```
    var osoba = {
        ime: "Perica",
        prezime : "P",
        id: 666,
        "adresa stanovanja": "Nepoznata",
        // this!!!
        toString : function() {
           return this.prezime + " " + this.ime;
        }
    };
    document.getElementById("osoba1").innerHTML =
osoba.toString();
  </script>
</body>
</html>
```

this

Zapazite da se pristup elementima objekta iz metode u objektu obavlja pomoću ključne reči `this`. Ključna reč `this` označava tekući (nadređeni) objekat. Ako se ključna reč `this` koristi u objektu označava taj objekat, ako se koristi u funkciji označava nadređeni objekat funkcije. U prethodnom primeru objekat `this` je iskorišćen za pristup svojstvima objekta iz jedne funkcije istog objekta:

```
return osoba.prezime + " " + osoba.ime;
ili
return this.prezime + " " + this.ime;
```

Pristup do vrednosti jednog svojstva vrši se pomoću imena svojstva: `obj[imeSvojstva]` ili `obj.imeSvojstva`. Pristup do svih svojstava može da se izvrši pomoću petlje `for.in`: `for(let p in objekat){.}`. Na primer:

```
for (let key in osoba) {
    alert(key + ': ' +osoba[key]);
}
```

Ako se `this` koristi u globalnom prostoru onda je predstavlja isti objekat kao i `window`. Pogledajte sledeći kod i proverite ovo tvrđenje:

```
if (this === window) {
    console.log("U glob. prostoru this == window");
}
```

Takođe, ako bi funkcija vraćala `this` objekat, onda bi sledeći kod bio važeći:

```
function knjiga(naslov, autor) {
    this.naslov = naslov;
    this.autor = autor;
    return this;
}
let k1 = knjiga("Pesme1", "Ršumović");
```

```
let k2 = knjiga("Pesme2", "Ršumović");
// k1 === k2, k2.naslov === 'Pesme2'
```

Umesto toga, koristimo operator `new` pomoću koga se kreira poseban objekta, tipa `knjiga`. U ovom slučaju dobijamo kod nalik onom koji se koristi u radu sa objektima nekih drugih programskih jezika.

```
function knjiga(naslov, autor) {
    this.naslov = naslov;
    this.autor = autor;
}
var k1 = new knjiga("Pesme", "Ršumović");
var k2 = new knjiga("Pesme", "Ršumović");
if (k1 != k2) alert("k1 != k2");
```

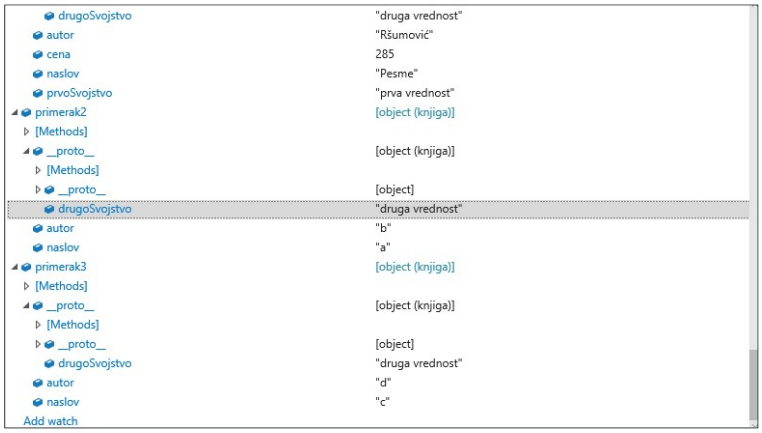
****Prototip - objekat delegiranja ponašanja

Svaki JavaScript objekat ima posebno svojstvo koje se naziva prototip. Prototip je sam objekat. JavaScript objekti dobijaju svojstva i metode od svojih prototipova. Na vrhu lanca prototipova se nalazi `Object.prototype` od koga svi nasleđuju.
Pri kreiranju jednog objekta, koristeći rezervisanu reč `new`, formira se odgovarajući prototip, na primer:

```
function knjiga(naslov, autor) {
    this.naslov = naslov;
    this.autor = autor;
}
var primerak1 = new knjiga("Pesme", "Ršumović");
```

Svojstvo `prototype` omogućava dodavanje novog svojstva već postojećem tipu. Dodavanje novog svojstva postojećem tipu nije isto kao proširenje postojeće objekta novim svojstvima. Na primer:

```
var primerak1 = new knjiga("Pesme", "Ršumović");
primerak1.prvoSvojstvo = "prva vrednost";
var primerak2 = new knjiga("a", "b");
knjiga.prototype.drugoSvojstvo = "druga vrednost";
// nakon promene prototipa promenjena su i svojstva tipa
„knjiga“
// to vazi za primerak3, ali i za primerak2 i primerak1
// Menjaju se samo korisnički prototipovi, a ne ugrađeni.
var primerak3 = new knjiga("c", "d");
var tmp3 = primerak3.drugoSvojstvo;
```



Slika 8.1. Prikaz vrednosti iz „debuger-a“

Funkcija eval

JavaScript može da obradi zadati string kao sopstveni kod. Ovo se obavlja pomoću funkcije eval. Na primer:

```
function testEval() {
    var a = 12, b = 4;
    console.log(eval("a - b"));
    console.log(eval("a * b"));
}
```

Nizovi

Array predstavlja tip objekta koji je niz u JavaScript-u. Niz se definiše pomoću uglastih zagrada. Koristeći niz, umesto da se definiše za svaku vrednost jedna promenljiva, može se definisati jedna promenljiva koja je niz, a koja omogućava rad sa više vrednosti. Ovaj tip podataka ima svoje osobine tj. svojstva i pripadajuće funkcije koje se mogu koristiti. Neke od njih su:

constructor	Vraća funkciju koja kreira Array - konstruktor
length	Broj elemenata niza
prototype	Dozvoljava dodavanje svojstava i metoda već formiranom tipu objekta
concat()	Spajanje dva ili više nizova
find()	Vraća vrednost prvog elementa u nizu koji zadovoljava uslov
findIndex()	Vraća indeks prvog elementa u nizu koji zadovoljava uslov
indexOf()	Pretražuje niz za neki element i vraća njegovu poziciju

sort() Sortira elemente niza

Na primer:

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

    <h1>1 Neuređena lista</h1>
    <p id="Smerovi1"></p>

    <script>
        var smer = viserSmerovi(false);

        html1 = "<ul>";
        for (i = 0; i < smer.length; i++) {
            html1 += "<li>" + smer[i] + "</li>";
        }
        html1 += "</ul>";
        document.getElementById("Smerovi1").innerHTML = html1;

        function viserSmerovi(sort)
        {
            smer = ["NRT","RT","EPO","ELITE","AVT","NET"];
            if(sort)
            {
                smer = smer.sort();
            }
            return smer;
        }
    </script>

</body>
</html>
```

1 Neuređena lista

- NRT
- RT
- EPO
- ELITE
- AVT
- NET

Primer niza sa prekidom i preskokom. Takođe, primer sadrži i petlju po svojstvima jednog objekta odnosno upotrebu operatora for...in.

```
<!DOCTYPE html>
<head>
<style>
  div{
   width:300px;
   height:300px;
   background-color:lightblue;
}
  ul, ol{
   margin:0;
  }
</style>
</head>
<html>
<body>
<a href="#"
id="link1">testiranje rada sa
nizom sa preskokom i
prekidom</a>
<div id="rez1">for petlja kroz niz</div>
<div id="rez2">for in petlja</div>
<script>
var niz = ["NRT", "RT", true, 54, 666, 77];
var mojObjekat = {
  ime: "Perica",
  prezime: "P",
  jmbg: 1231231231231,
  datum: new Date(2017,12,31)
};
var link1 = document.getElementById("link1");
var rez1 = document.getElementById("rez1");
var rez2 = document.getElementById("rez2");
link1.addEventListener("click", function () {
  var htmlRez = "<ul>";
  for (var i = 0; i < niz.length; i++)
  {
    if (i == 2) continue;
    if (i == 4) break;
    debugger;
    htmlRez += "<li>" + niz[i] + "</li>";
  }
  htmlRez += "</ul>";
  rez1.innerHTML = htmlRez;

  var htmlRez = "<ol>";
  for (var s in mojObjekat) {
    debugger;
    htmlRez += "<li>" + s + " : "
    + mojObjekat[s] + "</li>";
  }
  htmlRez += "</ol>";
```

testiranje rada sa nizom sa preskokom i prekidom

- NRT
- RT
- 54

1. ime : Perica
2. prezime : P
3. jmbg : 1231231231231
4. datum : Wed Jan 31 2018 00:00:00 GMT+0100 (Central Europe Standard Time)

```
  rez2.innerHTML = htmlRez;
});
</script>
</body>
</html>
```

U primeru se koristi metoda `addEventListener` kojom se povezuje događaj nekog HTML elementa sa odgovarajućom metodom za rukovanje događajem tzv. `callback` metoda. Metoda za rukovanje događajem se može pisati u samoj metodi `addEventListener` ali i van nje. U drugom slučaju treba koristiti samo naziv metode tzv. `callback`. Ovo treba imati u vidu ako je neophodno prosleđivanje argumenata. Na primer:

```
document.getElementById('dugme').addEventListener("click",
valid);
function valid() {
    let x =
document.getElementById('jmbg').value.match(/^\d{3}$/);
    console.log(x);
    if (x == null) {
        alert("Greska pri unosu!");
    }
}
```

Ukoliko se događaj propagira na više kontrola, kao na primer click događaj koji se evidentira na ugđeždenoj kontroli, onda se redosled evidentiranja događaja po kontrolama može definisati trećim parametrom ove metode.

Konverzije

Eksplicitna konverzija se obavlja primenom metoda za konverziju:

- `Number()` – konverzija u broj,
- `String()` – konverzija u string,
- `Boolean()` – konverzija u Boolean.

Tabela konverzije:

Vrednost/val	Number(val)	String(val)	Boolean(val)
false	0	"false"	false
true	1	"true"	true
0	0	"0"	false
1	1	"1"	true
"0"	0	"0"	true
"1"	1	"1"	true
NaN	NaN	"NaN"	false

```
Infinity          Infinity  "Infinity"        true
-Infinity         -Infinity "-Infinity"       true
""                0         ""                false
"20"              20        "20"              true
"twenty"          NaN       "twenty"          true
[ ]               0         ""                true
[20]              20        "20"              true
[10,20]           NaN       "10,20"           true
["twenty"]        NaN       "twenty"          true
["ten","twenty"] NaN       "ten,twenty"      true
function(){}      NaN       "function(){}"    true
{ }               NaN       "[object Object]" true
null              0         "null"            false
undefined         NaN       "undefined"       false
```

Regularni izrazi

Regularni izraz je string kojim se definišu pravila za proveru ili ispitivanje drugog stringa. Ova pravila su definisana određenom sintaksom.
Regularni izrazi se koriste za proveru ulaznih podataka na strani web čitača, dakle pomoću JavaScript-a ili na serverskoj strani. Podršku za regularne izraze imaju svi porgramski jezici.

Javascript koristi regularne izraze preko funkcija:

- match( )
- search( )
- replace( )

Primenu ovih funkcija razmotirćemo pošto definišemo regularne izraze.

Dakle, opšti oblik jednog regularnog izraza je:

let regIzraz = /uzorak/modifikator;

Modifikator nije obavezan i u tom slučaju je:
let regIzraz = /uzorak/;

Osim ovog zapisa, regularni izraz se može formirati preko konstruktora RegExp:
let regIzraz = new RegExp('uzorak');

Modifikator je jedan ili više simbola (mogu i sva tri simbola biti zajedno samo ih pišete jedan pored drugi) iz tablice "Modifikatori".

i	ignoriše razliku mala/velika slova
g	Obrađuje ceo string koji se ispituje, inače se ispitivanje završava kada se naiđe na prvo podudaranje sa šablonom podudaranje ili pretragu onda se staje
m	Obrađuje više redova

Uzorak regularnog izraza se formira koristeći posebnu sintaku opisanu narednim simbolima.

Uglaste zagrade.

Služi za nabrajanje karaktera ili navođenje opsega.

[abcd]	Šablon je bilo koji znak: a, b, c ili d
[^abcd]	Šablon je bilo koji znak koji nije: a, b, c ili d. (^ je negacija)
[0-9]	Cifra od 0 do 9
[a-z]	Sva mala slova od a do z
[A-Z]	Sva velika slova od A do Z
[a-zčćžšđ]	Sva mala slova od a do z uz srpska slova

Mala zagrada

Koristi se za grupisanje skupova pravila. Na primer
([1-9]|[a-z]) – šablon je cifra bez nule ili mali karakter od a do z.
Međutim, ako se karakteri zapisuju u malim zagradama, dakle van uglaste zagrade onda se ti karakteri grupišu u celinu koja se smatra šablonom.
Na primer:
(nrt|rt|is|avt|asuv|elite|net)
Šablon opisuje jedu od skraćenica koje su navedene.

Specijalni karakteri

.	bilo koji znak osim \n
\w	bilo koji alfanumerički znak: slova, brojevi i donja crta.
\W	bilo koji znak osim navedenih sa \w
\d	cifra
\D	Sve što nije cifra
\s	razmaci [\t\n\r]
\S	Sve što nisu razmaci
\b	ispitivanje na početku ili/i kraju reči
\B	Ispitivanje ali ne na početku i/ili kraju reči
\n	novi red
\r	pronadji povratak
\t	Pronadji tabulator

Karakteri podešavanja

n+	Označava ponavljanje jednom ili više puta prethodnog znaka. Na primer stringovi 12, i 1222 će odgovarati šablonu 12+, ali neće i string 1
n*	Nula ili više puta, odnosno, za prethodni primer odgovara i string 1
n?	Nula ili jedno ponavljanje.
n{X}	Znak n se ponavlja tačno X puta.
n{X,Y}	Opseg ponavljanja. Na primer a{1,3} pronalazi: a,aa,aaa
n{X,}	Pronalazi minimum X puta.
n$	Traži znak na kraju reči. Na primer abc$ podudariće se sa reči pegfabc
^n	Traži znak na početku reči. Na primer ^abc podudariće se sa reči abcdefr

match( )

Ova funkcija vraća podudaranja ako ih nadje, na primer izmedju regularnog izraza i stringa koji se kontroliše. Ako ne nadje vraća nulu. Evo primera:

```
<script>
var str="Ova i ovi su ovakvi ";
document.write(str.match(/ov/gi));
</script>
```

Rezultat: *Ov,ov,ov*

replace( )

Ova funcija traži podudaranje izmedju stringova, ili regularnog izraza i stringa, i ako nadje vrši zamenu tog nadjenog sa novin stringom. Evo primera:

```
<script>
      var str="Nikad neću reći nikad.";
      document.write(str.replace(/nikad/gi,
      "ipak"));
 </script>
```

Rezultat: *ipak neću reći ipak.*

search( )

search( ) funkcija traži podudaranje između regularnih izraza i stringa, i vraća položaj podudaranja: Znakove broji od 0. A ako ne nadje vraća -1.

```
<script>
  var str="Najbolji drugari: Joca, Aca, Pera
  i Gagi ";
  document.write(str.search(/Gagi/i));
```

```
</script>
```

Rezultat: *36*

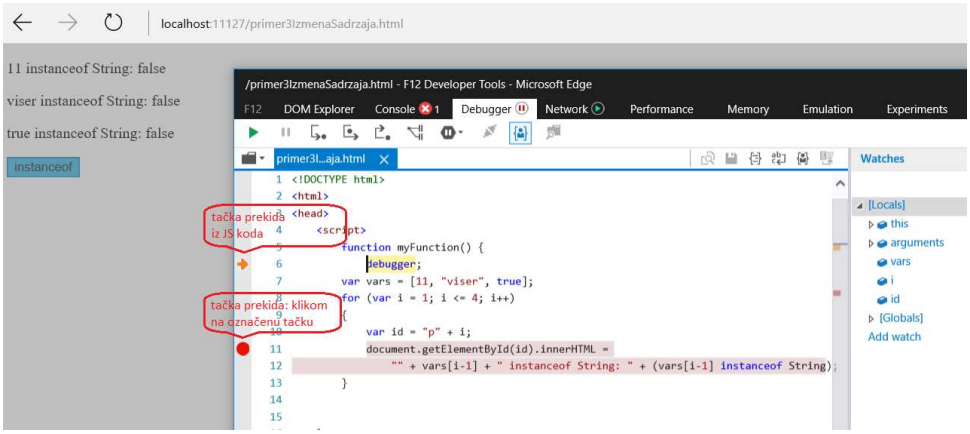
Otkrivanje grešaka

Otkrivanje grešaka u kodu naziva se i debagovanje. Novi web čitači imaju ugrađene mehanizme za debagovanje koji uključuju zaustavljanje izvršavanja na određenoj tački koda kao i izvršavanje liniju po liniju koda. Ovi mehanizmi se nazivaju još i debageri. Debageri mogu biti uključeni ili isključeni u toku izvršavanja koda. Kada su uključeni vrši se prikazivanje svih informacija o toku izvršavanja naročito vezanih za pojavu grešaka.

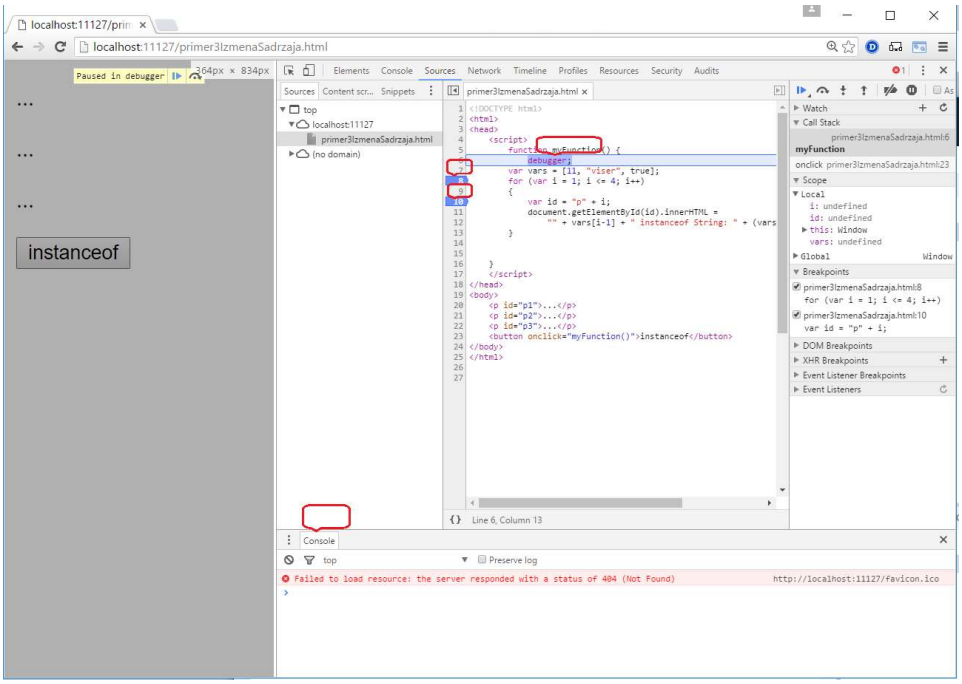
Pomoću debagera može se postaviti jedna ili više tačaka prekida (eng. *break point*) u kojima se program zaustavlja izvršavanje i u kojima se mogu pratiti stanja promenljivih. Rad pomoću debagera se ostvaruje prebacivanjem web čitača u poseban režim prikaza namenjen razvoju, a obično se može startovati skraćenicom sa tastature „F12“ .

Jedan od načina za ispis posebnih poruka u prozor **`console`** u toku debagovanja izvodi se primenom metode: **`console.log()`**.

Postavljanje tačaka prekida se izvodi u JS kodu pomoću komande **`debugger`**. Takođe, tačke prekida se mogu postaviti i direktno u prozoru debagera. Obično je dovoljan klik na liniju gde inače stoji oznaka za postavljenu tačku prekida ili nekom skraćenicom sa tastature.



Slika 8.2. Postavljanje tačaka prekida (eng. Break points)



Slika 8.3. Praćenje tekućih vrednosti i stanja

Objekti okruženja

Kada se JavaScript kod izvršava kod može pristupiti određenom sadržaju tj. izvršava se u određenom okruženju. Okruženje u kom se kod izvršava je okruženje nekog web čitača i obuhvata pristup sadržaju tekuće stranice, sadržajima otvorenih stranica ako ih ima više i na kraju pristup web čitaču.
Objekat window predstavlja jedan od objekata JavaScript-a koji su deo web čitača. Zajedno sa objektima screen, location, history čini BOM (eng. Browser Object Model) model. Neke važne metode i svojstva objekta window su:

- window.open() - otvara novi prozor,
- window.close() - zatvara tekući prozor,
- window.moveTo() - pomera tekući prozor,
- window.resizeTo() - menja veličinu tekućem prozoru,
- window.screen - predstavlja korisnikov ekran. Može se pisati bez prefiksa window, a neka njegova svojstva su:
  - screen.width - širina ekrana u pikselima,
  - screen.height - visina ekrana u pikselima,
  - screen.availWidth - širina ekrana u pikselima raspoloživa za kontrolu prikaza, npr. bez linije sa alatkama (taskbar),
  - screen.availHeight - visina ekrana u pikselima umanjena za delove interfejsa (npr. taskbar).
- window.print() – Metod koji obezbeđuje štampanje sadržaja tekućeg prozora.

Osim objekta window koji predstavlja prvi i najviši objekat u web čitaču važni objekti su:

- document – obuhvata tekući prikaz u web čitaču. Pristup ovom objektu znači pristup do svih elemenata tekućeg prikaza sa mogućnošću da se upravlja sa istim. Više detalja o metodama ovog objekta biće u poglavlju o DOM modelu. Neke metode su:
  - document.body – pristup body elementu,
  - document.forms - vraća sve <form> elemente,
  - document.head - vraća <head> element,
  - document.images - vraća sve <image> elemente.
- history – sadrži listu svih sajtova u istoriji web čitača. Omogućava kretanje unapred ili unazad po toj listi. Objekat history se može pisati bez prefiksa window (window.history isto je što i history).

Zbog zaštite privatnosti korisnika postoje ograničenja kako JavaScript pristupa ovom objektu. Osnovne metode:
  - history.back() - Isto kao klik na dugme back/nazad u pregledaču,
  - history.forward() - Isto kao klik na dugme forward/napred u pregledaču.
- location – čuva informacije o poziciji dokumenta koji se prikazuje. Lokacija obuhvata URL adresu kao i protokol, domen, putanju do fajla i port. Objekat location se može pisati bez prefiksa window. Osnovna svojstva su:
  - location.href - vraća href (URL) tekuće stranice,
  - location.hostname - ime domena,
  - location.pathname - Vraća putanju (path) i ime datoteke tekuće stranice,
  - location.protocol - web protokol koji se koristi (http:// ili https://).

DOM model

DOM model je skraćenica od engleskog naziva ***D**ocument **O**bject **M**odel*. U ovom modelu HTML dokument sa svim svojim sadržajima je predstavljen u obliku čvorova. Čvorovi su elementi, atributi, sadržaji, komentari. Čvorovi su hijerarhijski organizovani.

document

Korenski element je jedini element koji nema roditeljski element u hijerarhijskoj strukturi. U ovom modelu to je **`document`** objekat.

Objekat **`document`** sadrži svojstva i metode za pristup svim ostalim čvorovima preko JavaScript-a. Izuzetak: Dokument se prikazuje u prozoru čitača, pa se jedino sam objekat prozora čitača, **`window`**, može koristiti iznad dokumenta.

Do sada smo često koristili metodu **`getElementById()`** koja pripada dokumentu. U tabeli su navedene još neke često korišćene metode odnosno svojstva.

Svojstvo / Metod	Opis

document.activeElement	tekući element koji je u fokusu/aktivan,
document.body	**body** element,
document.cookie	parovi name/value svih kolačića u dokumentu,
document.createAttribute()	kreira jedan atribut,
document.createComment()	kreira komentar,
document.createElement()	kreira element,
document.forms	vraća kolekciju svih <form> elemenata,
document.getElementById()	element definisanog ID,
document.getElementsByClassName()	lista čvorova elemenata definisane klase,
document.getElementsByName()	vraća listu čvorova definisanog imena,
document.getElementsByTagName()	vraća listu čvorova definisanog tag naziva,
document.hasFocus()	vraća da li je dokument u fokusu tj. aktivan,
document.head	vraća element **<head>**,
document.images	vraća kolekciju svih **<img>** elemenata,
document.links	vraća kolekciju svih **<a>** i **<area>** elemenata koji imaju **href**,
document.open()	otvara HTML izlazni tok za prikupljanje sadržaja od **document.write()**,
document.querySelector()	vraća prvi element koji odgovara specifičnom **CSS** selektoru,
document.scripts	vraća kolekciju **<script>** elemenata,
document.title	vraća naslov,
document.URL	vraća puni **URL** tekućeg HTML dokumenta,

document.write()	piše HTML izraze ili JavaScript kod u dokument,
document.writeln()	isto kao write() samo dodaje novi red na kraju.