2016 - 2017

Programarea Clientului Web

conf dr. ing. Simona Caraiman

asist drd.inf. Tiberius Dumitriu

mailto: tiberiusdumitriu@yahoo.com

Universitatea Tehnica "Gheorghe Asachi" din Iasi
Facultatea de Automatica si Calculatoare

Curs 1-4 - overview

Internet vs. WEB

Website vs. Web app. vs. Desktop App

Programare (client) Web – Context

Programare (client) Web – Basics

- structura website (static vs. dinamic)
- comunicatia client-server pe Web
  - URI, DNS, HTTP
- dezvoltarea unui site Web
  - responsabilitati client/server
  - web stacks – tehnologii client/server

Curs 1-4 - overview

Web programming - client-side

- Browser-ul Web
  - arhitectura, exemple,
  - extensibilitate: plugins, extensii
- tehnologii
  - HTML

Curs 5+ - preview

Web programming - client-side

- Browser-ul Web
  - arhitectura, exemple,
  - extensibilitate: plugins, extensii
- tehnologii
  - HTML, CSS
  - Javascript
  - XML
  - AJAX
  - Web Workers, Web Storage

Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS)

- definesc **cum se afiseaza** elementele HTML/XML
- reguli stocate in **Style Sheets**
- adaugate in HTML 4.0 pentru a rezolva problema **separarii structurii de prezentare**
- **External Style Sheets** – stocate in **fisiere CSS**
- definitii multiple de stil vor fi **aplicate in cascada**

Cascading Style Sheets (CSS)

- Avantaje:
  - imbunatatirea accesibilitatii continutului
  - control si flexibilitate mai bune in specificarea caracteristicilor prezentationale
  - pagini multiple pot utiliza aceeasi formatare
  - reducerea complexitatii si a repetitiilor in continutul structurat
  - permite utilizarea de stiluri diferite pt. acelasi document markup (in print, on-screen, by voice, tactile)
  - permite afisarea documentului diferit in fct. de dimensiunea display-ului sau de device
  - permite definirea de catre utilizatori a unor stiluri customizate

CSS - standardizare

- CSS Level 1 (Recomm. 1996)
  - Proprietati fonturi, culoare, atribute text (spatiere cuvinte, litere, linii de text), aliniere, margini, borders, pozitionare, identificarea unica si clasificarea generica a grupurilor de atribute
- CSS Level 2 (Recomm. 1998)
  - pozitionare absoluta, relativa, fixa, z-index, tipuri media (aural, speech), text bidirectional, proprietati noi pt. fonturi (shadow)
- CSS Level 2.1 (Recomm. 2011)
  - trateaza erorile din CSS 2 (eliminarea unor elemente non-interoperabile, adaugarea extensiilor la standard deja implementate de browsere)

CSS - standardizare

- CSS 3
  - modularizare (peste 50 module cu diferite status-uri)
    - Selectors
    - Box Model
    - Backgrounds and Borders
    - Text Effects
    - 2D/3D Transformations
    - Animations
    - Multiple Column Layout
    - User Interface
- CSS 4
  - cateva module level 4 (Image Values, Backgrounds & Borders, Selectors, FlexBox)

Cascading Style Sheets (CSS)

Modul gresit de a produce stiluri:

```
<p><font face="Arial">Welcome to Greasy Joe's.
You will <b>never, <i>ever, <u>EVER </u></i></b>
beat <font size="+1" color="red">OUR</font>
prices!</font></p>
```

Welcome to Greasy Joe's. You will **never, *ever, EVER*** beat OUR prices!

- tag-urile de mai sus (`font`, `b`, `i`, `u`) sunt permise in versiuni mai vechi de HTML dar sunt depreciate in XHTML Strict si HTML5

Cascading Style Sheets (CSS)

Sintaxa de baza a unei reguli CSS:

```
selector {
    property: value;
    property: value;
    ...
    property: value;
}
```

```
p {
    font-family: sans-serif;
    color: red;
}
```

- o foaie de stil consta in una sau mai multe **reguli**
- fiecare regula incepe cu un **selector** ce specifica un element HTML caruia ii aplica **proprietati** de stil

Cascading Style Sheets (CSS)

Atasarea unui fisier CSS: <link>

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

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

<link rel="stylesheet" type="text/css"
   href="http://www.google.com/uds/css/gsearch.css" />
```

- tag-ul `<link>` apare in interiorul elementului `<head>`
- se pot lega mai multe foi de stil !

Cascading Style Sheets (CSS)

Incorporarea foilor de stil: <style>

```
<head>
<style type="text/css">
   p { font-family: sans-serif; color: red; }
   h2 { background-color: yellow; }
</style>
</head>
```

Stiluri specificate inline

```
<p style="font-family: sans-serif; color: red;">
This is a paragraph</p>
```

This is a paragraph

Cascading Style Sheets (CSS)

Gruparea stilurilor:

```
p,h1,h2 {
   color: blue;
}
h2 {
   background-color: yellow;
}
```

This paragraph uses the above style.

This heading uses the above style.

- un stil poate selecta mai multe elemente separate prin virgula
- proprietatile specificate vor fi aplicate tuturor elementelor
- elementele pot avea si stiluri individuale (ex. `h2`)

Cascading Style Sheets (CSS)

Selectorul **class**

```
p.right {text-align: right}
p.center {text-align: center}
```

```
<p class="right">This paragraph will be right-aligned.</p>
<p class="center">This paragraph will be center-aligned.</p>
```

- definirea de stiluri diferite pentru acelasi element
- aplicarea mai multor clase aceluiasi element
- aplicarea unui stil pentru toate elementele HTML care au o anumita clasa

Cascading Style Sheets (CSS)

Selectorul **class**

```
p.right {text-align: right}
p.center {text-align: center}
```

```
<p class="right">This paragraph will be right-aligned.</p>
<p class="center">Th                                    ter-aligned.</p>
```

```
<p class="center bold">
This is a paragraph.
</p>
```

- definirea de stiluri diferite pentru acelasi element
- aplicarea mai multor clase aceluiasi element
- aplicarea unui stil pentru toate elementele HTML care au o anumita clasa

Cascading Style Sheets (CSS)

Selectorul **class**

```
p.right {text-align: right}
p.ce
```

```
.center {text-align: center}
```

```
<p c
<p c
```

```
<h1 class="center"> This heading will be center-
   aligned </h1>
<p class="center"> This paragraph will also be
   center-aligned. </p>
```

```
p>
</p>
```

```
</p>
```

- definirea de stiluri diferite pentru acelasi element
- aplicarea mai multor clase aceluiasi element
- aplicarea unui stil pentru toate elementele HTML care au o anumita clasa

Cascading Style Sheets (CSS)

Adaugarea de stiluri elementelor cu un anumit atribut

```
[target] {background-color: blue}
```

```
input[type = text] {background-color: blue}
```

```
[title ~= flower] {background-color: blue}
```

```
[lang| = en] {background-color: blue}
```

Cascading Style Sheets (CSS)

Adaugarea de stiluri elementelor cu un anumit atribut

```
[target] {background-color: blue}
```

```
a[target]
{
background-color:yellow;
}
…
<p>The links with a target attribute get a yellow background:</p>

<a href="http://www.w3schools.com">w3schools.com</a>
<a href="http://www.disney.com" target="_blank">disney.com</a>
<a href="http://www.wikipedia.org"
   target="_top">wikipedia.org</a>
```

Cascading Style Sheets (CSS)

Adaugarea de stiluri elementelor cu un anumit atribut

```
[target] {background-color: blue}
```

```
input[type = text] {background-color: blue}
```

```
a[target=_blank]
{
background-color:yellow;
}
…
<p>The links with a target attribute _blank get a yellow
   background:</p>

<a href="http://www.w3schools.com">w3schools.com</a>
<a href="http://www.disney.com" target="_blank">disney.com</a>
<a href="http://www.wikipedia.org"
   target="_top">wikipedia.org</a>
```

Cascading Style Sheets (CSS)

Advanced use of attribute selectors

```
[lang|=en]
{
background:yellow;
}
…
<p>The elements with the lang attribute’s value starting with en
   get a yellow background:</p>

<p lang="en">Hello!</p>
<p lang="en-us">Hi!</p>
<p lang="en-gb">Ello!</p>
<p lang="us">Hi!</p>
<p lang="no">Hei!</p>
```

```
[lang| = en] {background-color: blue}
```

Cascading Style Sheets (CSS)

Selectorul **id**

```
#green {color: green}

sau

p#para1 {
   text-align: center;
   color: red
}
```

Cascading Style Sheets (CSS)

Selectori contextuali:

```
selector1 selector2 {
   properties
}
```

- se aplica *proprietatile* specificate *selectorului 2* doar atunci cand acesta este specificat in interiorul *selectorului 1*

```
selector1 > selector2 {
   properties
}
```

- se aplica *proprietatile* specificate *selectorului 2* doar atunci cand acesta este specificat *direct* in interiorul *selectorului 1* (*selectorul* 1 este parintele *selectorului* 2)

Cascading Style Sheets (CSS)

Selectori contextuali:

```
selector1 selector2 {
   properties
}
```

```
div p
{
background-color:red;
}
…
<h1>Welcome to My Homepage</h1>

<div>
<h2>My name is Donald</h2>
<p>I live in Duckburg.</p>
</div>

<p>My best friend is Mickey.</p>
```

```
div > p
{
background-color:red;
}
…
<div>
 <h2>My name is Donald</h2>
 <p>I live in Duckburg.</p>
</div>

<div>
 <span><p>I will not be styled.</p></span>
</div>

<p>My best friend is Mickey.</p>
```

```
selector1 > selector2 {
   properties
}
```

- se aplica *proprietatile* specificate *selectorului 2* doar atunci cand acesta este specificat *direct* in interiorul *selectorului 1* (*selectorul* 1 este parintele *selectorului* 2)

Cascading Style Sheets (CSS)

Selectori contextuali:

```
selector1, selector2 {
   properties
}
```

- se aplica *proprietatile* specificate atat *selectorului 1* cat si *selectorului 2*

```
selector1 + selector2 {
   properties
}
```

- se aplica *proprietatile* specificate *selectorului 2* doar atunci cand acesta este plasat imediat dupa *selectorul 1*

```
div + p
{
background-color:red;
}
…
<h1>Welcome to My Homepage</h1>

<div>
<h2>My name is Donald</h2>
<p>I live in Duckburg.</p>
</div>

<p>My best friend is Mickey.</p>

<p>I will not be styled.</p>
```

```
selector1 + selector2 {
   properties
}
```

- se aplica *proprietatile* specificate *selectorului 2* doar atunci cand acesta este plasat imediat dupa *selectorul 1*

Cascading Style Sheets (CSS)

Selectori contextuali (ex.):

```
li strong { text-decoration: underline; }
```

```
<p>Shop at <strong>Carrefour</strong>...</p>
<ul>
<li>The <strong>best</strong> prices in town!</li>
<li>Come check our offer!</li>
</ul>
```

Shop at **Carrefour**...

- The **best** prices in town!
- Come check our offer!

Cascading Style Sheets (CSS)

Selectori contextuali (ex.):

```
div#ad li.important strong { text-decoration: underline; }
```

```
<div id="ad">
<p>Shop at <strong>Carrefour</strong>...</p>
<ul>
<li class="important" >The <strong>best</strong> prices in
   town!</li>
<li>Come check our <strong>offer</strong>!</li>
</ul>
</div>
```

Shop at **Carrefour**...

- The **best** prices in town!
- Come check our **offer**!

Cascading Style Sheets (CSS)

Pseudo-clase:

```
a:link {color: #FF0000}       /* unvisited link */
a:visited {color: #00FF00}    /* visited link */
a:hover {color: #FF00FF}      /* mouse over link */
a:active {color: #0000FF}     /* selected link */
```

- `:active` : an activated or selected element
- `:focus` : an element that has the keyboard focus
- `:hover` : an element that has the mouse over it
- `:link` : a link that has not been visited
- `:visited` : a link that has already been visited
- `:first-child` : an element that is the first child of another

Cascading Style Sheets (CSS)

Noi selectori in CSS 3:

- `:enabled` : every enabled elements
- `:disabled` : every disabled elements
- `:checked` : every checked elements
- `::selection` : portion of an element that is selected by a user
- `:first-of-type` : every element that is the first child, of a particular type, of its parent
- `:last-child` : every element that is the last child of its parent
- `:nth-child(n)` : every element that is the *n*th child, regardless of type, of its parent

Cascading Style Sheets (CSS)

Noi selectori in CSS 3:

```
p:nth-child(2)
{
color:#ff0000;
}

<body>

   <h1>This is a heading</h1>
   <p>The first paragraph.</p>
   <p>The second paragraph.</p>
   <p>The third paragraph.</p>
   <p>The fourth paragraph.</p>

   <p><b>Note:</b> Internet Explorer does not
     support the :nth-child() selector.</p>
</body>
```

- `:nth-child(n)` : every element that is the *n*th child, regardless of type, of its parent

Cascad

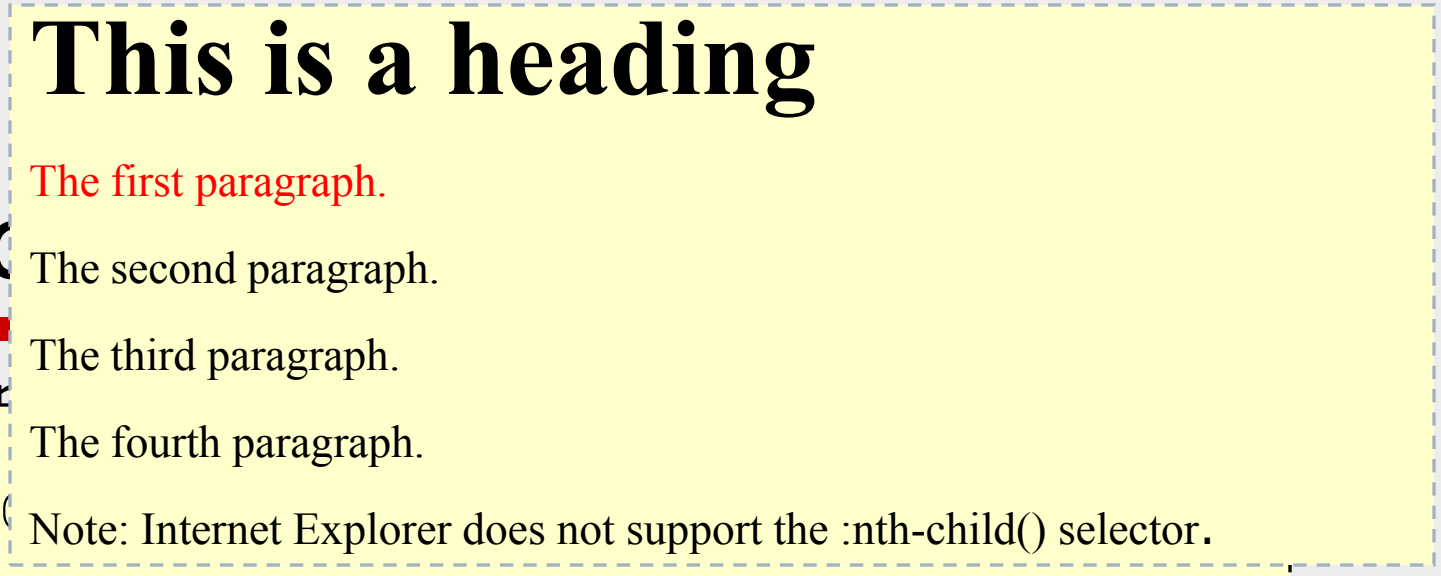
Noi selector

```
p:nth-child
{
color:#ff00
}

<body>

   <h1>This is a heading</h1>
   <p>The first paragraph.</p>
   <p>The second paragraph.</p>
   <p>The third paragraph.</p>
   <p>The fourth paragraph.</p>

   <p><b>Note:</b> Internet Explorer does not
     support the :nth-child() selector.</p>
</body>
```



- `:nth-child(n)` : every element that is the *n*th child, regardless of type, of its parent

Cascading Style Sheets (CSS)

Ordinea de cascadare:

- stilurile implicite ale browser-ului
- foile de stil externe (intr-un tag `<link>`)
- foile de stil interne (intr-un tag `<style>` in header-ul paginii)
- stilurile inline (atributul `style` al unui element HTML)

Cascading Style Sheets (CSS)

Ordinea de cascadare:

Fisier extern:

```
h3 {
color: red;
text-align: left;
font-size: 8pt;
}
```

Foaie de stil interna:

```
h3 {
text-align: right;
font-size: 20pt
}
```

- foile de stil interne (în heder-ul paginii)
- stilurile in[illegible] unui element HTML)

Proprietatile `h3` vor fi:

Cascading Style Sheets (CSS)

Ordinea de cascadare:

Fisier extern:

```
h3 {
color: red;
text-align: left;
font-size: 8pt;
}
```

Foaie de stil interna:

```
h3 {
text-align: right;
font-size: 20pt;
}
```

- foile de stil interne (în heder-ul paginii)
- stilurile in[line] unui element HTML)

Proprietatile `h3` vor fi:

```
color: red;
text-align: right;
font-size: 20pt;
```

Cascading Style Sheets (CSS)

Specificitate

- cand stilurile nu au aceeasi specificitate, ordinea nu mai conteaza!
- *se aplica stilul cel mai specific!*

Calcularea specificitatii unei reguli CSS:

- fiecare selector se situeaza pe unul dintre urmatoarele 4 niveluri:

(1) stiluri imbricate in codul html (ex. `<p style="color: red;">`)
(2) identificatori (ex. `#principal`)
(3) clase, atribute sau pseudo-clase (ex. `.clasa`, `[atribut]`, `:hover`)
(4) elemente si pseudo-elemente (`p`, `:first-letter`)

specificitate(X) = 1000*niv$_{(1)}$ + 100*niv$_{(2)}$ + 10*niv$_{(3)}$ + niv$_{(4)}$
niv$_{(i)}$ – numarul de selectori de pe nivelul *i*

Cascading Style Sheets (CSS)

Specificitate (exemple):

- `div p { ... }` = 2
- `.top { ... }` = 10
- `h3.bottom p.top { ... }` = 22
- `#a1.red { ... }` = 110

Fisier extern:

```
div p {color: red;}
```

Foaie de stil interna:

```
p {color: blue;}
```

```
<div>
<p>Something</p>
</div>
```

Something

Curs 5+ - preview

Web programming - client-side

- Browser-ul Web
  - arhitectura, exemple,
  - extensibilitate: plugins, extensii
- tehnologii
  - HTML, CSS
  - Javascript
  - XML
  - AJAX
  - Web Workers, Web Storage

Creare scenarii la nivel de client

- JavaScript -

JavaScript

- limbaj de programare *interpretat*
- implementare a standardului ECMAScript (v. 5.1, iunie 2011)
  - nu este suportat in aceeasi maniera de toate browserele
- NU are legatura cu Java decat prin denumire si cateva similaritati sintactice
- Utilizare
  - In browser
    - Interactiune cu utilizatorul
    - Controlul browser-ului
    - Comunicatii asincrone
    - Modificarea continutului unui document web
  - In afara browser-ului
    - Documente PDF
    - Componente SO (ex. desktop widgets)
    - Server-side web apps. (ex. Node.js)

JavaScript

Diferente intre Javascript si Java

- interpretat nu compilat
- sintaxa si reguli mai relaxate
  - mai putine tipuri de date si mai “lejere”
  - nu este necesara declararea variabilelor
  - erorile *often silent* (few exceptions)
- constructia cheie o reprezinta functia si nu clasa
- limbaj multi-paradigma (orientat-obiect, imperativ, functional)

JavaScript

Cuvinte cheie

```
abstract boolean break byte case catch char
class const continue debugger default delete
do double else enum export extends false
final finally float for function goto if
implements import in instanceof int interface
long native new null package private
protected public return short static super
switch synchronized this throw throws
transient true try typeof var void volatile
while with
```

JavaScript

Variabile

```
var name = value;
var clientName = "Connie Client";
var age = 32;
var weight = 137.4;
```

- *tipul* nu este specificat, desi JavaScript contine tipuri de date
  - valorile sunt deseori convertite intre tipuri in mod automat, in functie de necesitate (on-the-fly)
- *numele* variabilelor sunt case sensitive
- se declara *explicit* folosind cuvantul cheie `var`
- se declara *implicit* prin asignare (daca i se da o valoare, atunci exista!)

JavaScript

Tipuri de date

- Number
  - dubla precizie (64 biti)
  - Operatii avansate cu numere: obiectul predefinit Math:
    - Math.abs(x) Math.ceil(x) Math.cos(x) Math.exp(x) Math.floor(x) Math.log(x)
- String
  - secventa caractere Unicode (16 biti)
  - sirurile sunt obiecte (“Hello”.length <-> 5)
  - metode pt. siruri: s.charAt(pos) s.concat(s1, ..) s.match(regexp) s.replace(search, replace) s.split(separator, limit) s.substring(start, end) etc
- Boolean
- Object (Function, Array, Date, etc)

JavaScript

Tipuri de date

- Null
  - “nici o valoare”
- Undefined
  - “nici o valoare asignata inca”
- NaN
  - “not a number”

- Valori speciale: Infinity, -Infinity

- NU exista valori intregi
  - convertirea unui sir in numar: parseInt()

```
parseInt(“123”) <->  123

parseInt(“11”, 2) <-> 3
```

JavaScript

Operatori

```
+ - * / % ++ -- = += -= *= /= %= ==
!= > < >= <= && || !
```

- == verifica doar valoarea ("5.0" == 5 este adevarat)
- === verifica si tipul ("5" === 5 este fals)
- precedenta este similara cu cea din Java
- conversia de tip- “on-the-fly”

JavaScript

Operatori

```
+ - * / % ++ -- = += -= *= /= %= ==
!= > < >= <= && || !
```

- ea ("5.0" == 5 este

```
5 < "7" is true
"3" + 4 + 5  <->  345
3 + 4 + "5"  <->  75
```

" === 5 este fals)

a cu cea din Java

- conversia de tip-

```
3 + 4 + Number("5")  <->  12
3 + 4 + (+"5")       <->  12
3 + 4 + ("5"*1)      <->  12
```

JavaScript

Functia `typeof(value)`

- Fiind date declaratiile:
  - `function foo() { alert("Hello"); }`
  - `var a = ["Huey", "Dewey", "Louie"];`
  - `var b;`
- Urmatoarele propozitii sunt adevarate:
  - `typeof(3.14) == "number"`
  - `typeof("hello") == "string"`
  - `typeof(true) == "boolean"`
  - `typeof(foo) == "function"`
  - `typeof(a) == "object"`
  - `typeof(null) == "object"`
  - `typeof(b) == "undefined”`

JavaScript

Controlul executiei

- Testare: **if...else, switch**
- Ciclare: **while, do...while, for**
- Exceptii: **throw, try...catch...finally**

JavaScript

Obiecte JS

- perechi *nume – valoare*
- colectie de *proprietati* avand mai multe *atribute*

Global, Object, Function, String, Date, Array, Boolean, Number, Math, RegExp,

1.
```
var myFruits = new Array();
myFruits[0] = “apple”;
myFruits[1] = “cherry”;
myFruits[2] = “orange”;
```

2.
```
var myFruits = new Array(“apple”, “cherry”, “orange”);
```

3.
```
var myFruits = [“apple”, “cherry”, “orange”];
```

JavaScript

Obiecte JS - creare

1. Instantiere directa

```
var personObj = new Object();
var personObj = {}; //echivalent cu linia anterioara
personObj.firstname = "John";
personObj.lastname = "Doe";
personObj.age = 50;
personObj.eyecolor = "blue";
```

2. Creare sablon al unui obiect

```
function person(firstname,lastname,age,eyecolor)
{
   this.firstname = firstname;
   this.lastname = lastname;
   this.age = age;
   this.eyecolor = eyecolor;
}
myFather = new person("John","Doe",50,"blue");
```

JavaScript

Functii

```
function name(parameterName, ..., parameterName) {
    statements;
}
function quadratic(a, b, c) {
    return -b + Math.sqrt(b*b - 4*a*c) / (2*a);
}
```

- tipul parametrilor si tipul de retur nu sunt specificate
  - `var` **nu apare** la declararea parametrilor
  - functiile fara instructiune de retur returneaza o valoare *undefined*
- orice variabila declarata in interiorul functiei este locala (exista doar in acea functie)

JavaScript

Apelul functiilor

```
name(parameterValue, ..., parameterValue);

var root = quadratic(1, -3, 2);
```

- daca se transmite un numar gresit de parametri:
  - prea multi: parametrii in exces sunt ignorati
  - prea putini: cei netransmisi primesc o valoare nedefinita

JavaScript

Functii

- Argumentele se acceseaza via tabloul ***arguments***

```
function aduna(){
   var suma=0;
   for(var i=0,j=arguments.length; i<j; i++){
       suma += arguments[i];
   }
   return suma;
}
```

JavaScript

Functii

- argumentele functiilor sunt pasate prin *valoare*
  - schimbarea valorii unui argument nu este reflectata global sau in functia apelanta
  - modificarile asupra proprietatilor obiectelor referite sunt vizibile in afara functiei

JavaScript

Functii

```
function myFunction(someObject,
   someOtherObject)
 {
   someObject.brand = "Toyota";
   someOtherObject = {
   brand: "Suzuki",
   model: "Swift",
   year: 2008
   };
 }

var someCar = {
   brand: "Honda",
   model: "Accord",
   year: 1998
 };
```

```
var someOtherCar = {
   brand: "Ford",
   model: "Mondeo",
   year: 2005
 };


alert(someCar.brand); // 'Honda'
alert(someOtherCar.brand);//'Ford'

myFunction(someCar, someOtherCar);

alert(someCar.brand); //'Toyota'
alert(someOtherCar.brand);//'Ford'
```

JavaScript

Functii

- first-class functions: functia poate fi
  - stocata intr-o variabila
  - pasata unei alte functii
  - returnata de o functie – fiind argument pentru return
- Ex.: calcularea greutatii unui animal, dupa formula *greutate=marime*33*

```
var marimi = [17, 20, 7, 14];
var greutati = [ ];
for (var contor = 0; contor < marimi.length; contor++){
      greutati[contor] = marimi[contor] * 33;
}
```

JavaScript

Functii

```
function genereazaTablouGreutati (tablou, calcul) {
   var rezultat = [ ];
   for (var contor = 0; contor < tablou.length; contor++) {
       rezultat[contor] = calcul (tablou[contor])
   }
   return rezultat;
}

function calculGreutate (marime) {
   return marime * 33;
}

var greutati = genereazaTablouGreutati( marimi,
                                        calculGreutate);
```

JavaScript

Functii

- pot fi specificate *functii anonime*
  - o forma de functii imbricate (permit accesul la variabilele definite in scopul functiei continatoare)
  - utilizare: argumente ale altor functii, closures

```
var media = function () {
  // calculul mediei a N numere
  var suma = 0;
  for(var iter=0, lung=arguments.length; iter<lung; iter++){
       suma += arguments[iter];
  }
  return suma/arguments.length;
}
m=media(1,2,3,4);
```

JavaScript

Functii

- pot fi specificate *functii anonime*
  - o forma de functii imbricate (permit accesul la variabilele definite in scopul functiei continatoare)
  - utilizare: **argumente ale altor functii**, closures

```
alert( function(x){
         return x*x;
}(10) );
```

JavaScript

Incapsulare

- JavaScript ofera un singur spatiu de nume, la nivel global
  - conflicte privind denumirea functiilor/variabilelor specificate de programe diferite, concepute de mai multi dezvoltatori
  - nu trebuie afectat spatiul de nume global, pastrându-se codul sursa la nivel privat
  - codul poate fi complet încapsulat via functii anonime care “pastreaza” constructiile la nivel privat

JavaScript

Closures

- Declararea imbricata – ca expresii de tip functie - a functiilor anonime are denumirea ***closures***

```
// specificarea unei expresii de tip functie
( function () {
// variabilele & functiile vor fi vizibile doar aici
// variabilele globale pot fi accesate
}());
```

- via *closures*, simulam metodele private

JavaScript

Closures

```
function makeAdder(x) {
  return function(y) {
    return x + y;
  };
}

var add5 = makeAdder(5);
var add10 = makeAdder(10);

console.log(makeAdder(2)(3)); //5
console.log(add5(2));  // 7
console.log(add10(2)); // 12
```

JavaScript

Closures

```
var cod = (function () {
   var n = 0; // variabila privata
   function start (x) {
        // ... poate accesa 'n'
        // si functiile 'faAia' si 'faCeva'
   }
   function faAia (param) {
        //...invizibila din afara
   }
   function faCeva (x, y) {
        // ...
   }
   return {
        // sunt publice doar functiile 'start' si 'faCeva':
        'start': start,
        'faCeva': faCeva
   }
} ());
cod.start (x); // apelam 'start'
```

JavaScript

Closures

```
var Counter = (function() {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;
  }
  return {
    increment: function() {
      changeBy(1);
    },
    decrement: function() {
      changeBy(-1);
    },
    value: function() {
      return privateCounter;
    }
  };
})();
```

```
/* Alerts 0 */
alert(Counter.value());

Counter.increment();
Counter.increment();

/* Alerts 2 */
alert(Counter.value());

Counter.decrement();

/* Alerts 1 */
alert(Counter.value());
```

JavaScript

Closures

```
var makeCounter = (function() {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;
  }
  return {
    increment: function() {
      changeBy(1);
    },
    decrement: function() {
      changeBy(-1);
    },
    value: function() {
      return privateCounter;
    }
  };
});
```

```
var Counter1 = makeCounter();
var Counter2 = makeCounter();

/* Alerts 0 */
alert(Counter1.value());

Counter1.increment();
Counter1.increment();

/* Alerts 2 */
alert(Counter1.value());

Counter1.decrement();

/* Alerts 1 */
alert(Counter1.value());
/* Alerts 0 */
alert(Counter2.value());
```

JavaScript

Observatii

- Totul in Javascript este obiect (chiar si functiile)
- Toate proprietatile si metodele unui obiect sunt disponibile oriunde (*public scope*)
- Nu exista vizibilitate la nivel de bloc de cod (*block scope*), ci doar la nivel global ori la nivel de functie
- Functiile ascund orice e definit in interiorul lor
- Accesorul *this* este relativ la contextul executiei, nu al declararii

JavaScript

Observatii

- Totul in Javascript este obiect (chiar si functiile)
- Toate proprietatile si metodele unui obiect sunt disponibile oriunde (*public scope*)
- Nu exista vizibilitate la nivel de bloc de cod (*block scope*), ci doar la nivel global ori la nivel de functie
- Functiile ascund orice e definit in interiorul lor
- **Accesorul *this* este relativ la contextul executiei, nu al declararii**

JavaScript *this*

- refera obiectul la care este atasata/legata functia in care este utilizat
  - contine valoarea acestui obiect

```
var person = {
    firstName: "Penelope",
    lastName: "Barrymore",
    fullName: function () {

        console.log(this.firstName + " " + this.lastName);
        // We could have also written this:
        console.log(person.firstName + " " + person.lastName);
    }
}
```

JavaScript *this*

- refera obiectul la care este atasata/legata functia in care este utilizat
  - contine valoarea acestui obiect

```
function turnBlue(e){
  this.style.backgroundColor = '#A5D9F3';
}

//Get a list of every element in the document
var elements = document.getElementsByTagName('*');

//Add turnBlue as a click listener so when the element is clicked on,
//it turns blue
for(var i=0 ; i<elements.length ; i++){
  elements[i].addEventListener('click', turnBlue, false);
}
```

JavaScript *this*

- global scope (browser)
  - all global variables and functions are defined on the *window* object
  - *this* se refera la obiectul *window*
    - exceptie: strict mode

```
var firstName = "Peter", lastName = "Ally";

function showFullName () {
// "this" inside this function will have the value of the window object
// because the showFullName () function is defined in the global scope,
// just like the firstName and lastName
    console.log (this.firstName + " " + this.lastName);
}

var person = {
    firstName   :"Penelope",
    lastName    :"Barrymore",
    showFullName:function () {
    // "this" on the line below refers to the person object, because the
   showFullName function will be invoked by person object.
    console.log (this.firstName + " " + this.lastName);
    }
}

showFullName (); // Peter Ally

// window is the object that all global variables and functions are
   defined on, hence:
window.showFullName (); // Peter Ally

// "this" inside the showFullName () method that is defined inside the
   person object still refers to the person object, hence:
person.showFullName (); // Penelope Barrymore
```

JavaScript *this*

Principiu:

- *this* primeste o valoare doar in momentul in care un obiect invoca functia in care este definit
- *this* primeste valoarea obiectului care invoca

EXCEPTII:

1. la pasarea unei functii (care foloseste *this*) ca functie callback
2. cand *this* este utilizat intr-un closure
3. la asignarea unei functii (care foloseste *this*) unei variabile
4. la imprumutarea unei metode (care foloseste *this*)

JavaScript *this*

1. la pasarea unei functii (care foloseste *this*) ca functie callback

```
// We have a simple object with a clickHandler method that we want to use
   when a button on the page is clicked
var user = {
    data:[{name:"T. Woods", age:37},
          {name:"P. Mickelson", age:43}],
    clickHandler:function (event) {
    var randomNum = ((Math.random () * 2 | 0) + 1) - 1;

   // Prints a random person's name and age from the data array
    console.log (this.data[randomNum].name + " " +
   this.data[randomNum].age);
   }
}
// The button is wrapped inside a jQuery $ wrapper, so it is now a jQuery
   objectand the output will be undefined because there is no data
   property on the button object
$ ("button").click (user.clickHandler); // Cannot read property '0' of
   undefined
```

JavaScript *this*

1. la pasarea unei functii (care foloseste *this*) ca functie callback - FIX

```
// We have a simple object with a clickHandler method that we want to use
   when a button on the page is clicked
var user = {
    data:[{name:"T. Woods", age:37},
          {name:"P. Mickelson", age:43}],
    clickHandler:function (event) {
    var randomNum = ((Math.random () * 2 | 0) + 1) - 1;

   // Prints a random person's name and age from the data array
    console.log (this.data[randomNum].name + " " +
   this.data[randomNum].age);
   }
}
// The button is wrapped inside a jQuery $ wrapper, so it is now a jQuery
   objectand the output will be undefined because there is no data
   property on the button object
$("button").click (user.clickHandler.bind(user)); // P. Mickelson 43
```

JavaScript *this*

2. cand *this* este utilizat intr-un closure

```
var user = {
   tournament:"The Masters",
   data      :[{name:"T. Woods", age:37},
               {name:"P. Mickelson", age:43}],
   clickHandler:function () {
   // the use of this.data here is fine, because "this" refers to the
   user object, and data is a property on the user object.

    this.data.forEach (function (person) {
    // But here inside the anonymous function (that we pass to the
    // forEach method), "this" no longer refers to the user object.
    // This inner function cannot access the outer function's "this"
        console.log ("What is This referring to? " + this);
                //[object Window]
        console.log (person.name + " is playing at " + this.tournament);
                // T. Woods is playing at undefined
                // P. Mickelson is playing at undefined
    })     }      }
user.clickHandler(); // What is "this" referring to? [object Window]
```

JavaScript *this*

2. cand *this* este utilizat intr-un closure - FIX

```
var user = {
   tournament:"The Masters",
   data      :[{name:"T. Woods", age:37},
               {name:"P. Mickelson", age:43}],
   clickHandler:function () {
// To capture the value of "this" when it refers to the user object, we
   have to set it to another variable here:
// We set the value of "this" to theUserObj variable, so we can use it
   later
    var theUserObj = this;
    this.data.forEach (function (person) {
    // Instead of using this.tournament, we now use theUserObj.tournament
    console.log (person.name + " is playing at " +
        theUserObj.tournament);
})    }    }

user.clickHandler();    // T. Woods is playing at The Masters
```

JavaScript *this*

3. la asignarea unei functii (care foloseste *this*) unei variabile

```
// This data variable is a global variable
var data = [{name:"Samantha", age:12},
            {name:"Alexis", age:14}];
var user = {
// this data variable is a property on the user object
    data    :[{name:"T. Woods", age:37},
              {name:"P. Mickelson", age:43}],
    showData:function (event) {
        var randomNum = ((Math.random () * 2 | 0) + 1) - 1;
        console.log (this.data[randomNum].name + " " + this.data
        [randomNum].age);
    } }
// Assign the user.showData to a variable
var showUserData = user.showData;
// When we execute the showUserData function, the values printed to the
   console are from the global data array, not from the data array in the
   user object
showUserData (); // Samantha 12 (from the global data array)
```

JavaScript *this*

3. la asignarea unei functii (care foloseste *this*) unei variabile - FIX

```
// This data variable is a global variable
var data = [{name:"Samantha", age:12},
            {name:"Alexis", age:14}];
var user = {
// this data variable is a property on the user object
    data    :[{name:"T. Woods", age:37},
              {name:"P. Mickelson", age:43}],
    showData:function (event) {
        var randomNum = ((Math.random () * 2 | 0) + 1) - 1;
        console.log (this.data[randomNum].name + " " + this.data
        [randomNum].age);
    } }
// Assign the user.showData to a variable
var showUserData = user.showData.bind(user);
// When we execute the showUserData function, the values printed to the
   console are from the global data array, not from the data array in the
   user object
showUserData (); // Samantha 12 (from the global data array)
```

JavaScript *this*

4. la imprumutarea unei metode (care foloseste *this*)

```
// We have two objects. One of them has a method called avg () that the
   other doesn't have, so we will borrow the (avg()) method
var gameController = {
    scores  :[20, 34, 55, 46, 77],
    avgScore:null,
    players :[{name:"Tommy", playerID:987, age:23},
              {name:"Pau", playerID:87, age:33}]}
var appController = {
    scores  :[900, 845, 809, 950],
    avgScore:null,
    avg     :function () {
        var sumOfScores = this.scores.reduce (function (prev, cur, index,
   array) { return prev + cur;   });
        this.avgScore = sumOfScores / this.scores.length;
    }    }
//If we run the code below, the gameController.avgScore property will be
   set to the average score from the appController object "scores" array
gameController.avgScore = appController.avg();
```

JavaScript *this*

4. la imprumutarea unei metode (care foloseste *this*) - FIX

```
//Use the apply() method that this inside the appController.avg () method
   refers to gameController

// the 2nd argument has to be an array – the arguments to pass to the
   appController.avg () method.
appController.avg.apply(gameController, gameController.scores);

// The avgScore property was successfully set on the gameController
   object, even though we borrowed the avg () method from the
   appController object
console.log (gameController.avgScore); // 46.4

// appController.avgScore is still null; it was not updated, only
   gameController.avgScore was updated
console.log (appController.avgScore); // null
```

JavaScript in browser

- Programe Javascript rulate in navigatorul Web via un *script engine*
- are acces la
  - arborele DOM (*Document Object Model*) corespunzator documentului HTML
  - diverse obiecte oferite de mediul de executie pus la dispozitie de *browser*
    - *e.g.*, informatii privind contextul rularii (caracteristici ale navigatorului, latenta retelei), istoricul navigarii, fereastra de redare a continutului,..

Ce poate face un JavaScript?

- **inserare text dinamic intr-o pagina HTML**
- **reactie la evenimente**
- **citire/scriere elemente HTML**
- **validare date**
- **detectare browser**
- **creare *cookies***

JavaScript

Inserare JavaScript in HTML

- codul Javascript poate fi adaugat intr-o pagina web in trei moduri:
  1. in sectiunea body a paginii
  2. in antetul paginii
  3. intr-o legatura catre un fisier script extern (.js)

JavaScript

Exemplu – in HTML `body`

```
<body>
    ...
    <script type="text/javascript"> Javascript code
    </script>
    ...
</body>
```

- util pentru generarea textului dinamic

JavaScript

Exemplu – in HTML `head`

```
<head>
    ...
    <script type="text/javascript"> Javascript code
    </script>
    ...
</head>
```

- util pentru actiunile activate de evenimente
  - pop up alert message (ex. cand utilizatorul apasa un buton)
  - afisarea unui mesaj de intampinare la refresh-ul paginii

JavaScript

Exemplu – legarea unui fisier JavaScript

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

- poate fi plasat in sectiunile `head` sau `body` ale paginii web
- script-ul este stocat intr-un fisier `.js`

JavaScript in browser

Accesarea/modificarea arborelui DOM

- via obiectul ***document***
  - documentElement, getElementById(identificator), parentNode, nextSibling, previousSibling, childNodes, firstChild, lastChild, atributes
- informatii referitoare la nodurile arborelui DOM
  - nodeType, nodeValue, innerHTML, getAttribute(numeAtribut)
- modificarea structurii arborelui DOM
  - createElement(element), createTextNode(nod), appendChild(nod), removeChild(nod), cloneChild(), setAttribute(atribut, valoare)

JavaScript in browser

Injectarea dinamica a textului: `document.write()`

```
document.write("<p>message</p>");
```

- insereaza textul specificat in pagina web
- poate fi utilizat pentru a insera elemente HTML
- argumentul poate fi un sir de caractere intre ghilimele sau o variabila

JavaScript in browser

Tratarea evenimentelor

```
<h2 onclick="myFunction();">Click me!</h2>
```

- elementele HTML au atribute speciale denumite *evenimente*
- Functiile Javascript pot fi utilizate pentru tratarea evenimentelor
  - functia se va executa la interactiunea cu elementul respectiv
  - onclick este doar un exemplu de atribut HTML de tip eveniment

JavaScript in browser

Tratarea evenimentelor

- tratarea standardizata a evenimentelor
  - specificatia *DOM Level 2 Events*

```
obiect.addEventListener("eveniment", functie, mod);
```

JavaScript in browser

Event flow

- Elemente imbricate pt care se trateaza acelasi eveniment (ex. *click*)
  - Event capturing
    - Trigger elements from outer to inner (Netscape)
  - Event bubbling
    - Trigger elements from inner to outer (IE)

JavaScript in browser

Tratarea evenimentelor

- proprietati asociate evenimentelor privind actiunile mouse-ului
  - click, mousedown, mouseup, mouseover, mousemove, mouseout
- proprietati asociate evenimentelor vizand tastatura
  - keyup, keydown, keypress
- evenimente referitoare la interactiunea cu navigatorul
  - load, unload, select, change, submit, focus, blur, resize, scroll
- evenimente privitoare la modificarea arborelui DOM (*mutation events*)
  - DOMSubtreeModified, DOMNodeInserted, DOMNodeRemoved, DOMAttrModified, DOMCharacterDataModified, DOMNodeInsertedIntoDocument, DOMNodeRemovedFromDocument

JavaScript in browser

Tratarea evenimentelor

- proprietati utile ale obiectului ***Event***:
  - type (ex. "click", "load", "scroll", "submit")
  - currentTarget - indica nodul care trateaza evenimentul
  - target - desemneaza nodul asupra caruia evenimentul a fost declansat initial
  - bubbles - indica daca evenimentul se propaga spre elemente ascendente (valoarea *true*) ori catre descendenti (valoarea *false*)
  - cancelable - precizeaza daca evenimentul poate fi intrerupt

- eliminarea tratarii unui eveniment
  - removeEventListener()

JavaScript in browser

Tratarea evenimentelor

- evenimente tactile (*touch events*):
  - se extinde DOM cu concepte precum zona tactila (interfata Touch) ce poate emite evenimente (interfata TouchEvent) de tip
    - touchstart, touchend, touchmove, touchcancel
- evenimente nestandardizate inca:
  - cut, copy, paste
- evenimente specificate in cadrul HTML5
  - conectivitatea la retea: online, offline
  - interactiunea cu utilizatorul: redo, undo, drag, drop, mousewheel, contextmenu, pagehide, pageshow,...
  - starea dispozitivului – deviceorientation, devicemotion
  - utilizarea imprimantei – beforeprint, afterprint
  - ...etc

More Scripting

Intrebare:

Cum vreti sa scrieti codul JavaScript?

(I) Plug & Play

- adauga un “calendar widget” sau un “autocomplete”
  - experienta JavaScript necesara: putina sau deloc
  - doar se customizeaza niste optiuni si “gata”
  - flexibilitate zero

(II) Asamblare

- Scrierea de utilitare comune
  - incarcarea unei pagini via Ajax
  - construirea unui meniu dinamic
  - crearea de formulare interactive
- Folosirea *codului prefabricat* pentru suport cross-browser
- Flexibil (pana la intalnirea unui bug)

(III) “Down-and-Dirty”

- Tot codul JavaScript este scris de la zero
- Tratarea directa a bug-urilor din browsere
- Excesiv de flexibil

Cum vreti sa scrieti codul JavaScript?

- Widgets
- Biblioteci
- JavaScript brut

Cum vreti sa scrieti codul JavaScript?

- Widgets
- **Biblioteci**
- JavaScript brut

De ce sa folosim o biblioteca?

- scrierea codului JavaScript devine suportabila
- procesul este mai rapid
- simplifica suportul cross-browser

- ex: *stdlib* in C

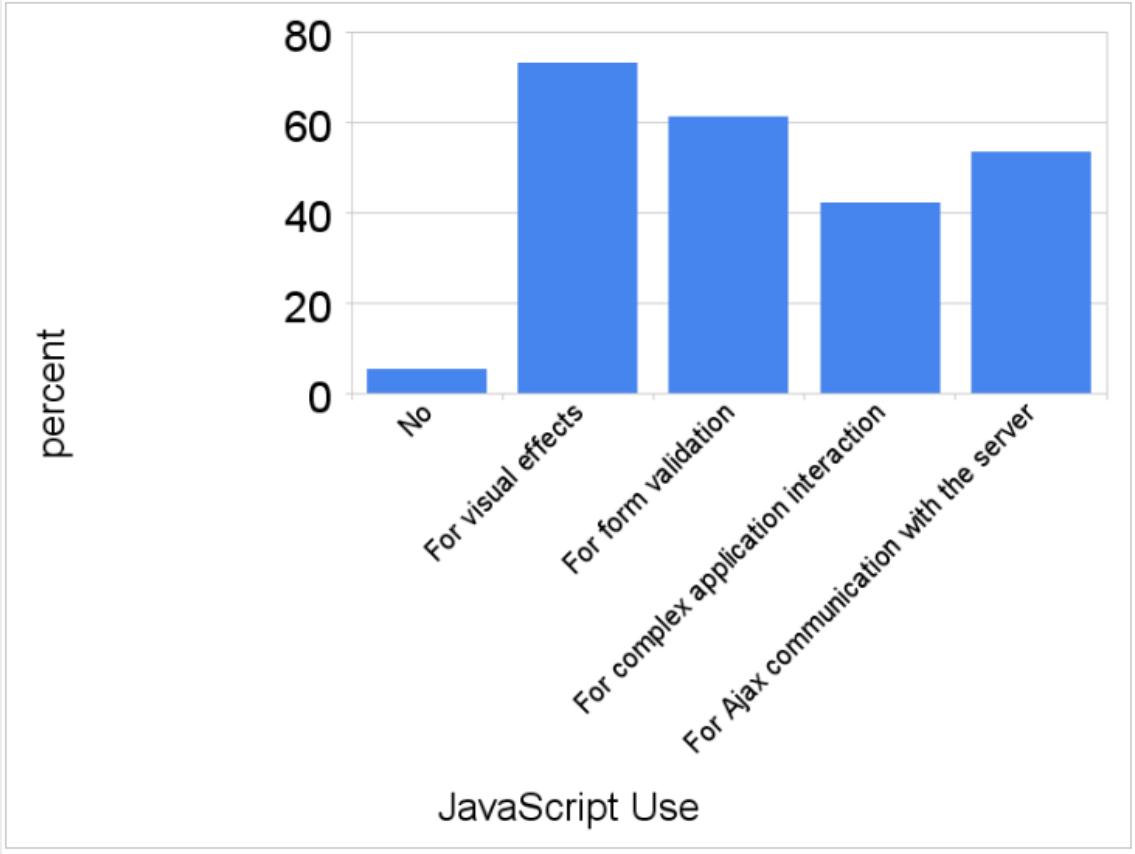
Most popular..

Prototype
JQuery
Yahoo UI
Dojo
Mootools
Ext JS
..

Ajaxian Survey

http://ajaxian.com/archives/state-of-the-web-2008

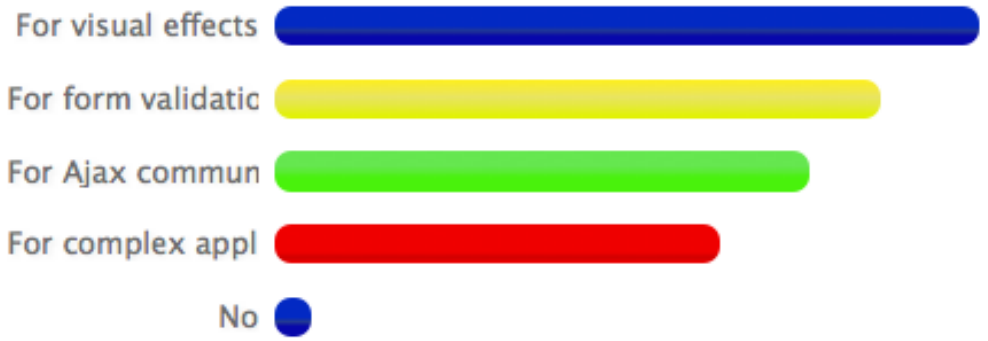
“Do you use JavaScript in your development?”



Ajaxian Survey
(2010)

“Do you use JavaScript in your development?”

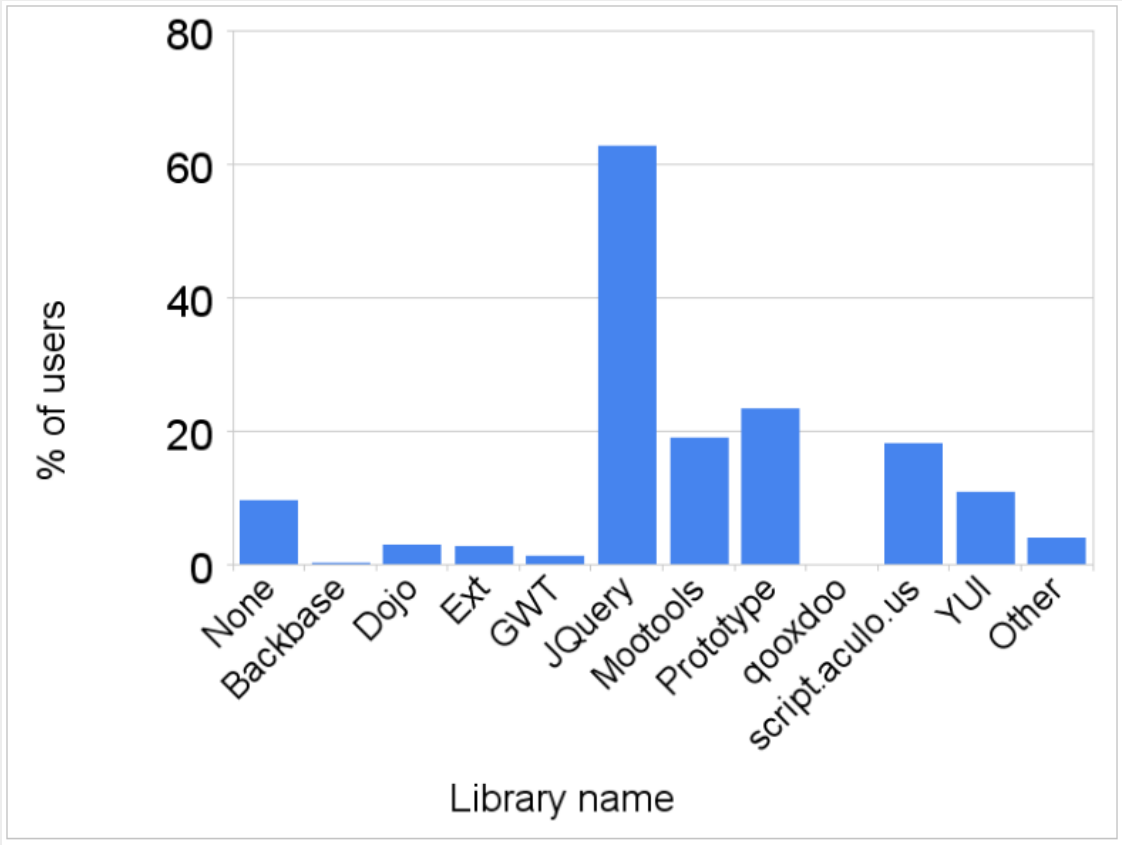
ANSWER	COUNT	%
For visual effects	1077	76.82%
For form validation	926	66.05%
For Ajax communication with the server	818	58.35%
For complex application interaction	683	48.72%
No	56	3.99%



Ajaxian Survey

http://ajaxian.com/archives/state-of-the-web-2008

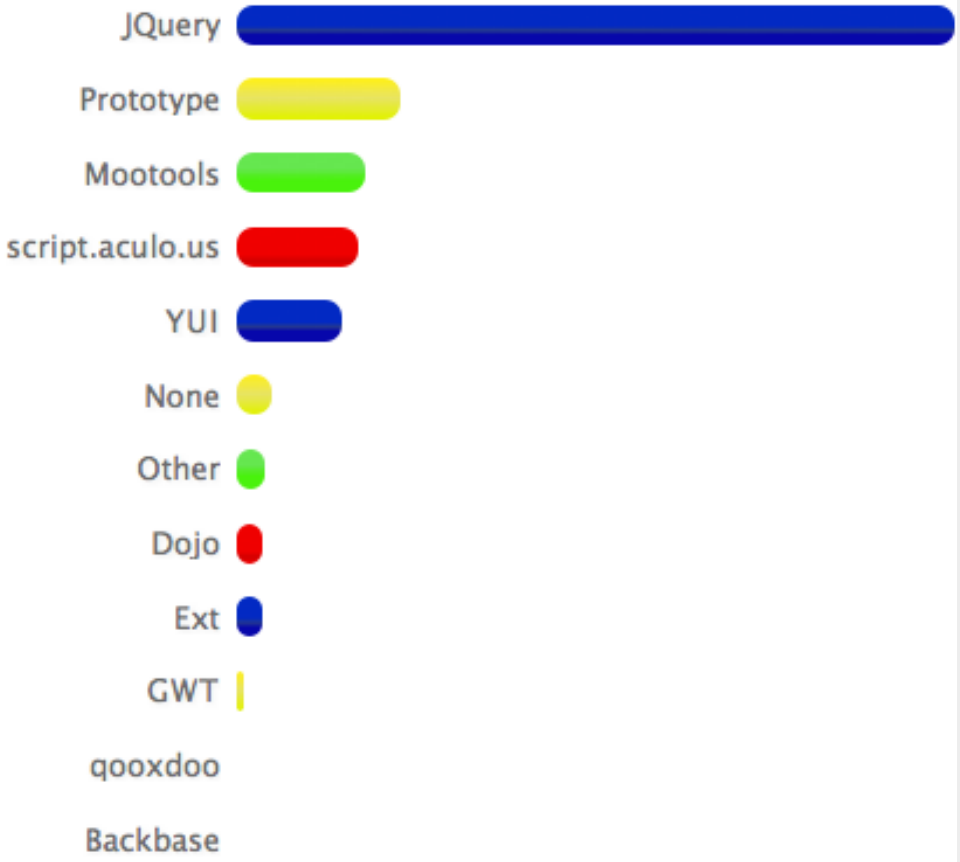
“What JavaScript libraries and frameworks do you use?”



Ajaxian Survey
(2010)

“What JavaScript libraries and frameworks do you use?”

FRAMEWORK	COUNT	%
JQuery	1091	77.82%
Prototype	249	17.76%
Mootools	196	13.98%
script.aculo.us	187	13.34%
YUI	161	11.48%
None	55	3.92%
Other	43	3.07%
Dojo	41	2.92%
Ext	39	2.78%
GWT	13	0.93%
qooxdoo	2	0.14%
Backbase	1	0.07%



Google Trends

Prototype JQuery YUI Dojo Mootools

http://www.google.com/trends?q=prototype+javascript%2C+jquery+javascript%2C+yahoo+ui+javascript%2C+dojo+javascript%2C+mootools+javascript&ctab=0&geo=all&date=all&sort=0

Angular JS

https://www.airpair.com/angularjs/posts/jquery-angularjs-comparison-migration-walkthrough

	jQuery	AngularJS
Abstracts the DOM	✓	✓
Unit Test Runner	✓	✓
Deferred Promises	✓	✓
Cross-Module Communication	✓	✓
Animation Support	✓	✓
AJAX / JSONP	✓	✓
RESTful API	✗	✓
Integration Test Runner	✗	✓
MVC Pattern Support	✗	✓
Templating	✗	✓
Two-way Data Binding	✗	✓
Dependency Management	✗	✓
Deep-Link Routing	✗	✓
Form Validation	✗	✓
Localization	✗	✓
File Size	32KB	38KB

Comparatie

Criterii:

- Code base
- Dezvoltare
- Documentatie
- Comunitate

Code base

- Functionalitatea de baza
  - DOM
  - Evenimente
  - Ajax
  - Animatii
- Widgets (interfete cu utilizatorul)

Code base

- Functionalitatea de baza
  - DOM
  - Eve
  - Aja
  - Animatii
- **Widgets (interfete cu utilizatorul)**

- componente uzuale:

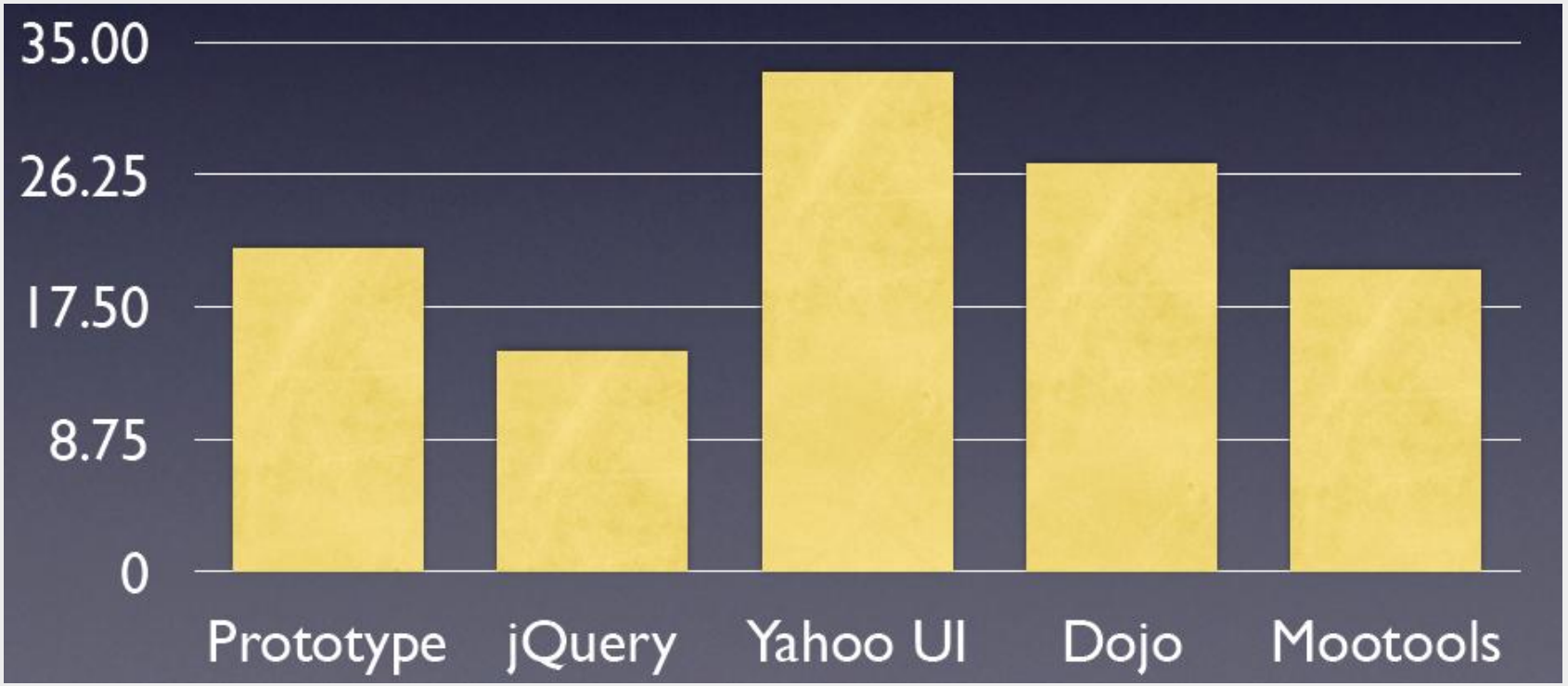
Drag & Drop, Grid, Modal Dialog, Calendar, Slider, Menu/Toolbar…

- Prototype – Script.aculo.us
- jQuery – jQuery UI
- Dojo – Dijit
- incluse in Yahoo UI
- capabilitati de “themeing”: jQuery, YUI, Dojo

Dezvoltare

- Open licensing
- Browser support
  - Toate suporta IE6+, Firefox 2+, Safari 2+, Opera 9+, Chrome 1+
  - Exceptii:
    - Prototype suport pentru Opera v. 9.25+
    - Dojo a renuntat la suportul pentru Safari 2

File size (kb)


35.00
26.25
17.50
8.75
0
Prototype
jQuery
Yahoo UI
Dojo
Mootools

Popularitate

- Cine ce foloseste:
  - *Prototype*: Apple, CNN, NBC, ESPN, Amazon
  - *jQuery*: Google, Dell, CBS, mozilla.org, Amazon
  - *Yahoo UI*: Yahoo, LinkedIn
  - *Dojo*: IBM, AOL, Shopping.com

Bibliografie

- ***, CSS Tehnici esentiale - Invata prin exemple practice; edit 3 D Media Communications

- Jim Keogh JavaScript fara mistere - ghid pentru autodidacti

http://www.elefant.ro/fragment?f=120120133636-fed79ee1e0f94611a3dab021934835aa

- ***. note de curs:

http://laurian.ro/wordpress/wp-content/uploads/2013/JavaScript_a5.pdf

- http://www.lec-academy.ro/10-car%C8%9Bi-gratuite-despre-limbajul-de-programare-java/

- http://www.w3schools.com