Математички факултет
Универзитет у Београду

Мастер рад

Електронски курс о програмском језику JavaScript

Студент: Никола Трипковић
Ментор: Мирослав Марић

Београд,
2015.

Комисија:
Доц. др Мирослав Марић - ментор
Проф. др Владимир Филиповић
Мр Сана Стојановић Ђурђевић

Садржај

Увод

Овај рад представља кратак приказ програмског језика JavaScript, једног од најчешће коришћених програмских језика, помоћу ког веб странице могу постати динамичније и корисније.
Уводни део рада је резервисан за кратко представљање програмског језика JavaScript, његов историјски развој и везу са програмским језиком Java.
Даље, од трећег до шестог поглавља, представљено је само језгро JavaScript-а, типови података које подржава JavaScript, декларисање и коришћење променљивих, контрола тока програма помоћу условних структура и петљи, као и декларисање и коришћење функција.
Објектно оријентисано програмирање на JavaScript-у је тема седмог и осмог поглавља. Ова два поглавља садрже дефиницију функције конструктора и објашњење механизама наслеђивања на основу прототипа, као и приказ објеката уграђених у JavaScript.
У другом делу курса говори се о клијентском JavaScript-у, тј. JavaScript-у који се користи у читачима веба који су описани као окружење за програмирање.
Од деветог до дванаестог поглавља се описују најважнији објекти читача веба и начини како се помоћу својстава и метода тих објеката може управљати прозорима читача и мењати садржај документа. Описани су и начини чувања и управљања подацима на страни клијента коришћењем колачића.
Изузеци и обрада изузетака је тема наредног поглавља, где је објашњено када може доћи до изузетака и како се они обрађују.
У последњем, тринаестом поглављу, објашњени су догађаји и процедуре за обраду догађаја, који представљају основу интерактивних програма на JavaScript-у.

1 Сајт Електронски курс о програмском језику JavaScript

Сајт Електронски курс о програмском језику JavaScript се налази на следећој адреси:

www.cleversolutions.rs/javascript/index.html

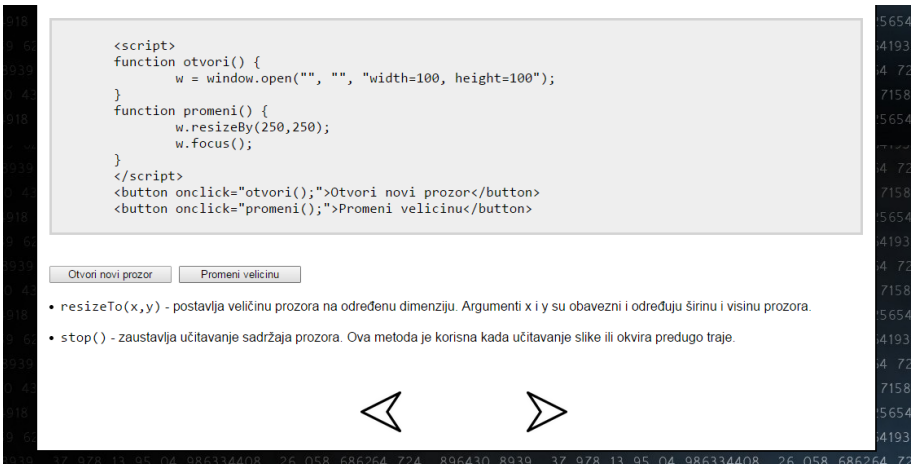
Почетна страна сајта је приказана на слици 1.

Слика 1. Почетна страна

На свакој страни се налази вертикални мени који садржи 12 наслова обрађених тема у електронском курсу о JavaScript-у. Од ставки у менију неке имају и подмените ради лакшег коришћења курса (Слика 2).

Слика 2. Вертикални мени са подменијем

Главни део садржи текст о изабраној теми и примере са кодовима на JavaScript-у.
Неки примери су директно уграђени у текућу страницу, па корисник, најчешће кликом на дугме, може видети резултат извршавања датог JavaScript кода (Слика 3).



Слика 3. Пример уграђен у текућу страницу

На дну сваке стране се налазе стрелице које воде ка претходној и следећој страни.

2 Увод у JavaScript

JavaScript припада групи такозваних скрипт језика. То су језици који се састоје од редова извршног рачунарског кода који се директно умећу у HTML странице. Примарно је дизајниран да би повећао интерактивност интернет страница и за креирање интернет апликација. Релативно нови програмски језик, такође из групе скрипт језика, је AJAX (Asynchronous JavaScript and XML) и његова основна карактеристика је што користи асинхрони пренос података (HTTP захтеве) између веб читача и сервера. Са AJAX-ом JavaScript програми могу директно комуницирати са сервером чинећи апликацију бржом и бољом.
WML (Wireless Markup Language) је језик настао из HTML-а али је заснован на XML-у и тиме доста ,,строжији". Овај скрипт језик служи за прављење страница које се приказују у WAP читачима и сматра се ,,лакшом"верзијом JavaScript-а.
Програмски језик изведен из Microsoft-овог језика Visual Basic назива се VBScript. Скриптови написани у овом језику умећу се у HTML, а веб читач читајући HTML чита и извршава скриптове VBScripta.
JavaScript је језик опште намене, користан и без HTML-а. Он се уграђује у сервере, развојне алате и веб читаче.
Најважније могућности које JavaScript пружа својим корисницима су:

- HTML дизајнерима пружа програмерски алат
- реагује на одређене догађаје (клик мишем, учитавање странице...)
- служи за читање и писање HTML елемената
- врши проверу података у обрасцу пре предаје серверу
- врши препознавање веб читача
- служи за прављење колачића (енг. cookies)

2.1 Историјат

Само име JavaScript може бити прилично конфузно за оне који не познају историју развоја JavaScript-а, јер упркос сличности у имену JavaScript нема никакве везе са програмским језиком Java.
NetScape је првобитно увео језик под називом LiveScript у бета издању Navigatora 2.0, 1995. године, и на почетку је служио за валидацију образаца. Језик је преименован у JavaScript због фасцинације целокупне јавности програмским језиком Java у том тренутку, као и због потенцијала

да заједно интегрисани служе за израду интернет апликација. Због речи Java у свом имену многи JavaScript сматрају неким смањеним обликом Jave.
Иако је назив језика доводио до забуне код неких корисника, JavaScript је био широко прихваћен од стране произвођача веб читача. Пошто је Netscape увео JavaScript у свој читач веба верзије 2.0, Microsoft је представио клон JavaScript-а назван JScript у Internet Explorer-u 3.0. Opera је такође представила JavaScript подршку током 3.х. генерације својих читача. Како је време пролазило, сваки од главних произвођача веб читача је направио своје додатке за JavaScript, и данас сви претраживачи имају подржане верзије JavaScript-а или JScript-а.
JavaScript је објектно оријентисан програмски језик са прототипским наслеђивањем. Подржава неколико објеката уграђених у сам језик, а програмери могу креирати своје објекте.

JavaScript је интерпретаторски језик са опционом JIT (енг. just in time) компилацијском подршком. У старијим имплементацијама (нпр. IE 8, Firefox 3.5 и ранији)
JavaScript је био чисто интерпретаторски језик. То значи да се скрипт извршавао без компајлирања, тј. без претварања текста скрипта у масхински код. Веб читач тумачи скрипт, тј. анализира га и одмах га извршава. У модерним имплементацијама JavaScript код се може интерпретирати или компајлирати користећи JIT компајлер. При покретању скрипта веб читач одлучује да ли ће код (или делови кода) бити компајлирани ради бољих перформанси. Ово знатно убрзава извршавање JavaScript програма чинећи га погодним за комплексне интернет апликације. Све новије верзије веб читача имају JavaScript JIT компајлер.

2.2 JavaScript и Java

Иако постоји сличност у имену, JavaScript и Java немају скоро ништа заједничко. У наредној табели упоредно су приказане основне карактеристике наведених програмских језика.

Табела 1. JavaScript и Java

JavaScript	**Java**
Најчешће се интерпретира на страни клијента	Компајлира се на серверу пре извршења код клијента
Објектно заснован. Користи постојеће објекте без класа и наслеђивања	Објектно оријентисан. Аплети садрже објекте састављене од класа са наслеђивањем
Код је интегрисан у HTML	Аплет је издвојен из HTML-а
Тип промењљиве се не декларише	Тип промењљиве мора бити декларисан
Динамичка повезаност. Објектне референце проверавају се у run-time	Статичка повезаност. Објектне референце морају постојати у време компајлирања
Заштићен. Не може записивати на хард диск	Заштићен. Не може записивати на хард диск

JavaScript је широко подржан у већини модерних веб читача:

- Netscape Navigator (почев од верзије 2.0)
- Microsoft Internet Explorer (почев од верзије 3.0)
- Firefox
- Safari
- Opera
- Google Chrome

Тако да већина корисника интернета данас има читаче који подржавају JavaScript. Зато је JavaScript један од најпопуларнијих програмских језика.

3 Основе језика

3.1 Изрази

Израз (енг. expression) је део кода програма који представља или рачуна неку вредност. Израз може бити константа, промењљива, позив

функције или више оваквих елемената у комбинацији са операторима. Неки једноставни изрази приказани су у следећем примеру:

```
1,7
"JavaScript programiranje"
true
null
[3, 5, 9, 12, 32, 54]
```

Сложенији изрази се могу правити комбинацијом једноставних израза, као на пример:

```
n+5
(i-3)*5
"tekst1"+"tekst2"
function(n) {return n*n;}
```

3.2 Коментари

JavaScript подржава коментаре у стилу језика C и C++. Сваки текст између // и краја реда сматра се коментаром, као што је приказано у наредном примеру:

```
// Ovo je komentar u jednom redu
```

Такође, сваки текст између знакова /* */ се сматра коментаром и ови коментари могу заузимати више редова, као на пример:

```
/* Ovo je komentar
koji zauzima
vise redova */
```

3.3 Типови података

Иако је JavaScript првенствено намењен за рад са текстом у облику HTML страница унутар веб читача, типови података које JavaScript нуди иду много даље од онога што је потребно за наведени задатак. JavaScript нуди широк спектар типова података који се могу наћи у другим модерним програмским језицима, као и робустан скуп функција са којима манипулише.
JavaScript подржава пет примитивних типова података:

- Numbers - цели бројеви и бројеви са покретним зарезом

- String - стрингови, тј. ниске карактера
- Boolean - логички тип, дефинише податке који могу имати вредности тачно или нетачно
- Undefined
- Null

Ови типови података се називају примитивнима јер су основни градивни блокови од којих се граде сложенији типови података. Од ових типова бројеви, стрингови и логички су реални типови података у смислу складиштења података. Undefined и Null су типови који се јављају под посебним околностима.
Поред основних типова података, JavaScript подржава и сложене типове података као што су објекти (Object) и низови (Arrays). Иако у суштини објекти и низови представљају исти тип података у JavaScript-у различито се понашају па се о њима углавном говори као о подацима различитог типа.

3.3.1 Бројеви

За разлику од језика као што су C и Java, JavaScript не прави разлику између целих бројева и бројева са покретним зарезом. Сви бројеви су представљени помоћу 64-битног формата са покретним зарезом дефинисаног стандардом *IEEE* 754. Ова репрезентација представља целе бројеве у опсегу од -2^{53} до 2^{53} и бројеве са покретним зарезом од $\pm 1,7967 \cdot 10^{308}$ до $\pm 2,2250 \cdot 10^{-308}$.
Бројеви у JavaScript-у могу бити представљени у три формата: у декадном (база 10), у окталном (база 8) и хексадецималном (база 16).

Декадни бројеви се представљају као секвенца цифара $(0-9)$.

Октални цели бројеви се представљају као низ цифара $(0-7)$ предвођен нулом. На пример, број 213 се у окталном систему представља као:

```
0325 // 5+2*8+3*64 = 213
```

Хексадецимални цели бројеви се представљају као низ цифара $(0-9)$ и слова (a-f или A-F) предвођен нулом коју следи слово x („0x"или „0X"). Слова a-f, односно A-F представљају бројеве од 10 до 15. Број 205 представљен у хексадецималном бројевном систему:

```
0xcd // 13+12*16 = 205
```

Пример представљања броја десет (10) у три бројевна система је:

```
Dekadni: 10
Oktalni: 012
Heksadecimalni: 0xA
```

Бројеви у формату са покретним зарезом могу се представити као реални бројеви са децималним зарезом. За њих се користи традиционална синтакса реалних бројева, као што се може видети у наредном примеру:

```
2,13
-888,38
```

Бројеви у овом формату могу се представити и помоћу експоненцијалног формата: прво се наводи реалан број, потом слово e или E, знак минус или плус и целобројни експонент. Оваквом нотацијом представљен је број помножен са 10 подигнутим на степен одређен експонентом, као на пример:

```
3,21e24 // 3,21x10^23
1,9847E-30 // 1,9847x10^-30
```

3.3.2 Специјалне нумеричке вредности

Када број има вредност већу од највећег броја који се може представити у JavaScript-у, придружује му се посебна бесконачна вредност са идентификатором `Infinity`. Слично томе, за негативне вредности добија се негативна бесконачна вредност `-Infinity`.
Друга специјална нумеричка вредност у JavaScript-у је `NaN`. Ова вредност се користи када је резултат математичке операције недефинисан или грешка (нпр. дељење нулом).
За испитивање да ли је нека вредност једнака `NaN` користи се функција `isNaN()`. Слична функција је `isFinite()` која проверава да ли је број различит од `NaN` и од позитивне и негативне бесконачне вредности.

3.3.3 Стринг тип

Стринг или ниска је уређен низ слова, цифара, интерпункцијских и других знакова ограничен наводницима или полунаводницима, као у следећем примеру:

```
" " // Prazan string
'poluprecnik'
"1024"
"true"
```

JavaScript не прави разлику између једног карактера и низа карактера, па је карактер `s` у наредном примеру типа стринг:

```
"s" // String duzine jedan
```

Стрингови могу садржати наводнике уколико се ставе између полунаводника (и обрнуто) као у следећем примеру:

```
'grad = "Beograd" '
"poruka = 'Dovidjenja!' "
```

У променљиву типа стринг се могу уградити и escape знакови. Escape знак је знак са специјалним значењем. У табели 2 приказани су неки escape знакови и њихова значења.

Табела 2. Escape знакови

Секвенца	Значење
/b	Знак за брисање уназад
/t	Хоризонтални табулатор
/n	Знак за нови ред
/v	Вертикални табулатор
/”	Наводник
/’	Полунаводник
/ /	Обрнута коса црта

3.3.4 Логички тип

За разлику од бројева и знаковних низова који имају неограничено много могућих вредности, логички тип може имати само две вредности - `true` (тачно) или `false` (нетачно).
Логичке вредности се обично добијају као резултат неког поређења на следећи начин:

```
s == 4; // Da li je s jednako 4
```

Ако је вредност променљиве `s` једнака броју 4, резултат поређења је логичка вредност `true`. У супротном, резултат поређења је `false`.
Логичке вредности се лако конвертују у друге типове и обрнуто, што

се често и аутоматски дешава. Ако се логичка вредност користи у нумеричком контексту, вредност `true` се конвертује у број 1, а вредност `false` у 0.
Уколико дође до конверзије у знаковни низ, вредност `true` се конвертује у знаковни низ "true", а вредност `false` у знаковни низ "false".
Број се конвертује у логичку вредност `true` ако није једнак 0 или `NaN` - те две вредности се конвертују у `false`.
Знаковни низ се увек конвертује у `true` осим у случају када је празан знаковни низ, и тада се конвертује у `false`.
Вредности `null` и `undefined` се увек конвертују у вредност `false`.
Било који објекат, низ или функција чија је вредност различита од `null` конвертују се у вредност `true`.
Логички тип је добио назив по имену енглеског математичара Була (George Boole, 1815-1864) који се сматра оснивачем математичке логике.

3.4 Променљиве

Променљиве су места у меморији у којима подаци могу да се чувају да би програм могао да их користи. Вредност променљиве може да се мења за време извршавања програма, па отуда и назив - променљива.
Да би било могуће радити са променљивима свакој променљивој је додељен идентификатор, име које се односи на њену вредност и омогућава скрипту да приступи и врши манипулацију над подацима. Идентификатор променљиве називамо именом променљиве.
Имена променљивих у JavaScript-у се могу састојати од произвољне комбинације слова и бројева, али постоје нека ограничења:

- Први знак мора бити слово енглеског алфабета или доња црта
- Резервисане речи не могу се користити као име променљиве
- Велика и мала слова се разликују, уобичајено је да се променљиве пишу малим словима

Резервисане речи

Резервисане речи у JavaScript-у приказане су у следећој листи:

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

Пре него што се променљива употреби у програму, мора се декларисати. Променљиве се декларишу помоћу резервисане речи `var` која претходи имену променљиве, на следећи начин:

```
var suma;
var e_mail;
```

Помоћу једне резервисане речи `var` може се декларисати више променљивих:

```
var suma, e_mail;
```

А приликом декларације променљивима се могу доделити почетне вредности као у следећем примеру:

```
var suma = 0;
var e_mail = "student@matf.bg.ac.rs";
```

JavaScript омогућава имплицитно декларисање променљивих, тј. без коришћења кључне речи `var`. Када преводилац види да програм вероватно држи податке у некој променљивој која није декларисана, он јој аутоматски додељује простор у меморији. Иако не представља добру програмерску праксу, следећи код неће довести до грешке:

```
/* Deklarisu se promenljive bez
koriscenja kljucne reci var */
ime = "Milica";
prezime = "Petrovic";
```

У већини језика на високом нивоу, као што су C или Java, променљива мора бити декларисана пре него што се користи, а тип променљиве мора бити одређен приликом декларисања. Када се променљива декларише, тип променљиве се не може мењати.
Свака променљива у JavaScript-у има тип, али типови података нису експлицитно декларисани. На пример, може се дефинисати променљива `duzina` која ће да чува растојање између два града и доделити јој вредност 130. Онда се може тој истој променљивој доделити стринг вредност ,,Beograd", као у следећем примеру:

```
// Deklarise se promenljiva duzina, tip podataka je undefined
var duzina;
// Tip podataka promenljive "duzina" je number
duzina = 130;
// Sada je tip podataka promenljive "duzina" string
duzina = "Beograd";
```

Као што се може видети, тип података се може закључити из садржаја, а тип променљиве се аутоматски преузима од типа податка који она садржи.
Тип променљиве може се експлицитно задати на следећи начин:

```
var duzina : number;
```

Променљива `duzina` је сада нумеричког типа и може јој се доделити било која нумеричка вредност. У случају додељивања неке друге вредности која није нумеричка, као у следећем примеру, програм ће упозорити на грешку:

```
// Ispravna dodela
duzina = 5;
// Program ce prijaviti gresku
duzina = "Novi Sad";
```

3.4.1 Област важења променљивих

Област важења променљивих је део програма у којем важи декларација те променљиве. Глобална променљива има глобалну област важења и њена дефиниција је важећа у читавом програму. Променљиве декларисане у функцији дефинисане су само у телу те функције. То су локалне променљиве. Параметри функције се такође сматрају локалним променљивима и њихова област важења је тело функције.
За разлику од програмских језика C , C++ , Java , у JavaScript-у не

постоји блок као област важења променљивих. Све променљиве декларисане у функцији, без обзира на то где су декларисане, дефинисане су у оквиру функције. У примеру 1 променљиве `i,j,k` имају исту област важења:

```
function test(n) {
    // Definicija promenljive i vazhi u funkciji
    var i = 0;
    if(n < 0) {
        /* Definicija promenljive j vazhi svuda u
        funkciji, ne samo u ovom bloku */
        var j = 0;
        for(var k = 0; k < 10; k++) {
            // I promenljiva k vazhi svuda, ne samo u petlji
            document.write(k);
        }
        document.write(k);

        }
    // Promenljiva j josh uvek vazhi
    document.write(j);
}
```

Пример 1. Област важења променљивих

3.5 Оператори

Оператори су симболи који омогућавају извршавање операција над вредностима и променљивима.
Оператори могу имати један, два или три аргумента, с тим што већина има два. На пример, оператор доделе има два аргумента - меморијску локацију на левој страни симбола = и израз на десној страни. Ти аргументи се називају операнди - вредности којима се оперише.
Оператори се могу поделити и према броју операнада са којима раде. Већина оператора су бинарни оператори, што значи да комбинују два израза у један, сложенији израз. Постоје и тзв. унарни оператори, који конвертују један израз у други. JavaScript подржава и један тернарни оператор, то је условни оператор ,, : ? ", који комбинује вредности три израза у један.
Оператори имају свој приоритет и асоцијативност.

3.5.1 Аритметички оператори

Аритметички оператори су прилично једноставни - то су обични знакови за рачунске операције. Приказани су у табели 3.

Табела 3. Аритметички оператори

Оператор	Опис	Пример
$+$	Сабирање	$x + y$
$-$	Одузимање	$x - y$
$*$	Множење	$x * y$
$/$	Дељење	x/y
$\%$	Модуло	$x\%y$
$++$	Увећање за 1	$++x, x++$
$--$	Умањење за 1	$--x, x--$

Оператор + сабира нумеричке операнде или надовезује стрингове. Ако је један операнд стринг, други се конвертује у стринг, па се надовезују један на други као у следећем примеру:

```
var x = 10;
var y = "broj";
// Promenljiva z ce imati vrednost "10broj";
var z = x + y;
```

Ако су оба операнда стрингови, оператор + их надовезује. На пример:

```
tekst1="Dobar ";
tekst2="dan";
// Promenljiva tekst3 ce biti "Dobar dan"
tekst3 = tekst1 + tekst2;
```

Оператор − одузима други операнд од првог. Уколико се одузимају ненумерички операнди, JavaScript покушава да их конвертује у бројеве.
Оператор $*$ множи своја два операнда. Ако се користи са ненумеричким операндима, покушава да их конвертује у бројеве.
Оператор / дели први операнд другим. Ако се користи са ненумеричким операндима, покушава да их конвертује у бројеве. Како су у JavaScript-у сви бројеви у формату са покретним зарезом, резултат сваког дељења ће увек бити у том формату. Резултат израза 11/2 ће бити 5.5, а не 5, као што би то било у програмском језику *C*.
Оператор % (модуло) рачуна остатак дељења првог операнда другим. На пример, резултат израза 5%2 је 1.
Овај оператор се најчешће користи са целобројним операндима, али

функционише и са вредностима у формату са покретним зарезом. На пример, вредност израза $6,4\%2,1$ је $0,1$.
Унарни оператори $+$ и $-$ се користе испред једног операнда и конвертују његову вредност у позитивну, односно негативну, исте апсолутне вредности.
Оператор $++$ увећава свој једини операнд за 1. Ако се оператор $++$ користи испред операнда, увећава тај операнд за 1 и враћа његову увећану вредност. Уколико се користи иза операнда, увећава операнд за 1, али враћа вредност која је била пре увећања.
Оператор $--$ умањује свој нумерички операнд за 1. Ако се користи испред операнда, умањује тај операнд и враћа његову умањену вредност. Ако се користи иза операнда, умањује тај операнд, али враћа његову неумањену вредност. Пример 2 показује коришћење оператора $++$ и $--$.

```
// Vrednost promenljive x je 1
x = 1;
// Vrednost promenljivih x i y je 2
y = ++x;

// Vrednost promenljive m je 5
m = 5;
/* Vrednost promenljive n je 5
a promenljive m je 4 */
n = m--;
```

Пример 2. Коришћење оператора $++$ и $--$

3.5.2 Оператори доделе

Табела 4. Оператори доделе

Оператор	Пример	Објашњење
$=$	$x = y$	$x = y$
$+=$	$x+ = y$	$x = x + y$
$-=$	$x- = y$	$x = x - y$
$*=$	$x* = y$	$x = x * y$
$/=$	$x/ = y$	$x = x/y$
$\%=$	$x\% = y$	$x = x\%y$

Оператор $=$ додељује променљивој са леве стране вредност произвољног типа на десној страни. У следећем примеру, променљивој `x` је додељена вредност 10:

```
x = 10;
```

Оператор доделе је десно асоцијативан, па се може користити у сложенијим изразима, као на пример:

```
// Promenljivima x, y, z je dodeljena vrednost 0
x = y = z = 0;
```

Оператор + = се може користити за бројеве или за знаковне низове. Израз:

```
cena += pdv;
```

једнак је следећем изразу:

```
cena = cena + pdv;
```

Слично се понашају и оператори `-=`, `/=`, `*=`, `%=`.

3.5.3 Оператори једнакости

У табели 5 су приказани оператори једнакости и неједнакости, који пореде две вредности покушавајући да одреде да ли су једнаке или се разликују, и враћају логичке вредности `true` или `false` у зависности од резултата поређења.

Табела 5. Оператори једнакости

Оператор	Опис
==	Једнако
===	Идентично
! =	Неједнако
! ==	Неидентично

Оператор једнакости == проверава да ли су два операнда једнака. Враћа `true` ако јесу или `false` ако нису.
Једнакост две вредности у случају оператора == одређује се по наредним правилима.

- Ако су две вредности истог типа, проверава се њихова идентичност. Уколико су две вредности идентичне, једнаке су; уколико нису идентичне, нису ни једнаке.

- Ако две вредности нису истог типа, могу бити једнаке. Једнакост се проверава на основу наредних правила и конверзије типова

  - Ако је једна вредност `null`, а друга `undefined`, једнаке су.
  - Уколико је једна вредност број, а друга стринг, стринг се конвертује у број, и понавља се поређење.
  - Ако је нека вредност једнака `true`, конвертује се у 1 и понавља се поређење. Уколико је нека од вредности `false`, конвертује се у 0 и понавља се поређење.
  - Ако је једна вредност објекат, а друга је број или стринг, конвертује се објекат у основни тип и понавља се поређење. Објекат се конвертује у основни тип помоћу метода `toString()` или `valueOf()`.
  - У свим другим случајевима, променљиве нису једнаке.

```
5 == "pet"; // false
8.0 == 8; // true
"1" == true; // true
null == undefined; // true
```

Пример 3. Коришћење оператора ==

Оператор идентичности === утврђује да ли су два оператора идентична према строгој дефиницији идентичности која не дозвољава конверзију типова. Такође враћа `true` ако јесу или `false` у супротном.
Идентичност две вредности у случају оператора === одређује се по наредним правилима.

- Уколико су две вредности различитог типа, нису идентичне.
- Ако су обе вредности бројеви и имају исте вредности, идентичне су, сем ако је бар једна од њих `NaN` - у том случају нису идентичне. Вредност `NaN` никада није идентична ни једној другој вредности, па ни самој себи.
- Када су обе вредности стрингови и садрже исте знакове на истим позицијама, идентичне су. Уколико се стрингови разликују по дужини или садржају, нису идентични.
- Ако су обе вредности једнаке логичкој вредности `true` или логичкој вредности `false`, идентичне су.

- Уколико обе вредности референцирају исти објекат, низ или функцију, идентичне су. Уколико референцирају различите објекте (низове, функције), нису идентичне, чак и ако оба објекта имају идентична својства или оба низа имају идентичне елементе.

- Када су обе вредности `null` или `undefined`, идентичне су.

```
5 === "pet"; // false
8.0 === 8; // true
"1" === true; // false
null === undefined; // true
```

Пример 4. Коришћење оператора ===

Оператори ! = и ! == раде супротно од оператора == и ===. Оператор неједнакости ! = враћа вредност `false` ако су две вредности једнаке, у супротном враћа `true`.
Оператор ! == враћа вреност `false` ако су две вредности идентичне, иначе враћа `true`.

3.5.4 Релациони оператори

Релациони оператори се користе за поређење две вредности. Изрази у којима се употребљавају ови оператори враћају логичку вредност `true` или `false`, у зависности од резултата поређења. Приказани су у табели 6.

Табела 6. Релациони оператори

Оператор	Опис
>	Веће
<	Мање
>=	Веће или једнако
<=	Мање или једнако

Могу се поредити само бројеви и стрингови, па операнди који нису тог типа морају се конвертовати.
Стрингови се пореде знак по знак, тако што се упоређују вредности нумеричких Unicode кодова знакова, као што се може видети у следећем примеру:

```
9 > 5 // true
"a" > "C" // false
```

3.5.5 Логички оператори

Логички оператори се користе за комбиновање резултата логичких израза.

Табела 7. Логички оператори

Оператор	Опис
&&	Логичка конјукција
||	Логичка дисјункција
!	Логичка негација

Логичка конјукција && враћа вредност `true` ако и само ако оба операнда имају вредност `true`. Уколико бар један операнд има вредност `false` резултат је `false`.

Табела 8. Логичка конјукција

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

Логичка дисјункција || враћа вредност `true` ако је вредност барем једног оператора `true`. Само уколико је вредност оба операнда `false`, резултат је `false`.

Табела 9. Логичка дисјункција

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Логичка негација ! је унарни оператор, тј. има само један операнд и његова сврха је инверзија логичке вредности. Ако променљива `x` има вредност `true`, резултат израза `!x` биће `false` и обрнуто.

3.5.6 Условни оператор

JavaScript подржава још један оператор, тзв условни оператор који додељује вредност некој променљивој на основу неког услова. Синтакса је следећа:

```
operand1 ? operand2 : operand3
```

Први операнд мора бити логичка вредност или се може конвертовати у логичку вредност. Обично је то резултат поређења. Други и трећи операнд могу имати било какву вредност.
Резултат зависи од логичке вредности првог операнда. Уколико је `true`, резултат условног израза је вредност другог операнда. У супротном, ако први операнд има вредност `false`, резултат условног израза је вредност трећег операнда. У следећем примеру, ако је `x` мање од 0, променљива `znak` ће узети вредност `-`, у супротном ће имати вредност `+`.

```
znak = x<0 ? "-" : "+"
```

3.5.7 Приоритет и асоцијативност оператора

- Унарни оператори имају виши приоритет од бинарних. У следећем примеру унарни оператор $-$ има виши приоритет од бинарног оператора $+$:

  ```
  - 3 + 5 // = 2
  ```

- Аритметички оператори су вишег приоритета од релационих што се може видети у следећем примеру:

  ```
  3 + 5 < 6 // false
  ```

- Релациони оператори су вишег приоритета од логичких. У примеру који следи, прво се врши поређење, па онда логичка дисјункција:

  ```
  0 < -1 || 0 // false
  ```

- Оператори доделе су најнижег приоритета. У наредном примеру, прво се врши поређење па онда додела:

  ```
  x = 3 > 2 // x ima vrednost true
  ```

- Ако два оператора имају исти приоритет, онда се у обзир узима асоцијативност која може бити с лева на десно и с десна на лево.

- Приоритет оператора се може променити коришћењем заграда.

3.6 Контролне структуре и петље

Управљачке структуре омогућавају управљање током извршења програма или скрипта. Груписали смо их у условне (структуре са гранањем) и структуре са понављањем (петље).

3.6.1 Наредба if

Наредба `if` се може употребљавати за доношење одлуке у зависности од одређеног услова. Основни облик ове наредбе је:

```
if (uslov)
    naredba;
```

Уколико је резултат услова у загради `true` или се може конвертовати у вредност `true`, извршава се наредба која следи. Уколико је вредност услова у загради `false` или се може конвертовати у `false`, наредба која следи се не извршава. У примеру који следи, ако корисничко име није регистровано, додељује му се име `Gost`:

```
if (username == null)
    username = "Gost";
```

Након `if` наредбе може следити блок наредаба, уоквирен витичастим заградама, као у следећем примеру:

```
if (n != 0) {
    zbir = zbir / n;
    n --;
}
```

Када је услов испуњен, извршавају се све наредбе у блоку, у супротном, ако услов није испуњен, ни једна наредба у блоку неће бити извршена.

3.6.2 Наредба else

Наредба `else` је нераскидиво повезана са `if` наредбом. Њоме се дефинише акција коју треба предузети ако услов наредбе `if` није испуњен. Основна синтакса је:

```
if (uslov)
    naredba1;
else
    naredba2;
```

Ако је вредност услова у загради `true` извршава се `naredba1`, у супротном, извршава се `naredba2`.
Пример 5 показује поступак одређивања апсолутне вредности броја коришћењем конструкције `if/else`.

```
if ( x >= 0)
    var apsolutna_vrednost = x;
else
    apsolutna_vrednost = -x;
```

Пример 5. Одређивање апсолутне вредности броја

Наредба `if/else` може да, зависно од резултата, изврши једну од две опције у коду. Ако је потребно извршити једну од више опција примењују се вишесмерна тестирања. Основна синтакса је:

```
if (uslov1) {
    // Blok naredbi #1
}
else if (uslov2) {
    // Blok naredbi #2

else if (uslov3) {
    // Blok naredbi #3
}
else {
    /* Ako ni jedan uslov nije ispunjen
    izvrshava se blok naredbi #4 */
}
```

3.6.3 Петља while

Као што је наредба `if` основна наредба која омогућава програму да доноси одлуке, тако је наредба `while` основна наредба за понављање акција. Петља `while` је најједноставнија врста петље у JavaScript-у. Она зависи само од једног услова и све док је тај услов испуњен извршава се блок наредби. Основни облик ове наредбе је:

```
while (uslov) {
    izraz;
}
```

Услов у загради се тестира и ако је испуњен тело петље се извршава. Затим се услов поново тестира и ако је испуњен поново се извршава тело петље. И тако све док услов није испуњен. Тада се излази из петље и наставља са првом следећом наредбом у програму. Следећи пример рачуна збир првих 100 природних бројева коришћењем `while` петље:

```
// Zbir prvih 100 prirodnih brojeva
var n = 0;
var suma = 0;
while (n <= 100) {
    suma += n;
    n ++;
}
```

Пример 6. Рачунање збира првих 100 природних бројева

Ако нема блока наредби уоквирених витичастим заградама након `while` петље, онда се прва наредба иза `while(uslov)` третира као тело петље. На пример, у следећем сегменту кода:

```
while (i<j)
i++;
j--;
```

само наредба `i++` се понавља у оквиру `while` петље, док се наредба `j--` извршава након изласка из `while` петље.

3.6.4 Петља do..while

Општа структура петље `do..while` је:

```
do
    izraz
while (uslov)
```

Ова петља се разликује од петље `while` због тога што се услов испитује на крају. То значи да се наредба или блок наредби унутар петље извршавају најмање једанпут.

```
var n = 10;
var faktorijel = 1;
do {
    faktorijel *= n;
```

```
    n--;
}
while (n);
```

Пример 7. Рачунање факторијела броја 10

3.6.5 Петља for

Основна структура `for` петље је:

```
for (izraz1; uslov; izraz2)
    izraz3;
```

- `izraz1` се извршава једном на почетку. Њиме се обично задаје почетна вредност бројача.
- `uslov` се испитује пре сваке итерације. Ако није испуњен извршавање петље престаје. Обично се испитује да ли је бројач дошао до одређене границе која је задата.
- `izraz2` се извршава на крају сваке итерације. Он обично мења вредност бројача.
- `izraz3` се извршава једном у свакој итерацији.

```
var zbir = 0;
for (var i = 0; i <= 100; i = i + 2)
    zbir += i;
```

Пример 8. Рачунање збира свих парних бројева мањих од 100

Вредности `izraz1`, `uslov`, `izraz2` могу бити изостављени, а наредба `for(; ;)` је циклус који се извршава бесконачно пута.

3.6.6 Наредба switch

Наредба `switch` делује слично као наредба `if`, али омогућава да услов има више од две вредности. То је наредба која омогућава да се тестира једна променљива, а затим се извршава једна од неколико грана зависно од њене вредности.
Наредба `switch` је слична истоименој наредби у програмским језицима Java или C . Основна синтакса наредбе `switch` је:

```
switch (izraz) {
    case vrednost1: naredba1;
        break;
    case vrednost2: naredba2;
        break;
            . . .
    deafult: podrazumevana naredba;
        break;
}
```

Како функционише наредба `switch`? Најпре се рачуна вредност израза `izraz`, а затим тражи лабелу `case` која одговара израчунатој вредности. Ако нађе лабелу са одговарајућом вредношћу, извршава наредбу која следи после те лабеле `case`.
Ако не нађе одговарајућу лабелу `case`, извршава наредбе које следе после специјалне наредбе `default`. Ако нема наредбе `default` прескаче се цео блок кода.

```
switch (znak) {
    case +:
        z = x + y;
        break;
    case -:
        z = x - y;
        break;
    case *:
        z = x * y;
        break;
    case /:
        z = x / y;
        break;
}
```

Пример 9. Коришћење наредбе `switch`

3.6.7 Наредба with

Наредба `with` има следећу синтаксу:

```
with (objekat)
    naredbe
```

У пракси наредба `with` скраћује код програма. На пример, математичким функцијама мора претходити објекат `Math`. У следећем примеру подразумева се `Math` испред функција и константи:

```
with (Math) {
    x = cos(a); // x = Math.cos(a)
    y = sin(a); // y = Math.sin(a)
    b = PI / 4; // b = Math.PI/4
}
```

3.6.8 Наредбе break и continue

Наредба `break` прекида петљу у којој се налази или наредбу `switch`. Има веома једноставну синтаксу:

```
break;
```

У примеру 10 траже се елементи низа одређене вредности. Петља се нормално завршава када дође до краја низа, али уколико пронађе жељену вредност у низу, прекида се помоћу наредбе `break`. Претпоставља се да низ `a` има 100 елемената.

```
for (i = 0; i < 100; i++) {
    if (a[i] == target)
        break;
}
```

Пример 10. Претрага низа

Наредба `continue` слична је наредби `break`. Међутим, уместо да прекине петљу, ова наредба започиње нову итерацију петље. Синтакса наредбе `continue` је, попут синтаксе наредбе `break`, веома једноставна:

```
continue;
```

Наредба `continue` може се користити само у телу петљи `while`, `do/while`, `for` и `for/in`. Коришћење на било ком другом месту доводи до синтаксне грешке. Наредни пример показује коришћење наредбе `continue` у `for` петљи:

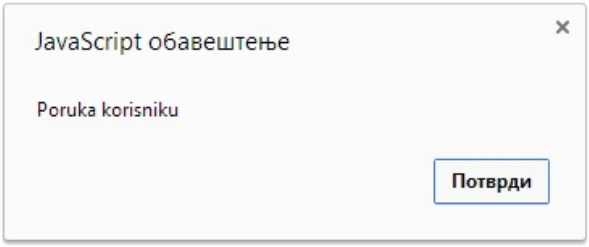
```
for (i = 0; i < a.length; i++) {
    if (a[i] == null)
        continue;
    suma += a[i];
}
```

3.7 Искачући прозор alert

Иако се о искачућим прозорима детаљније говори у наставку, у неким од наредних примера се користи искачући alert прозор. Основна синтакса за позивање искачућег alert прозора је:

```
alert("Poruka korisniku");
```

Изглед alert прозора зависи од веб читача, а на слици 4 је приказан alert прозор читача Gogle Chrome.



Слика 4. Alert прозор

4 Функције

Функција је блок кода који се дефинише једном, али се може позвати, тј. извршавати произвољан број пута. JavaScript подржава велики број уграђених функција, али корисник може дефинисати своје функције.
Функција се најчешће дефинише помоћу наредбе `function`. Ова наредба се састоји од резервисане речи `function`, после које следи име функције, затим ниједно или више имена параметара, унутар обичних заграда, раздвојених зарезима и тело функције унутар витичастих заграда које садржи наредбе на JavaScript-у:

```
function funkcija1(parametar1, parametar2, parametar3...) {
    Telo funkcije
    koje sadrzi
    JavaScript naredbe
}
```

Име функције мора бити идентификатор, не знаковни низ или израз. Аргументи унутар обичних заграда се прослеђују функцији приликом позивања и користе се у телу функције.
Тело функције се састоји од произвољног броја JavaScript наредби унутар витичастих заграда. Те наредбе се не извршавају приликом дефинисања функције, већ се преводе и повезују са том функцијом тек када

се она позове.
Витичасте заграде су обавезан део наредбе `function`.
Наредба `return` одређује вредност коју функција враћа. У следећем примеру, функција враћа збир аргумената који су јој прослеђени:

```
function saberi(x,y) {
    return x+y;
}
```

Наредба `return` се може налазити само у телу функције, у супротном прави се синтаксна грешка.
Извршавање функције се завршава када се изврши наредба `return`, чак и ако је у телу функције преостало још израза. Наредба `return` се може користити и без израза, ако је потребно само прекинути извршавање функције, и у том случају функција враћа вредност `undefined`.
Функција не мора да враћа вредност, другим речима у телу функције не мора постојати наредба `return`, као на пример функција чији је резултат извршавања искачући alert прозор:

```
function obavesti(poruka) {
    alert(poruka);
}
```

Пошто се дефинише, функција се може позивати помоћу оператора ().
Тако, на пример, претходна функција `saberi` се може позвати на следећи начин:

```
var rezultat = saberi(3,5);
```

Функцији сабери се прослеђују два аргумента, 3 и 5, а функција враћа њихов збир који се смешта у променљиву `rezultat`.
Пошто JavaScript није језик са строгим системом типова, не захтева се да се прецизира тип параметара функције, а такође не проверава да ли је прослеђен тип података који функција очекује.
JavaScript не проверава ни да ли је прослеђен прави број аргумената. Ако се проследи више аргумената него што функција очекује, она ће занемарити вишак аргумената. Уколико се проследи мање аргумената него што функција очекује, параметрима који су изостављени додељује се вредност `undefined`.

4.1 Аргументски објекат

У телу функције, идентификатор `arguments` представља такозвани аргументски објекат, који омогућава да се вредностима аргумената прослеђених функцији приступа помоћу броја уместо преко имена.

Иако се функција у JavaScript-у дефинише са одређеним бројем именованих аргумената, може јој се проследити произвољан број аргумената приликом њеног позивања. На пример:

```
// Funkcija sa jednim argumentom
function nekaFunkcija(x) {
    kod funkcije
}
// Funkcija se poziva i prosledjuju joj se 2 argumenta
nekaFunkcija(x,y);
```

Првом аргументу `x`, у претходном примеру, може се приступити преко имена параметра `x`, или помоћу израза `arguments[0]`. Другом аргументу се може приступити само помоћу израза `arguments[1]`.
Поред тога, објекат `arguments` има својство `length` које одређује број аргумената функције. Ово својство се најчешће користи да утврди да ли је функција позвана са одговарајућим бројем аргумената, пошто JavaScript то не проверава.

```
function f(a, b, c) {
    if (arguments.length != 3) {
        // Ako nisu prosledjena 3 argumenta
        // prekida izvrsavanje
        return;
        }
    // Sledi funkcija...
}
```

Пример 11. Коришћење својства `length` *објекта* `arguments`

Објекат `arguments` пружа важну могућност за JavaScript функције - може се написати функција која ради са произвољним бројем аргумената. У примеру 12 представљена је функција `max` која прихвата произвољан број аргумената и враћа вредност највећег прослеђеног аргумента.

```
function max () {
    m = arguments[0];

    for(var i = 0; i < arguments.length; i++) {
        if (arguments[i] > m)
                m = arguments[i];
        }
```

```
            // Vraca najveci argument
            return m;
}
```

Пример 12. Функција која проналази највећи аргумент

5 Објекти

JavaScript је језик базиран на објектима. Са изузетком конструкција као што су петље и оператори, скоро све JavaScript могућности су, у већој или мањој мери, имплементиране коришћењем објеката. Понекад се објекти експлицитно користе за обављање одређених задатака, као на пример манипулација (X)HTML објектима, док је у другим случајевима улога објеката мање очигледна, као на пример улога Number објекта током рада са бројевима.

Објекти у JavaScript-у се могу поделити у четири групе:

- Објекти које дефинише сам програмер.
- Објекти уграђени у JavaScript. Ту спадају објекти повезани са типовима података (String, Number, Boolean), објекти који омогућавају креирање кориснички дефинисаних објеката и сложених типова (Objects, Array) и објекти који поједностављују уобичајене задатке, као што су Date, Math.
- Објекти веб читача. Ови објекти нису део JavaScript језика, али их подржава већина веб читача. Примери таквих објеката су објекти Window, Navigator, Document, о којима ће бити речи касније.
- Документ објекти. Они су део Document Object Modela (DOM) дефинисани W3C стандардом. Ови објекти омогућавају JavaScript-у манипулацију над CSS-ом, и лакшу реализацију динамичког HTML-а (DHTML). Приступ објектима обезбеђује веб читач преко својства `document` објекта Window.

Објекти су композитни типови података, обједињују више вредности. Другим речима, објекат је неуређен скуп својстава од којих свако има име и вредност. Именоване вредности могу бити основног типа (бројеви, стрингови) или и саме могу бити објекти.
Објекат се најлакше прави помоћу литерала типа објекта. Литерал

типа објекта (иницијализатор објекта) састоји се од листе парова својство/вредност, раздвојених двотачком. Парови се међусобно раздвајају зарезима, а листа се наводи унутар витичастих заграда.
На пример:

```
var prazan = {}; // Objekat bez svojstava

var tacka = {x:0, y:0};

var osoba = {
    ime: "Petar",
    prezime: "Petrovic",
    datum_rodjenja: "1.1.1990",
    email: "petar@matf.bg.ac.rs"
};
```

Литерал типа објекат је израз који при сваком извршавању прави и иницијализује нов објекат.
Конструктор `Object()` прави празан објекат, као што то чини и литерал `{}`:

```
var osoba = new Object();
```

Објекат је скуп именованих вредности. Те именоване вредности се обично називају својства објекта. Својствима објекта се може приступити тако што се наведе објекат, потом тачка и назив својства. Ако објекту `osoba` треба доделити својства `ime` и `prezime`, то се може урадити на следећи начин:

```
osoba.ime = "Petar";
osoba.prezime =  "Petrovic";
```

Још један начин креирања објеката је позивање функције конструктора. Функција која се користи са оператором `new` назива се функција конструктор или само конструктор. Конструктор има задатак да иницијализује нови објекат, задајући вредности својствима за које је то неопходно пре него што се објекат употреби. Могу се направити неке специјализоване врсте објеката, као што су низ или објекат који представља текући датум:

```
var niz = new Array(15);
var danas = new Date();
```

Конструктор може бити дефинисан и од стране програмера, довољно је написати функцију која додаје својства објекту `this`. У примеру 13 дефинише се конструктор, а потом се позива два пута уз оператор `new`, да би се направила два нова објекта:

```
    // Definise se konstruktor
    function student(ime, prezime, brIndeksa) {
        this.ime = ime;
        this.prezime = prezime;
        this.brIndeksa = brIndeksa;
}
    // Poziva se konstruktor da napravi dva objekta tipa student
    var student1 = new student("Pera", "Peric", "1024/2010");
    var student2 = new student("Laza", "Lazic", "245/2009");
```

Пример 13. Прављење објеката помоћу конструктора

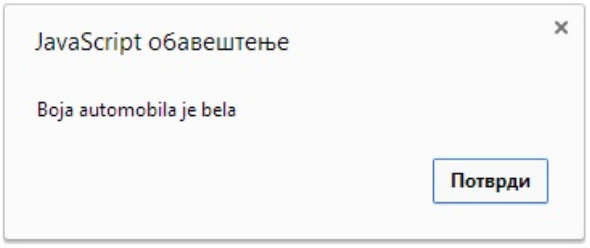
Објекту се могу доделити нова својства и на следећи начин:

```
    student1.godina = 4;
    student1.uslov = true;
```

Својства објекта се понашају као променљиве, у њега се могу уписивати вредности и читати одатле. Својства могу да садрже било који тип података, укључујући и низове, функције и друге објекте. За декларисање својстава се не користи резервисана реч `var`.

```
    // Pravi objekat automobil
    var automobil = new Object();

    // Zadaje vrednost svojstvu u objektu
    automobil.proizvodjac = "BMW";

    // Zadaju se vrednosti jos nekim svojstvima
    automobil.motor = new Object();
    automobil.motor.zapremina = 3000;
    automobil.motor.snaga = "170ks";

    automobil.karoserija = new Object()
    automobil.karoserija.vrsta = "limuzina";
    automobil.karoserija.boja = "bela";

    alert('Boja automobila je '+automobil.karoserija.boja);
```

Пример 14. Објекат automobil



Слика 5. Својство boja објекта automobil

Својство објекта се може променити тако што се својству додели нова вредност:

```
automobil.karoserija.boja = "crvena";
```

За приступање својствима објекта може се користити и оператор `[]`. Следећа два израза у JavaScript-у дају идентичан резултат:

```
automobil.proizvodjac
automobil["proizvodjac"]
```

Метода је функција придружена објекту. Метода се дефинише на исти начин као и функција, а придружује се објекту на следећи начин:

```
imeObjekta.imeMetode = nazivFunkcije;
```

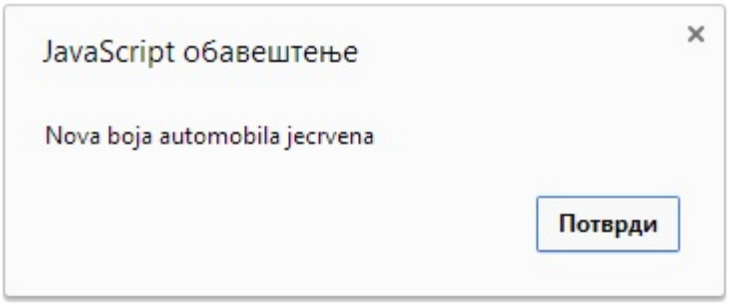
А метода придружена објекту се може позвати:

```
imeObjekta.imeMetode(parametri);
```

Може се додати метода `ofarbaj()` објекту аутомобил из претходног примера:

```
// Pravi objekat automobil
var automobil = new Object();

// Zadaje vrednost svojstvu u objektu
automobil.proizvodjac = "BMW";

// Zadaju se vrednosti jos nekim svojstvima
automobil.motor = new Object();
automobil.motor.zapremina = 3000;
automobil.motor.snaga = "170ks";
```

```
automobil.karoserija = new Object();
automobil.karoserija.vrsta = "limuzina";
automobil.karoserija.boja = "bela";

// Definise se metoda ofarbaj
automobil.karoserija.ofarbaj = ofarbaj;

// Definise se funkcija ofarbaj
function ofarbaj(boja) {
    this.boja = boja;
}
automobil.karoserija.ofarbaj("crvena");
alert("Nova boja automobila je" + automobil.karoserija.boja);
```



Слика 6. Боја аутомобила после позива методе ofarbaj

Може се приметити да објекат преко кога се позива метода постаје вредност резервисане речи `this` у телу методе.

5.1 Брисање својстава

Својство објекта може се обрисати помоћу оператора `delete`:

```
delete automobil.karoserija.vrsta;
```

Брисање својства значи да се то својство уклања из објекта.

5.2 Приступ својствима помоћу петље

Када се ради са објектима чија својства се не знају унапред, петља `for/in` омогућава да се итеративно приступа својствима објекта, тј. да се својства набрајају.

```
function PrikazhiSvojstvaObjekta(obj) {

    var ime = "";
    for(var ime in obj)
    ime += ime + "\n";
}
```

Пример 15. Набрајање својстава објекта

Да ли објекат има неко својство могуће је испитати помоћу оператора `in`, на пример:

```
/* Ako objekat automobil ima svojstvo proizvodjac
dodeliti mu vrednost "Alfa Romeo" */
if ("proizvodjac" in automobil)
    automobil.proizvodjac = "Alfa Romeo";
```

6 Низови

Низ је уређен скуп вредности. Те вредности се називају елементи низа. Сваки елемент низа има своју нумерички описану позицију - индекс. Елементи низа могу бити произвољног типа. Такође, не морају сви елементи једног низа бити истог типа. Најједноставније, низ се може декларисати тако што се експлицитно наброје елементи низа раздвојени зарезима, унутар угластих заграда. На пример:

```
// Niz koji ima 5 elemenata
var ocene = [1, 2, 3, 4, 5];

// Niz koji ima 3 elementa
var imena = ["Marija", "Branko", "Ivana"];

// Niz sa 4 elementa razlichitog tipa
var miks = [9, true, 3.5, "Milica"];

// Niz bez elemenata
var prazan = [];
```

Прва вредност у низу има позицију са индексом 0, друга вредност позицију са индексом 1 итд. Низ може садржати и недефинисане елементе који се праве тако што се изостави вредност за тај елемент између зареза:

```
// Drugi element niza je nedefinisan
var niz = [3, , 4];
```

Други начин за декларисање низова је помоћу конструктора `Array()`. Овај конструктор се може позвати на три начина:

- Без аргумената и тада прави празан низ:

  ```
  var niz = new Array();
  ```

- Експлицитно се наводе вредности првих n елемената низа:

  ```
  // Pravi se niz sa 3 elementa
  var niz = new Array(8, 13.4, "broj");
  ```

- Позива се коснтруктор `Array()` са аргументом који одређује дужину низа:

  ```
  /* Pravi se niz sa 7 elemenata
  cija je vrednost undefined */
  var niz = new Array(7);
  ```

  На овај начин се праве низови ако се претходно зна њихова дужина.

Елементу низа се приступа помоћу оператора `[ ]`, где се унутар заграда наводи његов индекс:

```
niz[0] // Prvi clan niza
niz[3] // Cetvrti clan niza
```

За разлику од програмских језика какви су C и Java , где низ има непроменљив број елемената који се мора задати приликом прављења низа, у JavaScript-у низ може имати произвољан број елемената и број елемената се може мењати у било ком тренутку.
Додавање елемента низу је веома једноставно, потребно је само доделити вредност том елементу:

```
// Elementu sa indeksom 10 dodjeljuje se vrednost "jabuka"
niz[10] = "jabuka";
```

Индекси низа не морају припадати непрекидном опсегу бројева, па су следеће наредбе исправне:

```
// Prvi element niza ima vrednost 1
niz[0] = 1;

// 569-ti elementi niza ima vrednost "a"
niz[568] = "a";

// 1000ti element niza ima vrednost false
niz[999] = false;
```

6.1 Дужина низа

Сви низови имају својство `length` које представља број елемената низа. Својство `length` низа се аутоматски ажурира приликом додавања нових елемената у низ.

```
var niz = new Array();
niz.length; // Vraca 0

niz = new Array(5);
niz.length; // Vraca 5

niz[5] = 10;
niz.length; // Vraca 6

niz[10] = 20;
niz.length; // Vraca 11
```

Пример 16. Коришћење својства `length`

6.2 Приступање елементима низа помоћу петље

Својство `length` се најчешће користи за приступање елементима низа помоћу петље:

```
var sport = ["football","basketball","tennis","box","snooker"];
for(var i = 0; i < sport.length; i++)
    alert(sport[i]);
```

Пример 17. Набрајање елемената низа помоћу петље

6.3 Вишедимензионални низови

JavaScript не подржава праве вишедимензионалне низове, али пружа могућност да се представе помоћу низова низова. Да би се приступило елементу у низу низова довољно је два пута применити оператор `[ ]`. У примеру 18 таблица множења је реализована применом дводимензионих низова:

```
var table = new Array(10);
for(var i = 0; i < table.length; i++)
    table[i] = new Array(10);
for(var row = 0; row < table.length; row++) {
    for(col = 0; col < table[row].length; col++) {
        table[row][col] = row*col;
        }
    }
// Racuna 7*8
var proizvod = table[7][8];
```

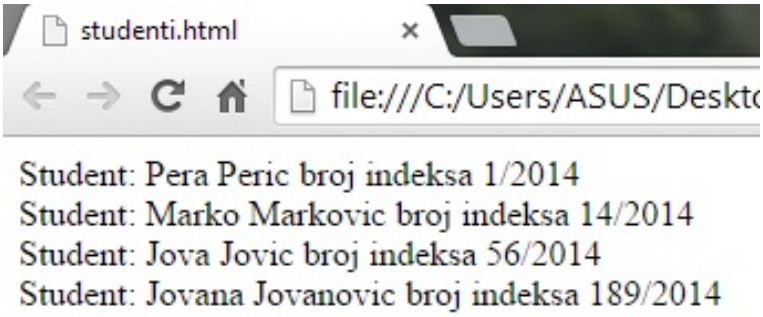
Пример 18. Дводимензиони низ

6.4 Асоцијативни низови

Асоцијативни низ користи стрингове уместо нумеричких вредности за индексирање елемената. Индекс се обично зове key, а додељена вредност value. Парови key/value су уобичајен начин за чување и приступ подацима. За итерацију кроз овакве низове користи се `for` петља.

```
var studenti = new Array();
studenti["1/2014"] = "Pera Peric";
studenti["14/2014"] = "Marko Markovic";
studenti["56/2014"] = "Jova Jovic";
studenti["189/2014"] = "Jovana Jovanovic";
    for (var i in studenti) {
        document.write("Student: "+studenti[i]+
        " broj indeksa "+i+"<br />");
    }
```

Пример 19. Асоцијативни низови



Слика 7. Асоцијативни низови

6.5 Методе објеката типа Array

Метода join()

Метода `Array.join()` конвертује све елементе низа у стрингове и надовезује их. Могуће је задати стринг који раздваја елементе у стрингу насталог надовезивањем. Уколико се не наведе ни један такав граничник за раздвајање ће се користити зарез. Резултат следећег кода је стринг "1,2,3":

```
var niz = [1, 2, 3];
var s = a.join();   // s = "1,2,3"
```

А резултат следећег примера је "1 i 2 i 3":

```
var niz = [1, 2, 3];
var s = a.join(" i ");   // s = "1 i 2 i 3"
```

Метода reverse()

Метода `Array.reverse()` обрће редослед елемената низа и враћа низ са обрнуто распоређеним елементима. Не прави се нови низ, већ се директно у постојећем низу мења редослед елемената. На пример:

```
var niz = [1, 2, 3]; // niz[0]=1; niz[1]=2; niz[2]=3
niz.reverse();   // niz[0]=3; niz[1]=2; niz[2]=1
```

Метода sort()

Метода `Array.sort()` сортира елементе низа директно у изворном низу. Када се метода `sort()` позове без аргумената, сортира елементе низа по абецедном реду. На пример:

```
 var automobili = new Array("Renault", "Alfa Romeo",
"Mercedes", "Fiat");
 automobili.sort();
```

Резултат претходног сортирања је низ аутомобили са елементима `Alfa Romeo`, `Fiat`, `Mercedes`, `Renault` тим редом.
Ако низ садржи недефинисане елементе ова метода их смешта на крај низа.

Метода slice()

Метода `Array.slice()` враћа подниз наведеног низа. Два аргумента ове методе одређују почетак и крај исечка који се добија. Резултујући низ садржи елемент одређен првим аргументом, и све наредне елементе све до елемента (али не и њега) одређеног другим аргументом.
Ако је наведен само један аргумент, резултујући низ садржи све елементе почев од оног одређеног тим аргументом до краја низа. На пример:

```
 var dani = ["ponedeljak", "utorak", "sreda", "cetvrtak",
"petak", "subota", "nedelja"];
 var radni_dani = dani.slice(0,5);
 var vikend = dani.slice(5);
```

Ако је аргумент негативан он представља елемент низа на одређеној позицији у односу на последњи елемент низа. На пример:

```
 var dan = dani.slice(-4,-3); // Rezultat je "sreda"
```

Метода splice()

Ова метода може да обрише елементе из низа, да уметне нове елементе у низ, или да изврши обе операције истовремено. То ради директно у изворном низу, не враћа нови низ. Елементи низа се после уметања или брисања премештају на одговарајуће позиције да би остали суседни осталим елементима. Први аргумент методе `splice()` одређује позицију у низу од које почиње уметање и/или брисање. Други аргумент одређује

број елемената који се бришу из низа. Уколико се изостави други аргумент, уклањају се сви елементи низа почев од елемента задатог првим аргументом, до краја низа. Ова метода враћа низ обрисаних елемената, или празан низ уколико ниједан елемент није обрисан. На пример:

```
var n = [1, 2, 3, 4, 5, 6, 7, 8, 9];
// Rezultat je [2,3]; n ima vrednost [1,4,5,6,7,8,9]
n.splice(1,2);

// Rezultat je [4]; n ima vrednost [1,5,6,7,8,9]
n.splice(1,1);

// Rezultat je [6,7,8,9]; n ima vrednost [1,5]
n.splice(2);
```

Прва два аргумента одређују који ће се елементи низа обрисати. Иза та два аргумента могу се написати и додатни аргументи да би се одредило који ће се елементи уметнути у низ, почев од позиције задате првим аргументом. На пример:

```
var n = [1, 2, 3, 4, 5, 6, 7, 8, 9];
// Rezultat je []; n ima vrednost [1,2,'a','b',3,4,5,6,7,8,9]
n.splice(2,0,'a','b');

// Rezultat je ['a','b'];
// n ima vrednost [1,2,'c','d',3,4,5,6,7,8,9]
n.splice(2,2,'c','d');
```

Методе push() и pop()

Методе `push()` и `pop()` омогућавају да се са низовима ради као да су стекови. Метода `push()` додаје један или више елемената на крај низа и враћа нову дужину низа. Метода `pop()` ради супротно - брише последњи елеменат низа и враћа уклоњену вредност. Обе методе директно мењају изворни низ уместо да праве измењену копију низа. Комбинацијом метода `push()` и `pop()` може се реализовати стек који ради по принципу ,,први унутра, последњи напоље"(First In, Last Out - FILO).

```
var stek = [];    // stek: []
stek.push(2,3);   // stek: [2,3]
stek.pop();       // stek: [2]
stek.push([5,6]); // stek: [2,[5,6]]
stek.push(10);    // stek: [2,[5,6],10]
```

```
stek.pop();       // stek: [2,[5,6]]
stek.pop();       // stek: [2]
```

Пример 20. Рад са стеком

Методе unshift() и shift()

Методе `unshift()` и `shift()` умећу и уклањају елементе од почетка низа. Метода `unshift()` додаје елемент или елементе на почетак низа, помера постојеће елементе на позиције с вишим индексима да би направила простор за нове елементе, и враћа нову дужину низа. Метода `shift()` уклања и враћа први елемент низа, померајући све наредне елементе на позицију са индексом умањеним за један. На пример:

```
var n = [7,8]     // n: [7,8]
n.unshift(3);     // n: [3,7,8]
n.unshift([4,5]); // n: [[4,5],3,7,8]
n.shift();        // n: [3,7,8]
```

7 JavaScript као објектно оријентисан језик

Објектно оријентисано програмирање је програмска парадигма која користи апстракцију за креирање модела заснованих на стварном свету. Користи неколико техника из претходно утврђених образаца, укључујући модуларност, полиморфизам и енкапсулацију. Данас многи програмски језици подржавају објектно оријентисано програмирање. Објектно оријентисано програмирање се може посматрати као дизајнирање софтвера коришћењем колекције кооперативних објеката, за разлику од традиционалног погледа у којем се програм може посматрати као колекција функција, или једноставно као листа инструкција за рачунар. У објектно оријентисаном програмирању, сваки објекат је способан да прима поруке, обрађује податке и шаље поруке ка другим објектима. Сваки објекат се може посматрати као независна мала машина са јасним улогама и одговорностима. Објектно оријентисано програмирање има за циљ да промовише већу флексибилност одржавања и проширивања постојећих програма. Захваљујући свом јаком нагласку на модуларности, објектно оријентисани код би требао да буде једноставнији за развој и лакши за разумевање.

Прототипско програмирање је стил објектно оријентисаног програмирања у коме нису присутне класе, а наслеђивање се постиже кроз процес

уређења постојећих објеката који служе као прототип. JavaScript подржава овај стил објектно оријентисаног програмирања.

Прототипови и наслеђивање

У поглављу 5 је дефинисано шта је функција конструктор. Следећи сегмент кода дефинише конструктор `Pravougaonik` који ће направити два објекта типа `Pravougaonik`:

```
function Pravougaonik(w, h) {
    this.sirina = w;
    this.visina = h;
}
/* Poziva se konstruktor da napravi dva objekta
tipa Pravougaonik */
var prav1 = new Pravougaonik(4,5);
var prav1 = new Pravougaonik(3.5, 5.6);
```

Да би се додала функција у конструктору која рачуна обим правоугаоника, дефинише се својство `obim` које ће референцирати функцију у конструктору која рачуна обим:

```
function Pravougaonik(w, h) {
    this.sirina = w;
    this.visina = h;
    this.obim = function() {
        return 2 * this.sirina + 2 * this.visina;
    }
}
```

Обим правоугаоника може се рачунати на следећи начин:

```
var prav1 = new Pravougaonik(4,5);
var o = prav1.obim();
```

Сваки правоугаоник направљен помоћу оваквог конструктора имаће три својства. Својства `sirina` и `visina` неће бити иста за све правоугаонике, али својство `obim` сваког објекта увек ће се односити на исту функцију. Али, није ефикасно да се користе својства објекта за методе које би требало да деле сви објекти које је направио исти конструктор.
Због тога сваки објекат JavaScript-а има референцу на други објекат, такозвани прототип или прототипски објекат. Сва својства прототипа су и својства објеката за које представља прототип. Другачије речено,

објекат наслеђује својства прототипа.
Прототип објекта је вредност својства `prototype` његовог конструктора. Све функције имају својство `prototype` које се аутоматски прави када се функција дефинише. Свако својство које се дода прототипском објекту биће и својство објеката иницијализованих помоћу конструктора.

```
function Pravougaonik(w, h) {
    this.sirina = w;
    this.visina = h;
    Pravougaonik.prototype.obim = function() {
        return 2 * this.sirina + 2 * this.visina;
        }
}
```

Пример 21. Дефинисање својства прототипског објекта

Својсва `sirina` и `visina` у претходном примеру не морају бити иста за све инстанце. Насупрот тога, прототипски објекат је повезан са конструктором, и сваки објекат који конструктор иницијализује наслеђује исти скуп својстава од прототипа. Својства прототипа се не копирају у нове објекте, већ се налазе у њима као да су њихова својства. Објекат прототип је идеалан за чување метода и других константних својстава. Коришћење објекта прототипа има неколико предности:

- Када се корите прототипови, смањује се количина меморије коју заузима сваки нови објекат, јер може да наследи многа својства прототипа.

- Објекат наслеђује својства чак и када се додају његовом прототипу након што се објекат направи.

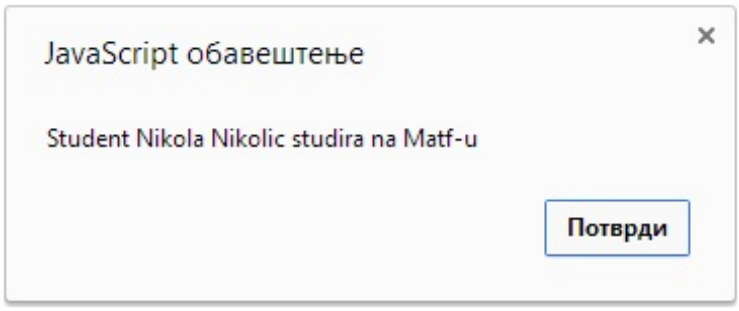
Наслеђена својства се понашају попут регуларних својстава објеката. Могу се разликовати само помоћу методе `Object.hasOwnProperty()`. На пример:

```
var prav1 = new Pravougaonik(4, 5);
prav1.hasOwnProperty("sirina"); // true
prav1.hasOwnProperty("obim"); // false
```

Приликом читања вредности својстава објекта, JavaScript прво проверава да ли објекат има тражено својство. Ако нема, проверава да ли

објекат прототип датог објекта има тражено својство. На овај начин је омогућено наслеђивање од прототипова.
Приликом задавања вредности својства које се наслеђује од прототипа, JavaScript не користи прототипски објекат већ се прави ново својство директно у објекту. Пошто тада објекат има сопствено својство, више не наслеђује то својство од прототипа. Приликом читања тог својства из објекта, JavaScript тражи то својство у самом објекту. Када се утврди да је такво својство дефинисано у објекту, неће бити потребно да претражује прототипски објекат, па никада неће прочитати вредност тог својства дефинисаног у прототипу.
У наредном примеру, својство `fakultet` се наслеђује од прототипског објекта:

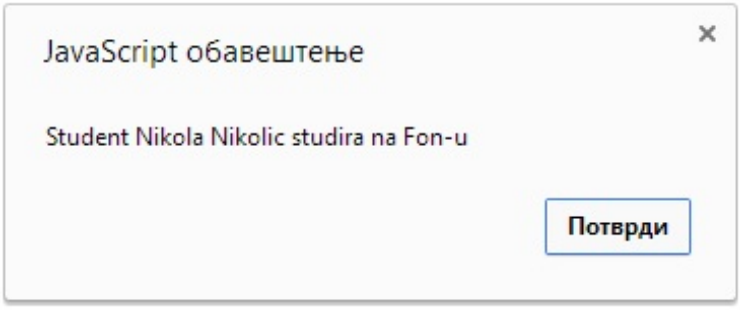
```
function Student(ime, prezime, indeks) {
    this.ime = ime;
    this.prezime = prezime;
    this.indeks = indeks;
    Student.prototype.fakultet = "Matf";
}
student1 = new Student("Nikola", "Nikolic", "203/2008");
alert('Student '+student1.ime+' '+student1.prezime+'
studira na ' +student1.fakultet+'-u');
```



Слика 8. Својство наслеђено од прототипа

Ако се зада вредност својства `fakultet`, које се наслеђује од прототипа, објекат `student1` неће више наслеђивати то својство:

```
student1.fakultet = "Fon";
alert('Student '+student1.ime+' '+student1.prezime+'
studira na ' +student1.fakultet+'-u');
```



Слика 9. Својство се више не наслеђује од прототипа

У објектно оријентисаним језицима, као што су Java и C++, нека својства класе се могу декларисати као приватна, тако да буду доступна само методама класе и да је немогуће управљати њима изван те класе. Помоћу технике зване капсулирање података својства се проглашавају приватним, тако да им се ради читања и уписивања вредности може приступати само преко посебних приступних метода. JavaScript може да симулира ту технику помоћу блокова closure, а приступне методе не могу се наследити од прототипа.

```
function Pravougaonik(w, h) {
    // Ove metode su blokovi closures
    this.getSirina = function() {return w;}
    this.getVisina = function() {return h;}
}
Pravougaonik.prototype.obim = function() {
    return 2 * this.getSirina() + 2 * this.getDuzina();
}
```

Пример 22. Капсулирање података

Конструктор `Pravougaonik` не додаје својства `visina` и `sirina` објекту који иницијализује. Уместо тога, конструктор дефинише приступне методе у објекту.

7.1 Заједничке особине и методе

Сви објекти у JavaScript-у имају заједничке особине и методе наведене у следећој листи. Већина је корисна само када се ради са објектима које програмер сам дефинише.

- `prototype` - референца на објекат од кога се наслеђују својства, тј. на свој прототип.
- `constructor` - референца на функцију конструктор.
- `toString()` - конвертује објекат у стринг.
- `toLocaleString()` - конвертује објекат у локализовани стринг.
- `valueOf()` - претвара објекат у одговарајући примитивни тип, најчешће број.
- `hasOwnProperty(prop)` - враћа `true` ако објекат има својство `prop` или `false` у супротном.
- `isPrototypeOf(obj)` - враћа `true` ако објекат служи као прототип објекта `obj`, у супроном `false`.
- `propertyIsEnumerable(prop)` - враћа `true` ако ће својство `prop` бити наведено у `for/in` петљи.

8 Објекти уграђени у JavaScript

8.1 Објекат Date()

JavaScript има класу објеката која представља датуме и време, и помоћу које се може радити са тим подацима. Датум је заснован на UNIX-овом датуму почев од 1.јануара 1970. и не подржава раније датуме. Date објекат враћа време и датум са локалног рачунара, а не са сервера. Објекат типа Date у JavaScript-у прави се помоћу оператора `new` и конструктора `Date()`.
Инстанце објекта Date се могу креирати на следећи начин:

```
new Date();
new Date(milisekunde);
new Date(podaciODatumu);
```

- `Date()` - конструктор `Date()` без аргумената креира објекат који садржи текући датум и време
- `Date(milisekunde)` - аргумент `milisekunde` представља број милисекунди (хиљадити део секунде) од 1. Јануара 1970. 00:00:00

- `Date(podaciODatumu)` - аргумент `podaciODatumu` представља датум у одговарајућем формату који подржава метод `Date.parse()`. Аргумент `podaciODatumu` може бити у следећем облику:
  - месец дан, година час:минут:секунда
  - година, месец, дан
  - година, месец, дан, час, минут секунда

Методе објекта Date()

- `getDate()` - враћа дан у месецу за тренутни Date објекат. Узима вредности 1-31.
- `getDay()` - враћа дан у недељи за тренутни Date објекат као цео број од 0 (недеља) до 6 (субота).
- `getHours()` - враћа сат из времена за Date објекат као цео број од 0 - 23.
- `getMinutes()` - враћа минуте из времена за Date објекат као цео број 0 - 59.
- `getSeconds()` - враћа секунде из времена за Date објекат као цео број 0 - 59.
- `getMonth()` - враћа месец из датума као цео број од 0 до 11 (0 - Јануар, 1 - Фебруар...).
- `getYear()` - представља годину из датума као цео број који представља годину минус 1900.

Коришћење наведених метода приказано је у следећем примеру:

```
var d = new Date("January 13, 2013 09:22:57");
var dan = d.getDate(); // dan uzima vrednost 13
var sat = d.getHours(); // sat uzima vrednost 9
var minut = d.getMinutes(); // minut uzima vrednost 22
var sekund = d.getSeconds(); // sekund uzima vrednost 57
var mesec = d.getMonth(); // mesec uzima vrednost 0
var godina = d.getYear(); // godina uzima vrednost 113
```

- `getTime()` - враћа време за тренутни Date објекат као цео број који представља број милисекунди протеклих од 1. јануара 1970. од 00:00:00.

- `setDate(dan)` - поставља дан у месецу за Date објекат. `dan` је цео број између 1 и 31.
- `setMinutes(minut)` - поставља минуте у време тренутног Date објекта. Узима целобројне вредности од 0 до 59.
- `setHour(cas)` - поставља сат у време тренутног Date објекта. Узима целобројне вредности од 1 до 23.
- `setMonth(mesec)` - поставља месец за тренутни Date објекат. Узима вредности од 0 до 11.
- `setYear(godina)` - поставља годину за тренутни Date објекат. Аргумент `godina` је цео број већи од 1900.
- `toGMTString()` - враћа вредност тренутног Date објекта као стринг следећег формата: Дан, дд месец година час : минут : секунд GMT.

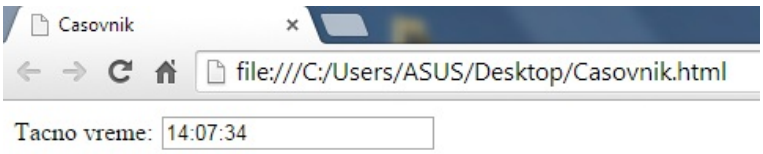
Пример 23 показује тачно време у облику HH:MM:SS.

```
<html>
<head>
    <title>Casovnik</title>
    <script type="text/javascript">
    function casovnik() {
        var danas = new Date();
        var sat = danas.getHours();
        var minut = danas.getMinutes();
        var sekund = danas.getSeconds();

        // Ako je minut jednocifren broj dodaje se nula ispred
        sat +=  ((minut < 10) ? ":0" : ":") + minut;
        sat += ((sekund < 10) ? ":0" : ":") + sekund;

    document.casovnikForma.vreme.value = sat;
    setTimeout("casovnik()",1000);
}
</script>
</head>
<body onload=casovnik()>
<form name="casovnikForma">
<input type="text" size="10" name="vreme" />
</form>
```

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

Пример 23. Часовник

Резултат примера 23 је приказан на слици 10.



Слика 10. Часовник

8.2 Објекат Math

Поред основних аритметичких операција, JavaScript подржава и сложеније математичке операције помоћу функција које су представљене као својства објекта Math. На пример, да би се израчунао квадратни корен броја `n`:

```
koren = Math.sqrt(n);
```

Објекат Math продржава многе математичке функције.

- `Math.abs()` - функција која рачуна апсолутну вредност.
- `Math.sin()`, `Math.cos()`, `Math.tan()` - тригонометријске функције синус, косинус и тангенс.
- `Math.asin()`, `Math.acos()`, `Math.atan()` - инверзне тригонометријске функције, аркус синус, аркус косинус и аркус тангенс.
- `Math.exp()` - рачуна степен броја e.
- `Math.log()` - рачуна природни логаритам.
- `Math.ceil()` - заокружује број на већу вредност.
- `Math.floor()` - заокружује број на мању вредност.

- `Math.min()`, `Math.max()` - враћа мањи, односно већи, од два прослеђена броја.
- `Math.random()` - враћа случајан број.
- `Math.round()` - заокружује на најближи цео број.

На пример:

```
Math.sqrt(25); // Vraca 5
Math.abs(-2); // Vraca 2
Math.floor(4.8); // Vraca 4
Math.round(3,1); // Vraca 3
Math.min(21,-13,100,0,43); // Vraca -13
```

Пример 24. Коришћење метода објекта Math

Не постоји конструктор `Math()`, као што постоји `Date()` или `String()`, а функције попут `Math.floor()` су само функције, а не методе које раде са објектима.
Поред великог броја функција, објекат Math садржи и константе са којима се може радити.

- `Math.E` - константа e, основа природног логаритма
- `Math.LN10` -логаритам са основом 10
- `Math.LN2` - логаритам са основом 2
- `Math.PI` - константа π
- `Math.SQRT1_2` - $\frac{1}{\sqrt{2}}$
- `Math.SQRT2` - $\sqrt{2}$

8.3 Омотач објекат

Основни типови података имају придружене објекте који носе исто име као тип података који представљају. Сва три основна типа података string, number, boolean имају објекте који се исто зову: String, Number, Boolean. Ови објекти се зову омотачи и пружају својства и методе који се могу користити за управљање тим подацима. Предност постојања омотач објеката је могућност примењивања и проширивања својстава објекта који утичу на основне податке, па је својство `length` дефинисано за било који стринг:

```
var txt = "Ovo je string koji predstavlja osnovni tip";
var duzhina = txt.length; // duzina=51
```

Уколико је то потребно, JavaScript аутоматски конвертује стрингове у објекте типа String. Исто важи и за обрнуту конверзију.
Све речено за стрингове и објекте типа String односи се на бројеве и на логичке вредности, и на одговарајуће објекте типа Number и Boolean.
Сваки број, стринг или логичка вредност може се конвертовати у одговарајући објекат омотач помоћу функције Object(). На пример:

```
// broj 12 postaje objekat tipa Number
var brojOmotac = Object(12);
```

8.4 Објекат String

Објекат String обезбеђује велики број својстава и метода који се могу користити за манипулацију над стринговима. `String()` конструктор има опциони аргумент који одређује почетну вредност:

```
var s = new String("Dobrodosli u svet JavaScripta!");
```

Ако се изостави аргумент, конструктор иницијализује празан стринг:

```
var s1 = new String();
```

У пракси се ретко користи наведено креирање стринг објекта, јер се сва својства и методе објекта String могу позвати преко стрингова као основних типова података. На пример, својство `length`, које враћа дужину стринга:

```
// s1 je objekat, a s2 je osnovni tip podataka
var s1 = new String("Dobrodosli u svet JavaScripta!");
var s2 = "Dobrodosli u svet JavaScripta!";
// Pozivamo svojstvo length
var duzina1 = s1.length;
var duzina2 = s2.length;
```

Вредности `duzina1` и `duzina2` из претходног примера су једнаке и узимају вредност 30.
Својство `length` се аутоматски ажурира када се промени стринг и не може бити директно измењено од стране програмера. У ствари, не постоји начин да се стрингом манипулише директно. Било која метода примењена на стринг не мења његову вредност већ враћа стринг који је резултат те методе. У наредном примеру се позива метода `toUpperCase()` која мења мала слова великим:

```
var s = "hello world";
/* s1 ima vrednost "HELLO WORLD",
ali s i dalje ima vrednost "hello world" */
s1 = s.toUpperCase();
```

Да би се променила вредност стринга резултат дате операције се мора доделити том стрингу:

```
var s = "hello world";
s = s.toUpperCase(); // s sada ima vrednost "HELLO WORLD"
```

Појединачним карактерима у стрингу се може приступити користећи методу `charAt()`, која прихвата целобројни аргумент и враћа карактер који се налази на задатој позицији. Нумерисање позиција почиње од нуле, слично као код низова. Метода враћа стринг, јер у JavaScript-у не постоји разлика између једног карактера и стрингова. На пример:

```
var s = "hello world";
s.charAt(6); // vraca "w"
```

Метода `indexOf()` прихвата стринг аргумент и враћа индекс првог појављивања аргумента у стрингу. Ако не пронађе аргумент у стрингу враћа -1. На пример:

```
var s = "hello world";
s.indexOf("worl"); // vraca 6
s.indexOf("JavaScript"); // vraca -1
```

Метода `substring()` омогућава издвајање дела стринга који се налази између задатог почетног и крајњег положаја. Први аргумент одређује индекс од кога тражени део почиње. Други аргумент је опциони, указује на позицију у којој се тражени део завршава. Следећи сегмент кода издваја карактере између позиција 0 и 5:

```
var s = "hello world";
s.substr(0,5); // vraca "hello"
```

Ако се други аргумент не наведе, метода враћа стринг од задате позиције до краја стринга:

```
var s = "hello world";
s.substring(3); // vraca "lo world"
```

Објекат String обезбеђује још корисних метода за манипулацију над стринговима, наведених у следећој листи:

- `charCodeAt(index)` - враћа Unicode карактер еквивалентан карактеру који се налази на позицији `index`. На пример:

```
var s = "HELLO WORLD";
var n = s.charCodeAt(0); // Unicode vrednost za H
```

  Вредност променљиве `n` је 72.

- `concat(string1)` - врши надовезивање стринга `string1` на постојећи стринг. На пример:

```
var string1 = "Hello ";
var string2 = "world";
var poruka = string1.concat(string2);
```

  Променљива `poruka` има вредност Hello world.

- `fromCharCode(codes)` - претвара број у одговарајући Unicode карактер. На пример:

```
var n = String.fromCharCode(65); // Rezultat je A
```

- `lastIndexOf(substr)` - проналази последње појављивање `substr`-а и враћа његов индекс.

- `replace(searchString, valueString)` - замењује `searchString` вредношћу `valueString`. На пример:

```
var s = "Programski jezik Pyton";
var res = s.replace("Pyton","JavaScript");
```

  Променљива `res` има вредност Programski jezik JavaScript.

- `slice(startpos, endpos)` - враћа део стринга између почетне и крајње позиције. На пример:

```
var s = "Hello world";
var res = s.slice(0,5);
```

  Променљива `res` узима вредност Hello.

- `split(granicnik)` - од стринга прави низ, при чему су елементи низа делови стринга одвојени знаком `granicnik`.
- `valueOf` - враћа вредност стринга датог објекта.
- `trim()` - брише белине са почетка и краја стринга. На пример:

```
var s = "   Hello world   ";
var res = s.trim(); // res uzima vrednost Hello world
```

8.5 Основни и референтни типови

Типови података које подржава JavaScript делимо у две групе: основни типови (бројеви, логичке вредности, `null` и `undefined`) и референтни типови (објекти, низови и функције).
Основни типови заузимају меморијски простор унапред предвиђене величине. Број заузима 8 бајтова меморије, а логичка вредност само један бит.
Референтни типови немају фиксну величину, па се њихове вредности не могу чувати директно у меморијској локацији унапред одређене величине. Уместо тога, променљива чува референцу на вредност.
Разлика између основних и референтних типова може се најбоље видети из следећа два примера:

```
// Inicijalizuje i deklarise promenljivu x
var x = 10;
// Kopira vrednost promenljive x u promenljivu y
var y = x;
// Menja vrednost promeljive x
x = 20;
```

Вредност променљиве `x` у претходном примеру ће бити 20, а променљиве `y` 10. Ако се измени претходни пример тако да уместо са бројевима ради са низовима, добијају се другачији резултати:

```
// Inicijalizuje i deklarishe niz n
var n = [a,b,c,d,e];
// Kopira referencu u novu promenljivu
var m = n;
// Menja niz pomocu izvorne reference
n[0] = 10;
// I niz m je promenjen
```

У претходном примеру, наредба `var m = n;` не прави два низа, већ се променљивој `m` додељује референца на вредност низа. После овог реда, још увек имамо само један низ, с тим што постоје две референце на њега.
Стрингови не припадају ни једном од наведена два типа. Могу бити различите дужине, па се не могу директно чувати у променљивој предвиђене дужине. Логичан закључак након претходне реченице је да је најефикасније копирати референце на стрингове, а не конкретан садржај стрингова.
С друге стране, стрингови се понашају као основни типови података гледано из више аспеката. На питање да ли су стрингови основни или референтни тип, не може се одговорити, јер су стрингови неизмењиви, тј. не постоји начин да се промени вредност знаковног низа. То значи да се не може смислити пример налик претходном који показује да се низови копирају по референци.
Ефикасности ради, може се претпоставити да је JavaScript имплементиран тако да се стрингови прослеђују по референци, а пореде по вредности.

9 Клијентски JavaScript

У првом делу је обрађено само језгро JavaScript-а. Сви представљени примери, сегменти кода, често нису имали одређени контекст, нити су били намењени извршавању у одређеном окружењу.
Други део је посвећен тзв. клијентском JavaScript-у, тј. JavaScript-у који се користи у веб читачима.

9.1 Начин укључивања JavaScript-а унутар HTML кода

Клијентски JavaScript код се може уградити у HTML документе на неколико начина:

- Између пара ознака `<script>` и `</script>`
- Из спољне датотеке задате атрибутом `src` ознаке `<script>`
- У процедури за обраду догађаја, наведен као вредност HTML атрибута
- У URL адреси која користи посебан протокол javascript:

JavaScript код се може сместити на два различита места унутар HTML странице.
Први начин је да се JavaScript код смести у `<head>` секцији, док је други начин смештање унутар `<body>` секције документа. Код првог начина, скрипт се учитава одмах, још пре учитавања читаве странице, док се код другог начина скрипт извршава чим читач наиђе на њега.

9.1.1 Директно укључивање скрипта у документ

Клијентски скриптови се директно укључују у HTML код помоћу ознака `<script>` и `</script>`.
Веб читач све унутар ознака `<script>` тумачи као неки облик скрипт језика. Не постоји ограничење броја `<script>` ознака унутар HTML документа. Код већине веб читача подразумеван је JavaScript. Међутим, могуће је да веб читач подржава друге скрипт језике, као што су VBScript, који је подржан од стране Internet Explorera. Да би се назначило на ком језику је написан скрипт употребљава се атрибут `type` ознаке `<script>` који означава MIME тип језика који се користи. У случају, када се ради о JavaScript-у, атрибут `type` ће имати вредност `text/javascript`:

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

Када се ознака `<script>` појавила, није подржавала атрибут `type`. Језик за скриптове се задавао помоћу атрибута `language`:

```
<script language="JavaScript">
```

Данас се све ређе користи атрибут `language`. Програмери га користе најчешће због усклађивања са старијим читачима, где се наводе оба атрибута:

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

Атрибут `language` се понекад користи за указивање на верзију JavaScript-а на којој је скрипт написан. Веб читачи занемарују скриптове написане на верзијама JavaScript-а које не подржавају. Старији читачи, који не подржавају JavaScript 1.6 неће покушати да изврше скрипт чији атрибут `language` има вредност "JavaScript1.6":

```
<script language = "JavaScript1.6"></script>
```

9.1.2 Укључивање скрипта из спољашњих датотека

Ознака `<script>` подржава атрибут `src` којим се задаје URL датотеке са JavaScript кодом. Датотека „script.js" може се укључити у HTML документ на следећи начин:

```
<script src = "../script.js"></script>
```

JavaScript датотека има екстензију .js и садржи искључиво JavaScript код, без ознака `<script>` и без икаквог HTML кода. Коришћење спољашњих датотека има неколико предности:

- Омогућава да једну датотеку користи више HTML страна. То знатно олакшава ажурирање кода.
- Раздвајају се садржај и понашање, и тако HTML датотеке постају знатно прегледније, без огромних блокова JavaScript кода.
- Ако више HTML страна користи једну датотеку, веб читач кешира ту датотеку, што убрзава учитавање.

9.1.3 Процедуре за обраду догађаја

Динамички програми дефинишу процедуре за обраду догађаја које веб читачи аутоматски позивају када се деси одређени догађај. Како догађаји у клијентском JavaScript-а потичу од HTML објеката, процедуре за обраду догађаја могу се дефинисати као вредности атрибута тих објеката. На пример, вредности атрибута `onclick` може се додати одређени JavaScript код:

```
<input type = "checkbox" name = "pol" value = "muski"
onclick = "pol = this.checked;">
```

Вредност атрибута `onclick` може да садржи једну или више наредби на JavaScript-у. Ако има више наредби оне морају бити раздвојене тачком и зарезом. Када се деси наведени догађај, у претходном примеру клик мишем на дугме, извршава се JavaScript код.
Иако је у дефиницији процедуре за обраду догађаја могуће уврстити произвољан број JavaScript наредби, уобичајено је, због прегледности кода, да се помоћу атрибута за обраду догађаја позивају функције које су дефинисане на другом месту, у ознакама `<script>` или у спољашњим датотекама. Атрибуту `onclick` може се доделити функција `saberi()` која је дефинисана на другом месту:

```
<input type="button" name="zbir" value="Saberi"
onclick = "zbir = saberi();">
```

Процедуре за обраду догађаја биће детаљније обрађене у поглављу 13.

9.1.4 JavaScript у URL адресама

Још један начин да се JavaScript користи на клијентској страни је да се наведе у URL помоћу псеудо протокола javascript: . Овај тип протокола означава да је тело URL адресе произвољан JavaScript код. Наредбе се морају раздвајати тачком и зарезом, а коментари писати унутар ознака `/*  */`.
Наредни URL отвара нов прозор читача без измене садржаја текућег прозора:

```
javascript:window.open("about:blank"); void:0;
```

URL на JavaScript-у се може употребити свуда где и регуларан URL. Псеудо протокол `javascript:` може се користити са HTML атрибутима чија би вредност требала да буде неки URL. На пример атрибут `href` хипервезе. Када корисник притисне ту везу, извршава се наведени JavaScript код:

```
<a href = javascript:alert("Dobrodosli");>Dobrodoslica</a>
```

9.2 Сакривање скрипта од старих веб читача који не подржавају JavaScript

Већина веб читача када наиђе на ознаке које не разуме, све унутар тих ознака приказује као обичан текст. Иако данас огромна већина веб читача подржава JavaScript, ипак програмери често прибегавају следећем трику.
Један лак начин да се маскира JavaScript код је да се сав код пише унутар HTML коментара:

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

JavaScript kod

//-->
</script>
```

Иако је овакав начин сакривања кода од старих веб читача веома чест на интернету, по строгим XHTML правилима није исправан. С' обзиром на то да је XHTML језик заснован на XML-у, многе ознаке у JavaScript-у, као што су > или & имају специјално значење, па би претходни приступ могао да резултира проблемом. Према XHTML спецификацији, може се користити следећа техника скривања кода од старих веб читача:

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

<![CDATA[

JavaScript kod

]]>
</script>
```

<noscript> елемент

У ситуацијама где веб читач не подржава JavaScript, или је JavaScript искључен, треба обезбедити алтернативну верзију онога што скрипт ради, или бар поруку упозорења која говори кориснику шта се догодило.
Ово се обезбеђује помоћу `<noscript>` елемента. Сви веб читачи који подржавају JavaScript игноришу садржај унутар ознака `<noscript>...</noscript>`, а читачи који не подржавају JavaScript ће приказати приложену поруку. Према правилима XHTML -а, `<noscript>` елемент се не сме налазити у одељку `<head>`.

```
<body>
<script type="text/javascript">
alert("Vas browser podrzava JavaScript!");
</script>
<noscript>
Vas web citac ne podrzava JavaScript ili je
JavaScript trenutno iskljucen
</noscript>
</body>
```

Пример 25. Коришћење `<noscript>` *елемента*

10 Објекат Window

Објекат Window се налази на врху хијерархије JavaScript-а и представља место за садржај HTML документа у прозору веб читача. Он служи као глобални објекат и глобални извршни контекст за клијентски JavaScript.
Објекат Window дефинише бројна својства и методе које омогућавају управљање прозором читача веба. Такође, дефинише својства која референцирају друге важне објекте.
Поред садржаја документа, сфера утицаја објекта Window укључује димензије прозора као и све остале саставне делове као што су скрол траке, траке са алатима, менији, статусне траке...
Како је објекат Window глобални објекат у клијентском JavaScript-у, помоћу њега се приступа свим осталим објектима.
Својства и методе објекта Window се најчешће позивају на следећи начин:

```
window.imesvojstva
window.imemetoda(parametri)
```

Када при референцирању програм показује на прозор у коме је смештен документ, за објекат Window постоји и синоним `self`, па се својства и методе текућег прозора могу позвати на следећи начин:

```
self.imesvojstva
self.imemetoda(parametri)
```

10.1 Својства објекта Window

- `closed` - логичка вредност која може само да се чита. Његова вредност је `true` ако је прозор затворен, у супротном је `false`.

  ```
  /* Vraca vrednost true ako je prozor zatvoren
  ili false ako nije zatvoren */
  if (window.closed)
  ```

- `document` - референца објекта Document који је садржан у прозору
- `history` - референца објекта History за дати прозор
- `location` - референца објекта Location за дати прозор
- `navigator` - референца објекта Navigator

- `status` - својство које може да се чита и у које може да се пише, односи се на садржај статусне линије

```
window.status = "Tekst u statusnoj liniji";
```

- `defaultStatus` - својство које може да се чита и у које може да се пише, представља подразумевану вредност која се појављује у статусној линији
- `name` - садржи име прозора
- `frames[]` - низ оквира (frame) који су садржани у прозору
- `length` - ово својство може само да се чита. Представља број елемената у низу `frames[]`. Може се позвати и преко `frames.length`
- `top` - референца на прозор који садржи текући прозор. Ово својство је корисно само када је текући прозор оквир

10.2 Методе објекта Window

Отварање прозора

Нов прозор читача може се отворити помоћу методе `open()` објекта Window. Због искачућих прозора који су честа појава док се крстари интернетом, а који се генеришу управо помоћу ове методе, метода `window.open()` се успешно извршава само ако је позвана као реакција на неку корисникову акцију, притискање дугмета или клик мишем. Метода `window.open()` има четири необавезна аргумента:

- Први аргумент је URL документа који треба да се прикаже у новом прозору. Ако се овај аргумент изостави прозор ће бити празан. На пример:

```
// Otvara prozor sa url adresom page1.html
window.open("page1.html");
```

- Други аргумент је име прозора. Ако се наведе име прозора који већ постоји, метода `open()` враћа референцу на постојећи прозор, уместо да отвара нови. На пример:

```
/* Otvara stranicu sa url adresom page1.html
   i imenom stranica1 */
window.open("page1.html", "stranica1");
```

- Трећи аргумент је листа карактеристика које одређују величину прозора и елементе графичког корисничког окружења. Ако се изостави овај аргумент, новом прозору се додељује подразумевана величина, са потпуним скупом стандардних карактеристика: трака са менијима, статусна трака, трака са алаткама итд. Када се наводи овај аргумент, све карактеристике које се не наведу се изостављају. Из безбедносних разлога обично није могуће да се отвори прозор који је премали или је постављен ван екрана, или прозор без статусне траке.
  Следећи пример показује позивање методе `window.open()` која отвара прозор подесиве величине, са статусном траком, али без траке са менијима и адресне траке:

  ```
  var prozor = window.open("page1.html", "stranica1",
  "width = 400,height = 400,status = yes,resizable = yes");
  ```

- Четврти аргумент методе `open()` се користи само ако је други аргумент име постојећег прозора. Његова вредност је логичка. Вредност `true` означава да URL из првог аргумента треба да замени текућу ставку у историји прегледања веб читача. У супротном, (вредност `false`), прави се нова ставка у историји прегледања читача. Подразумевана вредност је `false`.

Резултат методе `open()` је објекат Window који представља нови прозор.

```
var w = window.open("index.html","index",
"fullscreen=yes,scrollbars=yes,status=yes");
```

Пример 26. Променљива w је објекат типа Window

Затварање прозора

Слично методи `open()` која отвара нов прозор, метода `close()` затвара прозор. Ако постоји објекат w типа Window, може се затворити следећом наредбом:

```
w.close();
```

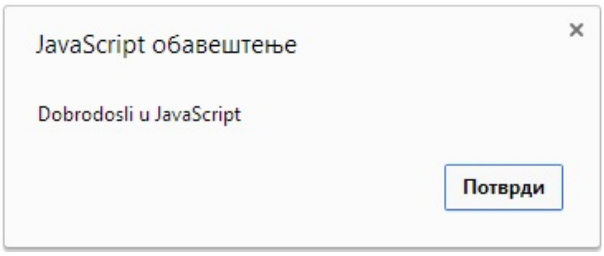
Већина читача дозвољава скрипту да затвори само оне прозоре које је сам направио. Ако се покуша затварање било ког другог прозора, захтев неће бити прихваћен или ће се приказати оквир за дијалог где ће се тражити да корисник потврди да жели да затвори прозор. Објекат Window постоји и пошто се прозор који представља затвори. Не препоручује се коришћење његових својстава и метода након затварања.

10.3 Оквири за дијалог

Објекат Window има три методе за приказивање једноставних оквира за дијалог - `window.alert()`, `window.confirm()`, `window.prompt()`. Метода `alert()` приказује поруку кориснику, тј. текст који се преда као параметар, а дугме `OK` омогућава кориснику да уклони упозорење. Метода `alert()` се може позвати на следећи начин:

```
window.alert("Dobrodosli u JavaScript");
```

и резултат позива је прозор на слици 11.

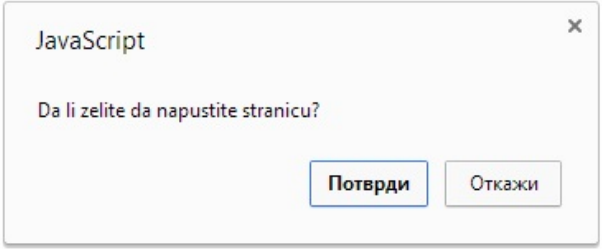


Слика 11. Alert прозор

Метода `confirm()` тражи од корисника да притисне дугме `OK` или `Cancel` како би потврдио или отказао операцију. Овај метод враћа `true` ако корисник притисне `OK`, или `false` ако притисне `Cancel`. Основна синтакса овог метода је:

```
window.confirm("Da li zelite da napustite stranicu?");
```

а резултат претходног примера је приказан на слици 12.



Слика 12. Confirm прозор

Како метода враћа логичке вредности `true` и `false`, може се користити и као услов у некој `if` конструкцији.

```
var r = confirm("Kliknite dugme!");
if (r == true) {
    alert("Kliknuli ste OK!");
}
else {
    alert("Klinkuli ste Cancel!");
}
```

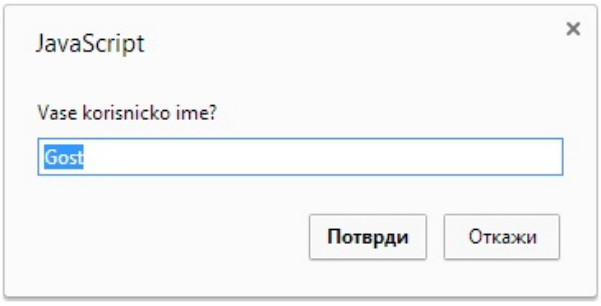
Пример 27. Коришћење методе `confirm()` *у* `if` *наредби*

Последњи оквир за дијалог, приказује задату поруку и обезбеђује поље за унос текста у које корисник уноси одговор. Метод `window.prompt()` има два параметра. Први је порука која се приказује кориснику, а други параметар је подразумевани одговор. Ако не постоји подразумевани одговор, наводи се празан низ ” ”. Притиском на дугме `OK` враћа се вредност унетог знаковног низа, а притиском на дугме `Cancel`, метод враћа null. Основна синтакса је:

```
window.prompt("Pitanje", "podrazumevani odgovor");
```

На пример:

```
var ime = window.prompt("Vase korisnicko ime?", "Gost");
if (ime != null) {
    alert("Dobrodosli " +ime);
}
```



Слика 13. Prompt прозор

Ове методе блокирају рад скрипта, тј. када се прикаже неки од ових оквира, скрипт престаје да се извршава, а учитавање документа се заустављa све док корисник не одговори на захтев из оквира за дијалог. Употреба искачућих прозора има неколико предности:

- Разумљиви су за корисника.
- Појављују се изван и изнад тренутног документа што им даје првостепени значај.

Поред тога, постоји и неколико недостатака при коришћењу ових прозора.

- Не могу бити стилизовани, већ зависе од веб читача.
- Зависе од JavaScript-а, а повратне информације би требало да буду на располагању и када је JavaScript искључен.

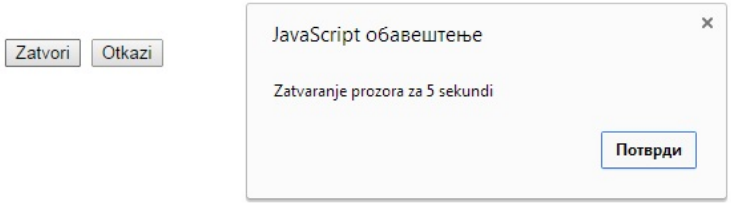
10.4 Тајмери

Важна одлика сваког окружења за програмирање је могућност заказивања извршавања кода у неком будућем тренутку. Клијентски JavaScript то омогућава помоћу глобалних функција `setTimeout()`, `clearTimeout()`, `setInterval()`, `clearInterval()`.
Метода `setTimeout()` објекта Window заказује извршавање функције после одређеног броја милисекунди. Ова метода враћа идентификатор који се може проследити методи `clearTimeout()` која може да откаже извршавање дате функције.

```
<input type="button" value="Zatvori"
onclick="timer = setTimeout('window.close()',5000);
alert('Zatvaranje prozora za 5 sekundi'); return true;" />
<input type="button" value="Otkazi"
onclick="clearTimeout(timer)"
alert('Otkazano zatvaranje prozora'); return true;" />
```

Пример 28. Коришћење метода `setTimeout()` *и* `clearTimeout()`

Резултат примера је приказан на слици 14.



Слика 14. Коришћење метода `setTimeout()` *и* `clearTimeout()`

Метода `setInterval()` је попут методе `setTimeout()`, осим што се позив одређене функције понавља у задатим интервалима израженим у милисекундама. Ова метода враћа идентификатор који се може проследити методи `clearInterval()` да би она отказала извршења функције.

```
<script type="text/javascript">
setInterval(function () {
    alert("Nepopularan nacin koriscenja
    metode setInterval");}, 3000);
</script>
```

Пример 29. Коришћење метода `setInterval()` *и* `clearInterval()`

Пример 29 избацује alert прозор у временском интервалу од 3000 милисекунди, односно 3 секунде.

Још неке методе објекта Window

У следећој листи су набројане још неке методе објекта Window које се најчешће користе.

- `focus()` - поставља фокус на дати прозор. Пример 30 отвара нови прозор и ставља фокус на њега.

  ```
  // Otvra novi prozor
  var w = window.open("", "", "width=200,height=200");
  // Stavlja fokus na novootvoreni prozor
  w.focus();
  ```

  Пример 30. Коришћење методе `fokus()`

- `blur()` - уклања фокус са прозора, супротно методи `focus()`.

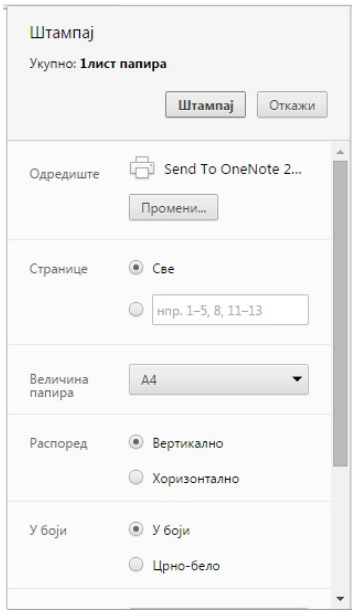
- `moveTo(x,y)` - помера прозор на задату позицију. Аргументи `x` и `y` су обавезни и одређују координате горњег левог темена прозора.

- `moveBy(x,y)` - помера прозор за одређено растојање у односу на тренутну позицију. Аргументи `x` и `y` су обавезни. Аргументи `x` и `y` могу бити позитивни или негативни и одређују за колико пиксела ће прозор бити померен хоризонтално, односно вертикално.

```
// Otvra novi prozor
var w = window.open("", "", "width=200,height=200");
// Pomera prozor 100px horizontalno udesno i
// 150px vertikalno nagore
w.moveBy(100,150);
// Stavlja fokus na novootvoreni prozor
w.focus();
```

Пример 31. Коришћење метода објекта `Window`

- `print()` - штампа текућу страницу. Ова метода отвара Print Dialog Box, за подешавања параметара штампања.



Слика 15. Print Dialog Box

- `resizeBy(x,y)` - мења величину прозора. Аргументи `x` и `y` су обавезни. Могу узимати позитивне и негативне вредности и одређују за колико пиксела ће прозор бити смањен у односу на хоризонталну, односно вертикалну осу. У следећем примеру су дефинисане функције које отварају нови прозор и мењају његову величину:

```
<script>
function otvori() {
    w = window.open("", "", "width=100, height=100");
```

```
}
function promeni() {
    w.resizeBy(250,250);
    w.focus();
}
</script>
<button onclick="otvori();">Otvori novi prozor</button>
<button onclick="promeni();">Promeni velicinu</button>
```

- `resizeTo(x,y)` - поставља величину прозора на одређену димензију. Аргументи `x` и `y` су обавезни и одређују ширину и висину прозора.
- `stop()` - зауставља учитавање садржаја прозора. Ова метода је корисна када учитавање слике или оквира предуго траје.

10.5 Објекат History

Својство `history` објекта Window референцира објекат History прозора. Објекат History има једно својство, `length`, које враћа број елемената у листи историје веб читача. Објекат History подржава три методе. Помоћу метода `back()` и `forward()` могуће је кретање уназад и унапред у историји прегледања, при чему се текући документ замењује претходно прегледаним документом. Слично се дешава када корисник притисне дугмад **Back** и **Forward** веб читача.

```
<script>
function nazad() {
    window.history.back();
}
function napred() {
    window.history.forward();
}
</script>
<button onclick="nazad()">Prethodna</button><br />
<button onclick="napred()">Sledeca</button><br />
```

Пример 32. Кретање кроз историју прегледања

Трећа метода `go()` има целобројни аргумент и може да прескочи произвољан број страна испред или иза текуће стране, у зависности да ли је аргумент позитиван или негативан, у историји веб читача.

```
<script>
function preskoci() {
    window.location.go(-2);
}
</script>
<button onclick="preskoci()">Nazad 2 strane</button>
```

Пример 33. Коришћење методе `go()`

10.6 Објекат Navigator

Својство `navigator` објекта Window референцира објекат Navigator који садржи информације о веб читачу. Објекат Navigator је назван по веб читачу Netscape Navigator, али га подржавају и сви остали читачи. Објекат Navigator садржи својства која садрже информације о веб читачу:

- `appName` враћа име читача веба. Код Internet Explorera то име је Microsoft Internet Explorer. У Firefoxu ово својство има вредност Netscape.
- `appVersion` враћа платформу и верзију читача.
- `userAgent` стринг који читач уписује у заглавље USER-AGENT HTTP. Ово својство обично садржи све информације из својстава `appName` и `appVersion`, а често и додатне детаље.
- `appCodeName` кодно име читача. Netscape користи име Mozilla. Због компатибилности Internet Explorer има исту вредност овог својства.
- `platform` хардверска платформа на којој се извршава читач.
- `cookieEnabled` враћа логичку вредност у зависности да ли су колачићи доступни у веб читачу.
- `browserLanguage` враћа текући језик веб читача.
- `systemLanguage` враћа језик који користи оперативни систем.

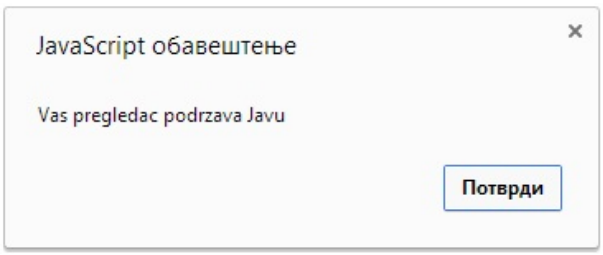
Следећи пример приказује вредност ових својстава објекта Navigator у прозору за дијалог:

```
var info = "Informacije o web pregledachu:\n";
for (var svojstvo in navigator) {
    info += svojstvo + ": " + navigator[svojstvo] + "\n";
}
alert(info);
```

Слика 16. Информације о веб читачу

Од метода објекта Navigator најчешће се користи `javaEnabled()` која враћа логичку вредност у зависности од тога да ли веб читач подржава Javu. На пример:

```
if (navigator.javaEnabled())
    alert("Vas pregledac podrzava Javu");
else
    alert("Vas pregledac ne podrzava Javu");
```



Слика 17. Подршка за Java аплете

10.7 Објекат Location

Location објекат је део Window објекта и представља URL адресу документа који је тренутно приказан у том прозору. Доступан је кроз својство `location` објекта Window.
Својства објекта Location:

- `hash` поставља или враћа део URL адресе од знака # укључујући и знак #.
- `href` поставља или враћа цео URL.
- `host` поставља или враћа назив хоста и број порта текућег URL.-а
- `hostname` поставља или враћа назив хоста текућег URL-а.
- `pathname` поставља или враћа путању текућег URL-а.
- `port` поставља или враћа број порта текућег URL-а.
- `protocol` поставља или враћа протокол текућег URL-а.
- `search` поставља или враћа URL од знака питања.

Методе објекта Location

- `assign()` - учитава нови документ. Има обавезан параметар који представља URL адресу новог документа. У следећем примеру, кликом на дугме учитава се нови документ који је на адреси www.matf.bg.ac.rs:

```
<script>
function newDoc() {
    window.location.assign("http://www.matf.bg.ac.rs");
}
</script>
</head>
<body>
<button onclick="newDoc()"> Matf </button>
```

Пример 34. Коришћење методе `assign()`

- `replace()` - мења текући документ новим. Разлика између ове методе и методе `assign()` је што ова метода уклања URL текућег документа из историје прегледа, што значи да није могуће да се коришћењем дугмета Back вратимо на оригинални документ. На пример:

```
// Menja tekuci dokument novim
location.replace("http://www.matf.bg.ac.rs");
```

- `reload()` - поново учитава текући документ. Текући документ се учитава из кеш меморије веб читача. Ако се жели да се текући документ поново учитава са сервера, овој методи се прослеђује параметар `true`:

```
location.reload(true);
```

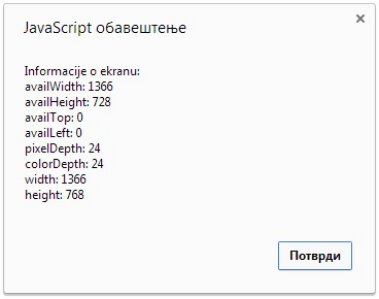
10.8 Објекат Screen

Својство `screen` објекта Window референцира објекат Screen који садржи информације о величини корисничког екрана и броју боја које подржава.
Својства `width` и `height` враћају величину екрана у пикселима. Слично, својства `availWidth` и `availHeight` враћају доступну ширину и висину екрана, без простора који заузимају елементи попут палете послова.
Помоћу својстава `availLeft` и `availTop` се задају координате прве доступне позиције на екрану. Ова два својства објекта Screen не подржавају сви читачи.
Својство `pixelDepth` враћа дубину пиксела, а својство `colorDepth` враћа дубину боје.
Пример 35 приказује информације о корисничком екрану:

```
var info = "Informacije o ekranu:\n";
    for (var svojstvo in screen) {
        info += svojstvo + ": " + screen[svojstvo] + "\n";
    }
    alert(info);
```

Пример 35. Информације о корисничком екрану

Резултат претходног примера је приказан на слици 18.



Слика 18. Информације о екрану корисника

11 Објекат Document

Сваки објекат Window дефинише својство `document` које референцира објекат Document придружен прозору, који садржи стварни садржај странице, тј. све оно што постоји у области прозора читача. Својства и методе овог објекта утичу на изглед и садржај документа који се налази у прозору. Својствима и методама овог објекта приступа се на следећи начин:

```
window.document.svojstvo
window.document.metoda(parametri)
```

У случају када се реферише објекат типа Document текућег прозора (оног у коме се извршава JavaScript) може се изоставити реферисање објекта Window:

```
document.svojstvo
document.metoda(parametri)
```

Својства објекта типа Document

- `body` - даје директан приступ `<body>` елементима.

  ```
  // Zadavanje boje pozadine
  document.body.backgroundColor = "red";
  ```

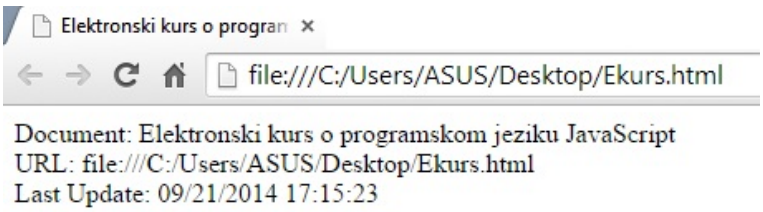
- `cookie` - својство које омогућава програмима на JavaScript-у да читају и уписују HTTP колачиће.

- `domain` - својство које омогућава веб серверима у истом домену да између себе пренебрегну поједина безбедносна ограничења која поставља правило о истом пореклу за своје веб стране.

- `lastModified` - стринг који садржи датум последње измене документа.
- `referrer` - URL адреса документа који садржи везу преко које је читач дошао на текући документ.
- `title` - текст између ознака `<title>` и `</title>` за текући документ.
- `URL` - Знаковни низ са URL адресом с које је документ учитан. Ово својство има исту вредност као својство `href` објекта Location.

Неколико ових својстава садрже опште информације о документу, што може бити корисно за посетиоца веб стране да процени колико је садржај документа актуелан или застарео. Наредни сегмент кода даје основне информације о документу и може се убацити било где на страни:

```
Document: <script type="text/javascript">
document.write(document.title);</script><br />
URL: <script type="text/javascript">
document.write(document.URL);</script><br />
Last Update: <script type="text/javascript">
document.write(document.lastModified);</script><br />
```



Слика 19. Информације о документу

У претходном примеру је коришћена је метода `document.write()` која ће бити представљена у следећем поглављу.

11.1 Методе објекта Document

Метода `document.write()` се користи од првих верзија JavaScript-а. Омогућава уписивање садржаја у документ. Може се користити на два начина.
Први начин је да се употреби у скрипту како би се HTML садржај

уписао у документ који се тренутно рашчлањује. Тада је позивамо у коду највишег нивоа у елементима `<script>` само зато што се ти скриптови извршавају у оквиру процеса рашчлањивања документа. Позив `document.write()` умеће текст у HTML документ на месту ознаке `<script>`.

```
<script>
    var datum = new Date();
    document.write("Danasnji datum: " + danas.toString());
</script>
```

Пример 36. Коришћење методе `write()`

Пример 36 умеће датум на страницу на месту где је ознака `<script>`. Други начин коришћења методе `document.write()` је у дефиницији функције која се позива из процедуре за обраду догађаја. Тада ће ова функција обрисати текући документ и скриптове које садржи, тј. отвориће нови, а одбацити постојећи документ.
Метода `write()` је погодна за прављење потпуно нових докумената у другим прозорима или оквирима, заједно са функцијама `open()` и `close()`. На пример:

```
function obavestenje() {
    var w = window.open(); // Pravi nov prazan prozor
    var d = w.document;
    d.write("<h2>Master rad</h2>");
    d.write("<p>Elektronski kurs o programskom
    jeziku JavaScript</p>");
    d.close(); // Zatvara dokument
}
```

Објекат Document подржава и методу `writeln()`, која је идентична методи `write()`, сем што после исписа аргумената додаје нови ред, па може бити корисна за исписивање унапред форматираног текста.

11.1.1 Колекције

Категорију својстава објекта типа Document, чија су вредност низови називамо колекције. Помоћу њих се може приступати одређеним елементима документа:

- `anchors[ ]` Низ објеката типа Anchor који представљају сидра у документу. Сидро се прави помоћу ознаке `<a>` која уместо атрибута `href` има атрибут `name`. Својство `name` објекта типа Anchor садржи вредност атрибута `name`. Својство `length` садржи број сидара у документу. На пример:

```
var x = document.anchors.length;
```

- `applets[ ]` Низ објеката типа Applet представљају Јавине аплете у документу.
- `forms[ ]` Низ објеката типа Form представљају елементе `<form>` у документу. Сваки објекат типа Form има сопствено својство колекције `elements[ ]`, са елементима који представљају елементе датог обрасца. Колекција `form[ ]` је најважнија у старом објектном моделу документа.
- `images[ ]` Низ објеката класе Image представљају елементе `<img>` у документу. Својство `src` објекта типа Image садржи URL слике.
- `links[ ]` Низ објеката класе Link представљају хипервезе у документу. Хипервезе се праве помоћу ознака `<a>`. Својство `href` објекта типа Link чува URL адресу везе.

Објекти који садрже ове старе колекције објектног модела документа могу се скриптовати, али ни један од њих неће омогућити да се измени структура документа. Може се изменити одредиште везе, читати и задавати вредности елемената обрасца, заменити једну слику другом, али се не може променити текст документа.
Приступање објектима документа помоћу броја позиције у колекцији је нестабилно, мале промене у редоследу његових елемената могу да обезвреде код који зависи од тог редоследа. Боље решење је доделити елементима документа имена како би им се могло приступити преко имена. У старом објектном моделу документа, за то служи атрибут `name` слика, аплета, веза, образаца и елемената образаца. На пример, слици која представља лого неке органицазије може се доделити вредност `logo` атрибуту `name`:

```
<img name="logo" src="logo.gif"</img>
```

Нека је наведена слика прва у документу. JavaScript може да позове дати објекат помоћу неког од следећих израза:

```
// Pristup pomocu pozicije u dokumentu
document.image[0]
// Pristup pomocu imena svojstva
document.image.logo
// Pristup pomocu imena kao indeksa niza
document.image["logo"]
```

И елементима у обрасцу се може доделити име:

```
<form name="login">
<input type="text" name="username">
<input type="text" name="password">
<input type="submit" name="submit">
</form>
```

Пољу за унос корисничког имена може се приступити помоћу наредне синтаксе:

```
document.login.username
```

Ако два елемента имају атрибуте `name` са истом вредношћу, на пример `m`, тада својство `document.m` постаје низ који чува референце на оба елемента.

11.2 Колачићи

HTTP (HyperText Transfer Protocol) је протокол без стања. То значи да након завршене комуникације између сервера и веб читача, веза између сервера и читача се губи, а ни сервер ни читач немају податке о завршеној комуникацији. Са развојем интернета јавила се потреба да се неки подаци, као што су наруџбине купаца, датум последње посете и слично, негде ускладиште. Netscape је 1994. управо из претходно наведених разлога увео колачиће.
Колачићи су мали делови текста које веб читач памти и који су повезани са одређеном интернет страном. Служе као меморија веб читача. Најчешће се користе за проверу аутентичности, праћење сесије, памћење информација о кориснику, његовог имена, последње посете итд. Сви модерни веб читачи подржавају колачиће, али сами корисници могу да их онемогуће. Од модерних читача се очекује да могу да ускладиште најмање 300 колачића, величине 4kb. За сваки сервер може се сачувати до 20 колачића.
JavaScript може управљати колачићима помоћу својства `cookie` објекта типа Document.
Сваки колачић има шест атрибута: `name` (име), `value` (вредност), `expire` (животни век), `domain` (име домена), `path` (путања) и `secure` (безбедност).
Прва два атрибута, име и вредност, су обавезна, остали атрибути су опциони и није битан редослед у коме се наводе. Основна синтакса је:

```
document.cookie = "IME=VREDNOST;expires=DATUM;
path=PUTANJA;domain=DOMEN;secure;"
```

Име (name)

Име дефинише уписани колачић, док је вредност информација коју треба запамтити како би се идентификовао посетилац. Име и вредност морају бити јединствен стринг без зареза, тачка-зареза или белина. На пример:

```
document.cookie = "ime=Pera";
```

Може се сачувати више парова име/вредност, и тада се раздвајају зарезима, на пример:

```
document.cookie = "ime=Pera, prezime=Peric, godine=48";
```

Датум истека колачића (expire date)

Атрибут `expire` је датум који дефинише до када колачић остаје меморисан у веб читачу. Колачић има ограничен животни век. Он ће нестати када корисник затвори веб читач. Да би се колачић сачувао на диску, мора се подесити датум истека у следећем формату:

```
dan_u_nedelji, DD-MON-YY HH:MM:SS GMT
```

На пример:

```
;expires=Thu, 1 Jan 2015 00:00:00 GTM;";
```

Претходни колачић истиче 1. Јануара 2015. године у поноћ.
Пример 52 приказује функцију `setCookie`, која поставља колачић, а којој прослеђујемо три параметра, `cname` - име, `cvalue` - вредност, `exdays` - колико дана ће колачић бити меморисан у читачу.

```
function setCookie(cname, cvalue, exdays) {
    var d = new Date();
    d.setTime(d.getTime() + exdays*24*60*60*1000);
    var expires = "expires="+d.toUTCString();
    document.cookie = cname + "=" +cvalue + ";" + expires;
}
```

Пример 52. Функција која поставља колачић

Уместо атрибута `expire` може се користити атрибут `max-age`, помоћу ког се задаје животни век колачића у секундама. Пошто прође датум престанка важења, или се достигне максимални животни век задат вредношћу атрибута `max-age`, читач аутоматски брише колачић из датотеке.
Колачић се може обрисати постављањем колачића са истим именом, доменом и путањом (ако су постављени) и са датумом истека у прошлости. На пример:

```
document.cookie = "username=Pera;
expires=Thu, 01-Jan-1970 00:00:00 GMT";
```

Путања (path)

Важан атрибут колачића је `path` (путања). Она одређује веб страну с којом је колачић повезан. Колачић је подразумевано придружен и доступан веб страни која га је направила и било којој другој веб страни у истом директоријуму, или поддиректоријумима тог директоријума.
На пример, ако је колачић направила веб страна www.javascript.com/podaci/string.html, онда је тај колачић видљив и страни www.javascript.com-/podaci/niz.html и страни www.javascript.com/podaci/objekti/metode.html, али не и страни www.javascript.com/index.html.

Име домена (domain)

Следећа опција је `domain`, која указује на домен за који би колачић требало да буде доступан, тј. домен са којег колачић може да се чита и да му се мења вредност. Подразумевана вредност је име домена веб сервера стране. Ова опција се најчешће не користи.

Заштићен (secure)

Последња опција је `secure`. За разлику од других опција, ова опција нема других вредности осим `secure`. Ако је ова опција наведена, колачић ће се слати само серверу чији је захтев направљен коришћењем SSL и HTTPS протокола (протоколи који се користе за пренос поверљивих информација). У пракси, поверљиве информације никада не треба да се чувају и преносе помоћу колачића, јер цео механизам може бити несигуран.

Као што је већ речено, колачићи се налазе на корисниковом рачунару и корисник може читати и мењати вредност колачића након што је постављен од стране сервера. Из тог разлога није добра идеја да се

осетљиве информације, као што су поверљиве шифре, бројеви банковних рачуна, чувају у колачићу.

11.3 W3C DOM стандард

Објектни модел документа (енг. Document Object Model, DOM) јесте интерфејс за програмирање апликација који дефинише начин приступања објектима који сачињавају документ. У првим данима веба, Netscape као водећи произвођач читача веба, дефинисао је интерфејсе за програмирање апликација за писање клијентских скриптова. То је био једноставан објектни модел документа који је омогућавао приступ само посебним елементима документа, попут хипервеза, слика и елемената образаца. Овај објектни модел документа усвојили су сви произвођачи читача и познатији је као објектни модел документа нултог нивоа - Level 0 DOM. Овакав објектни модел документа функционише у свим читачима.
Проширена верзија једноставног DOM-а је W3C Document Object Model - платформа и језички неутралан интерфејс који омогућава програмима и скриптовима динамички приступ и измену садржаја, структуре и изгледа документа. DOM дефинише објекте и својства свих елемената документа и методе за приступ тим објектима.
DOM је дизајниран да буде независан од било ког програмског језика.

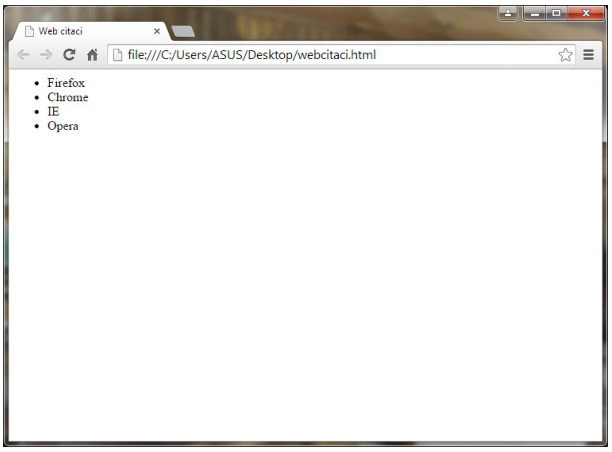
11.4 Представљање HTML документа у виду стабла

Приказ HTML документа у виду стабла садржи чворове који представљају HTML ознаке, тј. елементе, попут ознака `<p>` или `<h1>`, и чворове који представљају текст.

```
<html>
<head>
<title>Web citaci</title>
</head>
<body>
<ul id="browser">
    <li>Firefox</li>
    <li>Chrome</li>
    <li>IE</li>
    <li>Opera</li>
</ul>
</body>
```

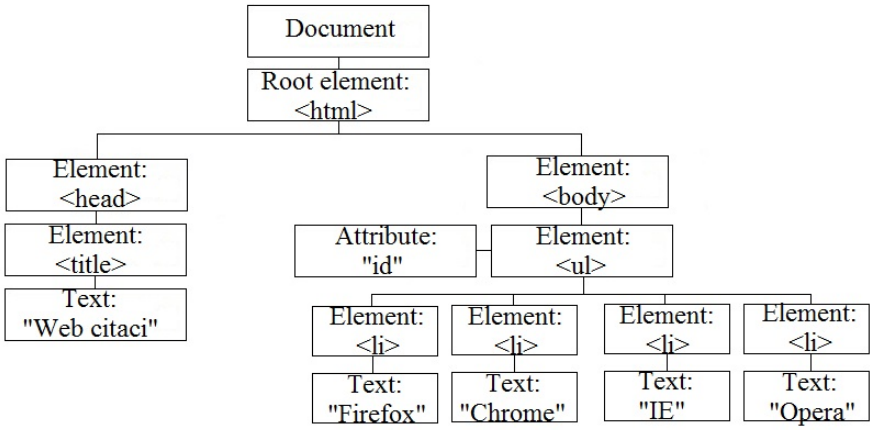
```
</html>
```

Пример 37. HTML документ



Слика 20. Изглед веб стране

Изглед овог документа у виду стабла је приказан на слици 21.



Слика 21. Приказ стране у облику стабла

Цео документ је документ чвор, а сваки HTML таг је елемент чвор. Текстови садржани у HTML елементима су текст чворови. Сваки HTML атрибут је атрибут чвор, а коментари су коментар чворови.

Терминологија у раду са структуром у виду стабла позајмљена је из породичних стабала. Чвор изнад датог чвора је родитељ тог чвора.
Сваки чвор изузев документ чвора има родитељски чвор. На пример, родитељски чвор од `<head>` чвора је `<html>` чвор, а родитељски чвор од текст чвора „Opera"је `<li>` чвор.
Чворови један ниво испод датог чвора су његови потомци. На пример, `<head>` чвор има `<title>` чвор као свој потомак. `<title>` чвор такође има један потомак текст чвор „Web citaci".
Чворови на истом нивоу, и са истим родитељем су братски чворови.
Чворови било ког нивоа испод текућег чвора су његови потомци.
Родитељ, прародитељ и сви остали чворови изнад текућег чвора су његови преци.

11.5 Информације о чворовима

Сваки чвор има својства која садрже информације о чворовима. То су следећа својства:

- `nodeName` садржи име чвора. У зависности од врсте чвора може имати различите вредности.
  - код елемент чвора ово својство има вредност тага.
  - код атрибут чвора ово својство је назив атрибута.
  - код текст чвора ово својство увек има вредност `#text`.
  - код документ чвора ово својство има вредност `#document`.

- `nodeType` својство враћа тип чвора.

Табела 10. Тип чвора

Тип елемента	Тип чвора
Елемент	1
Атрибут	2
Текст	3
Коментар	8
Документ	9

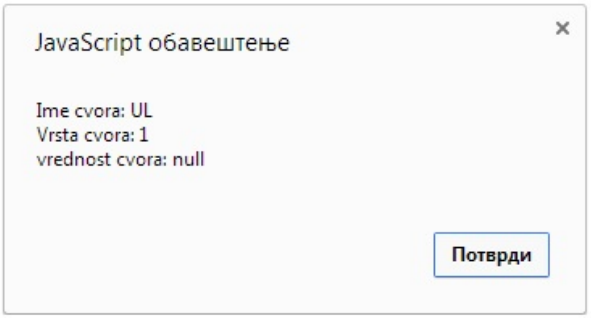
- `nodeValue` у зависности од чворова, ова својство може имати различите вредности.
  - код текст чворова, ова особина садржи текст.

- код атрибут чворова ова особина садржи вредност атрибута.
- код документ и елемент чворова ово својство није доступно.

```
function cvorInfo(cvor) {
    var info = "";
    info += "Ime cvora: "+cvor.nodeName+"\n";
    info += "Vrsta cvora: "+cvor.nodeType+"\n";
    info += "vrednost cvora: "+cvor.nodeValue+"\n";

    return info;
}
var cvor = document.getElementById('browser');
var poruka = cvorInfo(cvor);
alert(poruka);
```

Пример 38. Информације о чвору

Резултат примера 38 је приказан на слици 22.



Слика 22. Информације о чвору

11.6 Кретање по стаблу документа

Већ је напоменуто да објектни модел документа представља HTML документ као стабло са чворовима. Структура у виду стабла омогућава приступ сваком његовом чвору.
Својство `childNodes` враћа листу деце чвора. Сваки чвор може имати произвољан број деце. Својство `parentNode` враћа родитељски чвор елемента. На пример, ако је променљива `x` чвор:

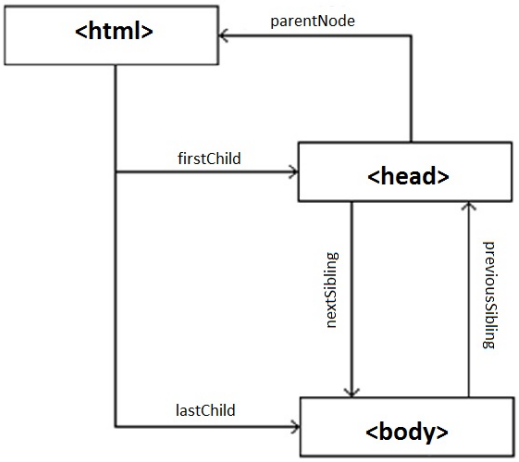
```
x.childNodes // Vraca listu cvorova dece cvora x
x.parentNode // Vraca roditeljski cvor cvora x
```

У примеру 39, чвор се прослеђује функцији која проверава да ли тај чвор представља HTML ознаку. Рекурзивно позива саму себе за сваки потомак датог чвора. Враћа укупан број елеменат чворова на који наиђе. Ако се функција позове прослеђујући јој објекат типа Document, прегледаће све чворове DOM стабла.

```
function izbroj(cvor) {
    var brojCvorova = 0;
    if (cvor.nodeType == 1)
        brojCvorova++;
    var children = cvor.childNodes;
    for(var i=0; i < children.length; i++) {
        brojCvorova += izbroj(children[i]);
    }
    return brojCvorova;
}
```

Пример 39. Функција која враћа укупан број елеменат чворова

Својства `firstChild` и `lastChild` представљају прво и последње дете чвора, док својства `nextSibling` и `previousSibling` представљају суседне братске чворове.



Слика 23. Однос између чворова

У примеру 40 функција проналази потомке елемента `body`

```
<html>
<body>
<div>Web citaci:</div>
<ul>
<li>Firefox</li>
<li>Chrome</li>
<li>IE</li>
<li>Opera</li>
</ul>
<!--  comentar cvor -->
<script>
function pronadji() {
    var childNodes = document.body.childNodes;
    var pom = "";
        for(var i=0; i<childNodes.length; i++) {
        pom +=childNodes[i] +"\n";
        }
    alert(pom);
    }
</script>
<button onclick="pronadji()">Go!</button>
</body>
</html>
```

Пример 40. Функција која проналази потомке елемента `<body>`

Резултат претходне функције је приказан на слици 24.

Слика 24. Потомци елемента `<body>`

11.7 Проналажење елемената у документу

Помоћу DOM-а могуће је приступити сваком чвору у HTML документу на неколико начина:

- користећи `getElementsByTagName()`, `getElementById()` и `getElementsByName()` методе.
- користећи `parentNode`, `firstChild` и `lastChild` својства елемент чворова.

Помоћу методе `getElementsByTagName()` може се направити листа HTML елемената произвољног типа. Следећи пример показује како могу да се пронађу све табеле у документу помоћу ове методе:

```
var tabele = document.getElementsByTagName("table");
alert('U dokumentu se nalazi ' + tabele.length + 'tabela');
```

Метода `getElementsByTagName()` враћа елементе редом којим се појављују у документу. Ако се методи `getElementsByTagName()` проследи специјалан знаковни низ "*", она враћа листу свих елемената у документу редом којим се појављују у њему:

```
var sviElementi = document.getElementsByTagName("*");
```

Ако се добро познаје структура документа, помоћу ове методе може се приступити одређеном елементу на следећи начин:

```
var tabela3 = document.getElementsByTagName("table")[2];
```

Променљива `tabela3` из претходног примера ће представљати трећу табелу по реду у документу.
Међутим, ово није најбоља техника јер зависи од структуре документа. Ако се уметне нова табела, или избрише нека постојећа, промениће се читав код.
Зато, када се ради са појединачним елементима документа, најбоље је да се тим елементима придружи атрибут `id` којим се задаје јединствено име тог елемента. Тада ће се жељеном елементу моћи приступити помоћу методе `getElementById()`. Ако се трећој табели у документу додели јединствено име `studenti`, као вредност атрибута `id`:

```
<table id="studenti" >
```

тада се тој табели најлакше може приступити на следећи начин:

```
tabela3 = document.getElementById("studenti");
```

Пошто је вредност атрибута `id` јединствена, метода `getElementById()` враћа само један елемент са одговарајућим атрибутом `id`. На пример, ако треба да се изброји број редова у табели, користиће се комбинација ове две методе:

```
var table3 = document.getElementById("studenti");
var redovi = table3.getElementsByTagName("tr");
var brojRedova = redovi.length;
```

Често се користи још једна метода, `getElementsByName()` која претражује елементе на основу атрибута `name`. Пошто се не очекује да атрибут `name` има јединствену вредност метода `getElementsByName()` враћа низ елемената. Следећи код проналази све елементе чији атрибут `name` има вредност `hidden`:

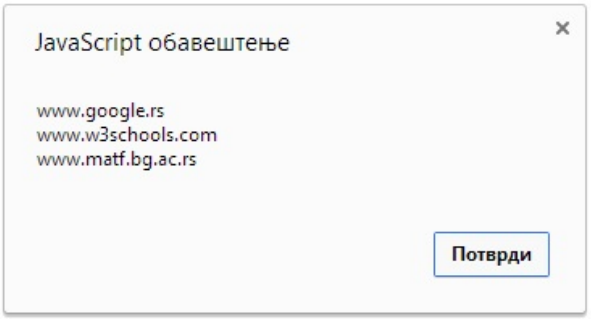
```
sakriveni = document.getElementsByName("hidden");
```

Ова метода се не понаша исто у свим веб читачима. У читачима Internet Explorer и Opera, метода `getElementsByName()` враћа и елементе чији је `id` једнак прослеђеном аргументу, па је треба пажљиво користити.

Атрибути елемената могу се читати, задавати или брисати помоћу метода `getAttribute()`, `setAttribute()` и `removeAttribute()`. Пример 41 проналази све линкове и исписује вредности атрибута `href`:

```
<html>
<head>
</head>
<body>
<a href="www.google.rs">Google</a><br />
<a href="www.w3schools.com">W3 Schools</a><br />
<a href="www.matf.bg.ac.rs">Matf</a>
<script type="text/javascript">
var pom = "";
    var x = document.getElementsByTagName("a");
    for(var i = 0; i < x.length; i++) {
        pom += x[i].getAttribute("href")+"\n";
    }
    alert(pom);
</script>
</body>
</html>
```

Пример 41. Проналажење линкова у документу



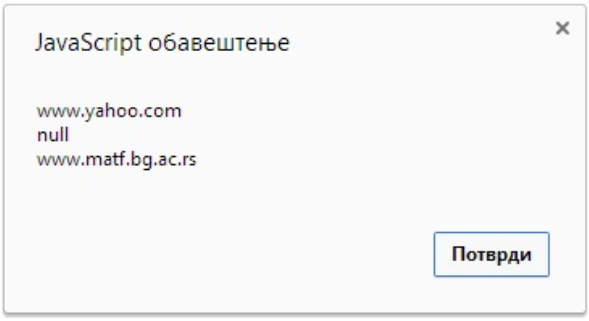
Слика 25. Сви линкови у документу

Комбинујући све три методе у примеру 42 може се мењати вредност или уклањати атрибут `href` ознаке `<a>`.

```
<html>
<head>
</head>
<body>
<a href="www.google.rs">Google</a><br />
<a href="www.w3schools.com">W3 Schools</a><br />
<a href="www.matf.bg.ac.rs">Matf</a>

<script type="text/javascript">
var x = document.getElementsByTagName("a");
    for(var i = 0; i < x.length; i++) {
    if (x[i].getAttribute("href")=="www.google.rs")
        {
            x[i].setAttribute("href","www.yahoo.com");
        }
    else if (x[i].getAttribute("href")=="www.w3schools.com")
        {
            x[i].removeAttribute("href");
        }
}
var pom = "";
    for(var i = 0; i < x.length; i++) {
        pom += x[i].getAttribute("href")+"\n";
}
```

```
alert(pom);
</script>
</body>
</html>
```

Пример 42. Коришћење метода `getAttribute()`, `setAttribute()` *и* `removeAttribute()`



Слика 26. Измена и брисање линкова у документу

11.8 Својство innerHTML

Својство innerHTML није део W3C DOM стандарда, али га због своје важности подржава већина модерних веб читача. Омогућава читање и задавање текстуалног садржаја HTML елемента. Следећи пример задаје текстуални садржај елемента `<p>` са јединственим атрибутом `id`:

```
<body>
    <p id = "dobrodoslica">Dobrodosli u JavaScript</p>
    <script type="text/javascript">
        poruka = document.getElementById("dobrodoslica");
        poruka.innerHTML = "Hvala sto koristite JavaScript";
    </script>
</body>
```

Слика 27. Својство innerHTML

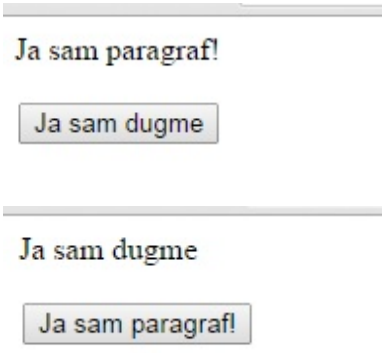
Следећи код врши приказивање свих линкова у документу користећи својство `innerHTML`:

```
<script type="text/javascript">
    var a = document.getElementByTagName("a");
    for (var i = 0; i < a.length; i++) {
        document.write(a[i].innerHTML +"<br />");
    }
</script>
```

Својство `innerHTML` подржавају само елемент чворови. За остале типове чворова се користи својство `nodeValue`. У примеру 43 комбинацијом својстава `innerHTML` и `nodeValue` замењује се текст у параграфу и текст дугмета:

```
<body>
    <p id = "para1">Ja sam paragraf!</p>
    <button onclick = "myFunction()">Ja sam dugme</button>
<script>
    function myFunction() {
        // Pronalazi prvo dugme u dokumentu
        var c = document.getElementsByTagName("BUTTON")[0];
        // Pronalazi paragraf ciji je id=para1
        var x = document.getElementById("para1");
        // Uobicajeni nacin razmene vrednosti
        var pom = x.innerHTML;
        x.innerHTML = c.childNodes[0].nodeValue;
        c.childNodes[0].nodeValue = pom;
    }
</script>
</body>
```

Пример 43. Коришћење својстава `nodeValue` *и* `innerHTML`



Слика 28. Замена текста параграфа и дугмета

11.9 Додавање и брисање чворова

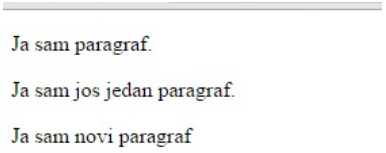
Методе `createElement()` и `createTextNode()` праве нове чворове типа Element и Text на следећи начин:

```
// Kreira novi div element
var noviDiv = document.createElement();

// Kreira novi tekst cvor
var txtCvor = document.createTextNode("Novi tekst cvor");
```

Помоћу метода `appendChild()`, `insertBefore()` и `replaceChild()` чворови се могу додати документу. Ове методе омогућавају да се направи DOM стабло с произвољним садржајем документа.
Метода `appendChild()` у примеру 44 додаје дете чвор постојећем елементу:

```
<div id="div1">
    <p id="p1">Ja sam paragraf.</p>
    <p id="p2">Ja sam jos jedan paragraf.</p>
</div>

<script>
    // Kreira novi <p> element
    var para = document.createElement("p");
    // Kreira novi tekst cvor
    var node = document.createTextNode("Ja sam novi paragraf.");
    // Dofaje tekst cvor elementu <p>
    para.appendChild(node);
    // Trazi element kome dodajemo novi paragraf
    var element = document.getElementById("div1");
    // Dodaje paragraf <div> elementu
    element.appendChild(para);
</script>
```

Пример 44. Коришћење методе `appendChild()`



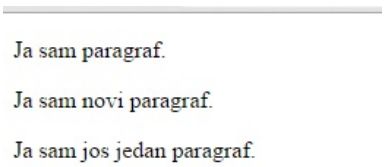
Слика 29. Додавање новог чвора

Метода `appendChild()` из претходног примера 44 додаје чвору `<div>` потомак чвор `<p>`, тако да `<p>` постаје последњи чвор потомак чвора `<div>`.
Ако, на пример, нови чвор треба да се уметне на одређену позицију, користи се метода `insertBefore()`. У примеру 45 умеће се нови чвор између два постојећа чвора:

```
<div id="div1">
    <p id="p1">Ja sam paragraf.</p>
    <p id="p2">Ja sam jos jedan paragraf.</p>
</div>

<script>
    // Kreiramo novi <p> element
    var para = document.createElement("p");
    // Kreiramo novi tekst cvor
    var node = document.createTextNode("Ja sam novi paragraf.");
    // Dofajemo tekst cvor elementu <p>
    para.appendChild(node);
    // Trazimo element kome dodajemo novi paragraf
    var element = document.getElementById("div1");
    // Trazimo element izpred koga dodajemo novi paragraf
    var child = document.getElementById("p2");
    // Umecemo cvor
    element.insertBefore(para,child);
</script>
```

Пример 45. Коришћење методе `insertBefore()`

Резултат претходног примера је приказан на слици 30.



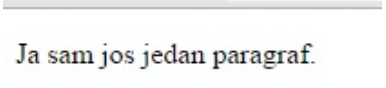
Слика 30. Уметање чвора на одређену позицију

Ако треба да се уклони неки чвор из постојећег стабла користи се метода `removeChild()`, као што се може видети у следећем примеру:

```
<div id="div1">
    <p id="p1">Ja sam paragraf.</p>
    <p id="p2">Ja sam jos jedan paragraf.</p>
</div>

<script>
    // Trazimo element sa id = div1
    var parent = document.getElementById("div1");
    // Trazimo element sa id = p1
    var child = document.getElementById("p1");
    // Uklanjamo element
    parent.removeChild(child);
</script>
```

Пример 46. Коришћење методе `removeChild()`

Резултат претходног примера је приказан на слици 31.



Слика 31. Уклањање чвора из документа

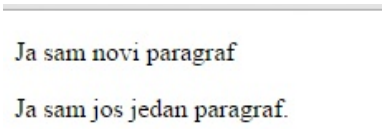
Ако треба да се измени већ постојећи елемент у DOM стаблу, користи се метода `replaceChild()`. Пример 47 замењује постојећи `<p>` чвор новим чвором:

```
<div id="div1">
    <p id="p1">Ja sam paragraf.</p>
    <p id="p2">Ja sam jos jedan paragraf.</p>
</div>

<script>
    // Kreiramo element cvor
    var para = document.createElement("p");
    // Kreiramo tekst cvor
    var node = document.createTextNode("Ja sam novi paragraf");
    // Dodajemo tekst cvor element cvoru
    para.appendChild(node);
```

```
    // Trazimo element sa id = div1
    var parent = document.getElementById("div1");
    // Trazimo element sa id = p1
    var child = document.getElementById("p1");
    // Menjamo element
    parent.replaceChild(para,child);
</script>
```

Пример 47. Коришћење методе `replaceChild()`

А резултат примера 47 је на слици 32.



Слика 32. Замена постојећег чвора новим чвором

12 Изузеци

Изузетак (енг. exception) указује на изузетну околност или на грешку. Изузеци се у JavaScript-у избацују када настане грешка при извршавању или када програм експлицитно изнуди бацање изузетка помоћу наредбе `throw`. Изузеци се хватају помоћу наредаба `try/catch/finally`. Наредба `throw` има следећу синтаксу:

```
throw izraz;
```

при чему израз може имати вредност произвољног типа. Најчешће је то објекат типа Error. Корисно је да се при избацивању изузетка испише порука о грешци. У следећем примеру, избацује се порука о грешци када је број који се коренује негативан:

```
function kvadratni_koren(x) {
    // Ako je argument neprihvatljiv, izbacuje izuzetak
    if (x<0) {
        throw new Error("x mora biti pozitivno");
    }
    // U suprotnom normalno racuna kvadratni koren
    else
```

```
        return Math.sqrt(x);
}
```

Пример 48. Квадратни корен

Наредба `try/catch/finally` представља механизам за обраду изузетака у JavaScript-у. Наредба `try` дефинише блок кода чији се изузеци обрађују. Иза блока `try` наводи се блок наредаба `catch` које се позивају када се јави изузетак у блоку `try`. Иза наредбе `catch` следи наредба `finally` са кодом за чишћење који се свакако извршава без обзира на то шта се дешава у блоку `try`. Може се изоставити блок `catch` или блок `finally`, али један од њих мора пратити блок `try`. Основна синтакса је:

```
try {
    // Ovaj blok koda bi se u normalnim okolnostima izvrsavao
    // od vrha ka dnu. Ali, ponekad moze da izbaci izuzetak
    // pomocu naredbe throw
}
catch(e) {
    // Ovaj blok se izvrsava samo ako blok try izbaci
    // izuzetak. Vrednost izbacenog izuzetka predstavlja
    // se promenljivom e
}
finally {
    // Ovaj blok sadrzi naredbe koje se uvek izvrsavaju
    // bez obzira sta se desava u bloku try
}
```

Пример 49. Основна синтакса наредбе `try/catch/finally`

Пример 50 показује коришћење наредбе `try/catch` без одредбе `finally`, која се употребљава знатно ређе од блока `catch`:

```
try {
    // Trazhi od korisnika da unese broj
    var x = prompt("Unesite pozitivan broj", "");
    // Racuna koren pretpostavljajuci da je unos ispravan
    var koren = kvadratni_koren(x);
    alert(x);
}
catch(ex) { /* Ako unos nije bio valjan kod se ovde zavrsava
             i saopstava gresku */
alert(ex);
}
```

Пример 50. Коришћење методе `try/catch`

У нормалном случају, ток програма долази до краја блока `try`, а потом прелази на блок `finally`. Ако се извршавање блока `try` заврши због наредбе `return`, `continue` или `break`, блок `finally` се извршава пре него што контрола програма пређе на ново одредиште.
Блокови `try` и `finally` могу се користити заједно без одредбе `catch`. У примеру 51, наредба `try/finally` обезбеђује да се бројач увећа на крају сваке итерације, чак иако се итерација прекине због наредбе `continue`:

```
var i = 0, suma = 0;
while(i < a.length) {
try {
    if ((tupeof a[i] != "number") || isNaN(a[i])) // Ako nije broj
        continue; // Sledi nova iteracija petlje
suma += a[i]; // U suprotnom, broj se dodaje ukupnoj sumi
    }
finally {
    i++; // Promenljiva i se uvek uvecava
}
}
```

Пример 51. Коришћење методе `try/finally`

13 Догађаји

Веб читачи имају могућност да позову JavaScript као одговор на корисникову акцију у оквиру веб стране. Акције корисника на које JavaScript може да одговори називају се догађаји (енг. events). Догађаји омогућавају интерактивност веб страница које реагују на оно што корисник ради. Програмер региструје процедуру за обраду догађаја (енг. event handler) - JavaScript функцију или сегмент кода. Затим, када се деси одређени догађај, читач позива код за обраду догађаја. Већина сложенијих програма на JavaScript-у у великој мери зависи од процедура за обраду догађаја.
Системски догађаји не захтевају никакву акцију корисника да би се активирали. То су, на пример, догађаји да је се документ учитао или да је протекао одређени временски период.
За разлику од системских догађаја, догађаји узроковани акцијама миша или тастатуре захтевају неку акцију корисника да би се активирали.

Процедуре за обраду догађаја као атрибути

У првобитном моделу догађаја, догађај је апстракција у оквиру читача веба и JavaScript не може директно да управља догађајем. У том моделу, код за обраду догађаја се задаје помоћу атрибута HTML елемената на следећи начин:

```
<a href = "http://matf.bg.ac.rs" onmouseover =
"alert("Web strana matematickog fakulteta");">
```

Првобитни модел догађаја дозвољава да се користи већи број различитих атрибута за обраду догађаја, наведених у следећој листи

- `onblur` - јавља се када елемент губи фокус, што значи да корисник активира неки други елемент, најчешће када кликне на неки други елемент.
- `onfocus` - указује да је елемент у фокусу, супротно од методе `onblur`. У следећем примеру поље за унос текста мења боју када се кликне на њега.

```
    <form id="myForm">
    Ime:
    <input type="text" id="ime" onfocus="style.backgroundColor='yellow'; "
onblur="style.backgroundColor= ' '; ">
    </form>
```

Пример 53. Методе `focus` *и* `blur`

- `onchange` - сигнализира да је поље обрасца изгубило фокус корисника, а вредност поља измењена током последњег приступа.
- `onclick` - указује да је кликнуто мишем на дати елемент.
- `ondbclick` - указује да је на дати елемент кликнуто двоструким кликом миша.
- `onkeydown` - указује да притиснут тастер са фокусом на датом елементу.
- `onkeypress` - указује да је тастер притиснут и пуштен са фокусом на датом елементу.

- `onkeyup` - указује да је тастер пуштен са фокусом на датом елементу.
- `onload` - указује да је објекат учитан у веб читач. Често се користи да би се позвала нека функција тек пошто се учита цео документ. Следећи пример проверава да ли веб читач дозвољава употребу колачића тек пошто се цео документ учита:

```
<body onload = "kolacic()" >
<script>
    function kolacic() {
        if (navigator.cookieEnabled == true) {
            alert('Vas pregledac podrzhava kolacice');
        }
        else {
            alert('Kolacici nisu omoguceni');
        }
    }
</script>
```

- `onmousedown` - указује на клик мишем без отпуштања на дати елемент.
- `onmousemove` - указује да је показивач миша померен док се налази изнад датог елемента.
- `onmouseout` - указује да је показивач миша померен ван елемента.
- `onmouseover` - указује да је показивач миша померен преко елемента.
- `onmouseup` - указује на отпуштање тастера миша са фокусом на датом елементу.
- `onreset` - указује да је образац ресетован, најчешће притиском на дугме `Reset`.
- `onselect` - указује на селектовање текста од стране корисника.
- `onsubmit` - указује на подношење обрасца, најчешће кликом на дугме `Submit`.
- `onunload` - указује да је веб читач напустио текући документ.

Вредност атрибута за обраду догађаја је произвољан JavaScript код. Ако се процедура састоји од више наредаба оне се морају раздвојити помоћу тачке и зареза као што је приказано у следећем примеру:

```
<input type="button" value="Press me"
onclick="if (window.brojKlikova) brojKlikova++;
else brojKlikova=1;">
```

Пример 54. Процедура за обраду догађаја

Уколико је за процедуру за обраду догађаја потребно више наредби, једноставније је да се дефинишу у телу функције, а онда се функција наведе као вредност атрибута за обраду догађаја, на пример:

```
<form action="login.php" onsubmit="return validiraj();">
```

Функцију `validiraj()` претходно дефинишемо, тако да проверава да ли су подаци у обрасцу исправни и враћа логичку вредност. У зависности од резултата форма се шаље или не шаље серверу.
Пошто HTML не разликује мала од великих слова свеједно је како се пишу атрибути за обраду догађаја. Уобичајено је да се користи комбинација великих и малих слова, на пример префикс `on` малим словима, а други део речи са почетним великим словом, `onClick`, `onSubmit`, `onChange`. Други начин је коришћење малих слова због компатибилности са XHTML-ом који разликује велика и мала слова.

Процедуре за обраду догађаја као својства

Да би се елементу документа додала процедура за обраду догађаја помоћу JavaScript-а, својству процедуре за обраду догађаја додељује се жељена функција. На пример, ако је дат следећи образац:

```
<form name="forma1">
<input type="text" name="username">
<input type="password" name="password">
<input type="button" name="registruj" value="Registruj">
</form>
```

дугме у овом обрасцу може се позвати помоћу израза `document.forma1.registruj`. То значи да је могуће доделити му процедуру за обраду догађаја помоћу следеће наредбе:

```
document.forma1.registruj.onclick=function() {
    alert('Hvala na registraciji!');
    };
```

Процедура за обраду догађаја може се доделити и на следећи начин:

```
function f() {
    document.forma1.registruj.value += " nalog";
}
document.forma1.registruj.value = f;
```

У последњем реду претходног примера, после имена функције `f` нема заграда `()`. Својству процедуре за обраду догађаја додељује се функција, а не резултат позивања функције.
Представљање процедура за обраду догађаја као својства JavaScript-а има неколико предности. Најважнија је то да раздваја HTML и JavaScript, што даје јаснији и флексибилнији код. Осим тога, омогућава да функције буду динамичке, тј. да се могу изменити у било ком тренутку.
Могуће је да се зада једна функција која ће бити процедура за обраду догађаја за више елемената документа. Пример 55 додаје процедуру за обраду догађаја свим линковима у документу:

```
function potvrdi() {
    return confirm("Da li ste sigurni da zhelite da
    posetite " + this.href + "?");
}
function potvrdiSvima() {
    for(var i = 0; i < document.links.length; i++) {
        document.link[i].onclick = potvrdi;
    }
}
```

Пример 55. Процедура за обраду догађаја за све линкове у документу

13.1 Регистрација процедуре за обраду догађаја

Претходне технике за обраду догађаја су део стандарда DOM Level 0, чије окружење подржава сваки веб читач. DOM Level 2 дефинише напредно окружење за програмирање намењено обради догађаја које се знатно разликује од оног дефинисаног по стандарду DOM Level 0. Већина модерних читача, осим Internet Explorer-а подржава оба стандарда.

Преношење догађаја

Преношење догађаја се одвија у три фазе. Прва фаза је хватање догађаја, где се догађај преноси од објекта типа Document низ стабло

до одредишног чвора. Ако било који предак одредишног чвора има посебно регистровану процедуру за обраду догађаја која служи за хватање, те процедуре се извршавају током ове фазе преношења.
Следећа фаза се одиграва на одредишном чвору. Свака одговарајуће регистрована процедура за обраду догађаја извршава се директно на одредишном елементу.
Трећа фаза је успињање. Тада се догађај преноси у супротном смеру, тј. од одредишног елемента до објекта типа Document. Ова фаза се дешава само за неке типове догађаја.

У објектном моделу документа другог нивоа, процедура за обраду догађаја за одређени елемент региструје се позивањем методе `addEventListener()` тог објекта. Ова метода има три аргумента.
Први је име типа догађаја за који се региструје процедура. Тип догађаја је стринг који садржи име HTML атрибута процедуре за обраду догађаја написано малим словима, без префикса „on". На пример, ако се користи HTML атрибут `onmousedown` или својство `onmousedown` у објектном моделу документа у нултом нивоу, у моделу другог нивоа употребљава се назив `mousedown`.
Други аргумент ове методе је функција која би требало да се позове када се деси одређени догађај.
Последњи аргумент методе `addEventListener()` је логичка вредност. Ако је то вредност `true`, наведена процедура за обраду догађаја хвата догађај током фазе преношења догађаја као у наредном примеру:

```
var mojDiv = document.getElementById("mojDiv");
mojDiv.addEventListener("mousedown", rukovanjeKlikom, true);
```

Претходни пример показује коришћење методе `addEventListener()` којом се хватају сви догађаји притискања тастера миша унутар `<div>` елемента чији атрибут `id` има вредност `mojDiv`. Ако је вредност овог аргумента `false`, позиваће се нормална процедура за обраду догађаја која се покреће када се догађај деси директно објекту као у наредном примеру:

```
var mojeDugme = document.getElementById("mojeDugme");
mojeDugme.addEventListener('click', function()
{alert("Hello World");}, false);
```

Подразумевана вредност је `false`.
Једном документу је могуће додати произвољан број процедура за обраду догађаја. На пример:

```
document.addEventListener("click",funkcija1());
document.addEventListener("keypress",funkcija2());
document.addEventListener("mousemove",funkcija3());
```

За веб читаче који не подржавају методу `addEventListener()` користи се `attachEvent()` метода приказана у наредном примеру:

```
// Vecina veb citaca
if (document.addEventListener) {
    document.addEventListener("click", myFunction);
}
// IE 8 i ranije verzije
else if (document.attachEvent) {
    document.attachEvent("onclick", myFunction);
}
```

Уз методу `addEventListener()` користи се метода `removeEventListener()`, која има иста три аргумента и која уклања функцију за обраду догађаја, као у следећем примеру:

```
// Pozivanje metode addEventListener()
document.addEventListener("mousemove", pomerajMisa);

// Uklanjanje funkcije za obradu dogadjaja
document.removeEventListener("mousemove", pomerajMisa);
```

Пример 56. Метода `removeEventListener()`

14 Закључак

Програмски језик JavaScript је незамењив програмски језик за све који се баве креирањем веб страна и интернет апликација. Пружа много могућности, објетно је оријентисан и има доста сличности са програмским језицима C и Java.
Када се ради о овом курсу, првенствено је намењен програмерима који имају одређено предзнање о HTML-у. За програмере почетнике које интересује веб програмирање, JavaScript је одличан избор за учење након савладавања HTML-а и CSS-а. Заједно интегрисани у веб стране, HTML и JavaScript представљају веома моћне алате за прављење привлачних и интерактивних веб страна.
Програмски језик JavaScript представља одличну основу и за учење неких других програмских језика и техника, као што су JQuery и AJAX, који знатно убрзавају и олакшавају комуникацију између клијента и сервера.

Литература

[1] Thomas Powell, Fritz Schneider, *JavaScript - The complete reference, Second Edition*, 2004, ISBN:0072253576

[2] Russ Ferguson, Christian Heilmann, *Beginning JavaScript with DOM Scripting and Ajax, Second Edition*

[3] David Flanagan, *JavaScript: The Definitive Guide, 6th Edition*, 2011, ISBN:978-0-596-80552-4

[4] http://www.w3schools.com

[5] https://developer.mozilla.org/en-US/docs/Web/JavaScript