Univ. Babeș-Bolyai,

Facultatea de Matematică și Informatică

Lect. dr. Darius Bufnea

Notițe de curs Programare Web: jQuery (săptămâna 9 de școală)

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 secţiunile din stânga, mai puţin secţiunea 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 nevoia 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 "filozofiile" jQuery este de a creste 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 librarii/tehnologii web.

Uneori numele fișierului este însoțit 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 înlătura 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 (exemplele 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>
```

Aceste 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 ca 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).

Exemplul 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 arătat 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 practica uzuală 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 înlănțuirea 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 sa 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

Exemplele 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 întrucât 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 „dispară” treptat într-un anumit număr de milisecunde
`fadeOut`	Face ca elementul/elementele selectate să „apară” treptat într-un anumit număr de milisecunde

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

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>

<table 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 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 uzuală 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 termina 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. Evenimentului `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 *shordhand* 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.

```
<div id="mydiv">Ana are mere</div>
<script type="text/javascript">
$("#mydiv").css("color", "red");
</script>
```

```
<script>
$(function() {
    $("#mydiv").css("color", "red");
})
</script>
<div id="mydiv">Ana are mere</div>
```

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”* (fără 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 față, este duplicarea secvenței 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 similara 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.

```
<table cellspacing="0" cellpadding="0" id="tabel">
    <tr><td>…</td></tr>
    <tr><td>…</td></tr>
</table>
<script>
    function init() {
        var table = document.getElementById('tabel');
        for (var i = 0; i < table.rows.length; i++) {
            var row = table.rows[i];
            row.onmouseover = function() {
                selectRow(this);
            };
            row.onmouseout = function() {
                unselectRow(this);
            };
        }
    }
    init();
</script>
```

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 primita 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(functio() {
    $("#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 linii, iar `find("td")` returnează un wrapper jQuery cu toate `td`-urile din cadrul acestei linii. 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 curenta; în timp ce `children` cobora în DOM un singur nivel).

Varianta finală 5 (disponibilă aici)

În această variantă, pe wrapper-ul jQuery construit în jurul fiecărei linii din tabel `$("#tabel tr")` setăm prin metoda `hover` ce primește ca parametru două funcții anonime ce specifică ce să se întâmple la `mouseover` și ce să se întâmple 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 tagurile `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 ca un tablou poate fi iterat după linii, dar nu după coloane.

Tot în arhiva cu exemple din pagină, 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-ul. 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:

- Tablesorter
- Datepicker
- Colorpicker

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