

# STANDARDNI KORISNIČKI INTERFEJSI

Predavanje broj: 10

Nastavna jedinica: JavaScript

Nastavne teme:

JSON. Objekti. Funkcije. Predefinisane metode. Unutrašnja funkcija. Closure. DOM. Baratanje elementima stranice. Nalaženje HTML elementa. Menjanje stila. Postavljanje align-contenta. Kreiranje elemenata. Animacija (backfaceVisibility). Događaji (dodeljivanje, propagacija, oslušivači). Interfejs fullscreen.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

J. D. Gauchat, "Integrisane tehnologije za izradu WEB strana", Mikroknjiga, Beograd, 2014.

W3C Tutorials, Internet, 2014

# JavaScript

- Paziti, JavaScript NE kreira novi opseg za svaki blok koda. Sledeći primer ispisuje 10.

```
<!DOCTYPE html>
<html><body>
<p id="demo"></p>
<script>
for (var i = 0; i < 10; i++) {
    // some code
}
document.getElementById("demo").innerHTML = i;
</script>
</body></html>
```

Mala ubrzanja koda:

```
sporiji: for (i = 0; i < arr.length; i++) }
brži:    l = arr.length;
         for (i = 0; i < l; i++) {
```

- Izbegavajte korišćenje ključne reči **with**. Negativno utiče na brzinu izvršavanja.
  - U strict modu nije dozvoljeno korišćenje **width**.

# JavaScript

- Ako se skript smesti na dno stranice, browser će prvo da učita stranicu pre nego počne izvršavanje skripta.
- Za vreme dok se skript učitava, browser neće startovati bilo koje drugo učitavanje.
  - Svo parsiranje i rendering mogu biti blokirani.
- HTTP specifikacije definišu da browser ne bi trebalo da učitava više od dve komponente paralelno.
- Alternativa je da se u script tagu koristi atribut **defer**="true".
  - Defer atribut specificira da bi skript trebalo da se izvrši nakon što je završeno parsiranje stranice, ali to radi samo za eksterne skriptove.
- Poziv funkcije nakon što je stranica učitana.

```
<script>
window.onload = downScripts;
function downScripts() {
    var element = document.createElement("script");
    element.src = "myScript.js";
    document.body.appendChild(element);
}
</script>
```

# JSON

- JSON: JavaScript Object Notation:
  - je sintaksa za razmenu podataka.
  - se često koristi kada se podaci šalju sa servera na web stranicu.
  - je "*lightweight*" format za razmenu podataka
  - ne zavisi od programskog jezika.
  - samoopisan je ("self-describing") i lak za razumevanje

- JSON primer:

```
{"employees":[  
  {"firstName":"John", "lastName":"Doe"},  
  {"firstName":"Anna", "lastName":"Smith"},  
  {"firstName":"Peter", "lastName":"Jones"}  
]}
```

- Sintaksna pravila:
  - podaci su u obliku parova "name":value ;
  - podaci su odvojeni zarezom;
  - velike zagrade sadrže objekte;
  - srednje zagrade sadrže nizove (arrays)

# Konverzija JSON teksta u JavaScript objekat

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>Create Object from JSON String</h2>
```

```
<p id="demo"></p>
```

```
<script>
```

```
var text = '{"employees":[" +  
'{"firstName":"John","lastName":"Doe" },' +  
'{"firstName":"Anna","lastName":"Smith" },' +  
'{"firstName":"Peter","lastName":"Jones" }]]}';
```

```
obj = JSON.parse(text);
```

```
document.getElementById("demo").innerHTML =
```

```
obj.employees[1].firstName + " " + obj.employees[1].lastName;
```

```
</script>
```

```
</body>
```

```
</html>
```

- Moglo bi i:      `obj.employees[1]["firstName"]`

# JSON

- JSON vrednosti mogu biti:
  - Boolean (true ili false)
  - broj (integer ili floating point)
  - string (u duplim navodnicima)
  - niz (array, u [] zagradama)
  - objekat (u {} zagradama)
  - null
- **Nemojte** koristiti eval za konverziju iz JSON u JavaScript objekat, jer eval kompajlira i izvršava JavaScript kod (opasno sa stanovišta sigurnosti):

```
<!DOCTYPE html>
<html><body><p id="demo"></p><script>
var txt = '{"employees":[" +
'{"firstName":"John","lastName":"Doe" },' +
'{"firstName":"Anna","lastName":"Smith" },' +
'{"firstName":"Peter","lastName":"Jones" }]]}';
var obj = eval ( "(" + txt + ")" );
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script></body></html>
```

# JSON Http Request

```
<!DOCTYPE html>
<html><body>  <div id="id01"></div>  <script>
  var xmlhttp = new XMLHttpRequest();
  var url = "mojipodaci.txt";
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      var myArr = JSON.parse(xmlhttp.responseText);
      myFunction(myArr);
    }
  }
  xmlhttp.open("GET", url, true);
  xmlhttp.send();
  function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i < arr.length; i++) {
      out += '<a href="' + arr[i].url + '>' +
        arr[i].display + '</a><br>';
    }
    document.getElementById("id01").innerHTML = out;
  }
</script></body></html>
```

# Datoteka mojipodaci.txt

- Sadržaj datoteke mojipodaci.txt je niz objekata čiji su atributi display i url sa pripadnim vrednostima.

```
[  
{  
  "display": "JavaScript",  
  "url": "js/index.html"  
},  
{  
  "display": "HTML5",  
  "url": "html5/index.html "  
},  
{  
  "display": "CSS3",  
  "url": "css3/index.html "  
}  
]
```



# JSON datoteke

- Ideja je da je u datoteci mojafunkcija.js napravljen samo poziv funkcije myFunction sa stvarnim argumentom, nizom objekata kao literalom:

```
myFunction( [{"display": "JavaScript","url": "js/index.html"},  
            {"display": "HTML5","url": "html5/index.html " },  
            {"display": "CSS3","url": "css3/index.html "}]  
            );
```

- Stranica koja učitava datoteku mojafunkcija.js je sada kao što sledi:

```
<!DOCTYPE html>  
<html><body><div id="id01"></div><script>  
function myFunction(arr) {  
    var out = "";    var i;  
    for(i = 0; i<arr.length; i++) {  
        out += '<a href="' + arr[i].url + '">' +  
            arr[i].display + '</a><br>';  
    }  
    document.getElementById("id01").innerHTML = out;  
}  
</script>  
<script src="mojafunkcija.js"></script>  
</body></html>
```

# 0 objektima

- Ranije su pomenuti objekti u JavaScriptu sa svojstvima i ponašanjem (metodama).
- Objekti se mogu kreirati kao što sledi:

- korišćenjem objekta kao literala;

```
var person = { firstName:"John", lastName : "Doe", age:50,  
              eyeColor:"blue" };
```

- korišćenjem ključne reči new;

```
var person = new Object();  
person.firstName = "John"; ... person.eyeColor = "blue";
```

- definisanjem konstruktora a onda kreiranjem konkretnog objekta;

```
class Point { constructor(x, y) { this.x = x; this.y = y; }  
              static distance(a, b) { var dx = a.x - b.x;  
                                     var dy = a.y - b.y;  
                                     return Math.hypot(dx, dy); }  
}
```

```
p1 = new Point(5, 5);
```

```
p2 = new Point(10, 10);
```

```
odabranidiv.innerHTML = ""+ Point.distance(p1, p2);
```

# 0 objektima

```
function person(first, last, age, eye) {  
    this.firstName = first;  
    this.lastName  = last;  
    this.age       = age;  
    this.eyeColor  = eye;  
}  
var myFather = new person("John", "Doe", 50, "blue");  
var myMother = new person("Sally", "Rally", 48, "green");
```

- Ključna reč this:
  - korišćena u funkciji predstavlja objekat koji poseduje tu funkciju;
  - korišćena u objektu predstavlja taj isti objekat;
  - u konstruktoru nema vrednost, ona je substitucija za novi objekat.
    - Vrednost **this** postaće novi objekat kada se konstruktor koristi da kreira objekat.
- JavaScript ima i ugrađene objekte npr:

```
var x1 = new Object();    var x2 = new String();  
var x3 = new Number();    var x4 = new Boolean();  
var x5 = new Array();     var x6 = new RegExp();  
var x7 = new Function();  var x8 = new Date();
```

# 0 objektima

- Bolje je koristiti primitivne tipove a ne kreirati složene objekte takvog tipa:

```
var x1 = {};           // objekat
var x2 = "";           // string
var x3 = 0;            // number
var x4 = false;        // boolean
var x5 = [];           // array objekat
var x6 = /()/          // regexp objekat
var x7 = function(){}; // function objekat
```

- JavaScript objekti su adresirani su po referenci a ne po vrednosti.
  - Npr. ako su x i y objekti,  
var x = y; // x nije kopija y, već referenciraju isti objekat.
  - Npr. `var person = {firstName:"Elvis", lastName:"Prisli"}`  
`var x = person;`  
`x.lastName = "Presley";` //promenjeno i za person
- Pristup svojstvima može biti kao što sledi:
  - `objectName.property` // person.age
  - `objectName["property"]` // person["age"]
  - `objectName[expression]` // x = "age"; person[x]

# 0 objektima

- Kretanje po svim svojstvima objekta:

```
var person = {fname:"John", lname:"Doe", age:25};  
for (x in person) {  
    txt += person[x];  
}
```

- Dodavanje novog svojstva:

```
var person = {  
    firstname:"John",  
    lastname:"Doe",  
    age:50,  
    eyecolor:"blue"  
};  
person.nationality = "English";
```

- Brisanje svojstva:

```
delete person.nationality;
```

- JavaScript objekti nasleđuju svojstva od njihovih prototipova.
  - Ako se obriše svojstvo prototipa to će uticati na sve objekte koji su nasleđeni iz prototipa.

# 0 objektima

- Kreiranje metoda objekta:
  - methodName : function(parameters) { code lines }
- Pristupanje metodima objekta:
  - objectName.methodName(parameters)
- Ako se poziv metoda navede bez zagrada, dakle kao da pristupamo svojstvu, vratiće se definicija funkcije, kao što sledi:

```
<!DOCTYPE html>
```

```
<html><body><p id="demo1"></p><p id="demo2"></p><script>
```

```
var person = {
```

```
  firstName: "John",
```

```
  lastName : "Doe",
```

```
  id       : 5566,
```

```
  fullName : function() {
```

```
    return this.firstName + " " + this.lastName;
```

```
  }
```

```
};
```

```
document.getElementById("demo1").innerHTML = person.fullName();
```

```
document.getElementById("demo2").innerHTML = person.fullName;
```

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

John Doe

```
function () { return this.firstName + " " +  
this.lastName; }
```

# 0 objektima

- Promena vrednosti svojstava pomoću metode:

```
<!DOCTYPE html>
```

```
<html><body><p id="demo"></p>
```

```
function person(firstName, lastName, age, eyeColor) {  
    this.firstName = firstName;  
    this.lastName  = lastName;  
    this.age       = age;  
    this.eyeColor  = eyeColor;  
    this.changeName = function (name) {  
        this.lastName = name;  
    }  
}
```

```
var myMother = new person("Sally","Rally",48,"green");
```

```
myMother.changeName("Doe");
```

```
document.getElementById("demo").innerHTML =
```

```
"My mother's last name is " + myMother.lastName;
```

```
</script>
```

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

# 0 objektima

- Standardni način da se kreira prototip objekta je korišćenje konstruktora (funkcije):

```
function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName  = last;  
    this.age       = age;  
    this.eyeColor  = eyecolor;  
}
```

sada se korišćenjem new kreiraju konkretni objekti:

```
var myFather = new person("John", "Doe", 50, "blue");  
var myMother = new person("Sally", "Rally", 48, "green");
```

- Dodavanje svojstva ili metode SAMO objektu je kao što sledi:

```
myFather.nationality = "English";  
myFather.name = function () {  
    return this.firstName + " " + this.lastName;  
};
```



# 0 objektima

- Dodavanje svojstva prototipu, gde svi objekti koji su nastali iz prototipa dobijaju dodatno svojstvo:

```
function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName  = last;  
    this.age       = age;  
    this.eyeColor  = eyecolor;  
    this.name      = function() {  
        return this.firstName + " " + this.lastName;};  
}
```

- **NE može sa** `person.nationality = "English";` jer `person` nije objekat.

Dodavanje svojstva prototipu je kao što sledi:

```
person.prototype.nationality = "English";
```

Dodavanje metode prototipu je kao što sledi:

```
person.prototype.uzmigodine = function() {  
    return this.age;  
};
```

# 0 funkcijama

- Funkcija može biti dodeljena promenljivoj, tako da se onda ta promenljiva može koristiti kao ta funkcija:

```
var x = function (a, b) {return a * b}; //x je funkcija
var z = x(4, 3);
```

- Konstruktor funkcije (ne preporučujem):

```
var myFunction = new Function("a", "b", "a++; return a * b;");
var x = myFunction(4, 3);
```

bolje je:

```
var myFunction = function (a, b) {a++; return a * b;};
var x = myFunction(4, 3);
```

- Funkcija može pozvati i samu sebe:

```
(function () {
    var x = "Hello!!";    // samopoziv
})();
```

ako su potrebni parametri i oni se navode, kao i kod tradicionalnog poziva funkcije

# 0 funkcijama

- Primer u kome funkcija ima jedan parametar, te je naveden i stvarni argument poziva (samopozivajuće anonimne) funkcije:

```
<!DOCTYPE html>
<html>
<body>
<p>Functions can be invoked automatically without being
called:</p>
<p id="demo"></p>
<script>
(function (smeh) {
    document.getElementById("demo").innerHTML =
        "Hello! I called myself "
        +
        smeh;
})("haha");
</script>
</body>
</html>
```

# 0 funkcijama

- Funkcije se koriste:

- kao vrednosti

```
function myFunction(a, b) {  
    return a * b;  
}  
var x = myFunction(4, 3);
```

- u izrazima

```
function myFunction(a, b) {  
    return a * b;  
}  
var x = myFunction(4, 3) * 2;
```

- Funkcije su objekti sa svojstvima i metodama:

```
function myFunction(a, b) {  
    return arguments.length;  
}  
function myFunction2(a, b) { return a * b; }  
var txt = myFunction2.toString();  
//function myFunction2(a, b) { return a * b; }
```

# 0 funkcijama

- Ako se ne navede stvarni argument pri pozivu funkcije, on će biti u datoj funkciji nedefinisan.

```
<script>
  function myFunction(x, y) {
    if (y === undefined) { y = 0; }
    return x * y;
  }
  document.getElementById("demo").innerHTML = myFunction(4);
</script>
```

- JavaScript funkcije imaju ugrađen objekat koji se zove objekat argumenata (arguments object) koji sadrži niz argumenata koji se koriste pri pozivu funkcije.

```
x = findMax(1, 123, 500, 115, 44, 88);
function findMax() {
  var i, max = 0;
  for (i = 0; i < arguments.length; i++) {
    if (arguments[i] > max) {
      max = arguments[i];
    }
  }
  return max;
}
```

# 0 funkcijama

- Argumenti se prosleđuju po vrednosti (ako nije objekat).
- Kod prosleđivanja objekata proslediće se kopija reference a to znači da će se kod u pozvanoj funkciji koji barata ovom kopijom reflektovati na original.
- Objekat stranice browser-a pripada objektu window.

```
function myFunction(a, b) {  
    return a * b;  
}  
myFunction(10, 2);           // ovako  
window.myFunction(10, 2);    // ili ovako
```

- U HTML-u vrednost **this** u globalnoj funkciji se odnosi na objekat window.

```
<!DOCTYPE html>  
<html><body>    <p id="demo"></p>  
<script>  
function myFunction() {    return this;    }  
document.getElementById("demo").innerHTML = myFunction();  
// ispisuje: [object Window]  
</script>  
</body></html>
```

# Poziv funkcije kao metoda

- Ako se upotrebi **this** u metodi, onda se **this** odnosi na objekat čija je metoda pozvana.

```
var myObject = {  
  firstName: "John",  
  lastName: "Doe",  
  fullName: function () {  
    return this.firstName + " " + this.lastName;  
  },  
  vratiobjekat: function () {  
    return this;  
  }  
}  
myObject.fullName();           // "John Doe"  
myObject.vratiobjekat();       // [object Object]
```

- Paziti, **this** u konstruktoru nema vrednost, dok za konkretan objekat ima:

```
function myFunction(arg1, arg2) {  
  this.firstName = arg1;    this.lastName = arg2; }  
var x = new myFunction("John", "Doe");    // nov objekat  
x.firstName;                             // "John"
```

# Predefinisane metode funkcije: call , apply

- JavaScript predefinisane **metode funkcije**:

- **call**
- **apply**

pozivaju funkciju gde je prvi parameter objekat (setuje this).

- Metoda call prihvata odvojene argumente
- Metoda apply prihvata argumente kao niz

```
function myFunction(a, b) {  
    return a * b;  
}
```

```
var myObject = new Object();  
myFunction.call(myObject, 10, 2);           // vraća 20
```

```
myArray = [10,2];  
myFunction.apply(myObject, myArray);        // vraća 20
```

```
var hm1 = { ime:"Elvis", pisi:function(){ console.log(this.ime); }};  
var hm2 = { ime: "Tom" };  
hm1.pisi.call(hm2);
```

- U JavaScript strict modu prvi argument postaje vrednost this pozvane funkcije, čak i ako argument nije objekt.



# Predefinisane metode funkcije: call , apply

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
  function myFunction(a,b) {
    return a*b;
  }
  var myObject = new Object();

  document.getElementById("demo").innerHTML =
    myFunction.call(myObject,10,2);

  alert("dalje");
  myArray=[10,4];
  document.getElementById("demo").innerHTML =
    myFunction.apply(myObject,myArray);
</script>
</body>
</html>
```

# Unutrašnja funkcija

- Iz funkcije se vidi sve iznad.
- U sledećem primeru data je unutrašnja (inner) funkcija.

```
<!DOCTYPE html>
<html>
<body>
<p>Counting with a local variable.</p>
<p id="demo">0</p>
<script>
  var be=5;
  document.getElementById("demo").innerHTML = add();
  function add() {
    var counter = 3;

    function plus() {counter += be;}

    plus();
    return counter;
  }
</script>
</body></html>
```

# Closure

```
<!DOCTYPE html>
<html>
<body>
<p>Counting with a local variable.</p>
```

Counting with a local variable.

Count!

19

```
<button type="button" onclick="myFunction()">Count!</button>
```

```
<p id="demo">0</p>
```

```
<script>
var add = (function () {
    var counter = 0;
    return function () {return counter += 1;}
})();
```

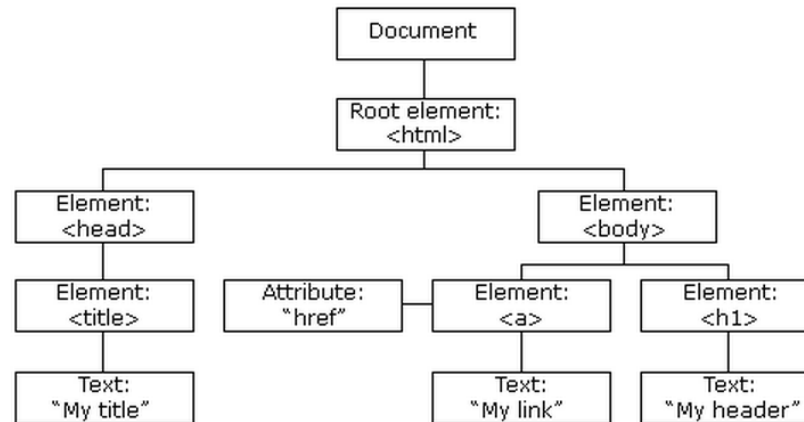
```
function myFunction(){
    document.getElementById("demo").innerHTML = add();
}
</script>
</body>
</html>
```

# Closure

- Objašnjenje prethodnog primera:
  - Neka je promenljiva `add` dodeljena povratnoj vrednosti samopozivajuće funkcije.
  - Samopozivajuća funkcija se pokreće samo jednom.
    - Ona postavlja brojač (`counter`) na nulu (`0`)
    - Vraća izraz iza `return`-a (`function () {return counter += 1;}`).
  - Ovim načinom je promenljiva `add` postala funkcija.
    - Ova funkcija može pristupiti promenljivoj `counter` u roditeljskom opsegu.
    - Ovo se zove JavaScript closure (zatvarač).
      - Ovim je postignuto da funkcija ima svoje "privatne" promenljive.
  - Promenljiva `counter` je zaštićena opsegom anonimne funkcije i može se promeniti samo ako se koristi `add` funkcija.
  - *Napomena:*  
Zatvarač (closure) je funkcija koja ima pristup roditeljskom opsegu čak i kada je roditeljska funkcija "zatvorena".
- Tako je u prethodnom primeru napravljen brojač koji može da broji koristeći lokalnu promenljivu samopozivajuće funkcije.

# DOM i JavaScript

- Kada se web stranica učitava, browser kreira Document-Object-Model stranice.
- HTML DOM model je konstruisan kao stablo objekata:



- JavaScript može:
  - menjati sve HTML elemente na stranici
  - menjati sve HTML atribute na stranici
  - menjati sve CSS stilove na stranici
  - ukloniti postojeće HTML elemente i attribute
  - dodavati nove HTML elemente i attribute
  - reagovati na sve HTML događaje na stranici
  - kreirati nove HTML događaje na stranici

# Selekcija, menjanje, dodavanje i brisanje HTML elementa

- Nalaženje HTML elemenata:
  - document.get**E**lementById()
  - document.get**E**lementsByTagName()
  - document.get**E**lementsByClassName()
- Menjanje sadržaja, atributa i stila HTML elementa:
  - element.innerHTML=
  - element.attribute=
  - element.setAttribute(attribute,value)
  - element.style.*property*=
- Dodavanje i brisanje HTML elementa:
  - document.createElement()
  - document.appendChild()
  - document.removeChild()
  - document.replaceChild()
  - document.write(text)       //upis u HTML output stream

# Baratanje HTML elemenatom

- Dodavanje event handler (obrađivača događaja):
  - `document.getElementById(id).onclick=function(){code}`
- Nalaženje HTML objekata:
  - `document.anchors` vraća sve `<a>` elemente koji imaju atribut `name`
  - `document.baseURI` vraća bazni URI dokumenta
  - `document.body` vraća `<body>` element
  - `document.cookie` vraća cookie dokumenta
  - `document.doctype` vraća document-ov doctype
  - `document.documentElement` vraća `<html>` element
  - `document.documentMode` vraća mod dokumenta
  - `document.documentURI` vraća URI dokumenta
  - `document.domain` vraća ime domena
  - `document.embeds` vraća sve `<embed>` elemente
  - `document.forms` vraća sve `<form>` elemente

# Baratanje HTML elemenatom

- `document.head` vraća `<head>` element
- `document.images` vraća sve `<img>` elemente
- `document.implementation` vraća DOM implementaciju
- `document.inputEncoding`
  - vraća karakter set dokumenta (encoding)
- `document.lastModified`
  - vraća datum i vreme ažuriranja dokumenta
- `document.links`
  - vraća sve `<area>` i `<a>` elemente koji imaju atribut `href`
- `document.readyState` vraća status učitavanja (loading) dokumenta
- `document.referrer` vraća URI linkovanog dokumenta
- `document.scripts` vraća sve `<script>` elemente
- `document.strictErrorChecking` vraća da li je postavljena provera strict-a
- `document.title` vraća `<title>` element
- `document.URL` vraća ceo URL dokumenta



# Nalaženje HTML elementa

- HTML element se može naći koristeći:

- id, ako se ne nađe vraća null

```
var x = document.getElementById("intro");
```

- tag name

```
var x = document.getElementsByTagName("p"); //svi <p>
```

kombinacija, npr. nađi sve <p> elemente unutar id "main"

```
var x = document.getElementById("main");
```

```
var y = x.getElementsByTagName("p");
```

- class name

```
var x = document.getElementsByClassName("intro");
```

- CSS selectors

```
var x = document.querySelectorAll("p.intro");
```

```
//svi <p> koji imaju atribut class="intro"
```

- HTML kolekcije objekata

```
var x = document.forms["frm1"]; var text = ""; var i;
```

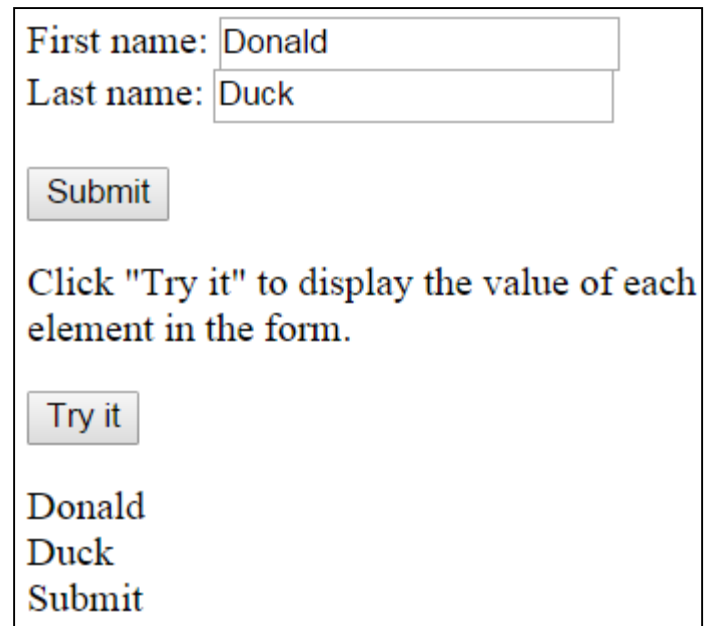
```
for (i = 0; i < x.length; i++) {
```

```
    text += x.elements[i].value + "<br>"; }
```

```
document.getElementById("demo").innerHTML = text;
```

# Nalaženje HTML elementa

```
<!DOCTYPE html>
<html><body>
<form id="frm1" action="form_action.php">
  First name: <input type="text" name="fname" value="Donald"><br>
  Last name: <input type="text" name="lname" value="Duck"><br><br>
  <input type="submit" value="Submit">
</form>
<p>Click "Try it" to display the value
  of each element in the form.</p>
<button onclick="myFunction()">Try it
</button>
<p id="demo"></p><script>
function myFunction() {
  var x = document.forms["frm1"];
  var text = "";
  var i;
  for (i = 0; i < x.length ;i++) {
    text += x.elements[i].value + "<br>";
  }
  document.getElementById("demo").innerHTML = text;
}</script></body></html>
```



The screenshot shows the rendered HTML form. It contains two text input fields: 'First name:' with the value 'Donald' and 'Last name:' with the value 'Duck'. Below these is a 'Submit' button. Underneath the button is a paragraph of text: 'Click "Try it" to display the value of each element in the form.' Below this text is a 'Try it' button. At the bottom of the form, the output of the script is displayed: 'Donald', 'Duck', and 'Submit' on separate lines.

# Nalaženje HTML elementa

- Ispis sorsa body elementa u alert-u, te sorsa html elementa u alert-u:

```
<!DOCTYPE html>
<html><body>
<p id="demo"></p>
<script>
    alert(document.body.innerHTML);           //u <body></body>
    alert(document.documentElement.innerHTML); //u <html></html>
</script>
</body></html>
```

- Prebrojavanje linkova sa atributom name (Number of anchors are: 3):

```
<!DOCTYPE html>
<html><body>
    <a name="html">HTML Tutorial</a><br>
    <a name="css">CSS Tutorial</a><br>
    <a name="xml">XML Tutorial</a><br>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Number of anchors are: " + document.anchors.length;
</script>
</body></html>
```

# Nalaženje HTML elementa

- Ispisuje koliko ima embed elemenata, te koliko ima formi na stranici:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo1"></p>
<p id="demo2"></p>

<form action="">
First name: <input type="text" name="fname" value="Donald">
<input type="submit" value="Submit">
</form>

<script>
document.getElementById("demo1").innerHTML =
    "Number of embeds: " + document.embeds.length;
document.getElementById("demo2").innerHTML =
    "Number of forms: " + document.forms.length;
</script>

</body>
</html>
```

# Nalaženje HTML elementa

- Naslov dokumenta, broj skriptova:

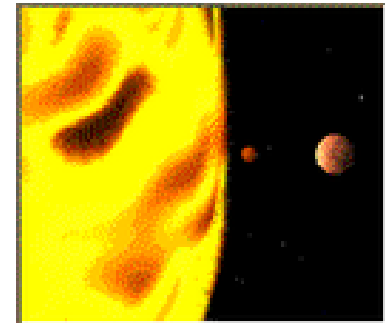
```
<!DOCTYPE html>
<html>
<head>
  <title>NASLOV</title>
  <script>
  </script>
</head>
<body>
<p id="demo" title="hm"></p>
<script>
  document.getElementById("demo").innerHTML =
    "The title of this document is: " + document.title;
  document.getElementById("demo").innerHTML += "<br>" +
    "Number of scripts: " + document.scripts.length;
</script>
</body>
</html>
```

# Nalaženje HTML dokumenta

- Broj elemenata `<a>` i `<area>` sa atributom href:

```
<!DOCTYPE html>
<html><body><p>

<map name="planetmap">
  <area shape="rect" coords="0,0,82,126"
    href="sun.htm" alt="Sun">
  <area shape="circle" coords="90,58,3"
    href="mercur.htm" alt="Mercury">
  <area shape="circle" coords="124,58,8"
    href="venus.htm" alt="Venus">
</map>
<br>
<a href="/css/space.html">CSS</a>
</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"Number of links: " + document.links.length;
</script></body></html>
```



CSS

Number of links: 4

# Nalaženje HTML dokumenta

- Broj slika u dokumentu:

```
<!DOCTYPE html>
<html><body>
  
  
  <p id="demo"></p>
  <script>
    document.getElementById("demo").innerHTML =
      "Number of images: " + document.images.length;
  </script>
</body>
</html>
```

- Menjanje atributa HTML dokumenta (setiti se paljenja sijalice):

```
<!DOCTYPE html>
<html><body>
  
  <script>
    document.getElementById("myImage").src = "landscape.jpg";
  </script>
</body></html>
```



# Menjanje stila HTML dokumenta

- Menjanje stila HTML dokumenta:

- selektovati dokument

- promeniti stil

```
<!DOCTYPE html>
```

```
<html><body>
```

```
<p id="p1">Hello World!</p>
```

```
<p id="p2">Hello World!</p>
```

```
<script>
```

```
document.getElementById("p2").style.color = "blue";
```

```
document.getElementById("p2").style.fontFamily = "Arial";
```

```
document.getElementById("p2").style.fontSize = "larger";
```

```
//ili
```

```
//document.getElementById("p2").setAttribute("style",
```

```
//"color:green;");
```

```
<button type="button"
```

```
onclick="document.getElementById('p1').style.color = 'red'">
```

```
Click Me!</button>
```

```
</script>
```

```
</body>
```

```
</html>
```

Hello World!

Hello World!

Click Me!

Hello World!

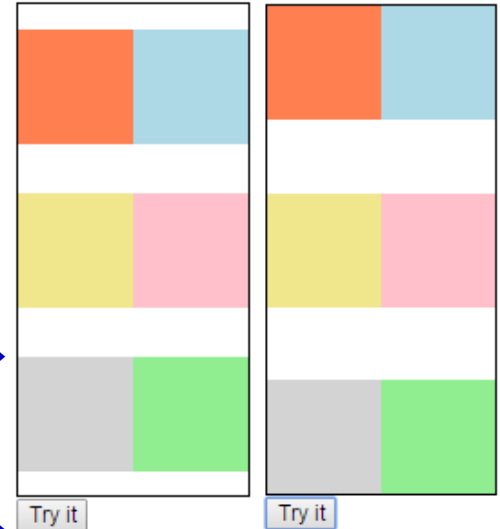
Hello World!

Click Me!



# Postavljanje: align-content

```
<!DOCTYPE html>
<html><head><style>
#main {
    width: 140px;    height: 300px;    border: 1px solid #000000;
    display: flex;    flex-flow: row wrap;
    align-content: space-around; }
#main div {    width: 70px;    height: 70px; }
</style></head><body>
<div id="main">
    <div style="background-color:coral;"></div>
    <div style="background-color:lightblue;"></div>
    <div style="background-color:khaki;"></div>
    <div style="background-color:pink;"></div>
    <div style="background-color:lightgrey;"></div>
    <div style="background-color:lightgreen;"></div> </div>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
    document.getElementById("main").style.alignContent =
                                                                    "space-between";
}
</script></body></html>
```



# Kreiranje elemenata

```
<!DOCTYPE html>
<html><body>
<p>Click the button to create a DETAILS, a SUMMARY and a P
element.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
    var x = document.createElement("DETAILS");
    document.body.appendChild(x);

    var summaryElmnt = document.createElement("SUMMARY");
    var txt1 = document.createTextNode("Copyright 1999-2014.");
    summaryElmnt.appendChild(txt1);

    var pElmnt = document.createElement("P");
    var txt2 = document.createTextNode(" - by Refsnes Data. All
Rights Reserved.");
    pElmnt.appendChild(txt2);
    x.appendChild(summaryElmnt);
    x.appendChild(pElmnt);
}</script></body></html>
```

Click the button to create a DETAILS, a SUMMARY and a P element.

Try it

Click the button to create a DETAILS, a SUMMARY and a P element.

Try it

► Copyright 1999-2014.

Click the button to create a DETAILS, a SUMMARY and a P element.

Try it

▼ Copyright 1999-2014.

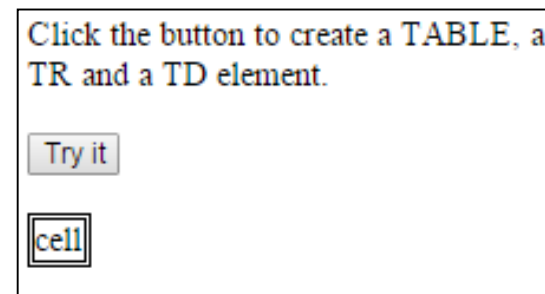
- by Refsnes Data. All Rights Reserved.

# Kreiranje elementa

```
<!DOCTYPE html>
<html><head><style> table, td { border: 1px solid black; }</style>
</head><body>
<p>Click the button to create a TABLE, a TR and a TD element.</p>
<button onclick="myFunction()">Try it</button><br><br>
<script>
function myFunction() {
    var x = document.createElement("TABLE");
    x.setAttribute("id", "myTable");
    document.body.appendChild(x);

    var y = document.createElement("TR");
    y.setAttribute("id", "myTr");
    document.getElementById("myTable").appendChild(y);

    var z = document.createElement("TD");
    var t = document.createTextNode("cell");
    z.appendChild(t);
    document.getElementById("myTr").appendChild(z);
}
</script></body></html>
```



# Animacija, backfaceVisibility

```
<!DOCTYPE html>
<html><head><style>
div {
    width: 100px; height: 100px; background: red; color: white;
    -webkit-animation: mymove 2s infinite linear alternate; /*CSO*/
    animation: mymove 2s infinite linear alternate;
}
@-webkit-keyframes mymove {/*CSO*/
    to {-webkit-transform: rotateY(180deg);} }
@keyframes mymove { to {transform: rotateY(180deg);} }
</style></head><body>
<div id="myDIV"> <h1>Hello</h1></div>
<input type="checkbox" onclick="myFunction(this)"
checked>backface-visibility
<script>
function myFunction(x) {
    if (x.checked === true)
        document.getElementById("myDIV").style.backfaceVisibility =
                                                                    "visible";
    else document.getElementById("myDIV").style.backfaceVisibility =
                                                                    "hidden";
}
</script></body></html>
```



☒ backface-visibility

# Dohvatanje elementa preko taga

- U primeru se uzima title

```
<!DOCTYPE html>
<html><head>
<title>HTML DOM Objects</title>
</head>
<body>
<h3>A demonstration of how to access a TITLE element</h3>
<p>Click the button to get the text of the document's title.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>

<script>
function myFunction() {
    var x = document.getElementsByTagName("TITLE")[0].text;
    document.getElementById("demo").innerHTML = x;
}
</script>

</body>
</html>
```

**A demonstration of how to access a TITLE element**

Click the button to get the text of the document's title.

Try it

HTML DOM Objects

# Događaji

- Promena teksta headera na klik u oblasti headera:

```
<!DOCTYPE html>
<html><body>
<h1 onclick="this.innerHTML='Oops!'">Click on this text!</h1>
</body></html>
```

ili:

```
<!DOCTYPE html>
<html><body>
<h1 onclick="changeText(this)">Click on this text!</h1>
<script>
function changeText(id) {
    id.innerHTML = "Oops!";
}
</script></body></html>
```

- U koracima: uzmi element preko id-a, postavi funkciju za događaj te napiši funkciju:

```
<!DOCTYPE html>...<body><button id="myBtn">Try it</button>
<p id="demo"></p>
<script>document.getElementById("myBtn").onclick = displayDate;function
displayDate() {    document.getElementById("demo").innerHTML = Date(); }
</script></body></html>
```

# onload, onchange

- Provera da li su cookie omogućeni po učitavanju stranice:

```
<!DOCTYPE html>
<html>
<body onload="checkCookies()"> <p id="demo"></p>
<script>
function checkCookies() {
    var text = "";
    if (navigator.cookieEnabled==true)text = "Cookies are enabled.";
    else text = "Cookies are not enabled.";
    document.getElementById("demo").innerHTML = text;
}
</script></body></html>
```

- Promena teksta u velika slova pri napuštanju input text-box-a:

```
<!DOCTYPE html>
<html><head><script>
function myFunction() {
    var x = document.getElementById("fname");
    x.value = x.value.toUpperCase();
}
</script></head><body>Enter your name: <input type="text"
id="fname" onchange="myFunction()"></body></html>
```

Enter your name:

Enter your name:

# Dodeljivanje više događaja elementu

- U datom primeru pritisak i otpuštanje tastera miša su dva događaja dodeljena elementu div.

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<div onmousedown="mDown(this)"
```

```
onmouseup="mUp(this)"
```

```
style="background-color:#FFFF00; width:90px; height:20px;
```

```
padding:40px;">Click Me</div>
```

```
<script>
```

```
function mDown(obj) {
```

```
    obj.style.backgroundColor = "#1ec5e5";
```

```
    obj.innerHTML = "Release Me";
```

```
}
```

```
function mUp(obj) {
```


```
    obj.style.backgroundColor="#00FF88";
```

```
    obj.innerHTML="Thank You";
```


```
}
```

```
</script>
```


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



Click Me



Release Me



Thank You



# Propagacija događaja

- Propagacija događaja definiše redosled elemenata koji obrađuju dati događaj pri pojavi događaja.
  - Npr. element `<div>` sadrži element `<p>` i korisnik je kliknuo na `<p>` element. Kojim redosledom će događaj "click" biti obrađen, odnosno, koji element prvi obrađuje ovaj događaj ?
- Postoje dva načina propagacije događaja u HTML DOM:
  - **bubbling**
    - događaj krajnjeg unutrašnjeg elementa se obrađuje prvi a onda se ide redom do krajnjeg vanjskog elementa (kao mehur, "bubble", koji isplivava)
      - u datom primeru `<p>` element prvi obrađuje događaj "click" a onda `<div>` element obrađuje događaj "click" .
  - **capturing**
    - događaj krajnjeg vanjskog elementa se obrađuje prvi a onda se ide redom do krajnjeg unutrašnjeg elementa
      - u datom primeru `<div>` element prvi obrađuje događaj "click" a onda `<p>` element obrađuje događaj "click" .
- Korišćenjem metode `addEventListener()` može se parametarski specificirati tip propagacije (parametar `"useCapture"`), koji je podrazumevano `false`, dakle, bubbling propagacija je podrazumevana.

# Postavljanje osluškivača događaja

- Metoda `addEventListener` elementa dodaje obrađivač događaja za dati element. Sintaksa je kao što sledi:

```
element.addEventListener(event, function, useCapture);
```

parametri su: događaj (**bez prefiksa on**), funkcija koja obrađuje dati događaj i boolean vrednost koja specificira da li se koristi event *bubbling* or event *capturing* (ovaj parametar je opcionalni i podrazumevano je false, bolje ga je eksplicitno navesti).

```
<!DOCTYPE html>
<html><body>
<p>This example uses the addEventListener() method to attach a
click event to a button.</p>
<button id="myBtn">Try it</button>
<script>
document.getElementById("myBtn").addEventListener("click",
    function(){      //resenje preko anonimne funkcije
        alert("Hello World!");
    },
    false);
</script>
</body></html>
```

# Primer propagacije događaja

```
<!DOCTYPE html>
<html><head><style>
div { background-color:yellow; border:1px solid; padding:20px; }
</style>
</head><body>
<div id="myDiv"><p id="myP" >Click this (Bubbling) </p></div><br>
<div id="myDiv2"><p id="myP2">Click this (Capturing)</p></div>
<script>
document.getElementById("myDiv").addEventListener("click",
    function() { alert("You clicked the DIV element!");}, false);
document.getElementById("myP").addEventListener("click",
    function() { alert("You clicked the P element!");} , false);
document.getElementById("myDiv2").addEventListener("click",
    function() { alert("You clicked the DIV element!");}, true );
document.getElementById("myP2").addEventListener("click",
    function() { alert("You clicked the P element!");} , true );
</script>
</body>
</html>
```

Click this (Bubbling)

Click this (Capturing)

# Osluškivači događaja

- Dodavanje osluškivača preko imenovane funkcije

```
<!DOCTYPE html>
<html><body>
<p>This example uses the addEventListener() method to execute a
function when a user clicks on a button.</p>
<button id="myBtn">Try it</button>
<script>
document.getElementById("myBtn").addEventListener("click",
                                                    myFunction);

function myFunction() {    alert ("Hello World!");}
</script></body></html>
```

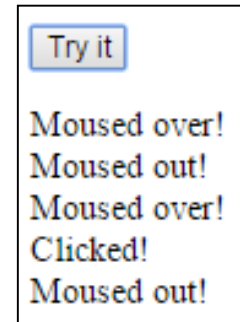
- Dodavanje više funkcija koje obrađuju isti događaj jednog elementa:

```
<!DOCTYPE html>
<html><body> <button id="myBtn">Try it</button> <script>
    var x = document.getElementById("myBtn");
    x.addEventListener("click", prvaFunkcija);
    x.addEventListener("click", drugaFunkcija);
    function prvaFunkcija() {    alert ("Prva funkcija!");    }
    function drugaFunkcija(){    alert("Druga funkcija!");    }
</script>
</body></html>
```

# Osluškivači događaja

- Dodavanje više osluškivača događaja jednom elementu za više događaja:

```
<!DOCTYPE html>
<html><body>
<button id="myBtn">Try it</button>
<p id="demo"></p>
<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);
function myFunction() {
    document.getElementById("demo").innerHTML += "Moused
over!<br>";
}
function mySecondFunction() {
    document.getElementById("demo").innerHTML += "Clicked!<br>";
}
function myThirdFunction() {
    document.getElementById("demo").innerHTML += "Moused
out!<br>";
}</script></body></html>
```



# Dodavanje osluškivača objektu window

- U datom primeru reaguje se na promenu veličine prozora:

```
<!DOCTYPE html>
<html><body>
<p>Try resizing this browser window to trigger the "resize" event
handler.</p> <p id="demo"></p> <script>
window.addEventListener("resize", function(){
    document.getElementById("demo").innerHTML = Math.random();
});
</script></body></html>
```

- U datom primeru preko anonimne funkcije se prosleđuju parametri funkciji koja obrađuje događaj:

```
<!DOCTYPE html>
<html><body><p>Click the button to perform a calculation.</p>
<button id="myBtn">Try it</button> <p id="demo"></p> <script>
var p1 = 5; var p2 = 7;
document.getElementById("myBtn").addEventListener("click",
    function() { myFunction(p1, p2);} );
function myFunction(a, b) {    var result = a * b;
    document.getElementById("demo").innerHTML = result;
}
</script></body></html>
```

# Uklanjanje funkcije koja obrađuje događaj

```
<!DOCTYPE html>
<html><head><style>
#myDIV { background-color:coral;
        border:1px solid;
        padding: 50px;    color: white;}
</style></head><body>
<div id="myDIV">Na onmousemove događaj ispisuje se random broj u
paragrafu "demo".</div>
<button onclick="removeHandler()" id="myBtn">Ukloni obradu
mousemove Try it</button>
</div> <p id="demo"></p>
<script>
document.getElementById("myDIV").addEventListener("mousemove",
myFunction);
function myFunction() {
    document.getElementById("demo").innerHTML = Math.random();
}
function removeHandler() {
    document.getElementById("myDIV").removeEventListener("mousemove",
                                                                myFunction);
}
</script></body></html>
```

Na onmousemove događaj ispisuje se random broj u paragrafu "demo".

Ukloni obradu mousemove Try it

0.8016582406125963

# Navigacija po čvorovima

- Navigacija po čvorovima dokumenta u JavaScript-u je kao što sledi:
  - parentNode
  - childNodes[nodenumbr]
  - firstChild
  - lastChild
  - previousSibling
  - nextSibling

```
<!DOCTYPE html>
<html><body>
<h1 id="intro">INTRO</h1> <p id="demo"></p> <script>
var myText =
    document.getElementById("intro").childNodes[0].nodeValue;
    //ili
    //document.getElementById("intro").firstChild.nodeValue;
document.getElementById("demo").innerHTML = myText; //INTRO
</script></body></html>
```



# Kreiranje, ubacivanje, brisanje i zamena elementa

```
<!DOCTYPE html>
<html><body>
<div id="div1">
  <p id="p1" >Prvi</p>
  <p id="p2" >Drugi</p>
</div>
<script>
var ediv1 = document.getElementById("div1");
var ep1    = document.getElementById("p1");
var ep2    = document.getElementById("p2");
var p3     = document.createElement("p");
var txt3   = document.createTextNode("Treci");
  p3.appendChild(txt3);
var p4     = document.createElement("p");
var txt4   = document.createTextNode("Cetvrti");
  p4.appendChild(txt4);
ediv1.insertBefore(p3,ep2);
ediv1.removeChild(ep1);
p3.parentNode.removeChild(ep2);
ediv1.replaceChild(p4,p3);
</script></body></html>
```

Prvi   Prvi   Treci   Treci   Cetvrti

Drugi   Treci   Drugi

Drugi

# Interfejs fullscreen

- Za interfejs fullscreen koriste se:
  - metode:
    - requestFullscreen (mozRequestFullScreen) - za sve elemente dokumenta
    - exitFullscreen (mozExitFullScreen) – izlazak iz prikaza preko celog ekrana
  - svojstva:
    - fullscreenEnabled
      - vraća boolean vrednost da li dokument omogućuje prikaz preko celog ekrana
    - fullscreenElement (mozFullScreenElement) vraća referencu na element koji se prikazuje preko celog ekrana
  - događaji
    - fullscreenchange
      - događaj prelaska u/iz fullscreen/a
    - fullscreenerror
      - događaj neuspeha prelaska u fullscreen

# Primer za fullscreen

```
<!DOCTYPE html>
<html lang="en"><head><meta charset="utf-8"><title>Ceo ekran</title>
  <script>
    var video;
    function initiate(){
      video = document.getElementById('media');
      video.addEventListener('click', gofullscreen);
    }
    function gofullscreen(){
      if(!document.mozFullScreenElement){
        video.mozRequestFullScreen();
        video.play();      }
    }
    addEventListener('load', initiate);
  </script>

</head><body>
  <section>
    <video id="media" width="720" height="400" loop autoplay>
      <source src="trailer.mp4">
    </video>  </section></body></html>
```

# Fullscreen za dati element

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>FS</title>
  <style>
    #player:-moz-full-screen,
    #player:-moz-full-screen #media {
      width: 50%;
      height: 50%;
    }
  </style>
  <script>
    var video, player;
    function initiate(){
      video  = document.getElementById('media');
      player = document.getElementById('player');
      player.addEventListener('click', gofullscreen);
    }
  </script>
</head>
<body>
  <div id="player">
    <div id="media">
      <video src="video.mp4">
    </div>
  </div>
</body>
</html>
```

# Fullscreen za dati element

```
function gofullscreen(){
    if(!document.mozFullScreenElement){
        player.mozRequestFullScreen();
        video.play();
    }else{
        document.mozExitFullScreen();
        video.pause();
    }
}
addEventListener('load', initiate);
</script>
</head>
<body>
    <section id="player">
        <video id="media" width="720" height="400" loop autoplay>
            <source src="trailer.mp4">
        </video>
    </section>
</body>
</html>
```