil, altfel, ea oricum va fi inclusa, dar fiind fara argumente, va genera eroare.  
- "super.numeMetoda()" apeleaza /include "numeMetoda()" din clasa parinte.

**Creare sub-clasa din clasa copil**

Dupa formula prezentata mai sus se pot defini mai multe clase copil ca extinderi a clasei de baza, dar se pot crea si subclase ale clasei copil, astfel, clasa copil devine "parinte" a sub-claselor extinse din ea, acestea fiind ca niste nepoti a clasei principale, de baza.  
• Clasele "nepot" nu pot mosteni direct proprietatile si metodele clasei "bunic", ci prin intermediul clasei din care sunt extinse.

Pentru a arata practic cum se construieste si functioneaza o sub-clasa extinsa din alta sub-clasa, vom continua cu exemplul prezentat pana acum in aceasta lectie.  
Clasa de baza, "Mover", face miscarea si oprirea obiectelor, cea extinsa din ea, "MoverChild", schimba directia de miscare cand obiectul ajunge in margini. Acum, vom defini o clasa copil din "MoverChild", denumita "MoverChildG". Cu aceasta se intentioneaza adaugarea unui efect de "gravitatie" care sa traga in jos obiectul si sa-l incetineasca pana la oprire.  
- Se creaza clasa "MoverChildG" din "MoverChild" si nu direct din cea de baza deoarece astfel mosteneste capacitatea ei de a nu lasa obiectul sa iasa din margini.  
Deschideti un nou document "ActionScript 3.0 Class", dati pt. numele clasei "**MoverChildG**", stergeti codul initial din el si copiati-l pe urmatorul:

// Se creaza package simplu (fara nume)

package {

// Importare clase a caror proprietati si metode vor fi apelate in aceasta clasa

import flash.display.MovieClip;

import flash.events.Event;

import MoverChild; // Importare clasa din care va fi extinsa

// Incepere definire sub-clasa MoverChildG (copil) a clasei MoverChild

public class MoverChildG extends MoverChild

{

// Se definesc proprietatile proprii acestei clase

// cu atribut "private" deoarece se folosesc doar in codul ei

private var streng: Number;

private var lastPosX: Number;

private var lastPosY: Number;

// Metoda constructor a clasei

// Pe langa atributele necesare clasei din care este extinsa, mai adauga un parametru "streng" (pt. puterea de miscare)

// ('stren' avand atribuita si o valoare, face optionala adauarea lui la crearea instantei in script)

public function MoverChildG(inMC:MovieClip, xV:Number, yV:Number, streng:Number=1)

{

// Apeleaza Metoda Constrctor a clasei parinte (la care este extinsa)

super(inMC, xV, yV);

// Atribuie proprietatii 'streng' valoarea parametrului "streng"

// S-a dat parametrului acelasi nume cu al proprietatii pt. a se sti mai bine rolul lui (dar poate avea si alt nume)

this.streng = streng;

}

// Rescrie metoda "updatePosition()", definita (tot ca rescriere) in clasa parinte, MoverChild

override protected function updatePosition(evtObj:Event):void

{

// Include /Pastreaza codul initial al acestei metode si adauga alte 3 instructiuni (apelari a unor functii interne)

super.updatePosition(evtObj);

this.applyGravity();

this.applyFriction();

this.checkForStop();

}

/\* Se definesc functiile interne, private, deoarece sunt necesare doar in interiorul acestui cod \*/

// Mareste viteza pe axa Y cu valoare proprietatii 'streng'

private function applyGravity():void

{

this.yVel += this.streng;

}

// Adauga coeficientul de gravitate G (0.98) la viteza de pe axele X si Y

// care, prin inmultire (fiind mai mica decat 1) are ca rezultat micsorarea valori lor

private function applyFriction():void

{

this.xVel \*= 0.98;

this.yVel \*= 0.98;

}

// Functia pt. verificarea momentului cand miscarea s-a oprit

// In acel moment apeleaza metoda "stopMove" din clasa de baza (pt. eficientizarea codului)

// ("stopMove" sterge inregistrarea evenimentului de detectare ENTER\_FRAME,

// ne fiind necesar dupa oprire, elibereaza memoria ocupata de acest eveniment)

private function checkForStop():void

{

// Daca pozitia X si Y a obiectului este aceeasi cu ultima inregistrata

if(this.insMC.x == this.lastPosX && this.insMC.y == this.lastPosY)

{

this.stopMove(); // Apeleaza metoda "stopMove"

}

// Retine in proprietatile 'lastPosX' si 'lastPosY' ultima pozitie a obiectului

this.lastPosX = this.insMC.x;

this.lastPosY = this.insMC.y;

}

}

}

- *Explicatii despre instructiunile folosite gasiti in documentatia din cod.*  
- Salvati aceasta clasa cu numele "**MoverChildG.as**" in acelasi director unde e "Mover.as" si "MoverChild.as".  
- In documentul FLA (din acelasi director) in care e creata instanta Movie Clip "sfer", dati click-dreapta pe primul Frame din Timeline si alegeti "Actions", stergeti daca e ceva adaugat in panoul pt. cod ActionScript si adaugati-l pe urmatoru:

// Creare instanta la clasa MoverChildG

var obj3:MoverChildG = new MoverChildG(sfer, 15, 30, 2);

obj3.startMove(); // Apelare metoda "startMove()"

- Daca apasati "Ctrl+Enter", rezultatul va fi miscarea obiectului precum se vede in prezentarea urmatoare (*click pe imagine*).  
  
  
• Din toate aceste exemple se vede cat de eficienta e folosirea mostenirii. Pot fi create /adaugate lucruri noi la ceea ce face o clasa, fara a trebui rescris tot codul sau sa fie aceasta modificata. Si oricand poate fi utilizata oricare din clase (parinte, copil, nepot).

*Exista posibilitatea de a bloca extinderea unei clase, sa nu mai transfere proprietatile si metodele ei vreunei sub-clase legata de ea. Pentru aceasta se adauga termenul****final****inaintea cuvantului****class*** *De exemplu:****package {  
                public final class numeClasa {  
                    // Instructiuni ...  
                }  
            }***

**Clase si Obiecte - Elemente statice**

Pe langa atributele: "public", "private", "protected" si "internal"; prin care se defineste domeniul de acces al principalelor elemente dintr-o clasa: variabile (proprietati), constante si metode (functii); acestea pot fi create si ca **statice**, astfel, vor apartine exclusiv clasei, si nu pot fi apelate prin instanta de obiect a clasei, ci direct prin ea.  
In continuare sunt prezentate pe rand modul cum se definesc proprietati, constante si metode ca fiind statice.  
• Ca regula generala, se foloseste cuvantul **static** cand sunt create aceste elemente.

**1. Variabile statice**

In mod normal, variabilele definite intr-o clasa sunt asociate cu proprietati ce pot fi apelate prin instanta de obiect la acea clasa (daca au atributul "public"), dupa formula:  
                **instantaObiect.proprietate**  
Variabilele, sau **proprietatile statice** se creaza adaugand cuvantul **static** inaintea lui **var**, sintaxa fiind:  
                **atribut static var prop\_statica:Tip = valoare;**  
Aceste proprietati nu pot fi apelate prin instanta de obiect deoarece nu apartin ei, ele apartin clasei si pot fi apelate cu formula:  
                **numeClasa.prop\_statica**

Iata un exemplu din care se intelege mai bine. In clasa urmatoare, denumita "elStatic" se definesc doua proprietati publice, a doua ca statica. - *Daca nu stiti cum se creaza si se foloseste o clasa in Flash, studiati mai intai lectia*[*OOP - Creare clase*](http://www.marplo.net/flash/oop-clase-obiecte_as)

package {

// Definire clasa "elStatic"

public class elStatic

{

// Definire proprietati

public var prop1:String = 'www.marplo.net';

public static var prop2:String = 'ActionScript'; // Variabila statica

}

}

- In documentul Flash (salvat in acelasi director in care e si fisierul cu aceasta clasa, "elStatic.as") se adauga urmatorul cod:

// Creare instanta de obiect la clasa "elStatic"

var obj:elStatic = new elStatic();

// Se verifica apelarea proprietatilor

trace(obj.prop1); // www.marplo.net

// trace(obj.prop2); // Va cauza eroare

// Apelare 'prop2' direct din "elStatic"

trace(elStatic.prop2); // ActionScript

- Apasati "Ctrl+Enter" ca sa vedeti rezultatul returnat in Output.  
- Daca expresia "*trace(obj.prop2);*" nu ar fi scrisa ca documentatie (cu // inainte) ar genera eroare, "prop2" avand declaratia "static" nu e recunoscuta in instanta la clasa ("obj"), dar e recunoscuta ca apartinand clasei si poate fi apelata direct prin ea insasi (*elStatic.prop2*).  
In interiorul codului clasei, variabila statica (aici "prop2") poate fi apelata normal, cu numele ei, sau cu formula "numeClasa.prop\_statica" (*dupa cum puteti vedea in exemplul de la capitolul 3 "Metode statice"*).

**2. Constante statice**

**Constantele** sunt variabile cu valoare fixa, care nu mai poate fi modificata, si se definesc cu instructiunea **const**.  
La fel ca variabilele, **constantele statice** se creaza adaugand cuvantul **static** inaintea lui **const**.  
- In script pot fi apelate doar cu expresia:  
                **numeClasa.CONSTANTA**  
- In interiorul clasei, constantele statice pot fi apelate direct cu numele lor sau cu formula de mai sus.

In urmatorul exemplu (continuare a celui de sus) sunt declarate 2 constante, ambele cu atribut "public", dar a doua e definita ca statica.

package {

// Definire clasa "elStatic"

public class elStatic

{

// Definire proprietati

public var prop1:String = 'www.marplo.net';

public static var prop2:String = 'ActionScript'; // Variabila statica

// Definire constante

public const TUTORIALE:String = 'web development';

public static const SITE:String = 'marplo'; // Constanta statica

}

}

- In documentul Flash se adauga urmatorul cod:

// Creare instanta de obiect la clasa "elStatic"

var obj2:elStatic = new elStatic();

// Se verifica apelarea constantelor

trace(obj2.TUTORIALE); // web development

// trace(obj2.SITE); // Genereaza eroare

// Apelare 'SITE' direct din "elStatic"

trace(elStatic.SITE); // marplo

- Daca expresia "*trace(obj2.SITE);*" nu ar fi scrisa ca documentatie (cu // inainte) ar genera eroare, constanta "SITE" avand declaratia "static" nu e recunoscuta in instanta la clasa ("obj2"), dar e recunoscuta ca apartinand clasei si poate fi apelata direct prin ea (*elStatic.SITE*).

**3. Metode statice**

Similar cu proprietatile si constanele, metodele de asemenea pot fi definite ca statice, prin adaugarea cuvantului **static** inaintea instructiunii **function**.  
Apelarea lor se face cu formula:  
                **numeClasa.metodaStatica(parametri)**  
- In codul metodelor statice se pot folosi doar proprietati si constante din clasa care si ele sunt statice (nici cuvanul **this** nu poate fi utilizat).

In exemplu urmator (continuare a celui precedent) se adauga o metoda statica (denumita "Cursuri"). Aceasta returneaza o variabila ('re') a carei valoare este data printr-o instructiune "switch" in functie de parametrul primit (*vedeti si explicatiile din cod*).

package {

// Definire clasa "elStatic"

public class elStatic

{

// Definire proprietati

public var prop1:String = 'www.marplo.net';

public static var prop2:String = 'ActionScript'; // Variabila statica

// Definire constante

public const TUTORIALE:String = 'web';

public static const SITE:String = 'marplo'; // Constanta statica

// Definire metoda

public static function Cursuri(param:String):String

{

var re:String; // Variabila care va fi returnata

// Instructi "switch" in functie de valoarea constantelor

switch(param)

{

case 'web':

re = elStatic.prop2; // "re" preia valoarea proprietatii statice "prop2"

break;

case 'marplo':

re = SITE; // "re" preia valoarea constantei statice "SITE" (se poate si elStatic.SITE )

break;

default: re = 'Fara valoare';

}

return re; // Returneaza valoarea lui re

}

}

}

- metoda "Cursuri" fiind definita statica, poate folosi doar variabile si constante care si ele sunt statice. Daca, de exemplu, adaugati in corpul ei "*re = this.prop1;*" sau "*re = TUTORIALE;*" va returna eroare.  
- In documentul Flash se adauga urmatorul cod:

var obj3:elStatic = new elStatic();

trace(elStatic.Cursuri(obj3.TUTORIALE)); // ActionScript

// trace(obj3.Cursuri(elStatic.SITE)); // Cauzeaza eroare

// Apelare metoda statica avand ca argument constanta statica

trace(elStatic.Cursuri(elStatic.SITE)); // marplo

- Daca expresia "*trace(obj3.Cursuri(elStatic.SITE));*" nu ar fi scrisa ca documentatie (cu // inainte) ar genera eroare, metoda "Cursuri" avand declaratia "static" nu e recunoscuta in instanta la clasa ("obj3"), dar e recunoscuta ca apartinand clasei si poate fi apelata direct prin ea (*elStatic.Cursuri('arg')*).

**Clase si Obiecte - Interface**

**Interface** (*Interfata*) este o clasa folosita ca tipar, sau template pentru clase cu functii similare, care trebuie sa respecte o anumita structura de baza.  
Sintetizat, Interface este o clasa cu o lista de metode obligatorii ce trebuie sa fie create in clasele unde este implementata. Toate metodele specificate in "Interface" trebuie sa fie definite in clasele in care e aplicata, avand aceeasi denumire, tip si parametri cum sunt indicate in "Interface".

**1. Creare interface**

**Clasa interface** se creaza similar cu celelalte tipuri de clase, intr-un "package", salvata in document AS separat. Diferenta e aceea ca la definirea ei, in loc de cuvantul "***class***" se foloseste cuvantul "**interface**"; in plus, in corpul ei se scrie doar o lista cu metodele fara, alt cod.  
Sintaxa generala este urmatoarea:

* **package {  
      public interface numeInterfata {  
          function numeMetoda(prop1:Tip, prop2:Tip, ...):Tip;  
          function altaMetoda(prop1:Tip, prop2:Tip, ...):Tip;  
          ...........  
      }  
  }**

- La declararea metodelor in Interface nu se adauga acoladele sau codul lor, si nici vreun atribut, deoarece in aceasta clasa se scriu doar metode care au atributul "public" in clasele unde e implementata.  
- Nu se pot specifica proprietati, dar pot fi adaugate metode Accesor (prin care se dau valori proprietatilor).  
  
Iata un exemplu cu o Interface, denumita "ITest", in care sunt definite 2 metode: "Links()" si "Tutoriale()".

package {

// Se creaza interface

public interface ITest {

// Lista cu metode

function Links():void;

function Tutoriale(gen:String, nota:Number):String

}

}

- Acest cod se adauga intr-un document "ActionScript 3.0 Class" (sau "ActionScript 3.0 Interface"), la numele clasei se scrie "ITest" si fisierul cu acest cod trebuie salvat cu denumirea "ITest.as", in acelasi director in care sunt salvate si clasele in care va fi implementata.

**2. Implementare interface**

Dupa ce a fost salvat tiparul "interface", se pot crea clase care implementeaza metodele stabilite in acel tipar.  
Implementarea se face adaugand cuvantul **implements** si numele Interfatei la definirea claselor, dupa numele lor.  
                **public class numeClasa implements numeInterfata {  
                    // Instructiuni  
                }**  
Acestea trebuie sa contina in corpul lor toate metodele definite in "interface", cu atribut "public", numarul de parametri si tipul lor stabilite in tipar. Pe langa acestea pot contine si alte metode.

In urmatorul exemplu se creaza o clasa (webDevelopment), intr-un nou document AS, cu numele "webDevelopment.as" si se salveaza in acelasi director unde e si interfata "ITest".

package {

// Creare clasa care aplica Interfata ITest

public class webDevelopment implements ITest

{

// Definire proprietate 'site' (cu atribut "protected")

protected var site:String = 'www.marplo.net';

/\* Definire metodele obligatorii (Links si Tutoriale), din interface \*/

// Returneaza in Output valoarea proprietatii 'site'

public function Links():void

{

trace(this.site);

}

// Returneaza valoarea unei variabile din ea (re), ce preia argumentele transmise

public function Tutoriale(gen:String, note:Number):String

{

var re:String = gen+'-'+note;

return re;

}

// Se creaza o alta metoda, suplimentara

// Afiseaza in Output argumentul primit

public function Diverse(val:\*):void

{

trace(val);

}

}

}

- Metodele obligatorii (aici "Links" si "Tutoriale") respecta exact parametri (numarul, ordinea si tipul) stabilite in "interface" "ITest". Alte metode (aiici "Diverse") si proprietati sunt optionale, in functie de rolul fiecarei clase.  
- Numele parametrilor nu conteaza (*observati ca in loc de "nota" s-a folosit "note"*), dar numarul, tipul si ordinea lor trebuie sa fie aceleasi ca in "interface".  
  
Pentru a testa acest exemplu, se foloseste urmatoarul script ActionScript intr-un document FLA nou, salvat in acelasi director cu interfata si clasa de mai sus.

// Creare instanta obiect la clasa webDevelopment

var obj:webDevelopment = new webDevelopment();

// Se apeleaza metodele

obj.Links(); // www.marplo.net

trace(obj.Tutoriale('Flash', 8)); // Flash-8

obj.Diverse(2010); // 2010

- Deoarece clasa "webDevelopment" indeplineste conditiile din interfata implementata (ITest), scriptul este functional.  
- Daca vreuna din conditii nu ar fi respectata in clasa, cum ar fi: nedefinirea unei metode, adaugarea de parametru in plus sau declararea cu alt tip; scriptul genereaza eroare.  
Astfel, implementarea de "interface" este utila mai ales cand sunt create mai multe clase cu roluri similare si dorim ca acestea sa aibe toate o anumita ordonare si structura minima de metode, mai usor de retinut.

**3. Interface ca tip de date**

Interfata poate fi aplicata si ca tip de data la variabile (sau parametri de functii), astfel, acea variabila (sau parametru) poate fi utilizata ca instanta de obiect la clasa precizata ca valoare, din cele ce folosesc acea "interface".  
Se intelege mai bine din urmatorul exemplu, in care e creata si folosita inca o clasa (limbiStraine) ce aplica Tiparul din "ITest"; contine o proprietate si metodele obligatorii stabilite.

package {

// Creare clasa care aplica Interfata ITest

public class limbiStraine implements ITest

{

// Definire proprietate 'site' (cu atribut "protected")

protected var adr:String = 'www.marplo.net/';

/\* Definire metodele obligatorii (Links si Tutoriale), din interface \*/

// Returneaza in Output cuvantul 'Cale buna'

public function Links():void

{

trace('Cale buna');

}

// Returneaza valoarea unei variabile din ea (re), ce preia argumentele "nr", "gen" si proprietatea "adr"

public function Tutoriale(gen:String, nr:Number):String

{

var re:String = nr+'-'+ this.adr+gen;

return re;

}

}

}

In documentul FLA se scrie urmatorul cod la panoul ActionScript.

// Creare instante obiecte la clasele folosite

var web\_development:webDevelopment = new webDevelopment();

var limbi\_straine:limbiStraine = new limbiStraine();

// Creare functie a carui parametru are ca tip de data Interfata "ITest"

function cursuri(cls:ITest):void

{

// Apeleaza metodele comune (stabilite in ITest) prin parametru "cls"

cls.Links();

trace(cls.Tutoriale('flash', 4));

}

// Apeleaza functia cu numele unei clase ce are aplicat ITest

cursuri(web\_development); // In Output afiseaza: www.marplo.net si flash-4

cursuri(limbi\_straine); // In Output afiseaza: Cale buna si 4-www.marplo.net/flash

- Observati ca apeland functia cu argumente diferite, reprezentand numele instantelor la clase, functia foloseste parametru "cls" (are ca tip de date interfata, ITest) ca instanta la clasa respectiva, si poate apela aceleasi metodele ("Links" si "Tutoriale") pt. fiecare deoarece aceste fiind specificate in "interface" ele trebuie sa existe in fiecare clasa ce apartine acelei Interfate, cu acelasi tip de parametri.  
- Prin aceasta tehnica nu mai e nevoie de a crea aceeasi functie pt. fiecare instanta.  
  
• Precum clasele, si "interface" poate fi creata prin extindere din alta Interfata, mostenind astfel lista de metode a aceleia, plus ce se mai adauga in ea. Sintaxa de definire fiind:  
                **public interface InterfataCopil extends InterfaceParinte {  
                    // Lista metode suplimentare:  
                }**  
- InterfataCopil va contine lista de metode mostenite din InterfataParinte, plus ce se mai adauga in ea. Nu va afecta in nici un fel InterfataParinte.  
  
• O "interface" poate fi aplicate si la clasele extinse (copil), cu formula:  
                **public clasaCopil extends clasaParinte implements Interfata**  
- clasaCopil poate fi o extindere a oricarei clase, indiferent daca clasaParinte are implementata sau nu Interfata.  
  
• O clasa poate implementa mai multe interface, separate prin virgula, dupa formula:  
            **public class numeClasa implements Interface1, Interface2**

*Atentie, daca modificati o "interface" (adaugand metode sau schimbandu-le definirea) care e deja aplicata in clase, acestea nu vor mai functiona deoarece nu mai respecta intru totul Interfata.*

**Clasa Object - Creare Obiecte**

Ierarhic, **Clasa Object** este radacina (root) tuturor claselor din ActionScript, toate sunt o extindere a clasei Object.  
**Object** este o clasa dinamica, adica permite adaugarea de noi proprietati si metode. Prin aceasta se pot crea **obiecte** cu proprietati si metode proprii.

**1. Creare obiecte**

Exista mai multe modalitati de a crea obiecte in ActionScript 3. Forma cea mai comuna este prin intermediul sintaxei:  
                **var numeObiect:Object = new Object();**  
- "numeObiect" devine un obiect la care se pot adauga proprietati si metode proprii, definite de programator.  
  
Un alt mod de a crea un obiect este cu urmatoarea formula:  
                **var numeObiect:Object = {prop1:val1, prop2:val2, ...}**  
- "prop1", "prop2" reprezinta proprietatile obiectului "numeObiect", iar "val1", "val2" sunt valorile lor. Pot fi adaugate oricate astfel de perechi "proprietate:valoare", intre cele doua acolade si separate prin virgula. Daca valorile sunt de tip sir trebuie scrise intre ghilimele, dar numerele nu.  
- Aceasta forma este utilizata de obicei cand obiectul respectiv contine o lista simpla de date, care pot fi accesate dupa cum este prezentat mai jos.

**2. Adaugare si accesare proprietati**

**Proprietatile** se adauga dupa ce a fost creat obiectul, cu "**new Object()**".  
Forma cea mai directa de a defini proprietati cu anumite valori unui obiect este cu notatia punct (.), dupa urmatoarea formula:  
                **numeObiect.nume\_proprietate = valoare;**  
      Sau cu paranteze patrate [] (precum la Array asociativ), cu sintaxa:  
                **numeObiect['nume\_proprietate'] = valoare;**  
  
- La varianta cu paranteze patrate, numele proprietatii trebuie scris intre ghilimele (simple sau duble).  
- "valoare" poate fi un numar, un sir (intre ghilimele), o variabila definita inainte, proprietatea vreunei instante de obiect, apelarea unei functii.  
• Apelarea proprietatilor se face cu oricare din modalitati: folosind caracterul punct (.) intre numele obiectului si proprietate (**Obiect.proprietate**), sau cu proprietatea intre paranteze patrate [] dupa numele obiectului (**Obiect["proprietate"]**).

• Indicat este aplicarea unei singure formule (fara combinatie), pentru o mai buna claritate si intelegere a codului. In general, la obiecte se foloseste notatia cu punct (.), parantezele patrate fiind utilizate la elementele Array, astfel se evita confuzia.  
In urmatorul exemplu, pentru a arata functionalitatea lor, sunt combinate ambele modalitati, atat la definire cat si la apelare (*explicatiile necesare sunt in cod*).

// Se creaza un obiect "obj"

var obj:Object = new Object();

// Se definesc 2 proprietati

// prima cu notatia punct, a doua cu []

obj.site = 'www.marplo.net';

obj['cursuri'] = 8;

/\*

Se poate crea acest obiect si prin formula cu acolade

var obj:Object = {site:"www.marplo.net", cursuri:8};

In acest caz reprezinta acelasi obiect

\*/

// Initializeaza o instanta "TextField" intr-o variabila "txt"

var txt:TextField = new TextField();

// Adauga ca text in "txt" valorile proprietatilor "site" si "cursuri"

// Pt. "site" se foloseste [] iar pt. "cursuri" notatia cu (.)

txt.text = obj["site"]+ ' - '+ obj.cursuri;

// cu "addChild()" adauga /afiseaza textul in prezentarea Flash

addChild(txt); // www.marplo.net - 8

- Daca adaugati acest script intr-un document Flash nou si apasati "Ctrl+Enter", in prezentarea Flash va apare textul "*www.marplo.net - 8*".

**3. Definire apelare metode**

In principiu, **metodele** sunt functii. Diferenta dintre functii si metode este aceea ca functia poate fi apelata direct (in zona unde e creata), dar metoda se apeleaza prin intermediul obiectului in care e definita.  
Valoarea obtinuta prin apelarea unei metode este rezultatul returnat de functia asociata metodei respective.  
Metodele se definesc prin notatia cu punct (.), dupa crearea obiectului. Iar ca valoare se asociaza o functie. Sintaxa generala este urmatoarea:  
                **numeObiect.numeMetoda = numeFunctie;**  
- Functia poate fi creata inainte si asociata prin numele ei, sau poate fi creata direct in momentul definirii metodei, cu formula:  
                **numeObiect.numeMetoda = function():Tip { ... codul functiei };**

Iata un exemplu din care se intelege mai bine. Se creaza un obiect cu o proprietate (id) si doua metode. La prima metoda (getSuma ) se asociaza o functie creata inainte, iar pentru a doua metoda (getId) functia e creata direct in codul definirii ei (dupa sintaxa de mai sus).

// Se creaza functia ce va fi asociata unei metode

function f\_suma(a:Number, b:Number):String

{

var suma:Number = a+b;

return 'Suma este: '+ suma;

}

// Se defineste obiectul "obj2" cu proprietatea "id" si metodele "getSuma" si "getId"

var obj2:Object = new Object();

obj2.id = 'tutoriale';

obj2.getSuma = f\_suma;

obj2.getId = function():String {

var idul = 'Id-ul: '+ this.id;

return idul;

};

// Initializeaza o instanta "TextField" intr-o variabila "txt2"

var txt2:TextField = new TextField();

// Adauga ca text in "txt2" valorile returnate de apelarea metodelor "getSuma()" si "getId"

// Cu "\n" se adauga rand nou

txt2.text = obj2.getSuma(7, 8)+ "\n"+ obj2.getId();

// cu "addChild()" adauga /afiseaza textul in prezentarea Flash

addChild(txt2);

- Prin apelarea celor doua metode: "getSuma" si "getId", vor fi executate functiile asociate lor, acestea returnand fiecare cate un sir.  
- Observati ca pentru asocierea functiei "f\_suma()" la metoda "getSuma" s-a folosit doar numele (fara parantezele rotunde), iar la apelul metodei se tine cont de cei 2 parametri ai functiei (a, b), si metoda se apeleaza cu 2 numere ca parametri.  
- Un alt aspect important in acest exemplu este utilizarea sintaxei "this.id". Cuvantul **this** se foloseste de obicei in interiorul metodelor si mereu face referire laobiectul curent prin care e apelata metoda. "id" este proprietatea obiectului "obj2". Astfel, formula "this.id" returneaza valoarea proprietatii "id" a obiectului curent (obj2).  
- Daca adaugati acest script intr-un document Flash nou si apasati "Ctrl+Enter", in prezentarea Flash va apare 2 randuri text, dupa cum se vede mai jos.

Suma este: 15  
Id-ul: tutoriale

**4. Modificare proprietati si metode**

Valoarea proprietatilor si metodelor dintr-un obiect creat cu "Object" pot fi oricand modificate, folosindu-se aceeasi formula ca pentru crearea lor:  
- Pentru proprietati  
                **numeObiect.nume\_proprietate = alta-valoare;**       sau       **numeObiect["nume\_proprietate"] = alta-valoare;**  
- Pentru metode  
                **numeObiect.numeMetoda = alta\_functie;**     sau     **numeObiect.numeMetoda = function():Tip { ... Alt cod };**

Iata un exemplu care arata practic acest lucru. Se defineste un obiect cu o proprietate "gen" si o metoda "Rasa". Dupa afisarea acestora in Output valorile lor sunt modificate si re-testate.

// Se defineste obiectul "obj3" cu proprietatea "gen" si metoda "Rasa"

var obj3:Object = new Object();

obj3.gen = 'Animale';

obj3.Rasa = function():String { return "Magari" };

// Verifica cu trace() valoarea lor

trace(obj3.gen+ ' - '+ obj3.Rasa()); // Animale - Magari

// Redefineste valoarea lui "gen" si metodei "Rasa"

obj3.gen = 'Pasari';

obj3.Rasa = function():String { return "Porumbei" };

// Verifica inca o data valoarea lor

trace(obj3.gen+ ' - '+ obj3.Rasa()); // Pasari - Porumbei

- Se observa ca la a doua verificare a dat alt rezultat, conform modificarilor.

**5. Stergere proprietati si metode, verificare existenta lor**

Stergere unei proprietati sau metode dintr-un obiect creat cu "Object" se face cu instructiunea **delete** urmata de apelul la acea proprietate sau metoda. Sintaxa fiind:  
                **delete numeObiect.element;**  
- "element" poate fi orice proprietate sau metoda din "numeObiect".

• In ActionScript 3 exista si un operator ( **in** ) prin care se poate verifica existenta unei proprietati sau metode dintr-un obiect sau in general din orice clasa. Sintaxa este:  
                **"element" in Clasa**  
- "element" trebuie adaugat intre ghilimele si poate fi orice proprietate sau metoda din "Clasa" (aceasta poate fi si un obiect creat cu Object).  
- Aceasta instructiune returneaza **true** daca "element" exista in "Clasa", in caz contrar returneaza **false**.  
  
In urmatorul exemplu se defineste un obiect (obj4) cu o proprietate "gen" si o metoda "Rasa". Se verifica cu "in" existenta lor, se sterg cu delete, apoi se face inca o verificare cu "in" si cu "trace()".

// Se defineste obiectul "obj4" cu proprietatea "gen" si metoda "Rasa"

var obj4:Object = new Object();

obj4.gen = 'Animale';

obj4.Rasa = function():String { return "Magari" };

// Verifica cu "in" existenta lor

trace("gen" in obj4); // true

trace("Rasa" in obj4); // true

// Sterge "gen" si metoda "Rasa"

delete obj4.gen;

delete obj4.Rasa;

// Verifica iar cu "in" existenta lor

trace("gen" in obj4); // false

trace("Rasa" in obj4); // false

// Se incearca si apelarea lor

trace(obj4.gen+ ' - '+ obj4.Rasa()); // TypeError: Error #1006: Rasa is not a function ...

- Se observa efectul lui "in" si "delete" din rezultatele returnate.

**Creare Obiecte - Imbricare si parcurgere**

In aceasta lectie sunt prezentate lucruri mai avansate legate de crearea obiectelor cu clasa Object, cum sunt imbricarea si parcurgerea lor.

**1. Imbricare obiecte**

Prin imbricarea obiectelor se intelege crearea de obiecte in interiorul altui obiect. Astfel, proprietatea unui obiect (valoarea ei) poate fi un Array sau chiar un alt obiect.  
Studiind exemplul urmator si explicatiile din el, se poate intelege modul de realizare a obiectelor imbricate.

// Se creaza un obiect principal "Cursuri", cu 2 proprietati: "tip" si "site", si o metota "getCurs()"

// Prima proprietate, "tip", va avea ca valoare un Array cu 2 elemente

// A doua, "site", va avea ca valoare un alt obiect ce va avea si el o proprietate "url"

// Metoda "getCurs()" va prelua, dupa un parametru, o valoare din prima proprietate si

// pe cea din "url" (din obiectul imbricat), prin a doua proprietate

var Cursuri = new Object();

Cursuri.tip = ['lectii', 'tutoriale']; // Prima proprietate, un Array

Cursuri.site = new Object(); // A doua proprietate (obiect intern)

Cursuri.site.url = 'www.marplo.net'; // proprietatea obiectului intern

Cursuri.getCurs = function(nr:Number) { // Metoda obiectului "Cursuri"

var sir1 = this.tip[nr]; // Preia o valoare din proprietatea "tip"

var sir2 = this.site.url; // Preia valoarea proprietatii "url" (a obiectului imbricat)

return sir1+ ' - '+ sir2

};

// Se afiseaza in Output rezultatul returnat de apelarea metodei "getCurs()"

trace(Cursuri.getCurs(1)); // tutoriale - www.marplo.net

- Proprietatea "site" este definita ca un sub-obiect in cel principal (Cursuri);  
- Observati ordinea apelarii din nstructiunea "this.site.url". Aceasta apeleaza proprietatea "url" care e definita in sub-obiectul "site" din obiectul principal (specificat prin "this").  
- Prin expresia "Cursuri.getCurs(1)", metoda "getCurs" preia numarul 1 si, conform instructiunii "this.tip[nr]" obtine elementul cu indice 1 din Array-ul stocat in proprietatea "tip".

**2. Parcurge obiecte**

Parcurgerea elementelor dintr-un obiect se face la fel ca la parcurgerea unui Array asociativ, cu instructiunea "**for ... in**" sau "**for each ... in**".  
Parcurgerea unui obiect este necesara cand se doreste utilizarea dinamica a elementelor din el, in functie de niste valori variabile. De obicei se foloseste pentru a verifica valorile proprietatilor sau numele lor si a metodelor.  
Studiati urmatorul exemplu si explicatiile din el, in care sunt prezentate ambele modalitati; "for ... in" in cadrul unei functii "parsObj()" si "for each ... in" separat.

// Functia pt. parcurgere obiecte

// Preia 2 argumente: "obj" pt. obiectul de parcurs si "val" pt. valoarea ce trebuie cautata in el

function parsObj(obj:Object, val:\*):String

{

// Se defineste o variabila in care se adauga valoarea ce trebuie returnata de functie

var re:String = 'no prop';

//parcurgere "obj" cu "for ... in"

for(var key:String in obj)

{

// Daca valoarea elementului parcurs este aceeasi cu cea din "val"

// Retine in "re" numele acelei proprietati si opreste parcurgerea

if(obj[key]==val)

{

re = key;

break;

}

}

return re;

}

// Definire obiect cu 2 proprietati ("curs", "tutorial") si o metoda "Suma"

var unObiect:Object = new Object();

unObiect.curs = 'Flash';

unObiect.tutorial = 'AS3';

unObiect.Suma = function(a:Number, b:Number):Number {

return a+b;

}

// Apeleaza functia "parsObj()" cu obiectul si valoarea 'AS3'

trace(parsObj(unObiect, 'AS3')); // tutorial

// Se parcurge elementele din "unObiect" cu "for each ... in"

// Afiseaza in Output valoarea fiecarui element parcurs

for each(var elm:\* in unObiect)

{

trace(elm);

}

/\* Va afisa:

Flash

function Function() {}

AS3

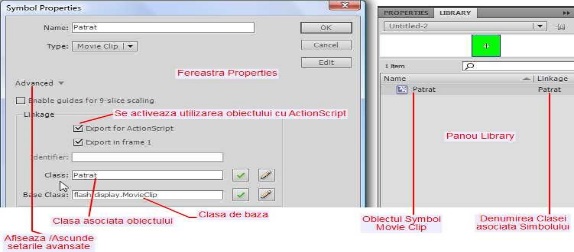
\*/

- Prin codul din corpul lui "for(var key:String in obj)" functia cauta proprietatea (key) din "obj" a carei valoare este "val" si returneaza numele ei; daca nu e gasita, va returna valoarea initiala din "re" ('no prop'). - A doua instructiune "for each(var elm:\* in unObiect)" afiseaza cu "trace()" valoarea din fiecare element parcurs (retinuta in "elm").

**Lucru in ActionScript 3 cu obiecte din Library**

Pentru a lucra in **ActionScript 3** cu un element din Scena se foloseste numele lui de instanta (cel care se scrie in panoul Properties, in casuta unde e scris "*<Instance Name>*"), dar se pot folosi si obiectele (imagini, simboluri) din panoul **Library**. Acest lucru e si mai avantajos deoarece pot fi adaugate dinamic si interactiv in prezentarea Flash mai multe instante ale aceluiasi obiect din Library, folosind ActionScript.  
Pentru a putea folosi in AS3 obiectele aflate in Library, trebuie ca acestea sa fie asociate cu o Clasa, prin activarea optiunii "**Export for ActionScript**", care se gaseste la sectiunea "***Advanced***" in fereastra ce se deschide cu click-dreapta pe obiectul respectiv din Library si se alege optiunea **Properties** (*vedeti imaginea de mai jos*).

Iata mai bine explicat printr-un exemplu simplu.

**1.** Deschideti un document Flash nou, desenati in Scena un patrat si transformati-l in Symbol Movie Clip (de la meniul **Modify -> Convert to Symbol**), la "***Name***" dati-i numele **Patrat** (e util ca fiecare Symbol sa aibe un nume sugestiv, legat de acel obiect, ca sa fie mai usor de recunoscut).  
**2.**Dupa ce e transformat in Symbol, acesta e adaugat automat in panoul Library. Stergeti desenul din Scena si deschideti panoul Library (din meniul **Window -> Library**).  
**3.** Click-dreapta pe numele Simbolului (Patrat) si alegeti **Properties**, se va deschide o fereastra ca cea din imaginea urmatoare, in stanga (daca e nevoie apasati pe **Advanced**).  
  
**4.** Bifati butonul de la "**Export for ActionScript**", acesta asociaza obiectul cu o clasa, a carei nume e adaugat la casuta "**Class**", dand posibilitatea utilizarii lui in scriptul ActionScript.  
Flash adauga automat numele obiectului din Library si ca nume al clasei. Poate fi modificat, dar de obicei se pune numele clasei acelasi cu cel dat la crearea Simbolului, pentru a sti mai usor ce obiect reprezinta. Acesta apare in coloana "*Linkage*" in dreptul acelui obiect din Library (dupa cum se vede in imaginea de sus).  
**5.** Dupa ce butonul "Export for ActionScript" e bifat si numele clasei e stabilit (Patrat), apasati OK. Clasa ce reprezinta acest obiect este creata si poate fi utilizata in ActionScript.  
- "Patrat" devine o clasa copil a celei de baza, adaugata de program in campul ***Base class*** (aici "*flash.display.MovieClip*"). Astfel, "Patrat", care reprezinta obiectul din Library, devine si o clasa in document, prin care se pot crea Instante cu instructiunea **new**, cu formula:  
                **var nume\_instanta:numeClasa = new numeClasa();**  
**6.** Apasati click-dreapta pe Cadru 1 in Timeline, alegeti **Actions** si in fereastra care apare pt. cod ActionScript adaugati urmatorul script:

// Definire variabile a caror valori vor fi folosite pt:

// distanta 'x' si 'y' si rotire

var dist\_x = 38;

var dist\_y = 38;

var rot\_z = 0;

// Definire bucla for care va fi executata de 3 ori

// La fiecare executie creaza, defineste si adauga cate o instanta la Patrat

for(var i:Number=0; i<3; i++)

{

// Creaza instanta de obiect pt. Patrat

// Patrat este clasa ce reprezinta obiectul din Library

var ins\_patrat:Patrat = new Patrat();

// Seteaza distantele 'x', 'y' si rotire

ins\_patrat.x = dist\_x;

ins\_patrat.y = dist\_y;

ins\_patrat.rotationZ = rot\_z;

addChild(ins\_patrat); // Adauga instanta in Scena

// Modifica variabilele prin care se definesc distantele 'x', 'y' si rotirea

// Astfel, la fiecare executie "for()" instanta e adaugata in pozitie diferita

dist\_x += 88;

dist\_y += 88;

rot\_z -= 23;

}

- Acest cod creaza si adauga in prezentarea Flash trei instante ale obiectului Patrat, fiecare cu proprietatile pentru distantele 'x', 'y' si rotire diferite. Explicatii mai detaliate sunt in script.  
- Apasati "Ctrl+Enter" ca sa vedeti rezultatul.

In prezentarea urmatoare se arata cum se face acest exemplu.  
  
- Aceasta metoda poate fi utilizata cu orice obiect care e adaugat in Library (Simboluri Button, Graphic, Imagini importate).  
  
Prin definirea unei clase la elementele din Library se combina grafica cu programarea. In Scena se creaza elementele grafice, se transforma intr-un Symbol si pot fi sterse din Scena daca acestea nu treuie sa apara initial in prezentarea Flash, deoarece sunt stocate in Library, iar prin clasa asociata lor scriptul ActionScript poate adauga interactiv cand si unde e nevoie Instante ale lor in prezentare.  
Important e ca obiectele din Library, din care vor fi create instante folosind ActionScript, sa aiba activata optiunea "*Export for ActionScript*".

*Clasa asociata unui element din Library poate fi editata, se pot adauga instructiuni si metode noi care sa fie aplicate instantelor create cu acea clasa.  
Se da click-dreapta pe obiectul din Library care are legatura cu o clasa, si se alege****Edit Class****.  
Dupa ce se adauga instructiunile necesare in documentul cu aceasta clasa, trebuie salvat (cu numele clasei) in acelasi director unde e si documenul FLA.  
- Vedeti tutorialul*[*Editare clasa la obiecte din Library*](http://www.marplo.net/flash/editare-clasa-obiecte-library_t)*.*

**Creare evenimente noi**

Pe langa evenimentele (**events**) prestabilite in ActionScript 3 (MouseEvent, KeyboardEvent, etc.) pot fi create si altele noi (*Custom Events*), prin extinderea clasei **EventDispatcher**.  
- *"EventDispatcher" e clasa de baza pt.****evenimente***.

Pentru a invata cum se creaza un eveniment nou, studiati urmatorul exemplu.  
Se creaza o clasa (Game) ca extindere a clasei "EventDispatcher". Aceasta trebuie apelata cu un numar ca argument (definit ca parametru "end" a metodei constructor).

**Clasa Game**

// Clasa Game (cu un "Custom Event")

package {

// Importa clasele a caror metode sunt folosite

import flash.events.\*; // Pt. a putea utiliza /extinde clasa EventDispatcher

import flash.utils.\*; // Necesara pt. setTimeout

// Creare clasa Game ca extindere la EventDispatcher

public class Game extends EventDispatcher

{

// Se defineste o constanta ce va reprezenta numele noului eveniment

// Valoarea ei da numele evenimentului

public static const GAME\_OVER:String = 'gameOver';

// Proprietate privata, folosta doar in clasa pt. retinerea numaratorii

private var sec:Number = 0;

// Metoda constructor a clasei

public function Game(end:Number)

{

// Apeleaza functia "numara()" cu proprietatea "sec" ca argument

numara(this.sec);

// Functia "numara()" in cadrul constructorului (apelata prin instructiunea de mai sus)

function numara(sec:Number):void

{

// Daca proprietatea 'sec' are valoarea 3, apeleaza metoda "endGame()"

if(this.sec==end) endGame();

// Altfel

else

{

trace(sec); // Afiseaza in Output valoaarea parametrului 'sec'

this.sec = sec+1; // Modifica valoarea proprietatii "sec" cu cea din parametru + 1

// "this.sec" este proprietatea, 'sec' simplu e parametru functiei

// Apeleaza functie "numara" la fiecare secunda, cu proprietatea "sec" ca argument

setTimeout(numara, 1000, this.sec);

}

}

}

// Metoda privata, apelata cand "sec" ajunge la valoarea parametrului "end"

// "end" e parametru metodei constructor si trebuie transmis la crearea instantei clasei

private function endGame():void

{

// Declanseaza evenimentul creat si reprezentat de aceasta clasa

dispatchEvent(new Event(Game.GAME\_OVER));

}

}

}

- Clasa Game contine o constanta "GAME\_OVER" care va reprezenta numele evenimentului, o proprietate privata "sec" (folosita doar in corpul clasei) si o metoda privata "endGame()" care declanseaza evenimentul creat si reprezentat de aceasta clasa.  
- Metoda constructor incrementeaza cu o unitate la fiecare secunda valoarea proprietatii "sec", afiseaza numarul in Output, iar cand acesta ajunge egal cu valoarea transmisa ca argument la crearea instantei, apeleaza metoda "endGame()", care va declansa evenimentul "Game.GAME\_OVER". Pt. alte ddetalii studiati explicatiile din cod.  
- Aceasta clasa trebuie adaugata intr-un document "ActionScript 3.0 Class", apoi salvat cu numele "Game.as" in acelasi director unde e si documentul FLA care o va folosi.  
  
In documentul Flash se adauga urmatorul cod ActionScript:

// Creare instanta la clasa Game, cu un numar ca argument

var game:Game = new Game(3);

// Inregistrare pt. detectarea evenimentului Game.GAME\_OVER

game.addEventListener(Game.GAME\_OVER, gameOverListener);

// Se poate folosi si ('gameOver', gameOverListener),

// cu numele evenimentului ca parametru 'gameOver', valoarea data in clasa la GAME\_OVER

// Functia apelata cand evenimentul e declansat

function gameOverListener (evt:Event):void

{

trace('Jocul s-a terminat!');

}

- Observati cum e inregistrat evenimentul creat de clasa Game ( Game.GAME\_OVER ), ca oricare alt tip de eveniment din ActionScript, cu "addEventListener()" aplicat la instanta acestei clase. "Game" este evenimentul si "GAME\_OVER" numele lui.  
- Cand proprietatea "sec" din clasa ajunge la numarul 3 (sau oricare e adaugat la 'new Game(nr)'), se apeleaza metoda "endGame()" care declanseaza evenimentul. Aceasta declansare e recunoscuta de inregistrarea facuta cu "addEventListener()" care apeleaza functia "gameOverListener()".  
- In Output va afisa: *0 1 2 Jocul s-a terminat!*.

*De obicei clasa prin care e creat evenimentul nou se foloseste in combinatie cu alta clasa pe care o apeleaza la declansarea evenimentului.*

**Creare date XML - E4X in ActionScript 3**

**XML** (*Extensible Markup Language*) este un format special de structurare si stocare a datelor in fisiere cu extensia *.xml* cu o sintaxa bazata pe tag-uri, este ca o baza de date simpla in care datele sunt adaugate ierarhic in tag-uri, precum la HTML.

Ca o sintaxa simpla a unui cod in format XML este:

<?xml version="1.0"?>

<baza>

<tag atribut="valoare">

<tag\_in atribut="valoare">Data stocata</tag\_in>

</tag>

</baza>

- Daca nu stiti ce face si cum se lucreaza cu documente XML, studiati tutorialul de la pagina [Documente XML](http://www.marplo.net/php-mysql/xml.html).  
  
Formatul XML este in general folosit pentru stocarea de date cum ar fi adrese de imagini sau pagini, adrese si nume de fisiere audio/video si altele, care pot fi utilizate /prelucrate in ActionScript cu functiile specifice E4X, ale claselor XML si XMLList.  
Datele in format XML pot fi utilizate in ActionScript prin preluarea lor dintr-un fisier extern sau create direct in script.

**Creare date XML cu E4X**

**E4X** este un "nickname" (un alt nume) folosit in ActionScript pentru XML in standardul ECMAScript. Este acelasi standard pentru acces XML folosit si in JavaScript 2.0.  
Cu E4X in ActionScript 3 sunt mai multe variante de a crea date in format XML direct in codul scriptului.  
Exista 2 optiuni principale:  
    1. Prin scrierea directa a intregii structuri XML (tag-uri, atribute, ...) ca valoare a unei variabile de tip XML.  
    2. Prin adaugarea separata a fiecarui element. Aici sunt doua modalitati: cu operatorul punct (.) si cu metode ale claselor XML, XMLList.  
• In aceasta lectie vor fi prezentate fiecare din aceste modalitati. Ca regula generala, toate folosesc o instanta de obiect a clasei XML.  
    - Un obiect XML reprezinta o portiune dntr-un format de date XML, precum: elemente (tag-uri), atribute, comentarii, text.  
    - XMLList e o lista de obiecte /elemente XML, indexate numeric, ca un Array.

**1.** Sa luam un exemplu simplu in care e folosita prima varianta.

// Instanta XML cu datele XML complete

var poze:XML = <imglist>

<image id="1">

<url>dir/img1.jpg</url>

<title>Titlu img1</title>

</image>

</imglist>;

trace(poze); // Afiseaza in Output structura XML

- Acest cod creaza o structura de date XML, in variabila "poze", in care e stocata adresa si titlu unei imagini, iar pt. verificare, daca testati, "trace(poze)" afiseaza in Output aceasta structura.  
  
• Datele in codul XML pot fi adaugate si prin variabile, astfel sunt mai usor de modificat si de folosit mai departe in script. Dar aceste variabile trebuie incadrate intre acolade, dupa cum e in exemplu urmator, ce creaza aceeasi structura de date XML ca cel de mai sus.

// Variabile cu date ce trebuie adaugate in XML

var img\_id:Number = 1;

var img\_url:String = 'dir/img1.jpg';

var img\_title:String = 'Titlu img1';

// Instanta XML cu datele XML complete

var poze:XML = <imglist>

<image id={img\_id}>

<url>{img\_url}</url>

<title>{img\_title}</title>

</image>

</imglist>;

trace(poze); // Afiseaza in Output structura XML

**2.** Iata acelasi format XML creat prin adaugarea separata a fiecarui element, cu operatorul punct:

// Instanta XML cu tag-ul root (de baza)

var poze:XML = <imglist/>; // Se poate si new XML(<imglist/>);

poze.image.@id = 1; // Creaza tagul <image>, include si un atribut "id" cu valoare 1

poze.image.url = 'dir/img1.jpg'; // Adauga in tagul <image> un tag <url>

poze.image.title = 'Titlu img1'; // Adauga in tagul <image> un tag <title>

trace(poze);

- Acest cod creaza si returneaza in Output aceeasi structura XML ca cea de la primul exemplu.  
Observati modalitatea de creare a fiecarui element, logica sintaxei e destul de simpla:  
    - Tagul de baza, numit si radacina (root) se adauga la declararea instantei, cu forma <numetag/>  
    - Celelalte tag-uri se creaza prin adaugarea lor la variabila cu instanta XML, sub forma ierarhica, separate cu notatia punct (.).  
    - Atributul se adauga la fel, cu notatia punct, dupa tag-ul unde trebuie inclus, dar cu un caracter "@" in fata, ca sa fie recunoscut ca e un element atribut.  
  
• O alta modalitate de a adauga elemente in structura XML este prin folosirea urmatoarelor metode din clasele XML, XMLList:

* **appendChild(childObj)** - adauga elementul "childObj" la sfarsitul celor din obiectul curent
* **insertChildAfter(child1, child2)** - adauga elementul "child2" dupa "child1" din obiectul XML curent
* **insertChildBefore(child1, child2)** - adauga elementul "child2" inaintea lui "child1" din obiectul XML curent
* **prependChild(childObj)** - adauga "childObj" la inceput in elementul XML la care e aplicata, inaintea oricaror alte elemente din acel obiect

- Aceasta abordare, prin metode, poate parea mai greu de inteles, dar este eficienta cand se doreste adaugarea unor date inainte sau dupa altele deja existente.  
Iata acelasi format XML din exemplele de mai sus, de data aceasta creat prin utilizarea metodelor.

// Instanta XML cu tag-ul root (de baza)

var poze:XML = <imglist/>;

// Se adauga fiecare element in parte

// Tagul <image>, apoi, in el: <url> si <title>

poze.appendChild(<image id="1"></image>);

poze.image.prependChild(<url>dir/img1.jpg</url>);

poze.image.insertChildAfter(poze.image.url, <title>Titlu img1</title>);

trace(poze);

*Se poate aplica oricare din aceste variante, sau combinate. Totusi, in general se folosesc date XML incarcate din fisier extern, deoarece acel fisier poate fi utilizat si cu alte aplicatii pe server (de ex. cu PHP).  
Preluarea datelor XML din fisiere externe este prezentata intr-o lectie urmatoare.*

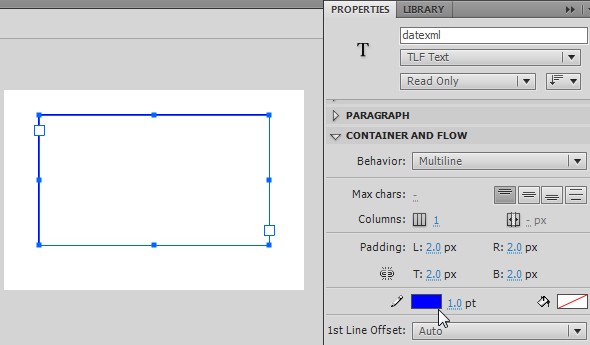
**Accesare date XML - E4X**

Cu functiile ActionScript 3 specifice E4X poate fi accesata si folosita orice data dintr-un continut XML, cum ar fi: numele tag-urilor, textul din ele, atributele si valorile lor, comentarii sau instructiuni de procesare.

**1. Accesare tag-uri si atribute**

**E4X** ofera 2 seturi generale de instrumente pentru accesarea datelor dintr-un format XML:

1. Prin metodele claselor XML si XMLList - *attribute(), attributes(), child(), children(), comments(), descendants(), elements(), parent(), processingInstructions(), text()*
2. Prin stilul ierarhic tip obiecte, cu operatorul punct (.). Fiecare element este considerat un obiect, iar pentru atribute se foloseste caracterul '@' in fata. Doua puncte (..) accesor pt. descendenti.

Cea mai convenabila este a doua varianta (cu punct), fiind mai usor de inteles. Dar sunt unele elemente care pot fi accesate doar cu metode din prima varianta.  
    - elementul parinte se acceseaza cu metoda **parent()**  
    - comentariile prin **comments()**  
    - instructiuni de procesare, cu **processingInstructions()**  
    - elementele si atributele a caror nume contin caractere rezervate se acceseaza doar cu metode  
  
Urmatorul exemplu afisaza intr-o zona de text in prezentarea Flash datele preluate din XML (folosind operatorul punct (.)).  
  1. Mai intai trasati in Scena documentului Flash o zona de text (cu "Text Tool") si dati-i numele de Instanta "datexml" (in panoul Properties). Pentru ca aceasta suprafata de text sa fie afisata cu o bordura, alegeti o culoare la "Container border color", in panoul Properties -> "Container and Flow". Vedeti in imaginea urmatoare indicat locul acestei optiuni, zona de text trasata in Scena si numele de Instanta 'datexml'.  
  
  2. In panoul pt. ActionScript adaugati urmatorul cod:

// Instanta in care e stocat continutul XML complet

var poze:XML = <imglist>

<image id="1">

<url>dir/img1.jpg</url>

<title>Titlu img1</title>

</image>

</imglist>;

// Adauga in "datexml" (zona de Text din Scena) date din XML

datexml.text = 'Tag radacina: '+ poze.name(); // Numele tag-ului radacina

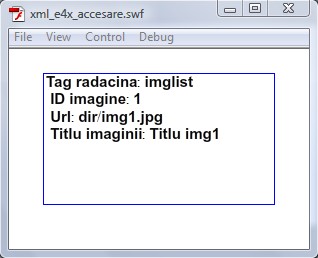
datexml.text += '\n ID imagine: '+ poze.image.@id; // Valoarea atributului 'id' in <image>

datexml.text += '\n Url: '+ poze.image.url; // Datele tag-ului <url>

datexml.text += '\n Titlu imaginii: '+ poze.image.title; // Datele tag-ului <title>

- Se foloseste '+=' pentru a adauga valoarea de dupa egal (=) la cele deja existente; '\n' adauga un rand nou.  
- Observati ca se foloseste si o metoda, "name()", aceasta returneaza numele elementului la care e aplicata (aici fiind chiar instanta cu tag-ul radacina). Operatorul punct (.) trebuie combinat cu metode pentru a lucra cat mai eficient. De exemplu, fara utilizarea functiilor nu se poate obtine numele vreunui element copil dintr-un obiect.

*Cu formula****obiect.\*****se obtin toate elementele (tag-urile) din "obiect", iar cu****@\*****se obtin valorile tuturor atributelor din tagul la care e aplicata.*

  3. Dupa ce se apasa "Ctrl+Enter", va rezulta urmatoarea prezentare Flash:  
  
  
• Accesarea datelor XML cu metode se face printr-o structura ierarhica de noduri (nodes), parinte, copil.  
Iata aceleasi date extrase din XML, dar folosind varianta cu metode.

// Instanta in care e stocat continutul XML complet

var poze:XML = <imglist>

<image id="1">

<url>dir/img1.jpg</url>

<title>Titlu img1</title>

</image>

</imglist>;

// Adauga in "datexml" (zona de Text din Scena) date din XML, preluate prin metode

datexml.text += 'Tag radacina: '+ poze.localName(); // Numele tag-ului radacina

// Valoarea atributului 'id' al primului element copil (<image>) in radacina

datexml.text += '\n ID imagine: '+ poze.children()[0].attribute('id');

// Datele din elementul <url> al primului obiect-copil (<image>) in radacina

datexml.text += '\n Url: '+ poze.children()[0].child('url');

// Datele din al doilea element ([1] <title> ) al primului obiect-copil (<image>) in radacina

datexml.text += '\n Titlu imaginii: '+ poze.children()[0].children()[1];

- Acest cod returneaza si afiseaza in prezentarea Flash aceleasi date ca exemplu precedent, in imaginea de sus.  
- "poze.children()[0]" returneaza continutul primului element copil din obiectul "poze" (primul element are index 0, al doilea 1, ...).  
- "attribute('id')" returneaza valoarea atributului "id" din obiectul la care e aplicata metoda.  
- "child('url')" returneaza datele elementului "url" din obiectul la care e aplicata metoda.  
Aceasta modalitate, cu metode ierarhice (child, parent) e utila in special cand nu se cunosc numele elementelor (de ex. la parcurgerea iterativa a continutului XML).  
*Lista completa cu metodele ce pot fi utilizate pentru lucru in ActionScript cu XML se gaseste la pagina oficiala*[*Clasa XML - AS3*](http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/XML.html)*.*

*Cu metoda****length****se poate prelua numarul de tag-uri sau atribute dintr-un element.  
De ex.:  
              trace(poze.image.\*.length());         // 2 (nr. tag-uri incluse in <image>)  
              trace(poze.image.attributes().length());         // 1 (nr. atribute din <image>)*

**2. Utilizare operator accesor si metoda pt. descendenti (..)**

O instructiune destul de utila in E4X este accesorul doua-puncte (..), care da posibilitatea accesarii directe a tuturor descendentilor dintr-un obiect (noduri copil, nepot, stranepot, ...).  
Cu operatorul accesor (..) se obtine o lista cu toate elementele sau atributele cu un anumit nume din toti descendentii inclusi in obiectul la care e aplicat.  
Clasa XMLList are si o metoda echivalenta la acest operator, denumita **descendants()**, care functioneaza in acelasi mod, dar pt. atribute trebuie adaugata si metoda "attribute()".

Iata cum se folosesc si ce fac operatorul doua-puncte (..) si metoda "descendants()". Studiati codul si explicatiile din urmatorul exemplu.

// Instanta in care e stocat continutul XML complet

var poze:XML = <imglist>

<image id="1">

<url>dir/img1.jpg</url>

<title>Titlu img1</title>

</image>

<image id="2">

<url>dir/img2.jpg</url>

<title>Titlu pt. img2</title>

</image>

</imglist>;

trace(poze..title); // sau trace(poze.descendants('title'));

/\* Returneaza toate tag-urile <title> aflate in oricare descendent in "poze"

<title>Titlu img1</title>

<title>Titlu pt. img2</title>

\*/

trace(poze.descendants().attribute("id")); // sau trace(poze..@id);

// Returneaza 12 (valoarea atributelor "id" aflate in fiecare descendent in "poze")

// Se obtine al doilea atribut "id"

trace(poze..@id[1]); // 2

- Expresia dupa '*sau*' arata echivalentul prin care se obtine acelasi rezultat.

**3. Accesare Comentarii si Instructiuni de procesare din XML**

In continutul XML , pe langa datele din tag-uri si atribute, pot fi adaugate Comentarii si Instructiuni de porocesare.  
Comentariile sunt cele adaugate cu sintaxa:  
                **<!-- Comentariu -->**  
Instructiunile de procesare sunt adaugate cu formula:  
                **<?app Date cu Instructiuni ?>**  
- Instructiunile de procesare sunt in general fragmente de cod care trebuie executate pe server. De exemplu, pt. PHP ar fi:     ***<?php Cod PHP ?>***  
  
Aceste doua forme de date auxiliare pot fi accesate utilizand metodele clasei XMLList: **comments()** (*pt. comentarii*) si **processingInstructions()** (*pt. instructiuni de procesare*). Ambele metode returneaza o lista cu aceste elemente.  
In mod implicit (default), parser-ul E4X ignora aceste tipuri de date, astfel, pentru a le face accesibile trebuie specificate proprietatile: **ignoreComments** (*pt. comentarii*) si **ignoreProcessingInstructions** (*pt. instructiuni de procesare*) cu valoare **false**. Ambele se aplica direct la clasa XML.  
                ***XML.ignoreComments = false;***  
                ***XML.ignoreProcessingInstructions = false;***

In exemplu urmator se foloseste acelasi document FLA creat la inceputul lectiei (cu zona de text "datexml" trasata in Scena), dar cu alte date XML, ce contin doua comentarii si doua instructiuni de procesare in tag-ul radacina <site>. Va afisa primul comentariu si a doua instructiune in campul de text "datexml".  
Stergeti alt cod ActionScript din documentul cu zona de text "datexml" si adaugati acest script:

// Se activeaza recunoasterea comentariilor si instructiunilor de procesare

XML.ignoreComments = false;

XML.ignoreProcessingInstructions = false;

// Instanta in care e stocat continutul XML complet

var marplo:XML = <site>

<!-- http://www.marplo.net -->

<?php Instructiuni procesare cod PHP ?>

<!-- Cursuri si Tutoriale -->

<cursuri>

<curs id="1" title="PHP-MySQL">www.marplo.net/php-mysql/</curs>

<curs id="2" title="JavaScript">www.marplo.net/javascript/</curs>

<curs id="3" title="FlashActionScript">www.marplo.net/flash/</curs>

</cursuri>

<?php Cod PHP ?>

</site>;

// Preia in 2 variabile comentariile si instructiunile PHP aflate direct in root (tagul <site>)

// Variabilele sunt de tip Object deoarece si datele sunt preluate cu acest tip

var coms:Object = marplo.comments();

var insp:Object = marplo.processingInstructions();

// Adauga primul Comentariu si a doua Instructiunea in 'datexml'

datexml.text = coms[0];

datexml.text += '\n'+ insp[1];

- Observati ca variabilele "coms" si "insp" sunt definite ca tip "Object", deoarece in E4X datele sunt preluate si stocate ca obiecte, cu indexare numerica (de la 0, precum la Array), de aceea se folosesc indexi [0], [1] pentru accesarea separata a fiecaruia.  
- Apasand "Ctrl+Enter" se obtine urmatorul rezultat:  


*Daca sunt comentarii si instructiuni in alte tag-uri incluse in cel radacina, acestea pot fi preluate cu operatorul accesor (..) sau metoda "descendants()", cu formula:****instanta\_xml..\*.comments()******instanta\_xml.descendants().processingInstructions()*** *- Acestea nu preia si pe cele incluse direct in tag-ul radacina.*

**4. Accesare date XML filtrate**

O alta abilitate importanta pe care o are E4X este posibilitatea de a filtra datele dintr-un obiect XMLList.  
Filtrarea se face cu sintaxa:  
                **elemente\_XMLList.(expresie\_conditionala)**  
- "elemente\_XMLList" reprezinta tag-urile si atributele dintr-un obiect XMLList  
- "expresie\_conditionala" sunt instructiuni conditionale prin care se returneaza doar elementele care indeplinesc acea conditie.  
Deoarece filtrarea se aplica la obiecte de tip XMLList, datele XML trebuie mai intai adaugate intr-o instanta XMLList

Iata cum se aplica aceasta formula la continutul XML folosit si in exemplu precedent.

// Instanta in care e stocat continutul XML complet

var marplo:XML = <site>

<!-- http://www.marplo.net -->

<?php Instructiuni procesare cod PHP ?>

<!-- Cursuri si Tutoriale -->

<cursuri>

<curs id="1" title="PHP-MySQL">www.marplo.net/php-mysql/</curs>

<curs id="2" title="JavaScript">www.marplo.net/javascript/</curs>

<curs id="3" title="FlashActionScript">www.marplo.net/flash/</curs>

</cursuri>

<?php Cod PHP ?>

</site>;

// Se creaza un obiect XMLList ce contine toate elementele din <site>

var site:XMLList = marplo.\*

// Se obtin doar tag-urie "curs" care au atributul id>1

var elms:XMLList = site.curs.(@id>1);

trace(elms);

/\* In Output va afisa:

<curs id="2" title="JavaScript">www.marplo.net/javascript/</curs>

<curs id="3" title="FlashActionScript">www.marplo.net/flash/</curs>

\*/

// Se obtine doar tag-ul care are in text cuvantul "flash"

var tag1:XMLList = site.curs.(text().search("flash") != -1);

trace(tag1); // www.marplo.net/flash/

// Se obtine atributul "title" aflat in tag-ul care are id<3 si in textul din tag cuvantul "php"

var tag2:XMLList = site.\*.(@id<3 && text().search("php") != -1).@title;

trace(tag2); // PHP-MySQL

- Expresia "*site.curs.(@id>1)*" returneaza elementele din tag-ul <curs> care au atributul "id" mai mare decat 1.  
- Aceasta "*site.curs.(text().search("flash") != -1)*" cauta in textul din tag-urile <curs> cuvantul 'flash' si returneaza pe cele gasite.  
- "*site.\*.(@id<3 && text().search("php") != -1).@title*" cauta in tag-urile din "site" pe cele care au atributul "id" mai mic decat 3 si in textul lor cuvantul "php", apoi returneaza valoarea atributului "title" din tag-urile gasite.

*Observati ca se pot face filtrari destul de complexe pt. a obtine exact elementele cautate. Dar pentru a filtra o lista in care nu toate tag-urile au atributul sau elementul copil specificat in conditia de filtrare, trebuie folosita metoda****hasOwnProperty()****, pentru a verifica existenta acelui atribut sau tag-copil. Altfel, returneaza eroare.  
De exzemplu, urmatorul cod returneaza toate elementele din "un\_xml" care au atributul "nr" mai mare decat 7.  
              un\_xml..\*.(hasOwnProperty("@nr") && @nr>7)*

**Incarcare si parcurgere XML din fisier extern**

De obicei, in majoritatea aplicatiilor, datele in format **XML** sunt **incarcate din fisiere externe**.  
ActionScript 3 contine functii speciale prin care poate fi incarcat si utilizat continutul dintr-un fisier extern.  
Pentru a prelua datele dintr-un fisier XML se executa urmatorii pasi:

1. Se creaza un obiect **URLRequest** in care e adaugata adresa fisierului XML.
2. Se creaza un obiect **URLLoader**, iar la declararea acestuia se adauga ca argument obiectul definit cu "URLRequest".
3. La instanta celui de-al doilea obiect (URLLoader) se inregistreaza cu **addEventListener()** evenimentul **Event.COMPLETE** (*prin acesta se detecteaza cand fisierul e incarcat complet*)
4. Se defineste functia care trebuie apelata de evenimentul inregistrat, in care, cu proprietatea **data** aplicata la instanta "URLLoader" se preia datele incarcate si le transfera la o instanta de obiect XML.

- Dupa ce au fost astfel preluate datele din fisierul extern si adaugate intr-o variabila de tip XML, elementele si atributele din continutul XML pot fi accesate si prelucrate cu metodele specifice E4X (*prezentate in lectia anterioara*).  
Iata si un exemplu practic. Se creaza un fisier denumit "xmltest.xml", in care se adauga urmatorul cod XML. Fisierul trebuie salvat in acelasi director unde e si documentul Flash in care va fi incarcat.

**Datele XML din xmltest.xml**

<?xml version="1.0"?>

<site>

<cursuri>

<curs id="1" title="PHP-MySQL">www.marplo.net/php-mysql/</curs>

<curs id="2" title="JavaScript">www.marplo.net/javascript/</curs>

<curs id="3" title="FlashActionScript">www.marplo.net/flash/</curs>

</cursuri>

</site>

In panoul pt. ActionScript din documentul Flash se adauga urmatorul script:

// Declarare instanta de obiect XML

var test\_xml:XML;

// Creare obiect pentru adresa fisierului

var url:URLRequest = new URLRequest('xmltest.xml');

// Obiectul pt. incararea adresei (la argument se adauga obiectul cu adresa)

var loader:URLLoader = new URLLoader(url);

// Inregistrare eveniment care detecteaza incarcarea completa a datelor

// Moment in care apeleaza functia "getXml()"

loader.addEventListener(Event.COMPLETE, getXml);

// Functia getXml() apelata de eveniment

function getXml(evt:Event):void

{

// Daca datele sunt incarcate (stocate in proprietatea "data" a obiectului "loader")

if(loader.data)

{

// Sterge inregistrarea evenimentului, ne mai fiind necesara, si elibereaza memoria

loader.removeEventListener(Event.COMPLETE, getXml);

// Adauga datele in instanta XML

test\_xml = XML(loader.data);

/\* Se acceseaza datele incarcate \*/

var nr\_c:Number = test\_xml.cursuri.\*.length(); // Preia Nr. elemente din <cursuri> (3)

// Returneaza numele tagului radacina

trace(test\_xml.localName()); //site

// Datele ultimului element din <cursuri> (nr\_c-1)

trace(test\_xml.cursuri.child(nr\_c-1)); // www.marplo.net/flash/

}

}

- Deoarece in variabila 'nr\_c' e retinut numarul de elemente (tag-uri) din <cursuri>, (nr\_c-1) reprezinta index-ul ultimului element, iar expresia "*test\_xml.cursuri.child(nr\_c-1)*" preia datele din el.  
- *Pentru alte explicatii despre incarcarea datelor si functiile folosite, sunt comentariile din cod.*  
- Acest exemplu va afisa in Output:

site  
www.marplo.net/flash/

**Parcurgere obiect XML**

Parcurgerea unui obiect inseamna accesarea pe rand a fiecarui element (unul dupa altu) din acel obiect.  
De exemplu, intr-un obiect XML, daca se doreste cautarea tag-urilor care contin un anumit atribut, trebuie parcurs obiectul pentru verificarea fiecarui elemet.  
Parcurgerea se face cu instructiunea "**for each ... in**", sintaxa generala este:

* **for each (*var elm:XML* in *ObiectXML..\**) {  
      // Instructiuni  
  }**

- Formula "..\*" aplicata la ObiectXML face ca parcurgerea sa se faca in toate elementele imbricate in el.  
- Variabila "elm" retine in ea elementul curent parcurs.  
  
Iata cum pot fi parcurse datele din fisierul "xmltest.xml" (folosit in exemplu precedent).  
Se aplica metoda de incarcare prezentata mai sus, iar in functia apelata dupa incarcarea completa (getXml()) se adauga **instructiunea de parcurgere "for each ... in"**, in corpul ei se verifica (cu "if()") daca elementul parcurs contine atributul "id" si acesta are valoarea 2. Daca sunt indeplinite conditiile, returneaza cu "trace()" numele si textul din acel element.

// Declarare instanta de obiect XML

var test\_xml:XML;

// Creare obiect pentru adresa fisierului

var url:URLRequest = new URLRequest('xmltest.xml');

// Obiectul pt. incararea adresei (la argument se adauga obiectul cu adresa)

var loader:URLLoader = new URLLoader(url);

// Inregistrare eveniment care detecteaza incarcarea completa a datelor

// Moment in care apeleaza functia "getXml()"

loader.addEventListener(Event.COMPLETE, getXml);

// Functia getXml() apelata de eveniment

function getXml(evt:Event):void

{

// Daca datele sunt incarcate (stocate in proprietatea "data" a obiectului "loader")

if(loader.data)

{

// Sterge inregistrarea evenimentului, ne mai fiind necesara, si elibereaza memoria

loader.removeEventListener(Event.COMPLETE, getXml);

// Adauga datele in instanta XML

test\_xml = XML(loader.data);

// Se parcurg elementele din fiecare tag descendent (..\*) din "test\_xml"

for each(var elm:XML in test\_xml..\*)

{

// Daca elementul (tagul) parcurs are atributul "id" si acesta are valoarea 2

// Afiseaza numele acelui tag si textul din el

if(elm.hasOwnProperty("@id") && elm.@id==2) trace(elm.localName()+ ' - '+ elm.text());

// Afiseaza: curs - www.marplo.net/javascript/

}

}

}

- Expresia "test\_xml..\*" face ca parcurgerea sa se faca in fiecare descendent din "text\_xml" (element copil, nepot, ...). Daca se scrie doar "test\_xml" nu sunt parcurse si verificate tag-urile mai imbricate, ci doar cele copil (*despre descendenti sunt explicatii in lectia anterioara*).  
- Tag-ul curent parcurs e retinut in variabila "elm", iar "*elm.hasOwnProperty("@id")*" verifica daca acesta contine atributul "id". Fara aceasta metoda de verificare, daca elementul parcurs nu are acel atribut, conditia "*elm.@id==2*" returneaza eroare.  
- "*elm.localName()*" preia numele tag-ului, "*elm.text()*" textul din el.  
- Scriptul afiseaza in Output: *curs - www.marplo.net/javascript/*

*Pentru ca parcurgerea sa se faca doar la atribute, se foloseste formula:****for each(var atr:\* in ObiectXML..@\*)*** *- Pentru exemplu de mai sus s-ar aplica:  
          for each(var atr:\* in test\_xml..@\*) {  
              if(atr==2) trace(atr.parent().localName()+ ' - '+ atr.parent().text());  
          }  
- "parent()" returneaza elementul parinte (cel in care e inclus).  
- Rezultatul afisat este acelasi.*

**Editare, Modificare XML - E4X**

Pe langa posibilitatea de a crea, incarca si accesa date **XML, E4X** are si functii de editare, modificare si stergere a lor in ActionScript.  
Pentru aceste operatii de **editare** pot fi utilizate /combinate doua variante: cu operatorul plus (.) si cu metode din clasele XML, XMLList.

**1. Modificare elemente si atribute**

Modificarea valorii unui element sau atribut se face cel mai usor cu operatorul plus (.), cand se cunoaste bine structura continutului XML.  
Se acceseaza elementul /atributul respectiv si cu egal (=) i-se atribue valoarea dorita. Dupa cum e prezentat in urmatorul exemplu.

// Instanta in care e stocat continutul XML complet

var marplo:XML = <site>

<cursuri>

<curs id="1" title="PHP-MySQL">www.marplo.net/php-mysql/</curs>

<curs id="2" title="JavaScript">www.marplo.net/javascript/</curs>

<curs id="3" title="Flash ActionScript">www.marplo.net/flash/</curs>

</cursuri>

</site>;

// Se modifica valoarea din al doilea tag <curs> (din <cursuri>)

marplo.cursuri.curs[1] = 'www.marplo.net/engleza/';

// Modificare atribut "title" din al doilea tag <curs>

marplo.cursuri.curs[1].@title = 'Limba Engleza';

// Se verifica daca datele au fost schimbate

trace(marplo);

/\* Va afisa:

<site>

<cursuri>

<curs id="1" title="PHP-MySQL">www.marplo.net/php-mysql/</curs>

<curs id="2" title="Limba Engleza">www.marplo.net/engleza/</curs>

<curs id="3" title="Flash ActionScript">www.marplo.net/flash/</curs>

</cursuri>

</site>

\*/

- Se observa din comentariu de sub "trace()" (pt. a arata rezultatul afisat) ca datele XML au fost modificate, mai exact, la al doilea tag <curs>.  
  
• Iata aceleasi schimbari facute cu metode (*explicatiile sunt in cod*):

// Instanta in care e stocat continutul XML complet

var marplo:XML = <site>

<cursuri>

<curs id="1" title="PHP-MySQL">www.marplo.net/php-mysql/</curs>

<curs id="2" title="JavaScript">www.marplo.net/javascript/</curs>

<curs id="3" title="Flash ActionScript">www.marplo.net/flash/</curs>

</cursuri>

</site>;

// Adauga intr-o variabila XMLList obiectul la care se fac modificari

// tag-urile "curs" din primul copil al tag-ului radacina

// Nu e obligatoriu, dar e mai usor cand se lucreaza de mai multe ori cu acelasi element

var curss:XMLList = marplo.children()[0].child('curs');

// Modifica valoarea din al doilea element stocat in variabila "curss"

curss[1] = 'www.marplo.net/engleza/';

// Schimba valoarea atributului 'title' din al doilea element stocat in variabila "curss"

curss[1].attribute("title")[0] = 'Limba Engleza';

// Se verifica daca datele au fost schimbate

trace(marplo);

- Va afisa in Output acelasi rezultat ca primul exemplu.

*Nici una din variante nu este stricta, depinde cat de bine se cunoaste continutul si ierarhia datelor XML precum si contextul in care sunt facute modificarile.  
Pot fi si combinate, de ex.:****marplo.\*.child('curs')[1].@title = 'Limba Engleza';***

**2. Adaugare tag-uri si atribute noi**

Cand trebuie adagate tag-uri noi intr-un obiect XML se folosesc de obicei metode (sau in combinatie cu operatorul (.)), deoarece ofera un control mai bun asupra locului unde sa fie adaugat (inainte sau dupa alte tag-uri din acel obiect).  
Pentru atribute, daca nu conteaza ordinea lor in tag, se adauga folosind mai mult varianta cu operatorul punct (.) si caracterul '@'.

In exemplu urmator se adauga un nou tag <curs> in <cursuri>, la inceput, inaintea celor deja existente (cu metoda "**prependChild()**"). Apoi, cu o instructiune "for()" se parcurg aceste tag-uri si la fiecare adauga un atribut "nr" (explicatiile sunt in cod).

// Instanta in care e stocat continutul XML complet

var marplo:XML = <site>

<cursuri>

<curs id="1" title="PHP-MySQL">www.marplo.net/php-mysql/</curs>

<curs id="2" title="JavaScript">www.marplo.net/javascript/</curs>

<curs id="3" title="Flash ActionScript">www.marplo.net/flash/</curs>

</cursuri>

</site>;

// Se creaza o variabila (tip XML) cu datele pt. un nou element "curs", apoi o adauga in <cursuri>

var new\_curs:XML = <curs id="0" title="HTML">www.marplo.net/html/</curs>;

marplo.cursuri[0].prependChild(new\_curs);

// Adauga intr-o variabila XMLList elementul cu tag-urile <curs>

var curss:XMLList = marplo.cursuri[0].curs;

// Parcurge "curss" pana la ultimul element (aflat prin "curss.length()")

for(var i:Number=0; i<curss.length(); i++)

{

// Adauga la fiecare un atribut "nr" cu valoarea curenta a lui i

curss[i].@nr = i;

}

trace(marplo);

- In Output va afisa urmatorul continut XML. Se observa noul tag <curs> (primul) si atributul "nr" la fiecare.

<site>  
    <cursuri>  
        <curs id="0" title="HTML" nr="0">www.marplo.net/html/</curs>  
        <curs id="1" title="PHP-MySQL" nr="1">www.marplo.net/php-mysql/</curs>  
        <curs id="2" title="JavaScript" nr="2">www.marplo.net/javascript/</curs>  
        <curs id="3" title="Flash ActionScript" nr="3">www.marplo.net/flash/</curs>  
    </cursuri>  
</site>

• Metodele clasei XML folosite pt. a adauga elemente noi sunt:

* **appendChild(childObj)** - include elementul "childObj" la sfarsitul celor din obiectul curent
* **insertChildAfter(child1, child2)** - adauga elementul "child2" dupa "child1" din obiectul XML curent
* **insertChildBefore(child1, child2)** - adauga elementul "child2" inaintea lui "child1" din obiectul XML curent
* **prependChild(childObj)** - include "childObj" la inceput in elementul XML la care e aplicata, inaintea oricaror alte elemente din acel obiect
* **setChildren(codXML)** - adauga datele specificate la "codXML" ca element copil in obiectul la care e aplicata functia, inlocuind alte elemente deja existente in el.

**3. Stergere elemente si atribute**

Pentru a sterge elemente si atribute din XML se foloseste operatorul **delete**, cu sintaxa:  
                **delete ObiectXML.element\_sau\_atribut**  
De exemplu:  
      ***delete xmlObj.elm[0];***         // Sterge primul tag <elm> din radacina  
      ***delete xmlObj.elm[0].tag[1];***         // Sterge al doilea <tag> din primul <elm> din radacina  
      ***delete xmlObj.elm[0].@atr;***         // Sterge atributul "atr" din primul tag <elm> din radacina  
      ***delete xmlObj.elm.tag.@atr;***         // Sterge atributul "atr" din toate elementele <tag> incluse in fiecare <elm> din radacina  
      ***delete xmlObj.\*;***         // Sterge toate elementele  
      ***delete xmlObj.\*.tag.@\*;***         // Sterge toate atributele din fiecare element <tag>

In exemplu urmator se sterge ultimul tag <curs> si atributul "id" din toate elementele "curs".

// Instanta in care e stocat continutul XML complet

var marplo:XML = <site>

<cursuri>

<curs id="1" title="PHP-MySQL">www.marplo.net/php-mysql/</curs>

<curs id="2" title="JavaScript">www.marplo.net/javascript/</curs>

<curs id="3" title="Flash ActionScript">www.marplo.net/flash/</curs>

</cursuri>

</site>;

// Preia nr. elemente <curs> din <cursuri> (ca sa poata fi cunoscut indexul pt. ultimu)

var nr\_c:\* = marplo.cursuri[0].curs.length();

// Sterge ultimul tag <curs> (indexul lui fiind [nr\_c-1])

delete marplo.cursuri.curs[nr\_c-1];

// Sterge atributul "id" din toate elementele <curs>

delete marplo.cursuri.curs.@id;

trace(marplo);

- Dupa ce sunt aplicate instructiunile de stergere (delete), continutul XML ramas si afisat este:

<site>  
    <cursuri>  
        <curs title="PHP-MySQL">www.marplo.net/php-mysql/</curs>  
        <curs title="JavaScript">www.marplo.net/javascript/</curs>  
    </cursuri>  
</site>

• Operatiile de **editare** pot fi combinate si cu instructiuni de **filtrare**.  
De exemplu:  
            *delete marplo.cursuri.curs.(hasOwnProperty('@id') && @id==2).\*;*  
- Sterge continutul din elementul "curs" care are atributul id="2".  
Iar urmatoarea instructiune:  
            *marplo.\*.curs.(hasOwnProperty('@id') && @id==2).setChildren(<alttag>Text</alttag>);*  
- Inlocuieste continutul din elementul "curs" care are atributul id="2" cu "*<alttag>Text</alttag>*".  
  
• Pentru a transforma datele dintr-un obiect XML in sir (String) se foloseste metoda **toXMLString()**, dupa formula:  
                **ObiectXML.toXMLString()**  
- Aceasta metoda permite utilizarea continutului XML ca un sir.  
  
• Pentru a transforma un sir cu date XML intr-un obiect XML-E4X, se adauga sirul la o instanta tip XML, dupa formula:  
                **var ObiectXML:XML = new XML(sirDateXML);**  
  
• Continutul XML editat in ActionScript poate fi salvat pe server prin transferul lui la adresa unui script de prelucrare date pe server (cum e PHP), folosind metoda**load()**. Gasiti un tutorial despre transferul de date XML din ActionScript la PHP, la pagina [Trimitere date XML din ActionScript la server PHP](http://www.marplo.net/flash/xml-actionscript-server-php_t)

**Namespace XML - E4X**

In general, **Namespace** e folosit in **XML** pentru a grupa intr-un set cu acelasi nume mai multe elemente sau atribute cu rol asemanator; e utilizat mai ales la RSS.  
In fisierele XML, Namespace se defineste cu un atribut **xmlns** la care se atribuie cu doua-puncte (:) un prefix (sau nume) pt. Namespace si ca valoare se adauga un sir, care de obicei e o adresa URI (*Uniform Resource Identifier*). Sintaxa este:  
                **xmlns:prefix="uri"**  
De exemplu:

<poze xmlns:ns="http://www.marplo.net">

<ns:image>img1.jpg</ns:image>

<image>img2.png</image>

</poze>

- "ns" reprezinta numele Namespace-ului (prefixul), poate fi orice cuvant.  
- "http://www.marplo.net" este o valoare de identificare pt. acest Namespace, poate fi orice sir de cuvinte, dar de obicei se folosesc adrese URI care pot fi si ne-valide, fara legatura cu site-ul (*de ce se obisnuiesc adrese? - ei stiu*).  
Elementele si atributele cu acelasi nume de Namespace sunt considerate ca facand parte dintr-un acelasi grup.  
Namespace poate fi declarat in elementul care-l foloseste sau in radacina, se pot defini mai multe namespace in acelasi tag.

**1. Lucru cu namespace in E4X**

Pentru lucru cu namespace folosind **sintaxa E4X**, in ActionScript exista mai multe metode a clasei XML.

* **namespace('prefix')** - returneaza valoarea namespace-ului a carui prefix e adaugat la parametru. Daca obiectul la care este aplicata nu are "namespace", returneaza "undefined".
* **addNamespace(obiectNS)** - adauga namespace-ul definit intr-o instanta de obiect "Namespace", (obiectNS).
* **removeNamespace(obiectNS)** - sterge namespace-ul definit intr-o instanta de obiect "Namespace", (obiectNS).
* **setNamespace(obiectNS)** - seteaza la elementul unde e aplicata namespace-ul definit in "obiectNS".

De exemplu:

// Variabila cu datele XML

var galery:XML = <poze xmlns:ns="http://www.marplo.net">

<ns:image>img1.jpg</ns:image>

<image>img2.png</image>

</poze>;

trace(galery.image[0].namespace('ns')); // http://www.marplo.net

trace(galery.image[1].namespace('ns')); // Error #1010: A term is undefined

• Pentru a crea namespace in ActionScript se foloseste clasa **Namespace**. Iar cu proprietatile ei: **prefix** si **uri** pot fi accesate si editate numele si valoare namespace din acea instanta.  
                **var nume\_var:Namespace = new Namespace();**  
- La declararea ei pot fi adaugate prefixul (numele) si valoarea pt. namespace ( **new Namespace("prefix", "valoare\_uri" )** ) sau pot fi setate ulterior cu proprietatile "prefix" si "uri".  
Dupa ce instanta de obiect Namespace e definita, poate fi adaugata in obiectul XML cu metoda **addNamespace()**.  
In urmatorul exemplu se creaza un obiect cu date XML, simplu, cu 2 taguri in radacina si un atribut la al adoile. Se defineste un namespace care va fi adaugat in primul tag si la atributul din al doilea.

// Variabila cu datele XML

var test:XML = <root>

<tag>Text elm1</tag>

<tag atr="val">Elm 2</tag>

</root>;

// Se defineste instanta cu namespace-ul

var ns1:Namespace = new Namespace('ns', 'www.marplo.net/flash');

// Adauga namespace-ul in obiectul XML (in tag-ul radacina)

test.addNamespace(ns1);

test.tag[0].setNamespace(ns1); // Seteaza "ns" la primul <tag>

// Seteaza "ns" la atributul "atr" din al doilea <tag>

test.tag[0].@atr.setNamespace(ns1);

// Sau: test.children()[1].@atr.setNamespace(ns1);

trace(test);

/\* Afiseaza:

<root xmlns:ns="www.marplo.net/flash">

<ns:tag>Text elm1</ns:tag>

<tag ns:atr="val">Elm 2</tag>

</root>

\*/

- Din rezultatul returnat de "trace()" se vede ca a fost adaugat namespace-ul in XML, la primul <tag> si la "atr".  
- De observat modul in care "ns1" e adaugat la atributul din al doilea <tag>. Desi ca ordine este al doilea element, s-a folosit index [0]. Asta deoarece inainte de aceasta instructiune, primul <tag> a primit un namespace, iar "tag[index]" acceseaza elementele <tag> fara namespace, ceea ce face ca index 0 sa fie la primul tag fara namespace. Pentru a evita confuzia ce poate sa apara in astfel de cazuri, ca sa se foloseasca ordine in care sunt elementele in continutul XML, se foloseste metoda "children()".  
De ex.:         *test.children()[1].@atr.setNamespace(ns1);*

**2. Utilizare operator doua-puncte (::)**

**E4X** contine si un operator special pentru lucrul cu nodurile (nodes) care contin namespace, operatorul doua-puncte (::).  
Cu acesta se pot accesa nodurile (elemente, atribute) care folosesc acelasi namespace. Se adauga intr-o instanta de tip Namespace acele noduri, cu metoda "namespace('prefix')", dupa care pot fi accesate cu sintaxa:  
                **ObiectXML.instantaNamespace::nod**  
Se intelege mai bine din urmatorul exemplu:

// Variabila cu datele XML

var galery:XML = <poze xmlns:ns="http://www.marplo.net">

<ns:image>img1.jpg</ns:image>

<ns:image ns:title="Img 2">img2.png</image>

</poze>

// Adauga intr-o instanta Namespace toate elementele (si atribute) cu "ns"

var name\_s:Namespace = galery.namespace('ns');

trace(galery.name\_s::image[0]); // img1.jpg

trace(galery.name\_s::image[1].@name\_s::title); // Img 2

- Observati in datele XML cum se aplica instanta Namespace la elemente si atribute (ns:tag , ns:atribut), apoi, instanta "name\_s" le stocheaza in ordine (ca un Array), iar accesarea lor se face cu operatorul (::), conform sintaxei de mai sus.  
  
Folosind acest operator, elementele stocate in instanta de obiect Namespace pot fi de asemenea editate si sterse.  
De exemplu:  
        *galery.name\_s::image[1].@name\_s::title = 'Alt titlu';*  
- Modifica valoarea atributului "title" din al doilea element <image>.  
Iar instructiunea:  
        *delete galery.name\_s::image[0];*  
- Sterge primul element <image>.  
  
• Elementele cu namespace pot fie accesate fi prin metode ierarhice ( child((), children() ), fie prin instanta de obiect cu acel namespace si operatorul doua-puncte (::). Simplu, cu "element[index]" se acceseaza doar elementele fara namespace, dupa cum e aratat in urmatorul exemplu.

// Variabila cu datele XML

var test:XML = <root xmlns:ns="www.marplo.net/flash">

<ns:tag>Text elm1</ns:tag>

<tag ns:atr="val">Elm 2</tag>

</root>

// Returneaza valoarea din primul element, folosind tag[0]

trace(test.tag[0]); // Elm 2 (e al doilea, dar primul fara namespace)

// Returneaza valoarea din primul element, folosind children()[0]

trace(test.children()[0]); // Text elm1

**RegExp - Expresii regulate in ActionScript**

Inainte de a prezenta modul de lucru cu **RegExp** in ActionScript, iata pe scurt cateva notiuni introductive, de baza, despre **Expresiile regulate** (RegExp - *Regular Expresions*). Acestea sunt un sir de caractere sablon care descriu multimea cuvintelor posibile care pot fi formate cu acele caractere, respectand anumite reguli. Aceste expresii regulate folosesc paranteze (rotunde, patrate, acolade) prin care formeaza regulile de formare a cuvintelor. Utilitatea cea mai frecventa a unei expresii regulate consta in a recunoaste daca un sir contine sau nu caractere, cuvinte sau sub-sir care se incadreaza in tiparul format prin expresia regulata respectiva.  
        *De ex.:* expresia **c[eo]r** poate forma cuvintele: *cer* si *cor*  
Un sablon pentru siruri care sa contina doar vocale se poate face utilizand expresia **[aeiou]** (*incluzand valorile posibile intre paranteze*). Daca se doreste si permiterea majusculelor, se adauga si acestea **[aeiouAEIOU]** (sau se poate folosi modificatorul "i", **/[aeiou]/i**, *despre modificatori, mai jos*).  
Pentru siruri care pot include orice litera scrisa cu minuscule, puteti scrie: **[abcdefghijklmnopqrstuvwxyz]**. Sau o forma mai compacta **[a-z]**, unde prin cratima "-" se intelege "*o serie de caractere consecutive de la 'a' la 'z'*".  
        Similar, tiparul **[0-9]** reprezinta siruri care contin doar numere.  
- Pentru a specifica numarul de caractere pe care un sir il poate contine, din cele incluse in RegExp, se adauga o pereche de acolade in care se includ minimul si maximul numarului de caractere permis.  
        De exemplu, expresia regulata **[aeiou]{1,3}** corespunde sirurilor care sunt formate din 1, 2 si 3 vocale.  
- Pentru a specifica repetarea mai multor parti ale unei expresii regulate, se includ partile respective intre paranteze rotunde.  
        De exemplu, **([sm]at){1 ,2}** corespunde unui numar de una sau doua repetari ale oricaruia dintre sirurile "sat", "mat".

Exista mai multe caractere speciale care se folosesc la formarea expresiei regulate.  
- Daca un accent circumflex (^) este primul simbol mentionat intre parantezele drepte, acesta are ca efect inversarea semnificatiei expresiei regulate plasate intre paranteze.  
        De exemplu, **[^a-z]** corespunde oricarui caracter care nu este o litera mica.  
Cand acest caracter este pus in afara parantezelor drepte, el reprezinta inceput de sir sau de linie.  
        Expresia regulata **^[sm]at** corespunde sub-sirurilor "sat" sau "mat" numai daca acestea apar la inceputul sirului subiect.  
Similar, pentru a forma o expresie regulata ce corespunde numai unui sir care include anumite caractere la sfarsit, se foloseste un simbol al dolarului "$".  
        De exemplu, **pa[nr]a$** corespunde sirurilor "pana" sau "para" numai daca acestea apar la sfarsitul sirului subiect.  
  
• Iata o lista cu mai multe caractere speciale si rolul lor in expresiile regulate:

* **^**   - indica inceputul liniei
* **$**   - indica sfarsitul liniei
* **.**   - (punct) orice caracter
* **[]**   - un caracter dintre cele din paranezele patrate
* **[^]**   - orice caracter, in afara celor din paranezele patrate
* **\**   - adauga in expresie caractere speciale, fara a mai avea rolul lor special
* **+**   - caracterul (expresia) anterior acestui semn se poate repeta cel putin odata si de cate ori e posibil (*de la 1 la infinit*)
* **\***   - caracterul (expresia) anterior acestui semn se poate repeta de cate ori e posibil sau niciodate (*de la 0 la infinit*)
* **?**   - caracterul (expresia) anterior acestui semn se poate repeta ce mult odata
* **<>**   - un cuvant intreg
* **(|)**   - lista de optiuni SAU
* **{m, n}**   - repetarea expresiei de la "m" la "n" ori
* **\n**   - rand nou (pt. Windows e \r\n)
* **\t**   - Tab

- De ex.: **[ca|ma]sa** corespunde cuvintelor: *casa* si *masa*.  
- Pentru a adauga aceste caractere in sablonul RegExp, dezactivand rolul lor special, trebuie prefixate cu un caracter backslash (\).  
        De ex.: **[0-9]\\*[0-9]** reprezinta o inmultire intre doua numere (\* nu mai e factor de repetitie).  
  
• Pe langa aceste caractere exista si formule speciale pentru scurtarea expresiei RegExp:

* **\w**   - Caractere pt. cuvinte (litere, numere si '\_'). Echivalent cu **[a-zA-Z0-9\_]**
* **\W**   - Caractere care nu sunt litere, numere sau '\_'. Echivalent cu **[^a-zA-Z0-9\_]**
* **\s**   - Spatiu, Tab, Rand-nou. Echivalent cu **[\t\n\r ]**
* **\S**   - Siruri fara Spatiu, Tab sau Rand-nou. La fel cu **[^\t\n\r ]**
* **\d**   - Caractere numerice. Ca si **[0-9]**
* **\D**   - Caractere non-numerice. Ca si **[^0-9]**
* **.**   - Orice caracter, in afara de rand-nou

- De exemplu: **[\d\s]+** se potriveste sirurilor ce contin doar numere si spatii.  
  
Iata cateva exemple de expresii regulate:

* **(.\*)**   - Reprezinta toate caracterele (*prin .*) repetate de cate ori e posibil (*dat de \**)
* **<bine>**   - Cuvantul "bine"
* **(ci|co)tim**   - Reprezinta "citim" si "cotim"
* **^www.+net$**   - Sirurile care incep cu "www" si se sfarsesc cu "net"
* **^www\.[a-z0-9]+\.ro$**   - Reprezinta sirurile "www.---.ro" unde '---' poate fi orice litera sau cuvant ce contine litere mici si numere
* **(^-\+[0-9]\*)**   - Orice numar care incepe cu "-" sau "+"
* **\<tag\>(.\*?)\<\/tag\>**   - Reprezinta continutul dintre <tag>...</tag>
* **\<tag\>(.[^\<]+)**   - Textul de la <tag>, pana la prima "</"
* **^([a-zA-Z0-9]+[a-zA-Z0-9.\_%-]\*@([a-zA-Z0-9-]+\.)+[a-zA-Z]{2,4})$**   - Expresie regulata pentru adrese de e-mail

In afara de caracterele speciale si formule de scurtare a expresiei se folosesc si modificatori. Acestia sunt cei listati mai jos; ei modifica interpretarea expresiei regulate.

* **g**   - (*Global*)   - determina continuarea cautarii expresiei si dupa prima potrivire
* **i**   - (*ignore-case*)   - nu mai face diferenta intre majuscule si litere mici
* **m**   - (*multiline*)   - schimba rolul lui "^" si "$". Daca multiline nu e specificat, acestia indica inceputul si sfarsitul textului din RegExp, dar cand e adaugat vor indica inceputul si sfarsitul intregii linii
* **s**   - (*dotall*)   - face caracterul punct "." sa reprezinte in expresia RegEsp si rand-nou
* **x**   - (*extended*)   - daca este adaugat, determina ignorarea spatiilor din formula expresiei (util cand se doreste un cod RegExp mai aerisit, dar spatiile sa nu fie interpretate ca spatiu, daca nu sunt prefixate cu '\')

- Acestia se adauga (unul sau mai multi) la sfarsitul exprexiei RegExp, de exemplu: **/\d{3}-[a-z]+/gi** (cauta toate sub-sirurile de forma "nnn-cuvant", 'nnn' fiind un numar de 3 cifre iar 'cuvant' poate fi si cu majuscule).

**Utilizare RegExp in ActionScript 3**

**RegExp** ( **Expresii Regulate** ) in **ActionScript** sunt instante ale clasei ***RegExp*** si pot fi definite literal (intre caracterele slash /.../) sau cu operatorul **new**:  
                **var reg1:RegExp = /expresie-regulata/g;**  
                **var reg2:RegExp = new RegExp("expresie-regulata", "g");**

Sunt 2 cai prin care expresiile regulate pot fi utilizate in ActionScript: folosind metode specifice din clasa String ("match()", "search()" "replace()"), sau proprietati si metode din clasa RegExp ("test()" si "exec()").  
  
• **test()** se foloseste pt. a verifica daca intr-un sir exista sub-sir care sa corespunda cu o expresie regulata, returneaza **true** sau **false** daca nu exista vreun sub-sir corespunzator expresiei. Sintaxa este:  
                **RegExp.test("sir")**

// RegExp cu expresie ce corespunde sablonului: "cuvant-nr3" (nr3 = numar cu 3 cifre)

var reg:RegExp = /[a-z]+-\d{3}/gi;

// Sirurile care vor fi verificate

var sir1:String = 'MarPlo - Cursuri-008 si Tutoriale-137';

var sir2:String = 'AS3 Lectii-37';

// Se verifica cu test() daca in siruri exista sablonul din 'reg'

trace(reg.test(sir1)); // true

trace(reg.test(sir2)); // false

- Dupa cum se vede, metoda trace() arata doar daca in sir exista sau nu vreun sub-sir potrivit expresiei din variabila "reg".  
- La 'sir1' returneaza true deoarece "*Cursuri-008*" (si "*Tutoriale-037*") se potriveste expresiei, dar pt. 'sir2' returneaza false (desi e asemanatoare, numarul nu are 3 cifre).  
  
• **search()** (apartinand clasei String) cauta sub-sirul corespunzator expresiei si returneaza un numar ce reprezinta locatia primei aparitii a lui in sir. Sintaxa este:  
                **Sir.search(RegExp)**

// RegExp cu expresie ce corespunde sablonului: "cuvant-nr3" (nr3 = numar cu 3 cifre)

var reg:RegExp = /[a-z]+-\d{3}/gi;

// Sirurile care vor fi verificate

var sir1:String = 'MarPlo - Cursuri-008 si Tutoriale-137';

var sir2:String = 'AS3 Lectii-37';

// Cauta cu search() locatia unui sub-sir dupa expresia din 'reg'

trace(sir1.search(reg)); // 9

trace(sir2.search(reg)); // -1

- Este returnata locatia primului sub-sir din 'sir1' care corespunde expresiei din 'reg'; *Cursuri-008*" e gasit ca incepe de la al noualea caracter (primul are locatia 0).  
  
• **exec()** returneaza un obiect in care e stocat sub-sirul gasit cat si locatia lui (retinuta intr-o proprietate **index**). In caz contrar returneaza *null*. Sintaxa este:  
                **RegExp.exec("sir")**  
Pentru a gasi toate sub-sirurile corespunzatoare expresiei regulate, se combina "**exec()**" cu o instructiune "**while**", dupa cum se vede in exemplu urmator.

// RegExp cu expresie ce corespunde sablonului: "cuvant-nr3" (nr3 = numar cu 3 cifre)

var reg:RegExp = /[a-z]+-\d{3}/gi;

// Sirul care va fi verificat

var sir1:String = 'MarPlo - Cursuri-008 si Tutoriale-137';

// Variabila in care vor fi retinute rezultatele valide date de exec

var exc:Object;

// Cu exec() se verifica tot 'sir1', printr-o bucla while()

while(exc=reg.exec(sir1))

{

trace(exc[0]+ ' = '+ exc.index); // Returneaza: sub\_sir = locatie

}

/\* Afiseaza:

Cursuri-008 = 9

Tutoriale-137 = 24

\*/

- Daca nu se foloseste cu "while()", "exec()" retine doar primul sub-sir gasit.  
  
• **match()** retine intr-un Array secvential sub-sirurile gasite. Daca modificatorul "Global" (g) e adaugat, cauta in tot sirul, daca nu e specificat se opreste la primul element gasit. Daca nici un sub-sir nu e gasit, returneaza *null*. Sintaxa este:  
                **Sir.match(RegExp)**

// RegExp cu expresie ce corespunde sablonului: "cuvant-nr3" (nr3 = numar cu 3 cifre)

var reg:RegExp = /[a-z]+-\d{3}/gi;

// Sirul care va fi verificat

var sir1:String = 'MarPlo - Cursuri-008 si Tutoriale-137';

// Adauga intr-o variabila tip Array sub\_sirurile gasite de match() aplicat cu 'reg' la 'sir1'

var ar\_matc:Array = sir1.match(reg);

// Daca 'ar\_matc' are cel putin un element, afiseaza pe primul

if(ar\_matc.length>0) trace(ar\_matc[0]); // Cursuri-008

• **replace()** inlocuieste sub\_sirul (sau sub\_sirurile) ce corespund unei expresii RegExp cu alt continut. Daca modificatorul "Global" (g) e adaugat, cauta in tot sirul, daca nu e specificat se opreste la primul element gasit. Returneaza sirul modificat, dar nu-l afecteaza pe cel original. Sintaxa este:  
                **Sir.replace(RegExp, 'un\_sir')**

// RegExp cu expresie ce corespunde sablonului: "cuvant-nr3" (nr3 = numar cu 3 cifre)

var reg:RegExp = /[a-z]+-\d{3}/gi;

// Sirul care va fi utilizat cu replace()

var sir1:String = 'MarPlo - Cursuri-008 si Tutoriale-137';

// Adauga intr-o variabila tip String sirul rezultat dupa inlocuirea partilor ce corespund expresiei din 'reg'

var sir1\_mod = sir1.replace(reg, 'alt\_text');

// Verifica cu trace sirul initial si pe cel cu rezultatul modificat

trace(sir1); // MarPlo - Cursuri-008 si Tutoriale-137

trace(sir1\_mod); // MarPlo - alt\_text si alt\_text

- Din rezultate se observa ca sirul original ramane neschimbat, dar se obtine si cel cu inlocuirea facuta.

**Animatie cu clasa Timer**

In acest tutorial e prezentat modul de creare a unei animatii a carei miscari sunt sincronizate cu un interval de timp specificat. Pentru acest tip de animatie facuta cu ActionScript se foloseste **Clasa Timer**.  
**Timer** este o clasa utilizata pentru executarea codurilor dupa un interval de timp specificat.  
Fiecare obiect "*Timer*" declanseaza evenimentul **TimerEvent.TIMER** la o anumita frecventa de timp specificata. Astfel, functia apelata prin acest eveniment se executa repetat, la acel interval de timp, iar prin modificarea frecventa a aspectului vizual al unui obiect poate crea efectul de **animatie**.  
Animatia creata cu **TimerEvent** nu depinde de numarul de cadre pe secunda (precum la cea cu ENTER\_FRAME), ci de frecventa de timp setata la utilizarea clasei Timer. Aceasta animatie poate fi afectata daca sistemul sau executia aplicatiei Flash este ocupata in momentul cand trebuie apelata functia la evenimentul TimerEvent.TIMER.

• Pasii generali pentru crearea unei animatii cu clasa Timer si evenimentul TimerEvent.TIMER sunt urmatorii:

1. Se creaza instanta de obiect la clasa Timer, la argument se adauga intervalul de timp (in miimi de secunda); pe langa acesta poate fi specificat si numarul de repetari.  
           **var nume\_var:Timer = new Timer(interval, repetari);**  
   - Intervalul de timp poate fi setat si cu proprietatea **delay**, iar numarul de repetari cu **repeatCount** (*valoarea 0 indica fara limita*)
2. Se creaza o functie care trebuie apelata periodic la declansarea TimerEvent.  
           **function numeFunctie(evt:TimerEvent):void {  
               // Codul functiei  
           }**
3. Se inregistreaza detectarea evenimentului TimerEvent.TIMER (la instanta-Timer), care apeleaza functia.  
           **instantaTimer.addEventListener(TimerEvent.TIMER, numeFunctie);**
4. Se aplica metoda **start()** la "instantaTimer". Aceasta porneste declansarea evenimentului *TimerEvent.TIMER* la intervalul de timp specificat.  
           **instantaTimer.start();**

• Alt eveniment ce poate fi utilizat la clasa Timer este **TIMER\_COMPLETE**. Acesta e declansat cand s-a efectuat complet numarul de repetari.  
• Metodele clasei Timer sunt:

* **start()** - Porneste declansarea evenimentului TIMER
* **stop()** - Opreste declansarea evenimentului TIMER
* **reset()** - Opreste TIMER si reseteaza numarul de repetari, aducandu-l inapoi la 0

Iata printr-un exemplu cum se aplica aceste evenimente si metode ale clasei Timer pentru a crea o animatie (*Explicatii pt. codul ActionScript sunt in documentatia din script*):

**1.** Deschideti un document Flash nou, desenati in Scena o figura geometrica sau orice desen doriti (*aici s-a folosit un dreptunghi, desenata cu "Rectangle Tool"*) si transformati-l in Movie Clip (de la **Modify - Convert to Symbol**. Dati instantei din Scena (desenului transformat in MovieClip) un nume, aici e folosit "***drept1***" (in panoul Properties, in casuta de sus, unde e scris "<Insance Name>")  
**2.** Dati click-dreapta pe Cadru 1 din Timeline, alegeti **Actions** si in panoul pt. script ActionScript adaugati urmatorul cod:

var dimens:Number = 0.12; // Coeficient de dimensionare a obiectului (12%)

var move\_y:Number = 23; // Distanta Y de deplasare

var rotire:Number = 30; // Grade pt. rotire

// Creare instanta la clasa Timer, cu intervalul jumatate de sec. si repetari 6

var jumpTimer:Timer = new Timer(500, 6);

// Functia care va fi apelata la evenimentul TimerEvent.TIMER

function jumpLand(evt:TimerEvent):void

{

// Modifica dimensiunile obiectului cu valoarea coeficientului (%) de la "dimens"

drept1.scaleX += dimens;

drept1.scaleY += dimens;

drept1.rotation -= rotire; // Roteste

drept1.y -= move\_y; // Muta pe verticala

}

// Functia care va fi apelata la evenimentul TimerEvent.TIMER\_COMPLETE

function atTimerComplete(evt:TimerEvent):void

{

// Schimba in sens opus valoarea de dimensionare si miscare pe verticala

dimens \*= -1;

move\_y \*= -1;

evt.target.reset(); // Reseteaza numarul de repetari (sa inceapa de la 0)

evt.target.start(); // Porneste iar declansarea TIMER

}

// Inregistrare eveniment TimerEvent.TIMER, la instanta "jumpTimer"

jumpTimer.addEventListener(TimerEvent.TIMER, jumpLand);

// Inregistrare eveniment TimerEvent.TIMER\_COMPLETE, la instanta "jumpTimer"

jumpTimer.addEventListener(TimerEvent.TIMER\_COMPLETE, atTimerComplete);

jumpTimer.start(); // Porneste declansarea evenimentului TIMER

- Cand numarul de repetari ajunge la valoarea specificata (aici 6), se declanseaza evenimentul TimerEvent.TIMER\_COMPLETE. In functia apelata de acesta (atTimerComplete) se schimba sensul miscarii animatiei, se reseteaza numarul de repetari si porneste iar detectarea TimerEvent.TIMER, care prin reset incepe de la 0. Si tot asa, se obtine o executie continua a miscarii.  
    - *Daca apasati "Ctrl+Enter" va apare o prezentare Flash ca aceasta:*

- Pentru o miscare mai rapida, se micsoreaza valoarea data la interval (delay).

• *In afara de utilizarea TimerEvent.TIMER, un alt eveniment prin care se pot crea efecte de animatie cu ActionScript 3 este****ENTER\_FRAME****, vedeti tutorialul:*[*Animatie cu ENTER\_FRAME in AS3*](http://www.marplo.net/flash/animatie-enter-frame-as3_t).