Univ. Babeș-Bolyai,

Facultatea de Matematică și Informatică

Lect. dr. Darius Bufnea

Notițe de curs Programare Web: JavaScript (săptămâna 7 de școală)

Pe lângă prezentul material legat de limbajul JavaScript, studenți sunt rugați „ferm" să parcurgă si următorul material: http://www.w3schools.com/js/default.asp (toate secțiunile din stânga până la JS AJAX (fără JS AJAX) plus JS Examples).

JavaScript s-a născut ca un limbaj interpretat client-side, inițial destinat browser-elor. A fost dezvoltat de Brendan Eich de la Netscape (Netscape fiind considerat bunicul „Firefox"), inițial denumit LiveScript. A apărut în decembrie 1995, sub denumirea de JavaScript, denumire data în urma unui agreement între Sun Microsystems (inventatoarea Java) și Netscape.

Paranteză: La acea vreme Sun dorea să își popularizeze applet-urile Java ca tehnologie client-side (in confruntare directă cu Macromedia Flash) și pentru acest lucru acordul cu Netscape prevedea ca browser-ul celor de la Netscape (Netscape Navigator) să suporte mașina virtuala Java care să permite rularea applet-urilor Java, iar in schimb Netscape să poate folosi in denumirea limbajului cuvantul „Java" pentru a face limbajul ce avea să fie cunoscut mai târziu sub denumirea de JavaScript mai popular. In timp, applet-urile Java au pierdut lupta cu Macromedia Flash aceasta din urmă impunându-se ca și tehnologie pe frontend, iar și mai tarziu Flash-ul (tehnologie ajunsa între timp in ograda Adobe) a pierdut definitiv „războiul" in favoarea JavaScript.

La început limbajul a fost destul de nestandardizat, existând mai multe variante, printre cele mai populare numărându-se:

- JavaScript – varianta Netscape
- JScript – varianta Microsoft implementata de primele versiuni de Internet Explorer
- ActionScript – varianta Macromedia – in perioada in care animațiile Flash ca tehnologie client-side erau la moda, ActionScript-ul fiind limbajul de facto pentru Macromedia Flash și mai târziu pentru Adobe Flash (când Adobe a cumpărat Macromedia)

Toate aceste „variante" (sau dialecte) s-au dovedit a fi o corvoadă pentru programator – nu erau rare situațiile in care trebuiau scrise variante de „script" diferite pentru a suporta multiple browsere (spre exemplu o variantă de script pentru Internet Explorer și o varianta de script pentru Netscape Navigator). S-a ajuns la situația / nevoia de a standardiza acest limbaj în ceea ce se numește ECMAScript.

Deși inițial limbajul a fost gândit pentru a fi folosit client-side (adică a fi rulat de către browser-e), in ultimul timp se folosește și ca tehnologie pe backend (NodeJS).

Ce este / ce nu este și ce se poate face in JavaScript?

Ce este:

- limbaj interpretat de către browserul Web (client-side);
- ca limbaj urmează mai multe paradigme: imperativ, funcțional, orientat obiect, case sensitive, orientat pe evenimente (event-driven).

Ce permite:

- interacțiunea cu pagina (documentul HTML);
- modificarea dinamică a conținutului documentului, crearea de noi elemente (tag-uri) in cadrul pagini, ștergerea unor elemente (tag-uri), modificarea atributelor HTML (adăugarea, ștergerea, schimbarea valorilor) a unor anumite tag-uri. Se permite astfel crearea de conținut dinamic in cadrul documentului HTML / crearea de documente HTML dinamice la nivelul clientului (DHTML);
- permite execuția anumitor funcții la apariția anumitor evenimente (interactivitate cu utilizatorul);
- validări pe frontend – spre exemplu validarea datelor introduse în câmpurile unui formular (observație: validările pe front-end sunt doar „de dragul" de a face pagina / interfața cât mai user-friendly. Validările de pe frontend trebuie OBLIGATORIU însoțite și de validări pe backend care sunt VITALE din punct de vedere al securității).

Ce nu este JavaScript:

- Nu este un limbaj înrudit cu limbajul Java - in afara de nume, singura legătura este sintaxa C comună specifică ambelor limbaje;
- nu este un limbaj de programare strong typed, e weakly typed - o variabilă poate primi inițial ca valoare un număr întreg, iar ulterior un string (în timp ce JavaScript e considerat weakly typed, Java este strongly type).

Inserarea codului JavaScript in documentul HTML

Codul JavaScript în interiorul documentului HTML se inserează cu ajutorului tag-ului `script`, fie in interiorul tag-ului, fie specificat cu ajutorul atributului `src` a acestui tag:

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

sau

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

Observații:

- atributul `src` poate indica spre un fișier local cu extensia .js (găzduit în același loc ca și documentul HTML) sau spre un URL absolut (care începe cu http:// sau https://);
- dacă, codul JavaScript se specifică cu ajutorul atributului `src`, nu uitați să închideți tag-ul script (trebuie sa apară și marcajul de sfârșit de tag, fiind un tag cu corp).

In unele situații este util după folosirea tag-ului script, folosirea tag-ului `noscript`, util in situații în care browser-ul nu suportă JavaScript din diverse motive – puține probabile în prezent (browser vechi, browser pentru dispozitive mobile mai vechi, engine-ul JavaScript al browserului este dezactivat). Exemplu:

```
<script>
   alert("Hello World!");
</script>
<noscript>I can't say hello because your browser doesn't support JavaScript!</noscript>
```

O practică des întâlnită pentru a preîntâmpina unele erori pe unele browser-e incapabile sa ruleze cod JavaScript este plasarea codului JavaScript în interiorul unui comentariu HTML `<!-- -->`. Exemplu:

```
<script type="text/javascript">
   <!--
   // începutul codului JavaScript
   alert("Hello World!");
   /*
   sfârșitul codului JavaScript
   */
   -->
</script>
```

Observație: pe exemplu de mai sus am exemplificat și folosirea comentariilor in limbajul JavaScript. Acestea sunt similar cu cele din limbajele C/C++/Java, respective folosire `//` pentru comentariu single-line și `/* */` pentru comentarii multi-line.

Localizarea codului JavaScript in cadrul pagini HTML

In mod tradițional, codul JavaScript era plasat în cadrul unui tag `<script>` in secțiunea `<head></head>` a documentului HTML. Acest lucru nu este obligatoriu, tag-ul `<script>` putând fi plasat oriunde in cadrul documentului.

IMPORTANT: CAT TIMP BROWSER-UL EXECUTA COD JAVA SCRIPT NU „FACE NIMIC ALTCEVA” – NU (RE)RANDEAZA PAGINA, NU INTERACTIONEAZA CU UTILIZATORUL (NU RAPSUNDE LA COMENZI/CLICK-URI DE MOUSE ETC).

Exemplu:

```
<a href="http://www.google.com">Click me</a>
<script>
while(1);
```

```
</script>
```

Cateva recomandari tinand cont de observatia importanta de mai sus:

- executia secventelor de cod JavaScript e important sa dureze cat mai putin pentru a da „sansa” browserului sa re(randeze) pagina si sa raspunda la input-ul utilizatorului;
- in sectiunea head pot fi incluse definitii de functii sau incluse fisiere .js care contin definitii de funcții JavaScript (aceste funcții nu se executa chiar atunci, sunt doar definite – vedem mai târziu ce se întâmplă cu ele);
- se recomandă pe cât posibil modelul de apel asincron a unor funcții: nu se apelează funcția direct așteptându-se („timp mort”) terminarea ei pentru obținerea rezultatului, ci se va seta apelul funcției ca funcție de „callback” la apariția unui anumit eveniment. Important în acest context e ca funcția dorită a fi apelată nu se executa instant, iar valoarea returnată de aceasta nu este disponibilă imediat, ci doar mai târziu când se termină de executat funcția – moment în care se poate utiliza și valoarea returnată de această funcție.
- crearea de elemente (tag-uri) `<script>` dinamic care sa se executa ulterior după randarea documentului HTML (o astfel de logica poate fi plasata la sfârșitul documentului HTML) – mai târziu un exemplu in acest sens.

Exemplul 1:

```
<html><head><title>Exemplul 1</title>
<script type="text/javascript">
    function clickMe() { // executia functiei nu are loc acum
        alert('Hello world');
    }
</script>
</head>
<body>
    <a href="javascript:clickMe();">Click here</a>
</body></html>
```

Exemplul 2:

```
<html>
    <head><title>Exemplul 2 JavaScript</title></head>
<body>
    Astazi este:
    <script type="text/javascript">
        document.write(new Date()); // executia are loc acum
    </script>
</body>
</html>
```

Elemente de bază ale limbajului JavaScript: Funcții, variabile, tablouri, obiecte, instrucțiuni de control JavaScript

Funcții

O funcție JavaScript se definește cu cuvântul rezervat `function` și poate fi apelată oricând după definiția acesteia:

```
<script>
function sum(a, b) {
    return a + b;
```

```
}
alert(sum(1,2));
</script>
```

Returnarea unei valori se face cu `return` (asemănător cu C/C++/Java), iar în lista parametrilor formali (`a` și `b` în exemplul de mai sus) aceștia nu trebuie sa aibă declarat un tip.

In JavaScript este deosebit de uzuală folosirea funcțiilor anonime. Exemplu:

```
element.onclick = function () {
    alert('Aceasta este o functie anonima');
}
```

O funcție anonimă poate fi chiar și apelată după definirea ei:

```
<script>
(function (a, b) { // a si b sunt parametrii formali ai functiei
    // functie anonima care afiseaza suma a doua numere
    var s = a + b;
    alert(s);
})(2, 3); // apelul cu parametrii actuali ai functiei
</script>
```

Variabile și tipuri in JavaScript

Variabilele JavaScript se declară cu cuvântul rezervat `var`. Tipul acestora este nedefinit (de fapt tipul unei variabile este dat de tipul valorii asociate acesteia). Astfel, o variabilă poate primi la un moment dat ca și valoare un număr, iar ulterior un șir de caractere – JavaScript fiind considerat un limbaj *weakly* și *dynamically typed*.

Exemplu:

```
var i = 7;
i = 'Ana are mere';
var s = "Cocosul canta";
// șirurile de caractere pot fi delimitate atât cu ' cat și cu "
s = 3.1415;
var c = true;
```

Printre tipurile de valori pe care le pot fi atribute unei variabile sunt: `number`, `string`, `boolean`, `object`, `function` (funcțiile sunt de fapt niște obiecte mai speciale), `undefined`.

Exerciții: Pentru a vă familiariza cu tipurile din JavaScript, operatorul `typeof`, precum și cu câteva funcții de conversie precum `eval()`, `Number()`, `String()`, puteți încerca să rulați următoarele linii de cod în consola JavaScript a browserului preferat. Consola JavaScript este accesibilă sub forma unui tab separat în cadrul *Developer Tools*-ului din cadrul browserului (F12 în orice browser – dar mă aștept să știți acest lucru daca ați făcut debugging la laboratorul de CSS ☺).

```
typeof 1
typeof 1.5
```

```
typeof '1.5'
typeof eval('1.5')
typeof Number('1.5')
typeof String(2)
typeof 'Ana are mere'
typeof "Cocosul canta"
typeof true
typeof {}
typeof x
f = function (){ return 2;}
typeof f
typeof [1, 2, 3, 4, 5]
punct = { x: 7, y: 9}
typeof punct
1/0
typeof Infinity
typeof 1/0
```

```
>> typeof 1
<- "number"
>> typeof 1.5
<- "number"
>> typeof '1.5'
<- "string"
>> typeof eval('1.5')
<- "number"
>> typeof Number('1.5')
<- "number"
>> typeof String(2)
<- "string"
>> typeof 'Ana are mere'
<- "string"
>> typeof "Cocosul canta"
<- "string"
>> typeof true
<- "boolean"
>> typeof {}
<- "object"
```

```
>> typeof x
<- "undefined"
>> f = function (){ return 2;}
<- ▸ function f()
>> typeof f
<- "function"
>> typeof [1, 2, 3, 4, 5]
<- "object"
>> punct = { x: 7, y: 9}
<- ▸ Object { x: 7, y: 9 }
>> typeof punct
<- "object"
>> 1/0
<- Infinity
>> typeof Infinity
<- "number"
>> typeof 1/0
<- NaN
```

Întrebare: Dacă `1/0` este `Infinity` și `typeof Infinity` este `number`, de ce `typeof 1/0` este `NaN` (Not a Number)? Si dacă asta vi se pare simpla ☺, găsiți alte „problemuțe drăguțe” specifice limbajului JavaScript aici.

In unele contexte, folosirea funcției `eval` este periculoasă (este posibil sa primiți un mesaj de eroare la execuția ei). `eval` în JavaScript face mult mai mult decât să convertească un string la număr, `eval` poate sa evalueze inclusiv o secvență de cod (adică să o execute).

Exemplu:

```
var s = "alert('Hello World')"; // acesta este un string
eval(s);
```

Observații:

- Variabilele pot fi declarate în JavaScript și cu cuvântul rezervat `let`, mai multe despre acesta mai târziu in acest material.
- exista limbaje „derivate" din JavaScript (sau mai degrabă construite peste JavaScript), a căror cod se compilează/translatează în cod JavaScript și care sunt *strongly typed*. Un exemplu in acest sens este TypeScript.

Tablouri in JavaScript

Tablourile in JavaScript sunt de fapt niște obiecte mai speciale. Prezentam mai jos câteva modalități de declarare a acestora:

```
var tari = new Array();
tari[0] = 'Romania';
tari[1] = 'Franta';
// tari.length este 2
tari[6] = 'Germania';
// tari.length este 7, cu 4 elemente ale tabloului tari undefined
var x=[1, 2, 3, 4]; // se declara un Array cu cele 4 elemente
var y = new Array(5, 6, 7, 8);
// la fel, se declara un Array cu cele 4 elemente
var z = new Array(11); // se declara un Array cu 11 elemente, toate undefined
```

Observații:

- lungimea unui tablou (`Array`) poate fi aflată prin intermediul proprietății `.length` a unui array. Atenție, aceasta se comporta ca o dată membră publică (în accepțiunea OOP), nu ca o metodă ce se va invoca cu: `tari.length()` – (paranteză: de fapt proprietatea `.length` este implementată folosind o metodă getter – mai multe detalii despre metodele getter și setter in JavaScript găsiți aici).
- Observații comportamentul diferit a constructorului `new Array()` in funcție de numărul de parametrii. Apelat cu un parametru, `new Array(11)` declară un tablou cu 11 elemente neinițializate (`undefined`), apelat cu mai mulți parametrii, spre exemplu `new Array(5, 6, 7, 8)` declară un `Array` cu 4 elemente inițializate cu valorile specificate;
- elementele unui `Array` pot sa fie de tipuri diferite. Exemplu:

```
var varza = new Array(1, 'Covid-19', false, {x:5, y:9});
```

Elementele tabloului de mai sus sunt de tip `number`, `string`, `boolean` și respetiv `object`.

Tablouri multidimensionale

JavaScript acceptă și tablouri multidimensionale. Spre exemplu, o matrice de 3x3 cu elemente întregi (tablou bidimensional) poate fi declarată astfel:

```
M = new Array(new Array(1, 2, 3), new Array(4, 5, 6), new Array(7, 8, 9));
// M.length va avea lungimea 3
// M[1][1] va avea valoarea 5
```

Elementele unui tablou multidimensional se accesează conform notației clasice folosind indecși numerici din limbajele C/C++/Java (spre exemplu `M[i][j]`). Pe exemplu de mai sus M este de fapt un `array` cu 3 elemente, fiecare element la rândul sau fiind un `array` cu alte 3 elemente.

Funcții uzuale de lucru cu tablouri in JavaScript

API-ul JavaScript oferă o serie de operații ce se pot efectua pe un `Array`. Lista completă a acestora este disponibilă aici. Printre cele mai populare operații (vă recomand totuși să vă uitați peste lista completă) sunt următoarele:

- `push(lista_elemente)` – adaugă un nou element (sau elementele) la sfârșitul unui tablou și returnează noua lungime a acestuia;
- `pop()` – șterge ultimul element dintr-un tablou și returnează elementul șters;
- `shift()` – șterge primul element al unui tablou și returnează acest element;
- `unshift(lista_elemente)` – inserează elementul (sau elementele) la începutul tabloului și returnează noua lungime a acestuia;
- `slice(poz, nrElemente)` – extrage un subsir de `nrElemente` începând cu poziția `poz`. Tablou pe care este apelată nu se modifică;
- `splice(poz, nrElemente, lista_elemente)` – șterge începând de la poziția `poz` un număr de `nrElemente` din tablou, returnând elementele șterse. Inserează pe poziția `poz` elemente din `lista_elemente`. Dacă este apelată doar cu doi parametrii (`listă_elemente` lipsește), șterge doar elementele din tablou (atenție!, spre deosebire de `slice` modifică tabloul pe care este apelată). Daca `nrElemente` este 0 (adică nu se dorește ștergerea niciunui element), `splice` este practic folosită pentru a insera elemente într-un tablou.
- `indexOf(elem)` – returnează indexul primei apariții a elementului in tablou sau -1 dacă acesta nu este găsit;
- `lastIndexOf(elem)` – returnează indexul ultimei apariții a elementului in tablou sau -1 dacă acesta nu este găsit;
- `isArray()` – returnează dacă un obiect este sau nu tablou (`true` sau `false`);
- `forEach(f)` – iterează elementele unui tablou și execută funcția `f` pentru fiecare element al acestuia

- `sort()` – ordonează un tablou. Funcției `sort` i se poate da ca parametru inclusiv o funcție care specifică cum ar trebui să fie comparate 2 elemente ale tabloului (utilă mai ales în situația în care elementele tabloului ce se dorește a fi sortat nu sunt elemente de tip `numeric` sau `string` ci obiecte mai complexe);
- `reverse()` – inversează ordinea elementelor unui tablou.

Obiecte JavaScript

Un obiect in JavaScript poate fi văzut ca o colecție neordonată de date (valori) ce pot avea tipuri diferite, dar împreună au o anumită semantică – spre exemplu "datele" despre o persoană și acțiunile întreprinse de persoana respectivă. Pentru a accesa fiecare dată / valoare din cadrul unui obiect este nevoie și de o cheie, astfel putem privi un obiect și ca o colecție de perechi (cheie, valoare). Este mai natural să ne referim la cheile cu care se accesează datele unui obiect cu numele de atribut-ul obiectului sau proprietatea obiectului, valorile asociate acestora putând fi primitive numerice, boolean-e, string-uri, dar și referințe la alte obiecte, tablouri sau funcții (acestea două din urmă fiind tot obiecte).

Exemplu:

```
var person = {
    name: 'Chuck Norris',
    strength : Infinity,
}
```

Proprietățile obiectului de mai sus sunt `name` și `strength` iar valorile asociate acestora sunt de tip `string`, și `number`. Unui obiect pot să îi fie atribuite proprietăți și mai târziu, astfel putem să-l facem pe Chuck Norris oricând nemuritor:

```
person.immortal = true;
```

Proprietățile unui obiect pot să primească ca și valori funcții, astfel adăugăm metode obiectului respectiv:

```
person.kick = function() {
    this.opponents = null;
}
```

In acest moment, obiectul dat ca exemplu, are un nume (`'Chuck Norris'`), putere (`Infinity`), este nemuritor (are proprietatea `immortal` setată la valoarea `true`) și prezintă o metodă numită `kick` care însă nu a fost apelată. În cadrul acestei metode, `this` (cuvânt rezervat) indică spre obiectul pe care se va apela funcția, proprietatea `opponents` adăugându-i-se acestuia (obiectul încă nu are aceasta proprietate, ea va fi adăugată la apelul metodei).

Dacă Chuck Norris dă cu piciorul, adică dacă se invocă metoda `kick` pe acest obiect:

```
person.kick();
```

obiectul dat ca exemplu va avea o proprietate nouă numită `opponents` cu valoarea `null` (Chuck Norris anihilându-și toți adversarii cum e și normal).

Proprietățile unui obiect pot să fie accesate și folosind o notație de forma (a se observa asemănarea dintre obiecte și Array-uri):

```
person["name"] // va returna 'Chuck Norris'
```

O proprietate pe un obiect poate fi și ștearsă cu (dacă Chuck Norris „o ia in freză”):

```
delete person.immortal
```

Revenind la tablouri, am specificat anterior că acestea sunt tot obiecte. Valorile memorate în cadrul unui obiect de tip tablou putând fi accesate prin intermediul indecșilor numerici (`x[0], x[1], x[2]…`), chiar și o expresie de forma `x["1"]` fiind corectă (nu și una de forma `x.1`).

Puțin mai târziu în cadrul acestui document vom relua discuția despre obiectele JavaScript după ce discutăm de scop global.

Instrucțiuni de control

Instrucțiunile de control `while`, `for`, `if` sunt identice cu cele din limbajele C/C++/Java. Insistăm însă pe o variantă de `for` care permite iterarea elementelor unui tablou sau a proprietăților unui obiect. Spre exemplu, pentru a vedea care sunt proprietățile obiectului `person` declarat mai sus, le putem itera cu (a se rula codul în consola JavaScript din *Developer Tools*):

```
for (i in person)
    console.log(i + ' are valoarea ' + person[i]);
```

Observație: In exemplu de mai sus valorile proprietăților iterate pot fi accesate cu o expresie de forma `person[i]` dar nu cu o expresie de forma `person.i` (aceasta din urmă s-ar referi la o proprietate `i` pe care are avea-o obiectul `person`, proprietate pe care obiectul nu o are).

In același, mod pot fi iterate elementele unui tablou:

```
x = [5, 6, 7];
x[10] = 0;
for (i in x)
    console.log('x[' + i + '] = ' + x[i]);
```

Mai târziu în acest material vom itera proprietățile a două obiecte importante JavaScript: `window` și `document`.

In contextul folosirii dese a unui obiect, se poate folosi instrucțiunea `with` pentru a simplifica codul și a nu repeta folosirea numelui obiectului. Spre exemplu, dacă dorim să vedem câți inamici are Chuck Norris după ce lovește fulgerător, putem folosi:

```
with(person) {
```

```
    kick();
    console.log(opponents);
}
```

In secțiunea de față a prezentului material mai insistăm pe folosirea operatorului de comparație `===` (față de folosirea operatorului `==`). Ambii operatori se folosesc pentru testarea egalității a două valori, dar `===` verifică în plus (pe lângă faptul că cele două expresii sunt evaluate la aceeași valoare) și faptul că cele două valori sunt de același tip. Un astfel de operator este in general specific limbajelor *weakly-typed* (mai este prezent de exemplu în PHP) și nu se regăsește in limbajele *strongly-typed*. Exemplu:

```
if (1 == true) alert('Sunt egale'); else alert ('Nu sunt egale');
// se afișează Sunt egale, pentru că true ca valoare de adevăr este evaluată
// la valoarea numerică 1
if (1 === true) alert('Sunt egale'); else alert ('Nu sunt egale');
// se afișează Nu sunt egale pentru că cele doua expresii au tipuri diferite,
// number, respectiv boolean
```

Obiectul window, scop global, DOM, manipularea DOM-ului

În exemplele date până în prezent în materialul de față am folosit unele funcții precum `alert()` sau obiecte precum `document` care par predefinite. Acestea nu sunt predefinite, ele de fapt există ca funcții membre, respectiv date (proprietăți) membre în cadrul unui obiect global numit `window` care abstractizează fereastra browser-ului. Nu este greșit de exemplu să folosim expresii de forma `window.alert()` sau `window.document`, dar specificarea explicită a obiectului `window` este redundantă.

Un exercițiu interesant este iterarea (folosind forma instrucțiunii `for` prezentată anterior) datelor și funcțiilor membre (proprietățile) ale obiectelor `window` și `document`. Astfel, pentru a realiza un fel de introspecție pe obiectul `window`, puteți încerca în consola JavaScript a browser-ului:

```
for (i in window) console.log(i + '=' + window[i]);
```

Puteți observa pe obiectul `window` existența unor date membre/proprietăți precum `document` sau `outerWidth` precum și a unor funcții membre precum `alert` sau `setTimeout`. Identic, se poate face introspecție pe `document`:

```
for (i in document) console.log(i + '=' + document[i]);
```

Aruncați in mare o privire peste tot ce „are” documentul ca proprietăți. Exemple de proprietăți ale obiectului `document` care ar trebui să vă fie intuitive: `title` sau `location`.

Toate variabilele și funcțiile care se declară in interiorul unui tag `<script>` spunem că sunt declarate in scopul global (ele sunt accesibile de oriunde din JavaScript). De fapt, ele ajung să fie definite ca date membre și funcții membre (proprietăți) ale obiectului `window`. Nu este greșit nici să afirmăm că scopul global in JavaScript (pentru codul care rulează intr-un browser) este reprezentat de obiectul `window`.

Pentru următorul exemplu de cod:

```
<script>
var x = 7;
function f() {
    // do something here
    console.log('Hello');
}
</script>
```

variabila `x` și funcția `f` ajung proprietăți ale obiectului `window`. Acest lucru se poate verifica ușor făcând din nou introspecția la proprietățile acestui obiect:

```
for (i in window) console.log(i + '=' + window[i]);
```

De altfel, aceste proprietăți/metode pot fi referite și cu `window.x` sau `window.f()`.

La nivelul scopului global, cuvântul rezervat `this` va fi referință chiar la obiectul `window`. Acest lucru se poate verifica ușor cu:

```
if (this === window) console.log('True');
```

DOM (Document Object Model), manipularea DOM-ului

După cum am văzut la iterarea proprietăților obiectului `document`, acesta are proprietăți precum `title`, `location`, `head`, `body`. DOM-ul (abreviere de la Document Object Model) reprezintă o structură ierarhică de obiecte construită de către browser pentru a facilita manipularea documentului (a paginii web) din JavaScript.

Exercițiu: care credeți ca este efectul rulării codului de mai jos în consola JavaScript din *Developer Tools*?

```
window.document.body.innerHTML='';
```

După cum am spus și la începutul acestui material, JavaScript permite (prin intermediul DOM-ului) modificarea dinamică a conținutului documentului, crearea de noi elemente (tag-uri) in cadrul pagini, ștergerea unor elemente (tag-uri), modificarea atributelor HTML (adăugarea, ștergerea, schimbarea valorilor) a unor anumite tag-uri, adăugarea/ștergerea de atribute (proprietăți CSS) sau modificarea valorilor atributelor CSS existente, sau permite execuția anumitor funcții la apariția anumitor evenimente. Vom da mai jos câteva exemple pentru toate scenariile înșirate mai sus.

O operație frecventă în JavaScript este obținerea referinței la un element (tag) din pagină pe baza id-ului său, acest lucru realizându-se cu funcția `document.getElementById()`. O astfel de operație este necesară pentru a manipularea din JavaScript a elementului respectiv. Exemplu:

```
<div id="somediv"></div>
<script>
    var mydiv = document.getElementById("somediv");
    mydiv.innerHTML = 'Ana are mere';
</script>
```

Este important în codul de mai sus ca tagul `script` să succeadă tag-ului `div`. Dacă tag-ul `script` ar fi plasat în secțiunea `head` a documentului HTML, e posibil ca efectul să nu fie cel așteptat și în consola JavaScript să aveți un mesaj de eroare legat de faptul ca variabila `mydiv` este `null`. Acest lucru se datorează faptului că in momentul execuției codului JavaScript browserul nu termină de construit DOM-ul (de parsat și încărcat pagina) și `div`-ul `somediv` nu este disponibil încă în DOM.

Va recomand să „vă jucați" și cu funcțiile `document.getElementsByTagName` și `document.getElementsByClassName`.

Proprietatea `innerHTML` este folosită pe un element container (nu se poate folosi pe elemente/tag-uri fără corp, doar pe tag-urile cu marcaj de început și sfârșit de tag) pentru a accesa / modifica conținutul din interiorul tag-ului. Folosind această proprietate se poate seta pe un element container ca și conținut inclusiv cod HTML. Exemplu:

```
<div id='somediv'></div>
<script>
    var mydiv = document.getElementById('somediv');
    mydiv.innerHTML = '<a href="http://www.google.com" id="somelink">Click
here</a>';
    var mylink = document.getElementById('somelink');
    mylink.style.color = '#00FF00';
    mylink.style.backgroundColor = 'red';
</script>
```

In exemplu de mai sus, in `div`-ul `somediv` s-a creat dinamic un tag ancora. Acesta este imediat disponibil in DOM – am dat și exemple de accesare a acestuia și de modificare a stilurilor CSS folosind JavaScript (am făcut la cursul de CSS observația că un atribut CSS ce conține liniuța în denumirea sa, precum `text-align` se transforma in JavaScript in proprietatea `textAlign`).

Crearea unui nou element HTML și integrarea sa in pagina/DOM

Pe lângă exemplul de mai sus în care am creat un nou element în pagina setând ca valoare pentru atributul `innerHTML` a unui container conținut HTML, un element in DOM mai poate fi creat și adăugat folosind metodele `document.createElement()` și `appendChild()`. `document.createElement()` primește ca parametru tag-ul (elementul) care se dorește a fi creat, iar `appendChild()` se apelează pe un container - spre exemplu `container.appendChild(elementnou)`.

Pentru exemplu de mai sus, linia de cod:

```
mydiv.innerHTML = '<a href="http://www.google.com" id="somelink">Click
here</a>';
```

poate fi rescrisă astfel:

```
var mylink = document.createElement('a');
mylink.setAttribute('href', 'http://www.google.com');
mylink.setAttribute('id', 'somelink');
mylink.innerHTML = 'Click here';
```