

# Osnove JavaScripta

C501



Ovaj priručnik izradio je autorski tim Srca:

Autor: Denis Stančer

Recenzent: Edin Mujadžević

Urednica: Sabina Rako

Lektorica: Jasna Novak Milić

TEČAJEVI **srca**

Sveučilište u Zagrebu

Sveučilišni računski centar

Josipa Marohnića 5, 10000 Zagreb

edu@srce.hr

ISBN 978-953-7138-56-1 (meki uvez)

ISBN 978-953-7138-57-8 (PDF)

Inačica priručnika: C501-20150609



Ovo djelo dano je na korištenje pod licencom *Creative Commons Imenovanje-Nekomercijalno-Dijeli pod istim uvjetima 4.0 međunarodna*.  
Licenca je dostupna na stranici  
<http://creativecommons.org/licenses/by-nc-sa/4.0/>.

# Sadržaj

<b>Uvod</b>	<b>1</b>
<b>1. Što je JavaScript?</b>	<b>3</b>
1.1. Naziv <i>JavaScript</i>	3
1.2. Povijest <i>JavaScripta</i>	3
1.3. Skriptni jezici	4
1.4. Preglednici ( <i>browsers</i> )	6
1.5. Aplikacije za uređivanje teksta ( <i>editors</i> )	6
1.6. <i>Document Object Model</i> u <i>JavaScriptu</i>	7
1.7. Sigurnost	10
<b>2. Upoznavanje s jezikom</b>	<b>11</b>
2.1. Način pisanja	11
2.2. Uključivanje <i>JavaScripta</i> u HTML-dokument	13
2.3. Pogreške	18
<b>3. Varijable i objekti</b>	<b>19</b>
3.1. Vrste podataka	19
3.2. Varijable	22
3.3. Objekti	23
<b>4. Operatori</b>	<b>27</b>
4.1. Aritmetički operatori	27
4.2. Operator pridruživanja	27
4.3. Operatori uspoređivanja	28
4.4. Logički operatori	30
4.5. Operator spajanja	30
4.6. Vježba 2	31
<b>5. Funkcije</b>	<b>33</b>
5.1. Definiranje funkcije	33
5.2. Poziv funkcije	34
5.3. Doseg varijabli	34
5.4. Vježba 3	36
<b>6. Naredbe za kontrolu tijeka</b>	<b>37</b>
6.1. Uvjetno izvođenje naredbi	37
6.2. Višestruka usporedba	38
6.3. Uvjetni operator	39
6.4. Petlja s uvjetom na početku	39
6.5. Petlja s uvjetom na kraju	40
6.6. Petlja s poznatim brojem ponavljanja	40
6.7. Vježba 4	41

<b>7. Obrasci .....</b>	<b>45</b>
7.1. Prvi obrazac .....	45
7.2. Unos kraćih nizova znakova .....	47
7.3. Odabir jedne od mogućnosti .....	52
7.4. Uključivanje i isključivanje jedne mogućnosti .....	53
7.5. Odabir jedne od mogućnosti na drugi način .....	54
<b>8. JavaScript biblioteka – jQuery .....</b>	<b>63</b>
8.1. Općenito o JavaScript bibliotekama .....	63
8.2. jQuery .....	63
8.3. Prerada obrazaca uz pomoć biblioteke jQuery .....	65
<b>9. Korisne skripte.....</b>	<b>69</b>
9.1. Rollover.....	69
9.2. Preusmjerenje.....	70
9.3. Provjera pomoću regularnih izraza .....	70
9.4. Upravljanje preglednikom .....	75
<b>10. Zadaci .....</b>	<b>79</b>
<b>11. Dodatak.....</b>	<b>80</b>
A. Operatori .....	80
B. Polja .....	81
C. Datumi.....	82
D. Matematičke funkcije .....	83
E. Događaji .....	84
F. Niz znakova .....	85
G. Aplikacija za uređivanje teksta – Brackets .....	86

# Uvod

U okviru ovog tečaja upoznat ćete se s osnovama programiranja u *JavaScriptu*. *JavaScript* je skriptni jezik (danas *de facto* standard) kojim se u statičke HTML-stranice mogu uvesti interaktivni elementi.

Za uspješno praćenje tečaja o *JavaScriptu* potrebno je znati osnovno o radu u operacijskom sustavu, poznavati sintaksu HTML-a, a poželjna su i osnovna znanja iz programiranja.

Ovaj se priručnik sastoji od jedanaest poglavlja koja će biti detaljno obrađena u 12 školskih sati.

U ovom se priručniku za označavanje važnijih pojmova rabe podebljana slova. Nazivi datoteka oblikovani su podebljanim slovima i kurzivom.

Prečaci na tipkovnici označeni su ovako: [Ctrl]+[Alt]+[Del], [F1], a programski se kôd prikazuje posebnim oblikovanjem:

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

Savjeti, upozorenja i zanimljivosti nalaze se u okvirima sa strane.



# 1. Što je JavaScript?

*JavaScript* je jednostavan, interpretiran programski jezik namijenjen ponajprije razvoju interaktivnih HTML-stranica. Jezgra *JavaScripta* uključena je u većinu današnjih preglednika (*Internet Explorer*, *Google Chrome*, *Mozilla Firefox*, *Opera*, *Safari* i drugi).

## 1.1. Naziv JavaScript

*JavaScript* omogućuje izvršavanje određenih radnji u inače statičnim HTML-dokumentima, npr. interakciju s korisnikom, promjenu svojstava preglednikova prozora ili dinamičko stvaranje HTML-sadržaja.

*JavaScript* nije pojednostavljena inačica programskog jezika *Java*. Povezuje ih jedino slična sintaksa i to što se koriste za izvršavanje određenih radnji unutar preglednika. Izvorno se *JavaScript* trebao zvati *LiveScript*, ali da bi se potakla uporaba novog skriptnog jezika, nazvan je slično jeziku *Java*, od kojeg se u tadašnje vrijeme dosta očekivalo.

## 1.2. Povijest JavaScripta

*JavaScript* se razvija od 1995. godine kada je *Netscape* objavio nekoliko prvih inačica jezika. Nedugo nakon toga *Microsoft* je objavio jezik sličan *JavaScriptu* pod nazivom *JScript*. Danas je za standardizaciju skriptnih jezika, pa tako i *JavaScripta*, zadužena organizacija *ECMA* (<http://www.ecma.ch/>). Standardi se objavljuju pod nazivom *ECMAScript* i do sada je objavljeno pet inačica standarda *ECMA-262*.

U ovom priručniku pojam *JavaScript* označava bilo koju implementaciju jezika, uključujući i *Microsoftov JScript*.

Inačica	Opis
<i>JavaScript 1.0</i>	Izvorna inačica jezika, bila je puna pogrešaka. Implementirana u preglednik <i>Netscape 2</i> .
<i>JavaScript 1.1</i>	Dodan novi objekt <code>Array</code> (za rad s poljima); popravljene ozbiljne pogreške. Implementiran u preglednik <i>Netscape 3</i> .
<i>JavaScript 1.2</i>	Dodana nova naredba <code>switch</code> , regularni izrazi i drugo. Djelomično poštuje <i>ECMA v1</i> uz neke nekompatibilnosti. Implementiran u preglednik <i>Netscape 4</i> .
<i>JavaScript 1.3</i>	Ispravljene nekompatibilnosti <i>JavaScripta 1.2</i> . Usklađenost sa standardom <i>ECMA v1</i> . Implementiran u preglednik <i>Netscape 4.5</i> .
<i>JavaScript 1.4</i>	Implementiran samo u preglednik <i>Netscape</i> za poslužiteljske proizvode.
<i>JavaScript 1.5</i>	Uvedeno upravljanje iznimkama ( <i>exception handling</i> ). Poštuje standard <i>ECMA v3</i> . Implementiran u preglednike <i>Mozilla Firefox</i> i <i>Netscape 6</i> .
<i>JScript 1.0</i>	Okviro ekvivalentan <i>JavaScriptu 1.0</i> . Implementiran u prve inačice preglednika <i>Internet Explorer 3</i> .
<i>JScript 2.0</i>	Okviro ekvivalentan <i>JavaScriptu 1.1</i> . Implementiran u kasnije inačice preglednika <i>Internet Explorer 3</i> .

Inačica	Opis
<i>JavaScript 3.0</i>	Okvirno ekvivalentan <i>JavaScriptu 1.3</i> . Usklađen sa standardom <i>ECMA v1</i> . Implementiran u preglednik <i>Internet Explorer 4</i> .
<i>JavaScript 4.0</i>	Nije implementiran ni u jedan preglednik.
<i>JavaScript 5.0</i>	Podržano upravljanje iznimkama ( <i>exception handling</i> ). Djelomično poštuje standard <i>ECMA v3</i> . Implementiran u preglednik <i>Internet Explorer 5</i> .
<i>JavaScript 5.5</i>	Okvirno ekvivalentan <i>JavaScriptu 1.5</i> . Potpuno usklađen sa standardom <i>ECMA v3</i> . Implementiran u preglednike <i>Internet Explorer 5.5</i> i <i>6</i> . ( <i>IE 6</i> zapravo ima <i>JavaScript 5.6</i> , ali <i>5.6</i> se značajno ne razlikuje od <i>5.5</i> za <i>JavaScript</i> programere koji pišu za preglednike).
<i>ECMA v1</i>	Prva standardna inačica. Standardizirane su osnove <i>JavaScripta 1.1</i> i dodano je nekoliko novih mogućnosti. Nisu standardizirani naredba <code>switch</code> i regularni izrazi. Implementacije koje poštuju standard <i>ECMA v1</i> su <i>JavaScript 1.3</i> i <i>JavaScript 3.0</i> .
<i>ECMA v2</i>	Razvojna inačica koja nije donijela nove mogućnosti, ali je razjasnila dvosmislenosti.
<i>ECMA v3</i>	Standardizirani naredba <code>switch</code> , regularni izrazi i upravljanje iznimkama. Implementacije usklađene sa standardom <i>ECMA v3</i> su <i>JavaScript 1.5</i> i <i>JavaScript 5.5</i> .
<i>ECMA v4</i>	Ova inačica nikada nije zaživjela zbog nesuglasica u radnoj skupini u kojem bi se smjeru jezik trebao razvijati.
<i>ECMA v5</i>	Dodan način rada <i>strict</i> , razjašnjene su mnoge dvosmislenosti iz standarda <i>ECMA v3</i> i dodana je podrška za <i>JSON</i> .
<i>ECMA v5.1</i>	Dorađena inačica 5 koja ne sadržava nikakve novitete vezane uz sâm jezik.
<i>ECMA v6</i>	Očekuje se sredinom 2015. godine, a treba dodati novu sintaksu za objektno programiranje i još neke dodatke. Ta je inačica poznata i kao <i>ES6 Harmony</i> .

### 1.3. Skriptni jezici

Program koji obrađuje i izvršava skripte zove se *interpreter*. *Interpreter* čita kôd i prevodi ga u strojni jezik svakog puta kada se pokrene skripta. Svaki jezik koji se *interpretira*, tj. koji izvršava *interpreter*, naziva se **skriptni jezik**. *Interpreter* za *JavaScript* ugrađen je u većinu današnjih preglednika.

Skriptni se jezici koriste jer je razvoj programa znatno jednostavniji. Za razliku od programa pisanih u pravim programskim jezicima, kod skriptnog jezika ne treba prevoditi skripte u strojni jezik:

#### Koraci kod programskih jezika

1. Napisati ili popraviti program.
2. Prevesti program u strojni jezik.
3. Pokrenuti prevedeni program.
4. Za popravke ponoviti od 1. koraka.

#### Koraci kod skriptnih jezika

1. Napisati ili popraviti skriptu.
2. Pokrenuti interpreter.
3. Za popravke ponoviti od 1. koraka.



HTML je jezik koji se koristi za opis dokumenata i nema dinamičnih elemenata. Davno se ukazala potreba za uvođenjem dinamičnog načina stvaranja HTML-elemenata i stvaranje interaktivnog sadržaja u HTML-u.

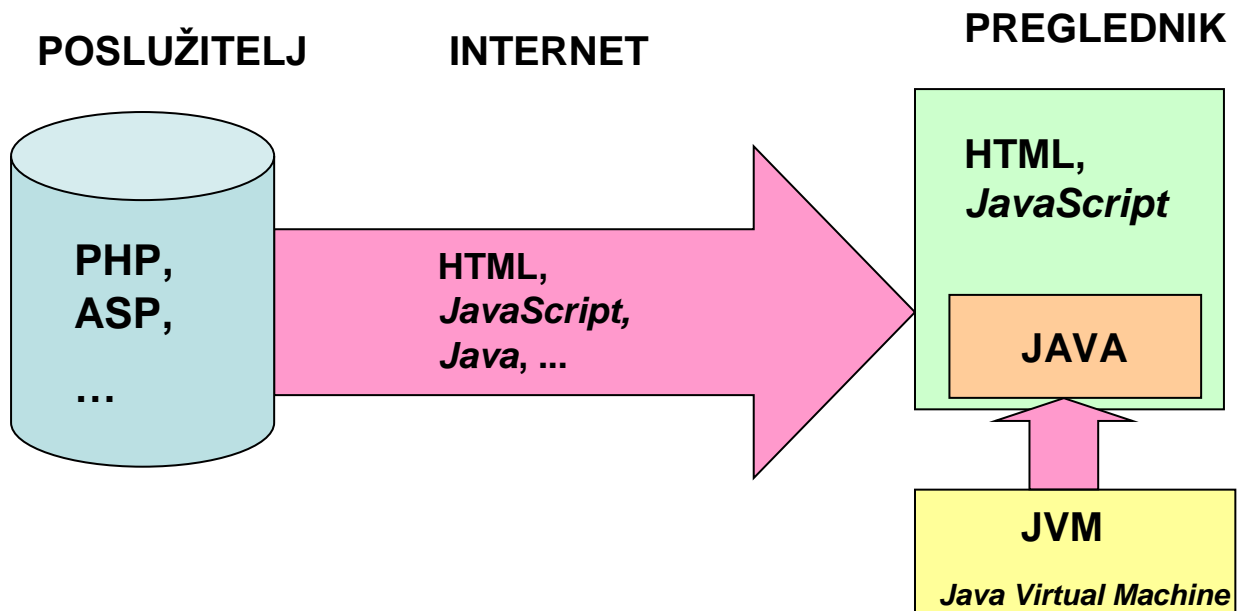
Danas postoji nekoliko tehnologija za stvaranje interaktivnog sadržaja u HTML-dokumentima:

1. U prvoj skupini su tehnologije za dinamično stvaranje HTML-a, tj. stranice su zapisane pomoću nekog (obično skriptnog) jezika, koji interpretira poslužitelj i šalje korisniku HTML-kôd. Tipični su predstavnici ove skupine ASP.NET, PHP (PHP: *Hypertext Preprocessor*), a mogu se koristiti i *Java*, *Ruby* (*Ruby on Rails*), *Python* i *Perl*.
2. U drugoj skupini su *Java* (u obliku *appleta*), *Flash* i *Shockwave* za čiji je prikaz potreban vanjski program (*plug-in*) koji zna interpretirati ili izvoditi navedene programe. Kad je sadržaj prikazan u pregledniku, on je dinamičan i neovisan o „okolnom” HTML-u.
3. U trećoj su skupini tzv. klijentski jezici, jer se njihov kôd interpretira na klijentskoj strani, tj. u pregledniku (klijentu). Glavni predstavnik klijentskih jezika je *JavaScript*. Nekada je bio značajan i skriptni jezik *VBScript* koji se koristio u starijim inačicama preglednika *Internet Explorera*.

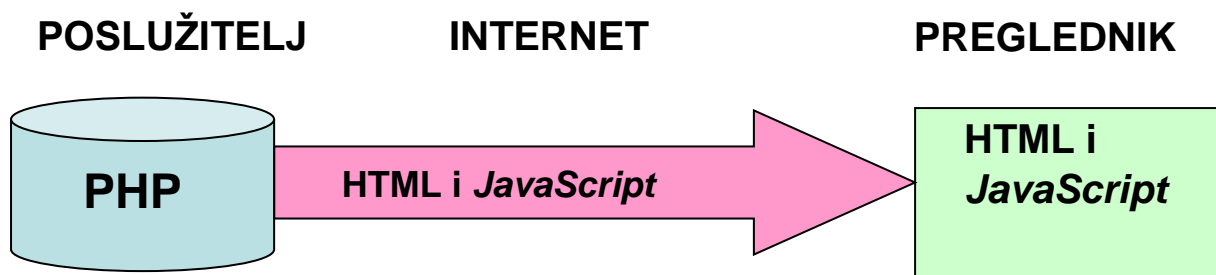
#### Napomena

Java *applet*i su manji programi napisani u *Javi* koji se izvršavaju u pregledniku.

Primjer toka podataka od *web-poslužitelja* do *web-preglednika*:



Za PHP to izgleda ovako:



## 1.4. Preglednici (*browsers*)

Danas je podrška za *JavaScript* izvrsna u svim preglednicima, tako da se autori preglednika više ne natječu u podršci nego u brzini izvođenja određenih algoritama.

Popis preglednika koji podržavaju standard *ECMA-262* inačica 5:

- *Internet Explorer 10+*
- *Firefox 21+*
- *Safari 6+*
- *Chrome 23+*
- *Opera 15+.*

Osim utrke u brzini autori preglednika natječu se u jednostavnosti razvoja, odnosno tko će omogućiti što više dodatnih značajki za razvoj. Što je bolja podrška za razvoj, to će se više razvijatelja koristiti tim preglednikom i preporučivati ga svojim klijentima. Tako raste i broj korisnika tog preglednika.

Tri najčešće korištena preglednika (*Internet Explorer*, *Google Chrome* i *Mozilla Firefox*) danas imaju alate za razvijatelje (*developer tools*) do kojih se u svim preglednicima dolazi jednako – odabirom tipke F12.

Od razvijateljskih značajki za ovaj su tečaj najzanimljivije *Console* i *DOM*.

## 1.5. Aplikacije za uređivanje teksta (*editors*)

*JavaScript* se može pisati u bilo kojem uređivaču teksta (*editor*) koji podržava standard *ASCII*. Blok za pisanje (*Notepad*) prisutan je u svim inačicama operacijskog sustava *Windows* pa se nameće kao pogodan i za programiranje u *JavaScriptu*.

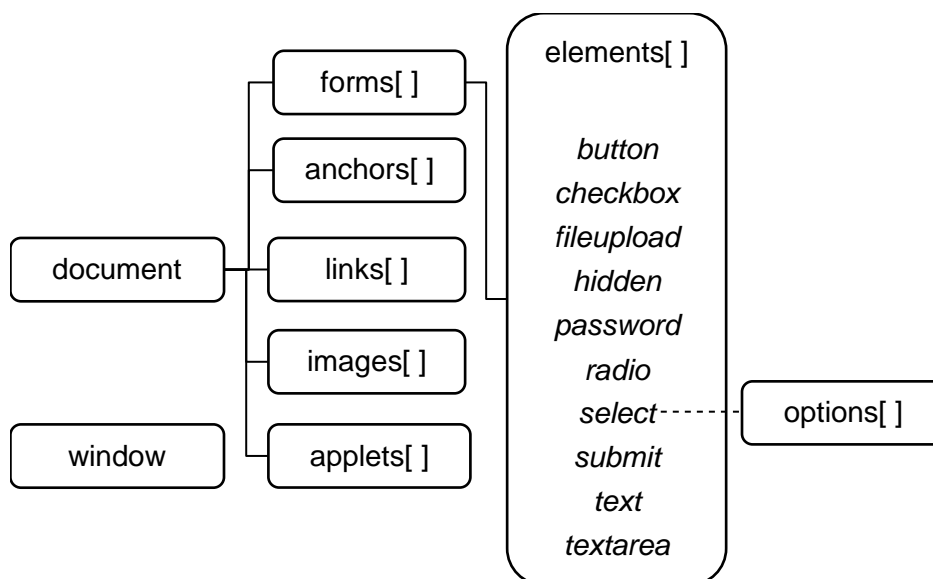
Da bi se programiranje ubrzalo i učinilo učinkovitijim, poželjno je rabiti uređivač teksta koji ima ova svojstva:

- mogućnost otvaranja više dokumenata u jednom prozoru
- razlikuje dijelove kôda i prikazuje ih u različitim bojama (bojanje sintakse)
- podrška za UTF-8
- automatsko poravnavanje (*tidy*)
- provjera stila kodiranja u skladu s preporučenim postupcima (*linter*).

## 1.6. *Document Object Model* u *JavaScriptu*

*Document Object Model* (DOM) je model za prikaz i interakciju s objektima u HTML-dokumentu. Omogućava jednoznačan i jednostavan pristup dijelovima (HTML-) dokumenta te rukovanje njegovim dijelovima (npr. elementi u HTML-dokumentu).

*JavaScript* definira svoj DOM u obliku ovakve hijerarhijske strukture:



Svakom objektu ili svojstvu pristupa se kroz taj model, tj. *document* je osnovni objekt preko kojeg se pristupa svim drugim objektima dokumenta.

Na primjer, u dokumentu koji sadrži ovaj programski kôd:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <script language="JavaScript" src="forma3.js"></script>
  </head>
  <body>
    <form action="">
      Unesite ime: <input type="text" value="" />
      <br />
      <input type="submit" value="Pošalji" onclick="return provjeri();" />
    </form>
  </body>
</html>
```

vrijednosti polja u koje se upisuje ime pristupa se ovako:

```
document.forms[0].elements[0].value
```

Vidljivo je da je tekstni objekt prvi u polju elemenata koji se nalaze u prvom obrascu. Da bi se programiranje pojednostavilo, omogućeno je imenovanje pojedinih objekata. Na primjer, dodavanjem naziva obrascu (`name="frm_a"`) i dodavanjem naziva tekstnom polju za unos imena (`name="ime"`) dobije se:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <script language="JavaScript" src="forma3.js"></script>
  </head>
  <body>
    <form name="frm_a" action="">
      Unesite ime:
      <input type="text" name="ime" value="" /><br />
      <input type="submit" value="Pošalji"
        onclick="return provjeri();" />
    </form>
  </body>
</html>
```

Vrijednosti polja u tom slučaju pristupamo jednostavnije:

```
document.frm_a.ime.value
```

Najjednostavniji način dohvata vrijednosti polja je pomoću jedinstvenog identifikatora (ID). Naime, preporuka je da vrijednost atributa ID mora biti jedinstvena na razini dokumenta. Prvobitni se dokument može dodatno urediti dodavanjem atributa ID (`id="ime"`) za polje za unos imena:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <script language="JavaScript" src="forma3.js"></script>
  </head>
  <body>
    <form action="">
      Unesite ime:
      <input type="text" name="ime" id="ime" value="" />
      <br />
      <input type="submit" value="Pošalji"
        onclick="return provjeri();" />
    </form>
  </body>
</html>
```

Vrijednosti polja u tom slučaju pristupamo još jednostavnije:

```
document.getElementById('ime').value
```

Potrebno je naglasiti da atribut `name` i dalje treba biti naveden zbog načina na koji se obrađuju obrasci.

Objekt `window` također je važan objekt pomoću kojeg se upravlja prozorom preglednika. U jednom od primjera detaljnije će se obraditi uporaba objekta `window`.

Dvije često korištene funkcije vezane su uz objekte `document` i `window`:

- `document.write` – funkcija koja upisuje niz znakova koji se proslijedi kao prvi argument u HTML-dokument na mjestu gdje se funkcija poziva
- `window.alert` – funkcija ispisuje niz znakova koji se proslijedi kao prvi argument u zasebnom prozoru i ne dopušta nastavak izvršavanja programa dok se ne zatvori.

## 1.7. Sigurnost

Kad god se programi (kao što su skripte *JavaScripta*, *Java*-programi ili makronaredbe programa *Microsoft Word*) nalaze u odijeljenim dokumentima, osobito u dokumentima koji se šalju Internetom, prisutna je opasnost od virusa i drugih zloćudnih programa. Tvorci *JavaScripta* bili su svjesni tih sigurnosnih problema i onemogućili su programima *JavaScripta* postupke koji bi za posljedicu imali brisanje ili izmjenu podataka na korisnikovu računalu. Kao što je već naglašeno, programi *JavaScripta* ne mogu pristupati lokalnim datotekama, tj. ne mogu zaraziti druge datoteke ili brisati postojeće.

Isto tako, programi *JavaScripta* ne mogu obavljati mrežne radnje, tj. *JavaScript* može učitavati adrese *web*-sadržaja (URL) i slati podatke iz HTML-obrazaca poslužiteljskim skriptama, ali ne može ostvarivati neposredne veze s drugim računalima i tako pokušati pogoditi lozinku na nekom lokalnom poslužitelju.

Međutim, budući da su preglednici složeni programi, u početku implementacija interpretera *JavaScripta* nije uvijek poštivala propisane standarde. Na primjer, preglednik *Netscape 2* (objavljen 1995. godine) omogućavao je pisanje programa *JavaScript* koji je automatski dohvatio adresu elektroničke pošte bilo kojeg posjetitelja određene stranice i u njegovo ime, bez njegove potvrde, poslao e-mail. Taj je propust, uz niz drugih, manje opasnih, popravljen. Međutim, ne postoji jamstvo da se neće otkriti novi propusti u implementaciji *JavaScripta* i tako omogućiti zlonamjernicima iskorištavanje tih propusta.

## 2. Upoznavanje s jezikom

### 2.1. Način pisanja

*JavaScript* se može pisati prema standardu *Unicode*, no preporuča se pisanje prema standardu *ASCII*, osim u komentarima i nizovima znakova.

#### 2.1.1. Velika i mala slova

Prilikom pisanja sintakse *JavaScripta* važno je zapamtiti:

#### ***JavaScript* razlikuje velika i mala slova!**

To znači da se ključne riječi, varijable, funkcije i drugi nazivi moraju pisati dosljedno s obzirom na velika i mala slova. Tako se, na primjer, ključna riječ mora pisati `while`, a ne `While` ili `WHILE`. Također, varijable `stranaa`, `stranaA`, `StranaA` i `STRANAA` četiri su različite varijable.

Radi izbjegavanja pogrešaka, preporuča se dosljednost prilikom korištenja malih i velikih slova u nazivima. Dobra praksa je slijediti način imenovanja koji se već koristi za nazive postojećih funkcija i svojstava u *JavaScriptu*, a to je tzv. *camelCasing*. Nazivi se pišu malim slovima, a ako naziv sadrži dvije ili više riječi, druga i svaka sljedeća riječ počinju velikim slovom (npr. `getElementById`).

#### **Napomena**

Velika i mala slova su najčešći uzrok početničkih pogrešaka u *JavaScriptu*.

#### **Napomena**

Iznimka od ovog načina pisanja su nazivi svojstava objekata DOM-a koji se svi pišu malim slovima, npr. `onclick`.

#### 2.1.2. Znak za završetak naredbe

Znakom kojim se u *JavaScriptu* označava kraj naredbe je točka-zarez (;). Točka-zarez nije obavezan znak, ali se njegova uporaba preporuča. Naime, *JavaScript* ubacuje točku-zarez na kraju retka ako pojedini kôd izgleda kao naredba, što u nekim situacijama nije to što je programer htio. Na primjer kôd:

```
return
true;
```

*JavaScript* će interpretirati kao:

```
return;
true;
```

(dakle ubačena je točka-zarez na kraju retka), iako je programer htio:

```
return true;
```

### 2.1.2. Komentari

U *JavaScriptu* se komentari označavaju jednako kao i u programskim jezicima C i C++. Bilo koji tekst između `//` i kraja retka smatra se komentarom. Bilo koji tekst između znakova `/*` i `*/` je komentar i zanemaruje se. Druga vrsta komentara može se protezati u više redaka, ali se ne smiju ugnježdivati. Evo nekoliko ispravnih komentara:

```
// Običan jednoređajni komentar.  
  
/*  
  
I ovo je komentar.  
  
Koji se proteže na više redaka.  
  
*/
```

### 2.1.3. Varijable

Nazivi varijabli i funkcija trebaju zadovoljavati ovo pravilo: prvi znak u nazivu treba biti slovo, podcrta ( `_` ) ili dolarski znak ( `$` ). Znakovi koji slijede mogu biti slova, brojevi, podcrta ili dolarski znak. Primjer su ispravnih naziva:

i	j
moja_varijabla	sImeIPrezime
v13	l1
_privremena	00

Nazivi ne smiju biti isti kao ključne riječi. U tablici je naveden popis ključnih riječi u *JavaScriptu* koje treba izbjegavati kod imenovanja varijabli:

break	do	if	switch	typeof
case	else	in	this	var
catch	false	instanceof	throw	void
continue	finally	new	true	while
default	for	null	try	with
delete	function	return		



Ova se imena trenutačno ne rabe u *JavaScriptu*, ali ih standard *ECMAScript v3* navodi kao moguće ključne riječi u budućnosti, pa bi i njih trebalo izbjegavati:

Abstract	double	goto	native	static
boolean	enum	implements	package	super
byte	export	import	private	synchronized
char	extends	int	protected	throws
class	final	interface	public	transient
const	float	long	short	volatile

Također treba izbjegavati nazive globalnih varijabli i funkcija predefiniranih u *JavaScriptu*:

arguments	encodeURIComponent	Infinity	Object	String
Array	Error	isFinite	parseFloat	SyntaxError
Boolean	escape	isNaN	parseInt	TypeError
Date	eval	Math	RangeError	undefined
DecodeURI	EvalError	NaN	ReferenceError	unescape
decodeURIComponent	Function	Number	RegExp	URIError

## 2.2. Uključivanje *JavaScripta* u HTML-dokument

*JavaScript* se ovako može uključiti u HTML-dokument:

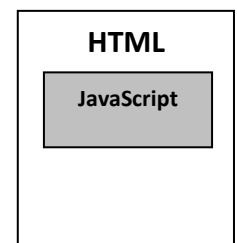
- pisanjem kôda između oznaka `<script>` i `</script>`
- iz vanjske datoteke uporabom atributa `src` u oznaci `<script>`
- preko obrade događaja navedenog kao HTML-atribut (npr. `onclick` ili `onmouseover`)
- u URL-u koji se koristi posebnim protokolom `javascript:.`

### 2.2.1. Unutar HTML-datoteke

Unutar HTML-dokumenta *JavaScript*-skripte pišu se između oznaka `<script>` i `</script>`. Proizvoljan broj *JavaScript*-naredbi obrađuje se prilikom učitavanja HTML-dokumenta i to redom kojim su napisane. Element `<script>` može se pojaviti u elementu `<head>` ili u elementu `<body>`.

U elementu `<head>` uobičajeno je pisati programe koji očekuju neku korisnikovu radnju, tj. učitaju se u preglednik i čekaju događaj koji bi ih pokrenuo.

U elementu `<body>` uobičajeno je pisati programe koji se izvršavaju odmah pri učitavanju. Najčešće je to generiranje nekog dijela HTML-dokumenta.



Jedan HTML-dokument može sadržavati nekoliko elemenata `<script>`. Te odvojene skripte obrađuju se redom kojim su napisane u dokumentu. Odvojene skripte čine jedan *JavaScript*-program u jednom HTML-dokumentu, tj. funkcije i varijable definirane u jednoj skripti dostupne su u svim sljedećim skriptama u istom HTML-dokumentu.

Iako je *JavaScript* najčešće rabljen skriptni jezik u HTML-dokumentima, on nije jedini. Vrsta skriptnog jezika navodi se u atributu `type` elementa `<script>`. Preglednici koji imaju *interpreter* za navedeni jezik obrade skriptu, a ostali ju zanemare.

Primjena atributa `type`:

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

    // JavaScript program

</script>
```

Danas više nisu u uporabi preglednici koji ne prepoznaju element `<script>` pa se sve rjeđe koristi „skrivanje” kôda *JavaScript*:

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

Ovako gornji kôd interpretira preglednik koji ne prepoznaje element `<script>`:

- Nepoznati element (`<script>`).
- W3C preporuka daje uputu da se sadržaj nepoznatog elementa prikaže u pregledniku.
- Preglednik prikazuje sadržaj elementa `<script>`, a to je komentar, odnosno ne prikazuje ništa.

S druge strane, preglednik koji prepoznaje element `<script>` kôd interpretira ovako:

- Preglednik zna kako obraditi element `<script>`.
- Je li prvi redak iz elementa `<script>` početak HTML-komentara (odnosno 4 znaka `<!--`).
- Ako da, tada se ignorira taj prvi redak i ostatak se sadržaja elementa `<script>` interpretira kao *JavaScript*.
- Zato posljednji redak u elementu `<script>` ima *JavaScript*-komentar (`//`) ispred završetka HTML-komentara (`-->`).

### 2.2.2. JavaScript u zasebnoj datoteci

Od inačice 1.1 *JavaScript* u elementu `<script>` podržava atribut `src`. Vrijednost tog atributa je URL datoteke koja sadrži *JavaScript*-program.

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

Datoteka *JavaScript* obično ima nastavak `.js` i sadrži čisti *JavaScript*, bez elementa `<script>`.

Bilo koji kôd unutar oznaka zanemaruje se. Nužno je primijetiti da je završna oznaka `</script>` obavezna iako je atribut `src` naveden i iako nema *JavaScript*-kôda između oznaka `<script>` i `</script>`.

Nekoliko je razloga za uporabu atributa `src`:

- Pojednostavljuje HTML-datoteku i smanjuje njezinu veličinu, jer nema velikih blokova *JavaScripta* u dokumentu. Smanjenjem veličine datoteka će se brže preuzeti s poslužitelja.
- Ako se neka funkcija *JavaScripta* rabi u više HTML-dokumenata, poziva se jedna datoteka, što omogućuje jednostavnije održavanje.
- Kada više HTML-dokumenata rabi jednu datoteku *JavaScripta*, ona se učitava u preglednik samo prilikom prve uporabe. Nakon toga se datoteka lokalno sprema na disk i svakog sljedećeg puta interpretira se iz lokalne datoteke.
- Budući da atribut `src` sadrži URL, program *JavaScript* s jednog poslužitelja može se rabiti na nekoliko drugih (npr. `src="http://www.server.hr/js/obrasci.js"`).

### 2.2.3. Obrada događaja

Program *JavaScript* izvršava se samo jednom i to kada se HTML-dokument učitava u preglednik. Takav način uporabe *JavaScripta* ne omogućuje interakciju s korisnikom. Zbog toga je za većinu HTML-elemenata definirano više događaja.

HTML-elementi imaju posebne attribute pomoću kojih se može povezati kôd *JavaScripta* s određenim događajem. Ti atributi počinju slovima `on...`, a kao vrijednost atributa navode se naredbe *JavaScripta* koje će se izvršiti kad se dogodi taj događaj.

```
<input type="button"
      value="Poruka"
      onclick="window.alert('JavaScript');" />
```

U gornjem primjeru pritiskom korisnika na dugme „Poruka“ dogodi se događaj `click`, izvršava se naredba navedena u atributu `onclick`. U primjeru se poziva funkcija `window.alert` koja korisniku prikazuje obavijest.

Kao vrijednost atributa može se navesti nekoliko naredbi *JavaScripta*, no ako je naredbi previše, uobičajeno je da se napiše funkcija (u elementu `<script>` ili u zasebnoj datoteci) te se zatim poziva ta funkcija. Tako se smanjuje miješanje programskog kôda *JavaScripta* i sadržaj HTML-dokumenta.

#### 2.2.4. *JavaScript* u URL-u

Kôd *JavaScripta* može se uključiti u HTML i pomoću pseudoprotokolne oznake `javascript` u URL-u. Taj posebni protokol označava da je tijelo stranice iza navedenog URL-a proizvoljan *JavaScript*-program koji će obraditi interpreter. Ako *JavaScript*-program u `javascript:` URL-u ima više naredbi, one moraju biti odvojene točka-zarezom. Na primjer:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
  </head>
  <body>
    <a href="javascript:var sada=new Date(); 'Sada je:' + sada;">JavaScript</a>
  </body>
</html>
```

Kad preglednik učitava takav *JavaScript* URL, izvede se *JavaScript*-program, izračuna se vrijednost posljednje naredbe i ta se vrijednost pretvori u niz znakova. Konačno, dobiveni niz znakova prikaže se u pregledniku.

Kad *JavaScript* URL sadrži program koji ne vraća nikakvu vrijednost, preglednik izvrši program, ali ne mijenja sadržaj dokumenta.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
  </head>
  <body>
    <a href="javascript:window.alert('Obavijest korisniku!');">JavaScript</a>
  </body>
</html>
```

Vrlo često se `javascript:` URL koristi za izvršavanje programa *JavaScripta* bez promjene sadržaja dokumenta. To se postiže tako da se osigura da posljednja naredba u URL-u ne vraća nikakvu vrijednost. Jedan od načina je uporaba operatora `void`.

Sljedeći primjer pokazuje `javascript:` URL koji otvara novi prazan prozor, bez promjene sadržaja trenutnog dokumenta:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
  </head>
  <body>
    <a href="javascript:window.open('about:blank'); void 0;">
      JavaScript
    </a>
  </body>
</html>
```

Bez operatora `void` u prozoru preglednika bila bi prikazana vrijednost posljednje naredbe pretvorena u niz znakova. U našem slučaju to je `window.open()`. Niz znakova bio bi: `[object]` (ili `[object Window]` u pregledniku *Mozilla Firefox*, a u pregledniku *Google Chrome* ta se vrijednost ne ispisuje).

Takav `javascript:` URL može se rabiti na svim mjestima gdje se koristi uobičajen URL. Na primjer, za brzo testiranje može se u pregledniku u polje *Adresa (Location)* upisati bilo koji *JavaScript*-program (naravno kraći).

Česta uporaba takvog oblika je kao vrijednost atributa `action` u elementu `<form>`.

## 2.3. Pogreške

Prilikom pogreške u izvođenju programa *JavaScripta*, pojedini preglednici to prikazuju različito. Ako se program *JavaScripta* ne ponaša kako se očekuje, najbolje je pokrenuti F12, konzolu *developer tools*, jer se tada u njoj mogu pratiti sve pogreške koje se javljaju tijekom izvođenja programa.

U ovoj skripti pogreška je u 11. retku (*Line 13*) i 5. stupcu (*Col 5*).

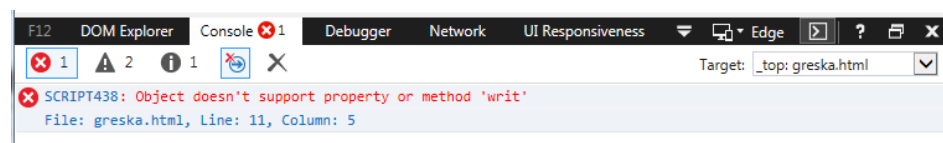
```

1  <!DOCTYPE html>
2  <html lang="hr">
3
4  <head>
5      <meta charset="UTF-8" />
6      <title>Greška</title>
7  </head>
8  <body>
9      <script type="text/javascript">
10         <!--
11         var i = 1;
12
13         document.writ('i: ' + i + '<br>');
14
15         -->
16     </script>
17 </body>
18 </html>

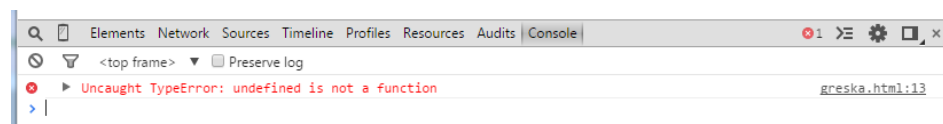
```

Pogreška je opisana u polju *Pogreška (Error)*. Naime, opis pogreške navodi da objekt ne podržava metodu (tj. objekt `document` ne podržava metodu `writ`). Drugim riječima, metoda `write` pogrešno je napisana.

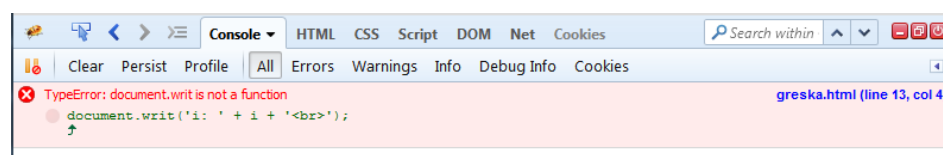
Izgled pogreške u pregledniku *Internet Explorer*:



Izgled pogreške u pregledniku *Google Chrome*:



Izgled pogreške u pregledniku *Mozilla Firefox*:



## 3. Varijable i objekti

### 3.1. Vrste podataka

#### 3.1.1. Brojevi

Svi brojevi u *JavaScriptu* prikazuju se kao brojevi s pomičnim zarezom (*floating-point*) u 64-bitnom formatu, tj. mogu se prikazati brojevi koji nisu veći (po apsolutnoj vrijednosti) od  $1,7976931348623157 \times 10^{308}$  i nisu manji od  $5 \times 10^{-324}$ .

Cijeli brojevi mogu se prikazati od  $-9007199254740992$  ( $-2^{53}$ ) do  $9007199254740992$  ( $2^{53}$ ).

Osim brojeva s bazom 10, *JavaScript* prepoznaje i brojeve s bazom 16 (heksadecimalne). Heksadecimalan broj mora započeti s „0x” ili „0X” i nastaviti se nizom heksadecimalnih znamenki (tj. 0-9 i A-F).

```
0xFF // 15*16 + 15 = 255 (u bazi 10)
```

```
0xCAFE911
```

Brojevi s pomičnim zarezom prikazuju se na dva načina: klasični način s točkom ili tzv. „inženjerski” način sa slovom E.

```
3.14
```

```
2345.789
```

```
.33333333333333333333
```

```
6.02e23 // 6.02 x 1023
```

```
1.4738223E-32 // 1.4738223 x 10-32
```

Kad rezultat operacije na brojevima postane prevelik za prikaz u *JavaScriptu*, broj se prikazuje kao posebna vrijednost `Infinity`. Slično tome, kad je negativan broj manji od najmanjeg broja koji se može prikazati u *JavaScriptu*, prikazuje se kao posebna vrijednost `-Infinity`.

Ako se prilikom neke matematičke operacije dobije nedefinirani rezultat (npr. dijeljenje s nulom), broj se prikazuje pomoću posebne vrijednosti `NaN` (*Not-a-Number* = nije broj). Ta vrijednost nije jednaka ni jednoj drugoj vrijednosti, pa čak ni samoj sebi. Zbog toga postoji posebna funkcija `isNaN()`, koja provjerava je li neka vrijednost broj ili ne. Funkcija `isFinite()` provjerava je li vrijednost broj i može li se prikazati kao konačan broj u *JavaScriptu*.

U tablici se navode još neke posebne vrijednosti:

Konstanta	Značenje
<code>Infinity</code>	Posebna vrijednost koja predstavlja beskonačnost.
<code>NaN</code>	Posebna vrijednost <i>nije-broj</i> .
<code>Number.MAX_VALUE</code>	Najveći broj koji se može prikazati.
<code>Number.MIN_VALUE</code>	Najmanji (najbliži nuli) broj koji se može prikazati.
<code>Number.NaN</code>	Posebna vrijednost <i>nije-broj</i> .
<code>Number.POSITIVE_INFINITY</code>	Posebna vrijednost koja predstavlja beskonačnost.
<code>Number.NEGATIVE_INFINITY</code>	Posebna vrijednost koja predstavlja negativnu beskonačnost.

Vrijednosti `Infinity` i `NaN` definirane su u standardu *ECMAScript v1* tako da nisu implementirane prije *JavaScripta 1.3*.

### 3.1.2. Nizovi znakova

U *JavaScriptu* se nizom znakova smatra bilo koji niz znakova *Unicode*, duljine 1 ili više, omeđen dvostrukim ili jednostrukim navodnicima ( " ili ' ). Niz znakova započet s dvostrukim navodnikom mora završiti dvostrukim navodnikom. Također, niz znakova započet jednostrukim navodnikom mora završiti jednostrukim navodnikom. Niz znakova započet dvostrukim navodnikom može sadržavati proizvoljan broj jednostrukih navodnika i obratno.

Niz znakova treba biti napisan u jednom retku. Ako postoji potreba za unosom novog retka, potrebno je rabiti znak `\n`.

```
"" // Ovo je prazan niz znakova, duljine 0

'test' // Niz znakova omeđen jednostrukim navodnicima

"3.14" // Ovo nije broj već niz znakova

'Potrebno je upisati "user"' // Dvostruki navodnici unutar niza znakova

"Običan niz znakova."

"Ovaj niz je\n u dva reda"
```

*JavaScript* se često kombinira s HTML-om, koji također rabi dvostruke ili jednostruke navodnike u vrijednostima argumenata. Stoga je poželjno imati stalan stil pisanja navodnika, na primjer:

```
<a href="javascript:window.open('about:blank'); void 0;">JavaScript</a>
```



Novi redak nije jedini slučaj kad treba napisati poseban znak. U JavaScriptu postoji nekoliko posebnih nizova znakova posebnog značenja:

Niz	Značenje
\0	NUL znak (\u0000).
\b	<i>Backspace</i> (\u0008).
\t	Vodoravni tabulator (\u0009).
\n	Novi red (\u000A).
\v	Okomiti tabulator (\u000B).
\f	<i>Form feed</i> (\u000C).
\r	<i>Carriage return</i> (\u000D).
\"	Dvostruki navodnici, čak i unutar niza znakova započelih dvostrukim navodnicima (\u0022).
\'	Jednostruki navodnici, čak i unutar niza znakova započelih jednostrukim navodnicima (\u0027).
\\	Obrnuta kosa crta (\u005C).
\xxx	Znak <i>Latin-1</i> naveden s dvije heksadecimalne znamenke xx.
\uXXXX	Znak <i>Unicode</i> naveden s četiri heksadecimalne znamenke XXXX.

Nizovi znakova spajaju se operatorom +:

```
poruka = "Dobrodošli u " + "Srce"; // Rezultat "Dobrodošli u Srce"
```

### 3.1.3. Logičke vrijednosti

Logičke vrijednosti (tzv. Booleove vrijednosti) imaju samo dvije moguće vrijednosti koje se prikazuju pomoću konstanti `true` i `false`. Logičke vrijednosti obično su rezultat uspoređivanja:

```
var a,b;
a = 4;
b = 7;
logVar = a > b; // logVar sadrži false jer je 4 < 7
```

### 3.1.4. Polja

Polje je skup vrijednosti u kojem svaki element ima jedinstveni redni broj koji se naziva indeks. Vrijednost određenog elementa dobije se tako da se iza naziva polja vrijednost indeksa stavi u uglate zagrade, tj. izraz `a[2]` označava treći element u polju `a`, jer su u **JavaScriptu elementi polja indeksirani od nule**.

Polja mogu sadržavati bilo koju vrstu podataka *JavaScripta*, uključujući i reference na druga polja, objekte ili funkcije. Budući da se u *JavaScriptu* ne navodi vrsta podatka pojedinih vrijednosti, nije nužno da su u jednom polju sve vrijednosti iste vrste (kao što je to u većini drugih jezika).

Polja se stvaraju pomoću objekta `Array`. Kad je polje stvoreno, može mu se dodati proizvoljan broj elemenata:

```
var a = new Array();

a[0] = 1.2;

a[1] = "JavaScript";

a[2] = true;

a[3] = iVarijabla;
```

Polja se mogu stvoriti i tako da im se odmah postave početne vrijednosti. Prethodni primjer mogao je izgledati ovako:

```
var a = new Array(1.2, "JavaScript", true, iVarijabla);
```

Ako pri stvaranju objekta `Array` proslijedite samo jedan parametar, on označava duljinu polja. Sljedeći primjer stvara novo polje od 10 elemenata:

```
var a = new Array(10);
```

### 3.1.5. Nepostojeća i nedefinirana vrijednost

Ključna riječ `null` označava posebnu vrijednost koja nema nikakvu vrijednost. Kad varijabla ima vrijednost `null`, tada ne sadrži ni jednu ispravnu vrstu podatka: objekt, polje, broj, niz znakova ili logičku vrijednost.

Druga je posebna vrijednost `undefined`, koja se dobije kad se uporabi varijabla koja je bila deklarirana, ali nema početnu vrijednost ili svojstvo objekta koje ne postoji.

Vrijednost `undefined` nije isto što i `null`, iako će test jednakosti vratiti istinu. Kao što će se pokazati u nastavku, za razlikovanje se koriste operatori `===` ili `typeof`.

## 3.2. Varijable

Varijabla je ime povezano s vrijednošću, tj. varijabla sadržava određenu vrijednost. Varijable omogućavaju pohranu i rukovanje podacima u programu.

Kao što je prije rečeno, prvi znak naziva varijable mora biti slovo, podcrta (`_`) ili dolarski znak (`$`). Znakovi koji slijede mogu biti slova, brojevi, podcrta ili dolarski znak. Ako se *JavaScript*- i HTML-datoteke

spremaju u kodnoj stranici UTF-8, tada se mogu koristiti i znakovi s dijakriticima (č, ž, š, ć i đ). Međutim, preporuka je rabiti samo znakove *ASCII*, odnosno 26 slova engleske abecede.

*JavaScript* je jezik u kojem se prilikom deklariranja varijable ne navodi vrsta podatka. Štoviše, jedna varijabla može sadržavati vrijednost bilo koje vrste, npr:

```
i = 10; // cijeli broj  
i = "deset"; // niz znakova  
i = false; // logička vrijednost
```

Varijabla dobije vrstu podatka na osnovi podatka koji sadrži. Štoviše, *JavaScript*, ako je to potrebno, automatski mijenja vrstu varijable. Prije nego se rabi varijabla, poželjno ju je deklarirati pomoću ključne riječi *var*, na primjer:

```
var i;  
  
var sum;  
  
var i, sum; // Ili obje odjednom  
  
var niz = "Opres!"; // Deklaracija s početnom vrijednosti
```

Ako pri deklaraciji varijable nije navedena početna vrijednost, ona se automatski postavlja na vrijednost *undefined*.

### 3.3. Objekti

Objekt je složena vrsta podatka koji skuplja više vrijednosti u jednu cjelinu. Također se može reći da je objekt skup imenovanih vrijednosti. Te se vrijednosti nazivaju svojstva objekta. Želi li se pristupiti određenom svojstvu, navede se ime objekta, točka i na kraju ime svojstva. Ako objekt *osoba* ima svojstva *ime* i *zanimanje*, tada se tim svojstvima pristupa ovako:

```
osoba.ime  
  
osoba.zanimanje
```

Objekti se stvaraju pozivom posebne funkcije (konstruktor). Npr. ovim se recima stvaraju objekti:

```
var objekt = new Object();  
  
var sada = new Date();  
  
var osoba = new Osoba('Pero', 'poštar');
```

Postupak ili metoda je funkcija *JavaScripta* koja se poziva kroz objekt:

```
// Funkcija

function identitet(){
    return "Ja sam " + this.ime + " po zanimanju " + this.zanimanje;
}

// Povezivanje s objektom

osoba.predstaviSe = identitet;
```

U gornjem primjeru ključna riječ `this` služi za pristup svojstvima objekta unutar njegove metode.

<b>O b j e k t</b>	<b>Svojstva</b>
	<b>Metode</b>

<b>A r r a y</b>	<b>length</b> ...
	<b>sort</b> <b>join</b> <b>reverse</b> ...

### 3.4. Vježba 1.

1. Izradite HTML-datoteku naziva `variable.html` ovog sadržaja:

```
<!DOCTYPE html>
<html lang="hr">
<head>
  <meta charset="UTF-8" />
  <title>Varijable</title>
</head>
<body>
  <script type="text/javascript">
    <!--
      . . .
    -->
  </script>
</body>
</html>
```

2. *JavaScript* program pišite unutar elementa `<script>`.
3. Deklarirajte varijablu `iBroj` i dodijelite joj vrijednost `1`.
4. Deklarirajte varijablu `sNiz1` i dodijelite joj vrijednost `'Niz znakova'`.
5. Deklarirajte varijablu `sNiz2` i dodijelite joj vrijednost `'3.14'`.
6. Deklarirajte varijablu `sNiz3` i dodijelite joj vrijednost `'U dva<br />retka'`.
7. Deklarirajte varijablu `bLogicka` i dodijelite joj vrijednost `true`.
8. Vrijednosti ispišite pomoću funkcije `document.write`.
9. Promijenite vrijednost varijable `bLogicka` u `false` te ju ponovo ispišite.

## Rješenje

```
<!DOCTYPE html>
<html lang="hr">
<head>
    <meta charset="UTF-8" />
    <title>Varijable</title>
</head>
<body>
    <script language="JavaScript">
var iBroj = 1,
    sNiz1 = 'Niz znakova',
    sNiz2 = '3.14',
    sNiz3 = 'U dva<br />retka!',
    bLogicka = true;

document.write('Broj: ' + iBroj + '<br />');
document.write('Niz 1: ' + sNiz1 + '<br />');
document.write('Niz 2: ' + sNiz2 + '<br />');
document.write('Niz 3: ' + sNiz3 + '<br />');
document.write('Logička: ' + bLogicka + '<br />');

bLogicka = false;

document.write('Logička: ' + bLogicka + '<br />');
    </script>
</body>
</html>
```

### Rezultat

**Broj: 1**

**Niz 1:** Niz znakova

**Niz 2:** 3.14

**Niz 3:** U dva  
retka!

**Logička:** true

## 4. Operatori

Operatori se rabe pri obavljanju određenih operacija nad varijablama i konstantama. Pregled svih operatora i njihovih svojstava nalazi se u dodatku ovog priručnika.

### 4.1. Aritmetički operatori

Aritmetički operatori su operatori osnovnih računskih matematičkih operacija.

Operator	Značenje
zbrajanje (+)	zbraja dva broja ili spaja dva niza znakova
oduzimanje (-)	oduzima drugi broj od prvoga ili kao unaran operator vraća negativnu vrijednost broja
množenje (*)	množi dva broja
dijeljenje (/)	dijeli prvi broj drugim i rezultat je uvijek decimalan broj; dijeljenje s nulom daje pozitivnu ili negativnu beskonačnost (ovisi o prvom broju) dok 0/0 daje NaN
modulo (%)	vraća ostatak od dijeljenja prvog broja drugim
inkrement (++)	povećava vrijednost operanda za jedan
dekrement (--)	umanjuje vrijednost operanda za jedan

```
window.alert(3 + 5);
```

### 4.2. Operator pridruživanja

Operator pridruživanja rabi se za pridruživanje vrijednosti varijabli.

```
a = 3;
```

Budući da je operator pridruživanja asocijativan (i to s desna), moguće je napisati:

```
var i = j = k = 0;
```

Sve tri varijable imat će vrijednost 0.

Operator pridruživanja ima i poseban oblik: pridruživanje s operacijom, tj. oblik:

```
a += b;

// je isto kao

a = a + b;
```

Operator pridruživanja s operacijom ne može se koristiti s operatorima inkrement (++) i dekrement (--).

### 4.3. Operatori uspoređivanja

Operatori uspoređivanja najčešće se koriste pri grananju (`if`) i uvjetnim petljama (`while` i `do...while`).

Operator	Značenje
manji od (<)	rezultat je true ako je prvi operand manji od drugoga; inače false
veći od (>)	rezultat je true ako je prvi operand veći od drugoga; inače false
manji ili jednak (<=)	rezultat je true ako je prvi operand manji ili jednak drugom; inače false
veći ili jednak (>=)	rezultat je true ako je prvi operand veći ili jednak drugom; inače false
jednak (==)	rezultat je true ako je prvi operand jednak drugom; inače false
identičan (===)	rezultat je true ako je prvi operand jednak drugom i jednake su vrste podatka; inače false
različit (!=)	rezultat je true ako je prvi operand različit od drugoga; inače false
ne-identičan (!==)	rezultat je true ako je prvi operand različit od drugoga ili su različite vrste podatka; inače false



Vrijednosti koje se uspoređuju mogu biti bilo koje vrste. No budući da se mogu uspoređivati samo brojevi i nizovi znakova, *JavaScript* prije uspoređivanja obavlja određena pretvaranja:

- Ako su oba operanda brojevi ili se mogu pretvoriti u brojeve, uspoređuju se kao brojevi.
- Ako su oba operanda nizovi znakova ili se mogu pretvoriti u niz znakova, uspoređuju se kao nizovi znakova.
- Ako je jedan operand niz znakova ili se može pretvoriti u niz znakova te ako je drugi operand broj ili se može pretvoriti u broj, niz znakova se pokušava pretvoriti u broj i uspoređuju se kao brojevi. Ako niz znakova nije broj, pretvara se u `NaN`, tj. uspoređivanje nije uspješno.
- Ako se objekt može pretvoriti u broj ili u niz znakova, *JavaScript* pretvara objekt u broj. To znači da se objekti *Date* uspoređuju kao brojevi.
- Ako se jedan od operanada koji se uspoređuju ne može uspješno pretvoriti u broj ili u niz znakova, uspoređivanje je uvijek neuspješno, tj. dobiva se *false*.
- Ako je jedan od operanada `NaN`, uspoređivanje je uvijek neuspješno.

Važno je primijetiti da se nizovi znakova uspoređuju znak po znak, rabeći kôd svakog znaka iz tablice *Unicode*. To znači da uspoređivanje nizova znakova može biti neobično. Naime, „Zagreb” je manji od „auto”.

Uspoređivanje također razlikuje velika i mala slova (jer su im kôdovi različiti).

Gore navedena pravila za pretvaranje tipova prilikom uspoređivanja ne primijenjuju se za operator identičnosti (`===`), koji zahtijeva da dvije varijable budu jednake i po vrijednosti i po vrsti podatka. U sljedećem primjeru dobiva se različit rezultat prilikom uspoređivanja s pojedinim operatorom:

```
var a = 2;
var b = "2";
document.write(a == b); // true
document.write(a === b); // false
```

## 4.4. Logički operatori

Logički operatori se najčešće koriste kod složenih uvjeta za grananja i petlje.

Operator	Značenje
logičko i (&&):	rezultat je <code>true</code> ako i samo ako su oba operanda <code>true</code> ; inače daje <code>false</code>
logičko ili (  ):	rezultat je <code>true</code> ako je jedan od operanada <code>true</code> ; inače daje <code>false</code>
logičko ne (!):	unarni operator kod kojeg je rezultat <code>true</code> samo ako je operand <code>false</code> ; inače daje <code>false</code>

```
var a = true;
var b = false;
document.write(a && b); // false
document.write(a || b); // true
```

## 4.5. Operator spajanja

Operator `+` spaja dva niza znakova u jedan novi. Npr:

```
a = "2"; b = "2";
c = a + b; // c je 22, a ne 4!!!
```

Operator `+` daje viši prioritet nizovima znakova nego brojevima. Dakle, ako je jedan operand niz znakova, onda se i drugi operand pretvori u niz znakova i obavi se spajanje. Kod operatora uspoređivanja je obratno. Naime, ako je jedan operand broj, drugi se pretvara u broj i obavlja se uspoređivanje. Stoga je važno pripaziti:

```
1 + 2 // Oba operanda su brojevi, rezultat je 3.
"1" + "2" // Oba operanda su nizovi znakova, rezultat je "12".
"1" + 2 // Drugi operand se pretvara u niz znakova,
// rezultat je "12".
11 < 3 // Oba operanda su brojevi, rezultat je false.
"11" < "3" // Oba operanda su nizovi znakova, rezultat je true.
"11" < 3 // Prvi operand se pretvara u broj, rezultat je false.
"one" < 3 // Prvi operand se pretvara u broj (postaje Nan),
// rezultat je false.
```

## 4.6. Vježba 2.

1. Izradite HTML-datoteku, naziva `operatori.html`, istog sadržaja kao u vježbi 1.
2. Deklarirajte dvije varijable `iA` i `iB` i dodijelite im proizvoljne cjelobrojne vrijednosti.
3. Deklarirajte varijable `iSuma`, `iRazlika` i `iModulo` koje će sadržavati redom: sumu, razliku, modulo varijabli `iA` i `iB`.
4. Postavite inicijalne vrijednosti varijabli `iSuma`, `iRazlika` i `iModulo` na nulu koristeći se operatorom pridruživanja.
5. Deklarirajte varijablu `bLogicka`.
6. Prikažite inicijalne vrijednosti varijabli `iA` i `iB` pomoću funkcije `document.write`.
7. Izračunajte tri aritmetičke operacije iz druge točke, dodijelite ih odgovarajućim varijablama i prikažite ih.
8. Inkrementirajte varijablu `iA` i dekrementirajte varijablu `iB` te prikažite nove vrijednosti varijabli `iA` i `iB`.
9. Već izračunatoj sumi dodajte novu vrijednost varijable `iA` koristeći se pridruživanjem s operacijom. Prikažite novu vrijednost varijable `iSuma`.
10. Varijabli `bLogicka` dodijelite rezultat usporedbe je li vrijednost varijable `iA` veća od 5. Prikažite vrijednost varijable `bLogicka`.
11. Prikažite `iA + iB` i uočite da se ne dobije zbroj već niz znakova.
12. Ispišite `(iA + iB)` i uočite da je zbroj točno ispisan.

**Rješenje**

```
<script type="text/javascript">
var iA = 10,
    iB = 3,
    iSuma,
    iRazlika,
    iModulo,
    bLogicka;

iSuma = iRazlika = iModulo = 0;
```

```
document.write('Početna vrijednost iA: ' + iA + '<br />');
document.write('Početna vrijednost iB: ' + iB + '<br />');

iSuma = iA + iB;
iRazlika = iA - iB;
iModulo = iA % iB;
document.write('Suma: ' + iSuma + '<br />');
document.write('Razlika: ' + iRazlika + '<br />');
document.write('Modulo: ' + iModulo + '<br />');

iA++;
iB--;
document.write('iA nakon ++: ' + iA + '<br />');
document.write('iB nakon --: ' + iB + '<br />');

iSuma += iA;
document.write('Sumi dodamo iA: ' + iSuma + '<br />');

bLogicka = iA > 5;
document.write('Je li iA > 5: ' + bLogicka + '<br />');

document.write('Paziti na konverzije iA + iB: ' +
    iA + iB + '<br />');
document.write('Paziti na konverzije (iA + iB): ' +
    (iA + iB) + '<br />');
</script>
```

**Rezultat**

```
Početna vrijednost iA: 10
Početna vrijednost iB: 3
Suma: 13
Razlika: 7
Modulo: 1
iA nakon ++: 11
iB nakon --: 2
Sumi dodamo iA: 24
Je li iA > 5: true
Paziti na konverzije iA + iB: 112
Paziti na konverzije (iA + iB): 13
```

## 5. Funkcije

### 5.1. Definiranje funkcije

Funkcija je konstrukcija u *JavaScriptu* pomoću koje grupiramo veći broj naredbi koje izvršavamo navodeći ime funkcije. Tako skraćujemo pisanje programa (ako više puta pozivamo istu funkciju) te pojednostavljujemo program (npr. cijeli program može se sastojati od poziva funkcija proizvoljnog imena te tako pokazuje logiku programa, što bi inače bilo skriveno u velikom broju naredbi). Najčešći je način definiranja funkcije pomoću ključne riječi `function`:

```
function <naziv_funkcije>(<argument 1>, <argument
2>, ...){
    // naredbe
}
```

Popis argumenata nije obavezan, ali okrugle zagrade jesu. Tijelo funkcije piše se u vitičastim zagradama `{ ... }` koje označavaju blok naredbi, tj. to je način da se nekoliko naredbi poveže u jednu cjelinu.

Funkcija može vratiti vrijednost naredbom `return`, ali i ne mora. Evo nekoliko primjera:

```
// Funkcija koja ništa ne vraća
function ispis(sPoruka) {
    document.write(sPoruka, '<br />');
}

// Funkcija koja vraća udaljenost dviju točaka
function fUdaljenost(x1, y1, x2, y2) {
    var fDx,
        fDy,
        fRezultat;

    fDx = x2 - x1;
    fDy = y2 - y1;
    fRezultat = Math.sqrt(fDx * fDx + fDy * fDy);

    return fRezultat;
}

// Rekurzivna funkcija (poziva samu sebe) koja računa faktorijel
// Podsjetite se:  $x! = x * (x-1) * (x-2) * \dots * 3 * 2 * 1$ 
function faktorijel(x){
    if (x <= 1){
        return 1;
    }
    return x * faktorijel(x-1);
}
```

## 5.2. Poziv funkcije

Funkcija se poziva tako da se navede njezin naziv, a argumenti funkcije u okruglim zagradaama. Ako funkcija nema argumenata, ne navodi se ništa, ali zagrade su obavezne. Ako se funkciji proslijedi manje argumenata nego ih sadrži definicija funkcije, drugi argumenti dobiju vrijednost `undefined`. Na primjer, gore definirane funkcije pozivaju se ovako:

```
ispis("Kako si, " + ime);
ispis("Pozdrav svima!");
ukupno = fUdaljenost(0,0,2,1) + fUdaljenost(2,1,3,5);
ispis("Vjerojatnost je: " + faktoriyel(39)/faktoriyel(52));
```

## 5.3. Doseg varijabli

U *JavaScriptu* vrijednost je varijable dostupna na dva načina – samo unutar određene funkcije ili u cijelom programu. Varijable koje su dostupne u cijelom programu nazivamo globalne varijable, a varijable koje su dostupne samo unutar funkcije nazivamo lokalne varijable. Prilikom uporabe varijabli poželjno je uvijek ih deklarirati pomoću ključne riječi `var`:

- Ako je varijabla deklarirana pomoću ključne riječi `var` ili joj je samo dodijeljena vrijednost u glavnom programu (izvan svih funkcija), varijabla je globalna.
- Ako je varijabla deklarirana unutar određene funkcije pomoću ključne riječi `var`, tada je varijabla lokalna.
- Ako varijabla nije deklarirana, nego joj je samo dodijeljena neka vrijednost unutar određene funkcije, tada je varijabla globalna.

```
var iGlobalna1 = 10;
iGlobalna2 = 34;

// dostupne su iGlobalna1 i iGlobalna2

function funkcija1(){
    var iLokalna1 = 4;
    iGlobalna3 = 15;
    // dostupne su iGlobalna1, iGlobalna2 i iLokalna1
}

// dostupne su iGlobalna1 i iGlobalna2
function funkcija2(){
    var iLokalna2 = 7;
    // dostupne su iGlobalna1, iGlobalna2 i iLokalna2
}
```

## 5.4. Vježba 3.

1. Izradite HTML-datoteku naziva `funkcije.html` istog sadržaja kao u vježbi 1.
2. Iskoristite definicije funkcija `ispis` i `fUdaljenost` iz točke 5.1.
3. Deklarirajte varijable `sMjesto`, `iMjerilo` i `iUkupno`.
4. Varijabli `sMjesto` dodijelite proizvoljan naziv grada.
5. Varijabli `iMjerilo` dodijelite vrijednost 25.
6. Izračunajte udaljenost mjesta od mora, tj. duljinu puta od točke (0,0) do točke (3,5) kroz točku (2,1) na karti mjerila pohranjenoj u varijabli `iMjerilo`.
7. Ispišite dobrodošlicu u grad.
8. Ispišite izračunatu udaljenost do mora (u kilometrima).

### Rješenje

```
<script type="text/javascript">
function ispis(sPoruka) {
    document.write(sPoruka, '<br />');
}

function fUdaljenost(x1, y1, x2, y2) {
    var fDx = x2 - x1,
        fDy = y2 - y1,
        fRezultat = Math.sqrt(fDx * fDx + fDy * fDy);
    return fRezultat;
}

var sMjesto = "Velegrad",
    iMjerilo = 25,
    iUkupno;

iUkupno = Math.round(fUdaljenost(0, 0, 2, 1) +
    fUdaljenost(2, 1, 3, 5)) * iMjerilo;

ispis("Dobrodošli u " + sMjesto + "!");
ispis(sMjesto + " je " + iUkupno + " kilometara udaljen od mora.");
</script>
```

#### Rezultat

Dobrodošli u Velegrad!  
Velegrad je 150 kilometara udaljen od mora.



## 6. Naredbe za kontrolu tijeka

### 6.1. Uvjetno izvođenje naredbi

Osnovna naredba za grananje je naredba `if`. Njezin je najjednostavniji oblik:

```
if (iA > iB) {  
    window.alert(iA + ' je veće od ' + iB);  
}
```

gdje je uvjet logički izraz čiji je rezultat istina (`true`) ili neistina (`false`). Kad operandi u izrazu nisu logički, prevode se u logičke vrijednosti. Ove se vrijednosti uvijek prevode u `false`:

- `null`
- `undefined`
- prazan niz znakova (`' '` ili `""`)
- broj `0`
- `NaN`.

Sve druge vrijednosti prevode se u `true`.

U ovom primjeru vrijednost varijable `ime` je prazan niz, što će se prevesti kao `false`.

```
var ime = '';  
if (ime) {  
    window.alert('Ime je uneseno');  
}
```

Vitičaste zagrade, koje označavaju blok naredbi, nisu potrebne ako iza uvjeta slijedi samo jedna naredba, ali se preporuča uvijek ih pisati radi jednoznačnosti.

Drugi oblik je oblik `if...else`:

```
if (a > b) {  
    window.alert(a + ' je veće od ' + b);  
} else {  
    window.alert(a + ' nije veće od ' + b);  
}
```

Taj oblik ima još jedan blok, koji se izvodi ako uvjet nije zadovoljen.

## 6.2. Višestruka usporedba

Ako jednu vrijednost treba usporediti više puta, rabi se nekoliko naredbi

`if...else`:

```
if(n == 1){
    // naredbe1
}
else if(n == 2){
    // naredbe2
}
else if(n == 3){
    // naredbe3
}
else {
    // naredbe4
}
```

Međutim, takav način pisanja višestruke usporedbe prilično je nepregledan. Stoga se rabi naredba `switch`:

```
switch(n) {
    case 1:
        // naredbe1
        break;
    case 2:
        // naredbe2
        break;
    case 3:
        // naredbe3
        break;
    default:
        // naredbe4
        break;
}
```

Ta naredba provjerava je li izraz u okruglim zagradama (odmah iza ključne riječi `switch`) istovjetan (koristi se operator `===`), dakle, i po vrsti jednak, jednoj od vrijednosti iza ključne riječi `case`. Ako takva vrijednost postoji, izvršava se blok naredbi iza te vrijednosti do kraja cijele naredbe `switch` ili do ključne riječi `break`, koja prekida blok koji se trenutno izvršava, tj. završava naredbu `switch`. Dakle, moguće je tu naredbu napisati tako da se jedan blok izvrši za više vrijednosti:

```
switch(n) {
    case 1:
    case 2:
    case 3:
        // naredbe1
        break;
    case 4:
        // naredbe2
        break;
    default:
        // naredbe3
        break;
}
```

**Treba pripaziti da se na kraju svakog bloka naredbi napiše naredba `break`.** Ako se ona nehotice izostavi, neće biti prijavljena pogreška, nego će se izvršiti i sljedeći blok naredbi.

Ako ne postoji vrijednost koja je istovjetna provjeravanom izrazu, izvršava se blok iza ključne riječi `default`. Iako je uobičajeno da se blok `default` piše posljednji (jer onda ne treba pisati naredbu `break`), to nije obavezno. Taj je blok ravnopravan drugim blokovima.

### 6.3. Uvjetni operator

Uvjetni operator jedini je operator s tri operanda u *JavaScriptu*:

```
<uvjet> ? naredba1 : naredba2;
```

što je jednako ovom:

```
if(<uvjet>)  
    naredba1;  
else  
    naredba2;
```

Taj operator najčešće se koristi pri inicijalizaciji varijabli:

```
sPunoIme = sIme != null ? sIme : 'Nepoznato';
```

### 6.4. Petlja s uvjetom na početku

Osnovna je petlja u *JavaScriptu* petlja `while`:

```
var i = 1;  
while(i <= 10){  
    document.write('Red ' + i + '<br />');  
    i++;  
}
```

Tijelo petlje izvršava se sve dok je uvjet zadovoljen. Stoga bi se naredbom `while(true){...}` napravila beskonačna petlja. Ako uvjet nije zadovoljen prije početka petlje, njezino tijelo se neće izvesti niti jednom.

## 6.5. Petlja s uvjetom na kraju

Često je potrebno izvršiti neke postupke barem jednom i tek tada provjeriti određeni uvjet. U takvom slučaju koristi se `do..while` konstrukcija:

```
var i = 1;
do {
    document.write('Red ' + i + '<br />');
    i++;
} while(i <= 10);
```

Konstrukcija `do..while` mora se završiti kao naredba točka-zarezom (zato jer završava uvjetom, a ne blokom kao osnovna petlja `while`).

## 6.6. Petlja s poznatim brojem ponavljanja

Kad je točno poznat broj ponavljanja nekog postupka ili su poznati početni i krajnji uvjet, koristi se petlja `for`:

```
for(<inicijalizacija>; <uvjet>; <korak>) {
    // naredbe
}
```

Petlja `for` istovjetna je ovoj petlji `while`:

```
<inicijalizacija>;
while(<uvjet>) {
    // naredbe
    <korak>;
}
```

Primjer uporabe:

```
for(i = 0; i <= 10; i++) {
    document.write('Red ' + i + '<br />');
}
```

## 6.7. Vježba 4.

1. Izradite HTML-datoteku naziva `petlje.html` istog sadržaja kao u vježbi 1.
2. Definirajte funkciju `ispis` kao u vježbi 3.
3. Deklarirajte varijable `iA`, `iB`, `sOperacija` i `iRazlika`.
4. Varijablama `iA` i `iB` dodijelite proizvoljne vrijednosti.
5. Varijabli `sOperacija` dodijelite niz znakova `+`.
6. Rabeći jednostavno grananje provjerite je li varijabla `iA` veća ili manja od `iB`.
7. Rabeći višestruko grananje ispišite koja je operacija zapisana u varijabli `sOperacija`.
8. Izračunajte i ispišite razliku brojeva `iA` i `iB`, ali tako da oduzmete veći broj od manjeg. Uporabite uvjetni operator.
9. Uporabom petlje `for` ispišite 10 redaka teksta: "*For: redak x*", gdje je *x* broj retka.
10. Napišite prethodnu petlju `for` pomoću petlje `while`.

## Rješenje

```
<script type="text/javascript">
function ispis(sPoruka) {
    document.write(sPoruka, '<br />');
}

var iA = 10,
    iB = 13,
    sOperacija = '+',
    iRazlika,
    iBrojac,
    jBrojac;

if (iA > iB) {
    ispis("iA je veci");
} else {
    ispis("iA je manji");
}

switch (sOperacija) {
case '+':
    ispis('Zbrajanje');
    break;
case '-':
    ispis('Oduzimanje');
    break;
case '*':
    ispis('Množenje');
    break;
case '/':
    ispis('Dijeljenje');
    break;
default:
    ispis('Nepoznato');
}
```

### Rezultat

iA je manji  
Zbrajanje  
Prava razlika: 3  
For: redak 1  
For: redak 2  
For: redak 3  
For: redak 4  
For: redak 5  
For: redak 6  
For: redak 7  
For: redak 8  
For: redak 9  
For: redak 10  
While: redak 1  
While: redak 2  
While: redak 3  
While: redak 4  
While: redak 5  
While: redak 6  
While: redak 7  
While: redak 8  
While: redak 9  
While: redak 10

```
iRazlika = iA > iB ? iA - iB : iB - iA;

ispis('Prava razlika: ' + iRazlika);

for (iBrojac = 1; iBrojac <= 10; iBrojac++) {
    ispis('For: redak ' + iBrojac);
}

jBrojac = 1;
while (jBrojac <= 10) {
    ispis('While: redak ' + jBrojac);
    jBrojac++;
}
</script>
```





## 7. Obrasci

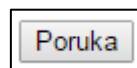
### 7.1. Prvi obrazac

HTML-obrazac dio je dokumenta koji sadrži tekst i posebne elemente zvane kontrole (potvrdni okvir (*checkbox*), izborna dugme (*radio button*), izbornik itd.) Korisnici obično „popune” obrazac mijenjajući kontrole (unose tekst, označavaju mogućnosti, odabiru od ponuđenog itd.). Na kraju „popunjavanja” korisnik šalje podatke na obradu. Obrasci su najčešći način prijave ili ispunjavanja podataka na HTML-stranicama. Sastoje se od niza elemenata od kojih svaki ima svoje posebnosti. Ovo je najjednostavniji obrazac koja ima samo jedno dugme *Poruka*:

```
<!DOCTYPE html>
<html lang="hr">
<head>
  <meta charset="UTF-8" />
  <title>Primjer</title>
  <script type="text/javascript">
    function prva() {
      window.alert('Osnove JavaScripta');
    }
  </script>
</head>

<body>
  <form action="">
    <input
      type="button"
      value="Poruka"
      onclick="prva();" />
  </form>
</body>

</html>
```



Budući da je pisanje *JavaScripta* u odvojenoj datoteci praktičnije, preporuča se zasebno pisanje HTML-a i *JavaScripta*:

```
<!-- obrazac.html -->
<!DOCTYPE html>
<html lang="hr">

<head>
  <meta charset="UTF-8" />
  <title>Primjer</title>
  <script type="text/javascript" src="obrazac.js"></script>
</head>
<body>
  <form action="">
    <input type="button"
      value="Poruka" onclick="prva();" />
  </form>
</body>
</html>

// obrazac.js
function prva() {
  window.alert('Osnove JavaScripta');
}
```

U sljedećim će primjerima uglavnom biti naveden dio samo unutar elementa `<form>`. Ostatak HTML-a se ponavlja.

## 7.2. Unos kraćih nizova znakova

Kraći nizovi znakova (nizovi koji imaju samo jedan red) unose se u tzv. tekstno polje. Primjer se nadogradi jednim poljem za unos imena:

```
<form action="">
  Unesite ime:
  <input
    type="text"
    name="ime"
    value="" />
  <br>
  <input
    type="button"
    value="Provjeri"
    onclick="provjeri();" />
</form>
```



```
function provjeri() {
  var sIme = '';

  sIme = document.forms[0].ime.value;
  window.alert('Ime je: ' + sIme);
}
```

Rabi se polje `forms` (indeks je 0, jer je taj obrazac prvi u polju `forms`), što je prilično nečitko, a može biti i problematično. Da bi se izbjegao takav način pisanja, svakom elementu obrasca dodijeli se atribut `id`. HTML preporuka definira `id` atribut kao atribut koji jedinstveno definira element na razini dokumenta, odnosno u jednoj HTML-datoteci ne smiju biti dva elementa s istom vrijednosti `id` atributa.

```
<form action="">
  <div>
    <label for="ime">Unesite ime:</label>
    <input
      type="text"
      name="ime"
      id="ime"
      value="" />
  </div>
  <input
    type="button"
    value="Provjeri"
    onclick="provjeri();" />
</form>
```

Element se tada dohvaća pomoću funkcije `getElementById`.

```
function provjeri() {
  var sIme = '';

  sIme = document.getElementById('ime').value;
  window.alert('Ime je: ' + sIme);
}
```

Prvi je korak u provjeri podataka koje upisuje korisnik: je li korisnik uopće upisao potrebni podatak:

```
function provjeri() {
  var sIme = '';

  sIme = document.getElementById('ime').value;
  if (sIme === '') {
    window.alert('Ime je prazno!');
  } else {
    window.alert('Ime je: ' + sIme);
  }
}
```

U sljedećem koraku obrazac se proširi poljem za prezime i elementu form se dodaju atributi action i method:

```
<form
  action="http://www.htmlcodetutorial.com/cgi-bin/mycgi.pl"
  method="GET">
  <div>
    <label for="ime">Unesite ime:</label>
    <input
      type="text"
      name="ime"
      id="ime"
      value="" />
  </div>

  <div>
    <label for="prezime">Unesite prezime:</label>
    <input
      type="text"
      name="prezime"
      id="prezime"
      value="" />

  </div>
  <input
    type="submit"
    value="Pošalji"
    onclick="return provjeri();" />
</form>
```

Vrijednost atributa action

`http://www.htmlcodetutorial.com/cgi-bin/mycgi.pl`

je putanja do poslužiteljske (*Perl*) skripte koja u tablici prikaže nazive i vrijednosti svih polja koja se pojavljuju u obrascu.

Da bi se mogla iskoristiti poslužiteljska skripta, potrebno je promijeniti vrstu dugmeta iz `button` u `submit`.

Svaki element u HTML-u (a to znači i u HTML-obracu) ima svoje podrazumijevano ponašanje. Na primjer, element `a` je hiperlink na drugi dokument i njegovo je ponašanje da se klikom mišem na taj element prikaže dokument. Klikom mišem na dugme za slanje (element `input` tipa `submit`) podaci iz obrasca šalju se na URL naveden u atributu `action`.

*JavaScriptom* se ta podrazumijevana ponašanja mogu promijeniti. Na primjer, ako je *Submit* dugme povezano s događajem `click`, dugme će se ponašati podrazumijevano samo ako funkcija u tom događaju vrati `true`. Ako funkcija vrati `false`, podrazumijevano ponašanje se prekida.

Stoga će funkcija provjeri vratiti `true` ako je provjera prošla, odnosno `false` ako nije.

```
function provjeri() {  
    var sIme = '';  
  
    sIme = document.getElementById('ime').value;  
    if (sIme === '') {  
        window.alert('Ime je prazno!');  
        return false;  
    } else {  
        return true;  
    }  
}
```

Još treba dodati i provjeru polja prezime:

```
function provjeri() {  
    var sIme = '',  
        sPrezime = '',  
        bOK = false;  
  
    sIme = document.getElementById('ime').value;  
    sPrezime = document.getElementById('prezime').value;  
    if (sIme === '') {  
        window.alert('Ime je prazno!');  
        bOK = false;  
    } else {  
        window.alert('Ime je: ' + sIme);  
        bOK = true;  
    }  
    if (sPrezime === '') {  
        window.alert('Prezime je prazno!');  
        bOK = false;  
    } else {  
        window.alert('Prezime je: ' + sPrezime);  
        bOK = bOK && true;  
    }  
    return bOK;  
}
```

Dakle, u slučaju da su i ime i prezime prazni, korisnik se upozorava dva puta. Porastom broja polja, povećao bi se i broj poruka za korisnike, što može biti neugodno. Bolje je neovisno provjeriti polja i zapisati upozorenja u niz znakova koji se ispisuje na kraju:

```
function provjeri() {
    var sIme = '',
        sPrezime = '',
        sPogreska = '',
        sPoruka = '';

    sIme = document.getElementById('ime').value;
    sPrezime = document.getElementById('prezime').value;
    if (sIme === '') {
        sPogreska += 'Ime je prazno!';
    } else {
        sPoruka += 'Ime je: ' + sIme;
    }
    if (sPrezime === '') {
        sPogreska += '\nPrezime je prazno!';
    } else {
        sPoruka += '\nPrezime je: ' + sPrezime;
    }
    if (sPogreska === '') {
        window.alert(sPoruka);
        return true;
    } else {
        window.alert(sPogreska);
        return false;
    }
}
```

Sada se poruka prikazuje samo jednom.

### 7.3. Odabir jedne od mogućnosti

Dodaje se element `input` tipa *radio* za vrstu računala:

Unesite ime:  
  
 Unesite prezime:  
  
 Računalo:  
☒ Stolno ☐ Prijenosno

```
<div>
  <label for="komp">Računalo:</label>
  <input
    type="radio"
    name="komp"
    value="stolno" /> Stolno
  <input
    type="radio"
    name="komp"
    value="prijenosno" /> Prijenosno
</div>
```

Pretpostavi se da korisnik nije odabrao dugme. Zato se varijabla `sKomp` postavi kao prazan niz znakova. Za svako dugme se provjeri je li ga korisnik označio i ako da, zapamti se vrijednost atributa `value` odabranog dugmeta:

```
function provjeri() {
  var sIme = '',
      sPrezime = '',
      sPogreska = '',
      sPoruka = '',
      sKomp = '',
      aoKomp,
      iBrojac;

  . . .

  aoKomp = document.getElementsByName('komp');
  for (iBrojac = 0; iBrojac < aoKomp.length; iBrojac++) {
    if (aoKomp[iBrojac].checked) {
      sKomp = aoKomp[iBrojac].value;
    }
  }
  if (sKomp === '') {
    sPogreska += '\nNiste izabrali vrstu računala!';
  } else {
    sPoruka += '\nRačunalo je: ' + sKomp;
  }
  . . .
}
```

U gornjoj obradi koristila se funkcija `getElementsByName` koja dohvaća sve elemente koji imaju atribut `name` postavljen na navedenu vrijednost.



Pojedini element `input` tipa *radio* imaju svojstvo `checked` koje je povezano sa stanjem tog elementa, odnosno ako je pojedini element odabran, tada svojstvo `checked` ima vrijednost `true`; inače ima vrijednost `false`.

Također pojedini elementi `input` tipa *radio* imaju svojstvo `value` koje odgovara vrijednosti atributa `value` u HTML-obrascu.

## 7.4. Uključivanje i isključivanje jedne mogućnosti

Ako je neka mogućnost proizvoljna, tada se upotrebljava tzv. element `checkbox`:

```
<div>
  <label>
    <input
      type="checkbox"
      name="internet"
      id="internet"
      value="sirokopojasni">
    Pristup Internetu je
    DSL tehnologijom.</label>
</div>
```

The screenshot shows a web form with the following elements:

- A label "Unesite ime:" followed by a text input field.
- A label "Unesite prezime:" followed by a text input field.
- A label "Računalo:" followed by two radio buttons labeled "Stolno" and "Prijenosno".
- A checkbox labeled "Pristup Internetu je DSL tehnologijom."
- A "Pošalji" button.

Obrada takvog elementa je jednostavna:

```
. . .
if (iOdabran === -1) {
  sPogreska += '\nNiste izabrali vrstu računala!';
} else {
  sPoruka += '\nRačunalo je: ' + aoKomp[iOdabran].value;
}

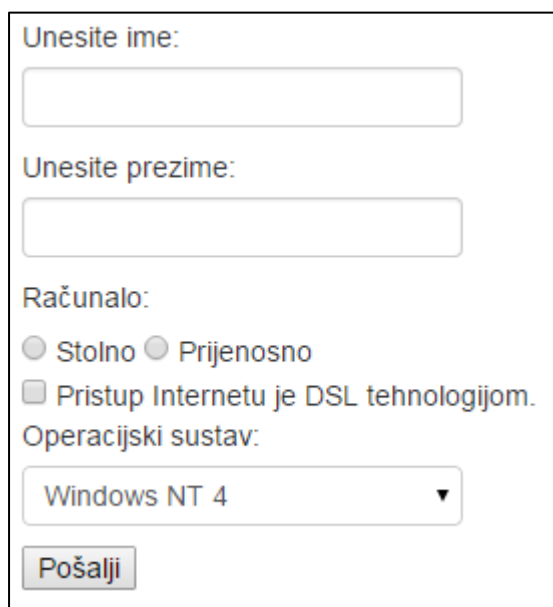
sPoruka += '\nPristup internetu je ';
sPoruka += document.getElementById('internet').checked ?
  'širokopojasni!' :
  'modemom!';

if (sPogreska === '') {
  window.alert(sPoruka);
}
. . .
```

## 7.5. Odabir jedne od mogućnosti na drugi način

Kad postoji veliki broj mogućnosti, nepregledno je sve ih ispisivati. Element `select` prikazuje samo izabranu mogućnost dok korisnik ne klikne mišem na njega.

```
<div>
  <label for="os">Operacijski sustav:</label>
  <select name="os" id="os">
    <option>Windows NT 4</option>
    <option>Windows 2000</option>
    <option>Windows XP</option>
    <option>Windows Vista</option>
    <option>Windows 7</option>
    <option>Windows 8</option>
  </select>
</div>
```



Unesite ime:

Unesite prezime:

Računalo:

☐ Stolno ☐ Prijenosno

☐ Pristup Internetu je DSL tehnologijom.

Operacijski sustav:

Windows NT 4 ▼

Pošalji

Obrada je elementa `select` složenija. Naime, svaki objekt `select` u *JavaScriptu* ima svojstvo `options`, polje koje sadrži vrijednosti i tekstove za svaku pojedinu mogućnost izbora.

```
function provjeri() {
    var sIme = '',
        sPrezime = '',
        sPogreska = '',
        sPoruka = '',
        sKomp = '',
        aoKomp,
        iStavka,
        iBrojac;

    . . .
    sPoruka += '\nPristup internetu je ';
    sPoruka += document.getElementById('internet').checked ?
        'širokopojasni!' :
        'modemom!';

    iStavka = document.getElementById('os').selectedIndex;
    sPoruka += '\nOS: ';
    sPoruka += document.getElementById('os').options[iStavka].text;

    if (sPogreska === '') {
        window.alert(sPoruka);
    }
    . . .
}
```

Svaki element `<option>` može imati atribut `value`, što je korisno kad je korisniku potrebno prikazati određeni tekst (tekst unutar elementa), a postaviti vrijednost `<select>` elementa na drugu vrijednost (tekst u atributu `value`). Vrijednosti teksta pristupamo svojstvom `text`, a vrijednosti atributa svojstvom `value`. Prepravljeni obrazac izgleda ovako:

```
<div>
    <label for="os">Operacijski sustav:</label>
    <select name="os" id="os">
        <option value="nt4">Windows NT 4</option>
        <option value="w2k">Windows 2000</option>
        <option value="xp">Windows XP</option>
        <option value="vista">Windows Vista</option>
        <option value="win7">Windows 7</option>
        <option value="win8">Windows 8</option>
    </select>
</div>
```

Prilagodi se i skripta:

```
iStavka = document.getElementById('os').selectedIndex;  
sPoruka += '\nOS: ';  
sPoruka += document.getElementById('os').options[iStavka].text;  
sPoruka += ' (';  
sPoruka += document.getElementById('os').options[iStavka].value;  
sPoruka += ')!';
```

Međutim, operacijski sustavi imaju i zakrpe o kojima ovisi kako se ponašaju sustav i pojedine aplikacije. Stoga se dodaje još i izbor „Service pack”:

Unesite ime:

Unesite prezime:

Računalo:

☐ Stolno ☐ Prijenosno

☐ Pristup Internetu je DSL tehnologijom.

Operacijski sustav:

Windows NT 4 ▼

Service pack:

Service pack 1 ▼

```

. . .
<div>
  <label for="os">Operacijski sustav:</label>
  <select name="os" id="os">
    <option value="nt4">Windows NT 4</option>
    <option value="w2k">Windows 2000</option>
    <option value="xp">Windows XP</option>
    <option value="vista">Windows Vista</option>
    <option value="win7">Windows 7</option>
    <option value="win8">Windows 8</option>
  </select>
</div>
<div>
  <label for="srv_pack">Service pack:</label>
  <select name="srv_pack" id="srv_pack">
    <option value="sp1">Service pack 1</option>
    <option value="sp2">Service pack 2</option>
    <option value="sp3">Service pack 3</option>
    <option value="sp4">Service pack 4</option>
    <option value="sp5">Service pack 5</option>
    <option value="sp6">Service pack 6</option>
    <option value="sp6a">Service pack 6a</option>
  </select>
</div>
. . .

```

I prilagodi se skripta:

```

function provjeri() {
  var sIme = '',
      sPrezime = '',
      sPogreska = '',
      sPoruka = '',
      sKomp = '',
      aoKomp,
      iStavka,
      iSPStavka,
      iBrojac;
  . . .

```

```

iStavka = document.getElementById('os').selectedIndex;
sPoruka += '\nOS: ';
sPoruka += document.getElementById('os').options[iStavka].text;
sPoruka += ' (';
sPoruka += document.getElementById('os').options[iStavka].value;
sPoruka += ')!';

iSPStavka = document.getElementById('srv_pack').selectedIndex;
sPoruka += '\nSP: ';
sPoruka += document.getElementById('srv_pack').options[iSPStavka].text;
sPoruka += ' (';
sPoruka += document.getElementById('srv_pack').options[iSPStavka].value;
sPoruka += ')!';

if (sPogreska === '') {
    window.alert(sPoruka);
}
. . .

```

U takvom obrascu korisnik može izabrati neobične kombinacije, kao što su: *Windows XP SP 6* ili *Windows 7 SP3*. Dakle, za svaki operacijski sustav treba definirati njegov skup zakrpa, koji se dinamički mijenja ovisno o odabranom sustavu. Stoga se u obrascu isprazni popis zakrpa.

Da bi se drugi popis (*service pack*) ispunio u ovisnosti o odabranom operacijskom sustavu, potrebno je obraditi događaj `change` (postavljanjem atributa `onchange`) na popisu operacijskih sustava (element `select` s atributom `id` postavljen na vrijednost `os`).

#### Napomena

Događaj `load` objašnjen je u dodatku E, na kraju priručnika.

S obzirom na to da je drugi popis na početku prazan, postaviti će se kao da je izabran prvi sustav na popisu, tj. onaj s indeksom 0, odnosno prvi *service pack*. Budući da to zahtijeva dodatnu intervenciju korisnika, početni odabir za *service pack* postaviti će se na zadnji. To se postiže obradom događaja `load` (postavljanjem atributa `onload`) za element `body`.

U primjeru se pojavljuje i ključna riječ `this`. U ovom slučaju ona označava objekt na kojem se dogodio događaj (element `select` s popisom operacijskih sustava, na kojem se dogodio događaj `change`). U ovom primjeru umjesto `this` moglo bi pisati

```
document.getElementById('os').selectedIndex
```

```

<body onload="puniSelect(document.getElementById('srv_pack'),
    asSP, document.getElementById('os').selectedIndex);">
<form
    action="http://www.htmlcodetutorial.com/cgi-bin/mycgi.pl"
    method="GET">
    . . .
    <div>
        <label for="os">Operacijski sustav:</label>
        <select
            name="os"
            id="os"
            onchange="puniSelect(document.getElementById('srv_pack'),
                asSP, this.selectedIndex);">
            <option value="nt4">Windows NT 4</option>
            <option value="w2k">Windows 2000</option>
            <option value="xp">Windows XP</option>
            <option value="vista">Windows Vista</option>
            <option value="win7" selected="selected">Windows 7</option>
            <option value="win8">Windows 8</option>
        </select>
    </div>
    <div>
        <label for="srv_pack">Service pack:</label>
        <select
            name="srv_pack"
            id="srv_pack">
        </select>
    </div>
    <input
        type="submit"
        value="Pošalji"
        onclick="return provjeri();" />
    . . .

```

Prvo se definira novo polje u kojemu će se nalaziti podaci o zakrpama (*service pack*) (na početku programa):

```
// Definiramo novo polje
var asSP = new Array(1);

// Napunimo polje, članovima koji su polje
asSP = [
    // NT 4
    ["(Nema);nema", "Service Pack 1;sp1", "Service Pack 2;sp2",
        "Service Pack 3;sp3", "Service Pack 4;sp4",
        "Service Pack 5;sp5", "Service Pack 6;sp6",
        "Service Pack 6a;sp6a"],
    // 2000
    ["(Nema);nema", "Service Pack 1;sp1", "Service Pack 2;sp2",
        "Service Pack 3;sp3", "Service Pack 4;sp4",
        "Service Pack 4 & Updates;sp4upd"],
    // XP
    ["(Nema);nema", "Service Pack 1;sp1", "Service Pack 2;sp2",
        "Service Pack 3;sp3"],
    // Vista
    ["(Nema);nema", "Service Pack 1;sp1", "Service Pack 2;sp2"],
    // Se7en
    ["(Nema);nema", "Service Pack 1;sp1"],
    // 8
    ["(Nema);nema", "8.1;81", "8.1 Update 1;81u1"]
];
```



**Funkcija `puniSelect` na osnovi indeksa odabranog operacijskog sustava puni drugi popis stavkama iz polja `asSP`:**

```
// Funkcija koja puni oderedjeni select element
// Argumenti:
//      oMenu:      objekt kojega punimo
//      asStavke:   niz iz kojega se puni
//      iStavka:    iz kojeg podniza se puni
function puniSelect(oMenu, asStavke, iStavka) {
    var iSeparator, iBrojac, sStavkaTekst, sStavkaVrijednost;

    // Postavimo duljinu niza na 0
    oMenu.options.length = 0;
    for (iBrojac = 0; iBrojac < asStavke[iStavka].length; iBrojac++) {
        // Provjerimo gdje je separator
        iSeparator = asStavke[iStavka][iBrojac].indexOf(';');
        if (iSeparator === -1) { // Nema ga
            sStavkaTekst = asStavke[iStavka][iBrojac];
            sStavkaVrijednost = sStavkaTekst;
        } else { // Separator postoji
            sStavkaTekst =
                asStavke[iStavka][iBrojac].substr(0, iSeparator);
            sStavkaVrijednost =
                asStavke[iStavka][iBrojac].substr(iSeparator + 1);
        }
        oMenu.options[iBrojac] =
            new Option(sStavkaTekst, sStavkaVrijednost);
    }
    oMenu.selectedIndex = oMenu.options.length - 1;
}
```

Pri dnu funkcije `puniSelect` izrađuju se nove stavke u izborniku uporabom objekta `Option`, čiji konstruktor ima dva argumenta:

1. tekst stavke
2. vrijednost koju poprima atribut `value`.



## 8. JavaScript biblioteka – jQuery

### 8.1. Općenito o JavaScript bibliotekama

*JavaScript* je dugo vremena bio samo dodatak HTML-u. *Google* je 2004. godine izradio *Gmail* u kojemu se koristila asinkrona komunikacija između preglednika i poslužitelja. Ista tehnologija upotrijebljena je godinu dana poslije za *Google Maps*. Ta je tehnologija izazvala malu revoluciju i pokrenula postupak koji je kasnije nazvan revolucija *Web 2.0*.

Navedena tehnologija dobila je naziv *AJAX – Asynchronous JavaScript and XML*.

Budući da je uporaba *AJAX*-a bila složena, počele su se pojavljivati gotove skripte koje su olakšavale programiranje. Takve su skripte bile jezgra za pojavu niza *JavaScript* biblioteka (*libraries*).

Među razlozima za pojavu *JavaScript* biblioteka bilo je i ujednačavanje načina rada s različitim preglednicima. Korištenjem biblioteke programer više ne mora brinuti o različitostima među pojedinim preglednicima.

Najpoznatije su:

- *jQuery* (<http://jquery.com/>)
- *Dojo Toolkit* (<http://dojotoolkit.org/>)
- *Prototype* (<http://prototypejs.org/>).

*jQuery* je danas gotovo standard, naročito jer je kompanija *Twitter* odlučila upotrijebiti upravo *jQuery* u svojoj besplatnoj distribuciji za standardizirani razvoj *web*-stranica pod nazivom *Twitter Bootstrap* (<http://getbootstrap.com/>).

### 8.2. jQuery

*jQuery* se sastoji od osnovne distribucije i od velikog broja dodataka (*plug-ins*).

Osnovna distribucija može se koristiti na dva načina:

- Klasično snimanje na lokalni disk *web*-sjedišta
- *CDN (Content Delivery Network)*.

Uporaba preko *CDN*-a ima dvije prednosti:

- Za smanjenje količine podataka koja putuje od poslužitelja do preglednika koristi se tzv. minimizirana inačica datoteke *JavaScript*. To znači da su iz datoteke (programa) uklonjene sve nevažne praznine, a ponekad se još obavi i skraćivanje naziva varijabli tako da se redom dodjeljuju nazivi oblika *a*, *b*, *c*, ..., *z*, odnosno nadalje *aa*, *ab*, *ac*, ..., *az*, *ba*, *bb*, *bc*, ...

- Uporabom CDN-a postoji velika vjerojatnost da je korisnik posjetio *web*-sjedište koje se koristi istom CDN-datotekom pa korisnikov preglednik ne dohvaća ponovo istu datoteku, nego se koristi postojećom u lokalnoj predmemoriji (*cache*) te se tako dodatno smanjuje količina podataka koja putuje od poslužitelja do preglednika.

Međutim, postoji i jedan nedostatak – minimizirani kôd teško je čitati pa ako postoji potreba za upoznavanjem s bibliotekom *jQuery*, preporuča se snimiti neminimiziranu inačicu.

*jQuery* se najčešće koristi tako da se za određene elemente dokumenta definiraju određene akcije.

U *jQuery*ju se elementi određuju CSS-selektorima, odnosno na isti način kao i u CSS-u. U HTML-dokumentu:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jQuery primjer 1</title>
</head>
<body>
  <h3>Naslov</h3>
  <ul>
    <li>stavka 1</li>
    <li>stavka 2</li>
    <li>stavka 3</li>
    <li>stavka 4</li>
    <li>stavka 5</li>
  </ul>
  <script
    type="text/javascript"
    src="http://cdn.jsdelivr.net/jquery/1.11.2/jquery.min.js">
  </script>
  <script>
    jQuery('li').on('click', function(){
      jQuery('li').css('color', 'green');
    });
  </script>
</body>
</html>
```

vidi se da se *jQuery* rabi pomoću funkcije *jQuery*.

Međutim, postoji i kraći način uporabe (tim se načinom koristi većina korisnika): znak `$` koji je alias za funkciju `jQuery`:

```
$('li').on('click', function(){
    $('#prviNaslov').css('color', 'green');
});
```

#### Napomena

Primjeri u datotekama:  
[jQuery/primjer2.html](#) i  
[jQuery/primjer3.html](#)

U prepravljenom primjeru pokazano je kako se identificiraju elementi pomoću atributa `id`. Ako je potrebno identificirati elemente kojima je atribut `class` postavljen na određenu vrijednost, to se postiže ovako:

```
$('li').on('click', function(){
    $('.naslov').css('color', 'green');
});
```

Da bi se dobio uvid u to kako izgleda rad s *jQuery*jem i njegovim dodacima, prepravit će se obrazac iz glavnog dijela tečaja.

### 8.3. Prerada obrazaca uz pomoć biblioteke *jQuery*

Za provjeru obrasca koristit će se dodatak *jQuery Validate*. Za ulančano povezivanje elemenata `select` koristit će se dodatak *Chained*.

Prvo treba učitati *jQuery* i potrebne dodatke:

```
<!DOCTYPE html>
<html lang="hr">

<head>
  <meta charset="UTF-8" />
  <title>Obrazac</title>
</head>
```

Tekstna polja, potvrdni okviri (*check button*), izborna dugmad (*radio button*) ne mijenjaju se. Elemente *select* treba prilagoditi (tj. upisati sve potrebne podatke) da bi dodatak *jQuery* mogao popunjavati drugi izbornik:

```
<div>
  <label for="os">Operacijski sustav:</label>
  <select
    name="os"
    id="os">
    <option value="nt4">Windows NT 4</option>
    <option value="w2k">Windows 2000</option>
    <option value="xp">Windows XP</option>
    <option value="vista">Windows Vista</option>
    <option value="win7" selected="selected">Windows 7</option>
    <option value="win8">Windows 8</option>
  </select>
</div>
```

Drugi se izbornik s prvim povezuje preko atributa `class`:

```
<div>
  <label for="srv_pack">Service pack:</label>
  <select
    name="srv_pack"
    id="srv_pack">
    <option value="nema" class="nt4">(Nema)</option>
    <option value="sp1" class="nt4">Service Pack 1</option>
    <option value="sp2" class="nt4">Service Pack 2</option>
    <option value="sp3" class="nt4">Service Pack 3</option>
    <option value="sp4" class="nt4">Service Pack 4</option>
    <option value="sp5" class="nt4">Service Pack 5</option>
    <option value="sp6" class="nt4">Service Pack 6</option>
    <option value="sp6a" class="nt4">Service Pack 6a</option>

    <option value="nema" class="w2k">(Nema)</option>
    <option value="sp1" class="w2k">Service Pack 1</option>
    <option value="sp2" class="w2k">Service Pack 2</option>
    <option value="sp3" class="w2k">Service Pack 3</option>
    <option value="sp4" class="w2k">Service Pack 4</option>
    <option value="sp4upd" class="w2k">Service Pack 4 &
      Updates</option>

    <option value="nema" class="xp">(Nema)</option>
    <option value="sp1" class="xp">Service Pack 1</option>
    <option value="sp2" class="xp">Service Pack 2</option>
    <option value="sp3" class="xp">Service Pack 3</option>

    <option value="nema" class="vista">(Nema)</option>
    <option value="sp1" class="vista">Service Pack 1</option>
    <option value="sp2" class="vista">Service Pack 2</option>

    <option value="nema" class="win7">(Nema)</option>
    <option value="sp1" class="win7"
      selected="selected">Service Pack 1</option>

    <option value="nema" class="win8">(Nema)</option>
    <option value="8.1" class="win8">8.1</option>
    <option value="8.1u1" class="win8">8.1 Update 1</option>
  </select>
</div>
```

Budući da se *jQuery*-kôd ne bi izvršio prije nego što su učitani svi HTML-elementi, program *JavaScript* piše se na kraju HTML-dokumenta (tj. učitava se iz datoteke):

```
. . .
<script type="text/javascript"
    src="http://cdn.jsdelivr.net/jquery/1.11.2/jquery.min.js"></script>
<script type="text/javascript"

src="http://cdn.jsdelivr.net/jquery.validation/1.13.1/jquery.validate.js">
</script>
<script type="text/javascript"
    src="http://cdn.jsdelivr.net/jquery.chained/0.9.9/jquery.chained.js">
</script>
<script type="text/javascript" src="obrazac.js"></script>
</body>
```

Obrada je povezanih elemenata *select* jednostavna:

```
$("#srv_pack").chained("#os");
$("#os").on('change', function () {
    $("#srv_pack option:last-child").attr("selected", "selected");
});
```

Provjera je li korisnik ispunio potrebna polja nešto je složenija, jer se trebaju navesti pravila, a da bi se zadržala funkcionalnost iz originalnog primjera, ispisuju se upisani podaci (ako su upisani svi potrebni podaci).

Općenita obrada podataka pomoću dodatka *jQuery Validate* ima oblik:

```
$("#obrazac1").validate({
    rules: {
        // pravila koja moraju zadovoljavati polja
    },
    messages: {
        // poruke koje se ispisuju za pojedinu pogrešku
    },
    submitHandler: function (form) {
        // popis naredbi koje se izvršavaju prije nego se pošalju podaci
        form.submit();
    }
});
```

Konkretno, za gornji su primjer pravila:

```
rules: {  
    ime: "required",  
    prezime: "required",  
    komp: "required"  
},
```

Poruke:

```
messages: {  
    ime: "Niste upisali ime!",  
    prezime: "Niste upisali prezime!",  
    komp: "Niste odabrali vrstu računala!"  
},
```

Budući da se dio obrasca koji u sebi ima izbornu dugmad (*radio button*) sastoji od većeg broja elemenata (odnosno od pojedine izborne dugmadi), nije dobro pogrešku ispisivati uz pojedino izbornu dugme. Stoga je potrebno promijeniti mjesto gdje će se ispisivati pogreška (iza elementa label):

```
errorPlacement: function (error, element) {  
    if (element.attr('type') === 'radio') {  
        error.insertAfter(  
            element.siblings('label')  
        );  
    } else {  
        error.insertAfter(element);  
    }  
},
```

Konačno se ispišu podaci koje je korisnik upisao (prije nego se podaci pošalju na odgovarajući URL):

```
submitHandler: function (form) {  
    var sMessage = 'Ime je: ' + $('#ime').val();  
    sMessage += '\nPrezime je: ' + $('#prezime').val();  
    sMessage += '\nRačunalo je: ' +  
        $('#obrazac1 input[name='komp']:checked').val();  
    sMessage += '\nPristup internetu je: '  
    sMessage += $('#internet').is(':checked') ?  
        'širokopojasni!' :  
        'modemom!';  
    sMessage += '\nOS: ' + $('#os>option:selected').text();  
    sMessage += ' (' + $('#os').val() + ')';  
    sMessage += '\nSP: ' + $('#srv_pack>option:selected').text();  
    sMessage += ' (' + $('#srv_pack').val() + ')';  
    window.alert(sMessage);  
    form.submit();  
}
```



## 9. Korisne skripte

### 9.1. Rollover

*Rollover* (okretanje) je u prošlosti jedan od najčešće korištenih vizualnih efekata kod kojih se rabio *JavaScript* (danas se češće rabi CSS). To je efekt kad se slika (obično je to slika s tekstom koja stoji u izborniku) promijeni u trenutku kad korisnik postavi pokazivač miša na tu sliku.

Primjer:

```

```

Napisana je funkcija koja postavlja argument `src` za određeni objekt (u našem slučaju je to `this`, tj. objekt na kojem se dogodio događaj).

```
function roll(oImg, sNewSrc) {
    oImg.src = sNewSrc;
}
```

Uporabom iste funkcije može se upravljati i drugim objektima. Na primjer, napravi se mali album:

```
<table border="0" align="center">
  <tr>
    <td colspan="3">
      
    </td>
  </tr>
  <tr>
    <td class="hot_spot"
      onmouseover="roll(document.velikaslika, 'slika_a.gif');"
      >Slika A</td>
    <td class="hot_spot"
      onmouseover="roll(document.velikaslika, 'slika_b.gif');"
      >Slika B</td>
    <td class="hot_spot"
      onmouseover="roll(document.velikaslika, 'slika_c.gif');"
      >Slika C</td>
  </tr>
</table>
```

#### Napomena

Događaji `mouseover` i `mouseout` objašnjeni su u dodatku E na kraju priručnika.

## 9.2. Preusmjerenje

Ponekad postoji potreba da se na osnovi određenih kriterija (npr. jezik u kojemu su stranice pisane ili vrsta preglednika kojom se korisnik služi) korisnici upute na druge stranice. U tu svrhu rabi se preusmjerenje pomoću objekta `location` i njegovog svojstva `href`:

```
<div>
  <input
    type="button"
    value="SRCE"
    onclick="location.href = 'http://www.srce.hr/';" />
</div>
<div>
  <input
    type="button"
    value="CARNet"
    onclick="location.href = 'http://www.carnet.hr/';" />
</div>
```

## 9.3. Provjera pomoću regularnih izraza

Prilikom unosa podataka korisnici slučajno unose nepravilne podatke. Bilo bi poželjno da se može provjeriti što su korisnici unijeli te da ih se može upozoriti ako su podaci neispravni. *JavaScript* od inačice 1.2 ima u sebi podršku za regularne izraze (*regular expressions*), koji omogućuju pisanje određenih pravila (izraza) i provjeru pridržava li se određeni niz znakova tih pravila. Srž je regularnih izraza suradnja dvaju dijelova: što želimo i koliko toga želimo. Prvi se naziva kvalifikator, a drugi kvantifikator.

Kvalifikator	Značenje
<znak>	Točno taj znak
[...]	Skup znakova unutar uglatih zagrada
[^...]	Komplement skupa znakova, tj. ne uključuje znakove unutar uglatih zagrada
.	Bilo koji znak osim kraja reda
\w	Bilo koje slovo <i>ASCII</i> , podcrta ili broj, isto kao [a-zA-Z0-9_]
\W	Bilo koji znak koji nije slovo <i>ASCII</i> , podcrta ili broj, isto kao [^a-zA-Z0-9_]
\s	Razmak
\S	Bilo koji znak koji nije razmak
\d	Broj, isto kao [0-9]
\D	Bilo koji znak koji nije broj, isto kao [^0-9]

Kvalifikator	Značenje
$\{n, m\}$	Prethodni kvalifikator mora se pojaviti najmanje $n$ puta, a najviše $m$ puta.
$\{n, \}$	Prethodni kvalifikator mora se pojaviti najmanje $n$ puta (ili više).
$\{n\}$	Prethodni kvalifikator mora se pojaviti točno $n$ puta.
$?$	Prethodni kvalifikator može se pojaviti jednom, a i ne mora; isto kao $\{0, 1\}$ .
$+$	Prethodni kvalifikator mora se pojaviti barem jednom; isto kao $\{1, \}$ .
$*$	Prethodni kvalifikator može se pojaviti bilo koliko puta, a i ne mora; isto kao $\{0, \}$ .

Ako je u proizvoljan izraz potrebno staviti nekoliko mogućnosti, tada se one grupiraju u okrugle zagrade i odvoje okomitom crtom. Na primjer, sljedeći izraz odgovara brojevima 099, 098, 091 i 092:

09(9|8|2|1)

Sljedeći izraz odgovara bilo kojem načinu pisanja riječi *JavaScript*:

`^[Jj](ava)?\s?[Ss]cript$`

Analiza izraza:

<code>^</code>	Znak za početak (retka ili niza znakova).
<code>[Jj]</code>	Na početku se može pojaviti bilo koji znak (samo jedan) iz skupa (ovdje J i j).
<code>(ava)?</code>	Niz znakova „ava” (grupirani okruglim zagradama); može se, ali i ne mora pojaviti.
<code>\s?</code>	Jedan razmak; može se pojaviti, ali i ne mora.
<code>[Ss]</code>	Može se pojaviti bilo koji znak (samo jedan) iz skupa (ovdje S i s).
<code>\$</code>	Znak za kraj (retka ili niza znakova).

Dakle, bilo koji od sljedećih nizova odgovara izrazu: „JavaScript”, „javascript”, „JScript”, „Java Script”, pa čak i „javaScript”.

Nedostaje funkcija koja će uneseni niz znakova usporediti s navedenim izrazom i vratiti informaciju odgovara li niz znakova izrazu. Najprije obrazac:

```
<form name="frm_regex" action="">
  <div>
    Ime i prezime
    <input
      type="text"
      name="ime"
      value=""
      onblur="test(this.value, this, '^[a-zA-ZčžćšđČŽŠĆĐ -]+$');" />
    </div>
    <div>
      Telefonski broj
      <input
        type="text"
        name="telefon"
        value=""
        onblur="test(this.value,
          this,
          '^(0([0-9]{1,2}) ( |\/))?[1-9][0-9]{2,3}\-[0-9]{3}$');" />
      </div>
      <input type="button" value="Provjeri" onClick="provjeri();" />
    </form>
```

U ovom obrascu dopušteno je da se za ime i prezime upišu samo slova (ASCII i slova s dijakritičkim znakovima), razmak i crtica – barem jednom (uočite znak + pred kraj izraza). Za telefonski broj može se unijeti predbroj i znak za razdvajanje (uočite ? iza grupirajućih zagrada). Predbroj počinje znakom 0 nakon kojega slijedi 1 ili dvije znamenke. Znak za razdvajanje može biti razmak ili kosa crta (ovdje osigurana s obrnutom kosom crtom, jer kosa crta označava početak i kraj regularnog izraza). Nakon toga slijedi jedan broj od 1 do 9 (prva nula nema smisla) te nakon njega dva ili tri broja koje može, ali ne mora, slijediti crtica za razdvajanje. Na kraju slijede točno tri broja. Ovdje se nalazi program koji provjerava izraze (program uz provjeru izraza i vraća fokus na polje koje nije ispravno, jer se hvata događaj `blur` koji se okida u trenutku napuštanja polja):

```
function test(sTekst, oPolje, sRegExp) {
    var re = new RegExp(sRegExp); // regular expression

    if ((sTekst !== '') && (re.test(sTekst) === false)) {
        window.alert('Unešena vrijednost nije pravilnog oblika!');
        // Ovo je hack jer focus ne radi
        // po standardu u Mozilli/Firefoxu
        setTimeout(function () {
            oPolje.focus();
        }, 10);
    }
}
```

Metoda `test` objekta `RegExp` provjerava zadovoljava li niz znakova, proslijeđen kao argument, zadani regularni izraz.

Program se doradi tako da se doda argument koji predstavlja poruku koja će se prikazati prilikom pogreške te se korisniku pomogne tako da se iz njegova neispravnog niza znakova uklone znakovi koji tu ne smiju biti (no, to još uvijek ne znači da je niz pravilnog oblika):

```
function test(sTekst, oPolje, sRegExp, sZadrzi, sPoruka) {
    var sReFind = new RegExp(sRegExp),
        sReKeep = new RegExp(sZadrzi),
        sZnak,
        i,
        sIzlaz = sTekst;

    if ((sTekst !== '') && (sReFind.test(sTekst) === false)) {
        window.alert(sPoruka);
        sIzlaz = '';
        for (i = 0; i < sTekst.length; i++) {
            sZnak = sTekst.charAt(i);
            if (sReKeep.test(sZnak)) {
                sIzlaz += sZnak;
            }
        }
        // Ovo je hack jer focus ne radi
        // po standardu u Mozilli/Firefoxu
        setTimeout(function () {
            oPolje.focus();
        }, 10);
    }
    return (sIzlaz);
}
```

## Odgovarajući obrazac:

```
<form name="frm_regex" action="">
  <div>
    Ime i prezime
    <input type="text" name="ime" value=""
      onblur="this.value = test(
        this.value,
        this,
        '^[a-zA-ZčžćšđČŽŠĆĐ -]+$ ',
        '^[a-zA-ZčžćšđČŽŠĆĐ -]$ ',
        'Ime i prezime su nepravilnog oblika!\nDozvoljena su samo slova!');" />
  </div>
  <div>
    Telefonski broj
    <input type="text" name="telefon" value=""
      onblur="this.value = test(
        this.value,
        this,
        '^(0([0-9]{1,2})( |\/))?[1-9][0-9]{2,3}\-?[0-9]{3}$ ',
        '[0-9 -\/ ]',
        'Tel. broj je nepravilnog formata (0XX/YYYY-YYY)!');" />
  </div>
  <input type="button" value="Provjeri" onclick="provjeri();" />
</form>
```

## 9.4. Upravljanje preglednikom

Objekt `window` omogućuje rukovanje prozorom preglednika. Rukovanje trenutačnim prozorom preglednika često su zlorabili oglašivači pa je u današnjim preglednicima moguće isključiti odziv na mijenjanje izgleda prozora preglednika. Zato je u primjeru prvo potrebno otvoriti novi prozor da bi se mogla mijenjati njegova svojstva:

```
<form name="frm_win" action="">
  <input type="button" value="Novi prozor" onclick="novi();" />
  <br />
  <input type="text" name="wd" value="800" size="3" maxlength="3" />
  <input type="text" name="ht" value="600" size="3" maxlength="3" />
  <input type="button" value="Veličina" onclick="velicina();" />
  <br />
  <input type="text" name="wd_by" value="-20" size="3" maxlength="3" />
  <input type="text" name="ht_by" value="-40" size="3" maxlength="3" />
  <input type="button" value="Promijeni veličinu za" onclick="promijeniZa();" />
  <br />
  <input type="text" name="x" value="300" size="3" maxlength="3" />
  <input type="text" name="y" value="300" size="3" maxlength="3" />
  <input type="button" value="Pomakni" onclick="pomakni();" />
  <br />
</form>
```

Pomoću ovih se funkcija *JavaScripta* upravlja stanjem preglednika:

```
var oProzor;  
  
function novi() {  
    oProzor = window.open("", "", "width=250, height=250");  
    oProzor.moveTo(300, 400);  
}  
  
function velicina() {  
    oProzor.resizeTo(document.frm_win.wd.value,  
        document.frm_win.ht.value);  
    oProzor.focus();  
}  
  
function promijeniZa() {  
    oProzor.resizeBy(document.frm_win.wd_by.value,  
        document.frm_win.ht_by.value);  
    oProzor.focus();  
}  
  
function pomakni() {  
    oProzor.moveTo(document.frm_win.x.value,  
        document.frm_win.y.value);  
    oProzor.focus();  
}
```

Često je potrebno kakav sadržaj (najčešće hijerarhijski organiziran sadržaj) prikazati u novom prozoru. To se postiže uporabom funkcije `window.open` koja ima općeniti oblik:

```
var oWin = window.open("<URL>", "<naziv_prozora>", "<parametri>");
```

Treći argument je niz znakova u kojem su zapisani parametri kao parovi ključa i vrijednosti (`width=100, height=50`). Od parametara su najzanimljiviji:

- `top, left, height, width`: y koordinata gornjeg ruba, x koordinata lijevog ruba, visina, širina
- `menubar, toolbar, location, scrollbars, status, resizable`: izbornik, alatna traka, adresno polje, trake za pomak, statusno područje; prozoru se može mijenjati veličina.



Sljedećim obrascem kontrolira se izrada novog prozora:

```
<form name="frm_win" action="">
  Novi prozor:
  <br/>
  <input type="checkbox" name="menubar" />
    Prikaži izbornik (Menubar)<br />
  <input type="checkbox" name="location" />
    Prikaži adresno polje (Location)<br />
  <input type="checkbox" name="resizable" />
    Dozvoli promjenu veličine (Resizable)<br />
  <input type="checkbox" name="scrollbars" />
    Prikaži trake za pomak (Scrollbars)<br />
  <input type="checkbox" name="status" />
    Prikaži statusno područje (Status)<br />
  <input type="checkbox" name="toolbar" />
    Prikaži alatnu traku (Toolbar)<br/>
  Širina: <input type="text" name="wd_new"
    value="" size="3" maxlength="3" />
  <br />
  Visina: <input type="text" name="ht_new"
    value="" size="3" maxlength="3" />
  <br/>
  <input type="button" value="Novi prozor"
    onclick="newWin();" />
</form>
```

Ovim se programom *JavaScripta* izrađuje novi prozor:

```
function newWin() {  
    var sOptions = 'menubar=';  
    sOptions += document.frm_win.menubar.checked ? '1' : '0';  
    sOptions += ',location=';  
    sOptions += document.frm_win.location.checked ? '1' : '0';  
    sOptions += ',resizable=';  
    sOptions += document.frm_win.resizable.checked ? '1' : '0';  
    sOptions += ',scrollbars=';  
    sOptions += document.frm_win.scrollbars.checked ? '1' : '0';  
    sOptions += ',status=';  
    sOptions += document.frm_win.status.checked ? '1' : '0';  
    sOptions += ',toolbar=';  
    sOptions += document.frm_win.toolbar.checked ? '1' : '0';  
    if (document.frm_win.wd_new.value !== '') {  
        sOptions += ',width=' + document.frm_win.wd_new.value;  
    }  
    if (document.frm_win.ht_new.value !== '') {  
        sOptions += ',height=' + document.frm_win.ht_new.value;  
    }  
    window.open("", "new_win", sOptions);  
}
```

## 10. Zadaci

### Pogađanje brojeva

- Napišite program koji će naći slučajan broj (funkcijom `Math.random()` koja vraća slučajan broj od 0.0 do 1.0) do 100. Ispišite dobiveni broj.
- Napišite program koji će dopustiti korisniku unos broja od 0 do 100 u jedno tekstno polje. Program mora pamtit koji je to unos po redu te ispisivati sve prije unesene brojeve.
- Združite programe pod A i B te napišite novi program koji korisniku dopušta da pogodi koji je slučajan broj program našao. Korisnik mora u pregledniku vidjeti koji mu je to pokušaj, koji broj je unio te sve svoje prijašnje pokušaje. Najveći broj pokušaja je 7, nakon toga se ispiše slučajan broj.

### Odabir većeg broja vrijednosti

- Napišite program koji će na kraj elementa `<select>` dodavati vrijednosti iz tekstnog polja.
- Napišite program koji će na kraj elementa `<select>` dodavati vrijednosti iz tekstnog polja. Omogućite i brisanje odabrane stavke iz popisa.
- Napišite program (rabeći postupke iz A i B) koji će prebacivati vrijednosti iz jedne liste u drugu.

# 11. Dodatak

## A. Operatori

U ovoj tablici stupac označen s „P” je prioritet operatora, a stupac „A” je asocijativnost operatora (koja može biti lijeva ili desna).

P	A	Operator	Tip operanda	Opis
15	L	.	objekt, varijabla	svojstvo ili metoda
15	L	[ ]	polje, cijeli broj	indeks polja
15	L	( )	funkcija, argumenti	poziv funkcije
14	D	++	broj ili varijabla	pre- ili post-inkrement (unaran)
14	D	--	broj ili varijabla	pre- ili post-dekrement (unaran)
14	D	-	broj	unarni minus (negacija)
14	D	+	broj	unarni plus (nema-operacije)
14	D	~	cijeli broj	bit komplement (unaran)
14	D	!	boolean	logički komplement (unaran)
14	D	typeof	bilo koji	tip podatka (unaran)
14	D	void	bilo koji	nedefinirana vrijednost (unaran)
13	L	*, /, %	broj	množenje, dijeljenje, ostatak
12	L	+, -	broj	zbrajanje, oduzimanje
12	L	+	niz znakova	spajanje niza znakova
11	L	<<	cijeli broj	pomak u lijevo
11	L	>>	cijeli broj	pomak u desno (puni se 1)
10	L	>>>	cijeli broj	pomak u desno (puni se 0)
10	L	<, <=	broj ili niz znakova	manje i manje ili jednako
10	L	>, >=	broj ili niz znakova	veće i veće ili jednako
10	L	in	niz znakova, objekt	provjera da li postoji svojstvo
9	L	==	bilo koji	test jednakosti
9	L	!=	bilo koji	test nejednakosti
9	L	===	bilo koji	test identičnosti
9	L	!==	bilo koji	test neidentičnosti
8	L	&	cijeli broj	bit AND
7	L	^	cijeli broj	bit XOR
6	L		cijeli broj	bit OR
5	L	&&	boolean	logički AND
4	L		boolean	logički OR
3	D	?:	boolean, bilo koji, bilo koji	uvjetni operator (3 operanda)
2	D	=	broj ili varijabla, bilo koji	pridruživanje
2	D	*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=,  =	broj ili varijabla, bilo koji	pridruživanje s operacijom
1	L	,	bilo koji	višestruki izračun izraza

## B. Polja

Pokazano je da se polja stvaraju preko objekta `Array`. Ovdje je navedeno koja svojstva ima polje i koje sve metode postoje za rukovanje poljima.

Najvažnije svojstvo polja je duljina polja. Duljina polja je broj za jedan veći od najvećeg indeksa u polju. Budući da su polja u *JavaScriptu* dinamička, duljina se osvježi svakog puta kad se promijeni polje:

```
var a = new Array();    // a.length == 0
                        // (nema elemenata)
a = new Array(10);     // a.length == 10
                        // (postoje prazni elementi 0-9)
a = new Array(1,2,3);  // a.length == 3
                        // (postoje elementi 0-2)
a = [4, 5];            // a.length == 2
                        // (postoje elementi 0 i 1)
a[5] = -1;             // a.length == 6
                        // (postoje elementi 0, 1, i 5)
a[49] = 0;             // a.length == 50
                        // (postoje elementi 0, 1, 5, i 49)
```

Tipična primjena bila bi ovakva:

```
var aVoce = ["mango", "banana", "jabuka", "kruška"];
for(var i = 0; i < aVoce.length; i++){
    alert(aVoce[i]);
}
```

Metode su za rukovanje poljima:

Metoda	Primjer	Značenje
<code>join</code>	<code>a.join("&lt;separator&gt;")</code>	Metoda koja spaja sve elemente polja u jedan niz znakova odvojenih nizom znakova <code>&lt;separator&gt;</code> .
<code>reverse</code>	<code>a.reverse()</code>	Metoda koja razmješta elemente polja obratno, tj. prvi element postaje zadnji, zadnji element prvi.
<code>sort</code>	<code>a.sort(&lt;funkcija&gt;)</code>	Razvrstava polje po redoslijedu <i>ASCII</i> . Ako je potreban drugi redoslijed, kao argument se navodi ime funkcije koja će usporediti vrijednosti.
<code>concat</code>	<code>a.concat(&lt;polje&gt;)</code>	Spaja polje <code>a</code> i polje koje je navedeno kao argument.
<code>slice</code>	<code>a.slice(&lt;početak&gt;, &lt;kraj&gt;)</code>	Vraća elemente polja od indeksa <code>&lt;početak&gt;</code> do indeksa <code>&lt;kraj&gt;</code> kao novo polje. Kad je indeks negativan, indeks se računa od kraja polja.
<code>push</code>	<code>a.push(&lt;polje&gt;)</code>	Dodaje na kraj polja elemente u polju <code>&lt;polje&gt;</code> .
<code>pop</code>	<code>a.pop()</code>	Briše zadnji element iz polja i vraća njegovu vrijednost.
<code>shift</code>	<code>a.shift()</code>	Briše prvi element iz polja i vraća njegovu vrijednost.
<code>unshift</code>	<code>a.unshift(&lt;polje&gt;)</code>	Dodaje na početak polja elemente u polju <code>&lt;polje&gt;</code> .

## C. Datumi

Rukovanje datumima vrši se preko objekta `Date`. Novi se objekt radi ovako:

```
sada = new Date();
novaGodina = new Date(2005, 0, 1);
// 0 - siječanj, ..., 11 - prosinac
```

Operacije na datumima obavljaju se kao i na brojevima, jer su interno datum i vrijeme broj milisekundi proteklih od 1. siječnja 1970.

```
danas = new Date();
bozic = new Date(); // Novi datum s trenutnom godinom
bozic.setMonth(11); // Postavi mjesec na prosinac
bozic.setDate(25); // Postavi dan na 25

if (danas.getTime() < bozic.getTime()) {
    razlika = bozic.getTime() - danas.getTime();
    razlika = Math.floor(razlika / (1000 * 60 * 60 * 24));
    // milisekunde * sekunde * minute * sati = dana
    alert('Do Božića je ostalo još ' + razlika + ' dana!');
}
```

Neke metode za rukovanje datumima i vremenom:

Metoda	Primjer	Značenje
<code>getTime</code>	<code>d.getTime()</code>	Vraća broj proteklih milisekundi od 1.1.1970.
<code>getSeconds</code>	<code>d.getSeconds()</code>	Vraća broj sekundi (0-59) od određenog datuma.
<code>getMinutes</code>	<code>d.getMinutes()</code>	Vraća broj minuta (0-59) od određenog datuma.
<code>getHours</code>	<code>d.getHours()</code>	Vraća broj sati (0-23) od određenog datuma.
<code>getDate</code>	<code>d.getDate()</code>	Vraća dan u mjesecu (1-31) od određenog datuma.
<code>getDay</code>	<code>d.getDay()</code>	Vraća dan u tjednu (0:nedjelja – 6:subota) od određenog datuma.
<code>getMonth</code>	<code>d.getMonth()</code>	Vraća mjesec u godini (0:siječanj-11:prosinač) od određenog datuma.
<code>getFullYear</code>	<code>d.getFullYear()</code>	Vraća godinu od određenog datuma.
<code>setSeconds</code>	<code>d.setSeconds()</code>	Postavlja broj sekundi za određeni datum.
<code>setMinutes</code>	<code>d.setMinutes()</code>	Postavlja broj minuta za određeni datum.
<code>setHours</code>	<code>d.setHours()</code>	Postavlja broj sati za određeni datum.
<code>setDate</code>	<code>d.setDate()</code>	Postavlja dan u mjesecu za određeni datum.
<code>setDay</code>	<code>d.setDay()</code>	Postavlja dan u tjednu za određeni datum.
<code>setMonth</code>	<code>d.setMonth()</code>	Postavlja mjesec u godini za određeni datum.
<code>setFullYear</code>	<code>d.setFullYear()</code>	Postavlja godinu za određeni datum.
<code>toString</code>	<code>d.toString()</code>	Vraća niz znakova koji predstavlja određeni datum.

## D. Matematičke funkcije

Objekt `Math` definira nekoliko konstanti i funkcija koje su potrebne za složenije matematičke operacije.

Konstante:

Konstanta	Značenje
<code>Math.E</code>	Baza prirodnog logaritma ( $e$ ).
<code>Math.LN10</code>	Prirodni logaritam od 10.
<code>Math.LN2</code>	Prirodni logaritam od 2.
<code>Math.LOG10E</code>	Logaritam s bazom 10 od $e$ .
<code>Math.LOG2E</code>	Logaritam s bazom 2 od $e$ .
<code>Math.PI</code>	Konstanta $\pi$
<code>Math.SQRT1_2</code>	Konstanta kvadratni korijen od $1/2$ .
<code>Math.SQRT2</code>	Konstanta kvadratni korijen od 2.

Funkcije:

Metoda	Primjer	Značenje
<code>abs</code>	<code>Math.abs(n)</code>	Vraća apsolutnu vrijednost broja.
<code>acos</code>	<code>Math.acos(n)</code>	Vraća arkus kosinus broja.
<code>asin</code>	<code>Math.asin(n)</code>	Vraća arkus sinus broja.
<code>atan</code>	<code>Math.atan(n)</code>	Vraća arkus tangens broja.
<code>atan2</code>	<code>Math.atan2(x, y)</code>	Vraća arkus tangens dva broja (kut koji tvore $x$ i $y$ ).
<code>ceil</code>	<code>Math.ceil(n)</code>	Vraća najbliži cijeli broj koji je veći ili jednak $n$ .
<code>cos</code>	<code>Math.cos(n)</code>	Vraća kosinus broja.
<code>exp</code>	<code>Math.exp(n)</code>	Vraća $e^n$ .
<code>floor</code>	<code>Math.floor(n)</code>	Vraća najbliži cijeli broj koji je manji ili jednak $n$ .
<code>log</code>	<code>Math.log(n)</code>	Vraća prirodni logaritam broja ( $\ln n$ ).
<code>max</code>	<code>Math.max(a, b)</code>	Vraća veći broj od dva.
<code>min</code>	<code>Math.min(a, b)</code>	Vraća manji broj od dva.
<code>pow</code>	<code>Math.pow(x, y)</code>	Vraća $x^y$ .
<code>random</code>	<code>Math.random(n)</code>	Vraća slučajan broj od 0 do 1.
<code>round</code>	<code>Math.round(n)</code>	Zaokružuje broj na najbliži cijeli broj.
<code>sin</code>	<code>Math.sin(n)</code>	Vraća sinus broja.
<code>sqrt</code>	<code>Math.sqrt(n)</code>	Vraća kvadratni korijen broja.
<code>tan</code>	<code>Math.tan(n)</code>	Vraća tangens broja.

## E. Događaji

Važniji događaji definirani u DOM-u:

Događaj	DOM svojstvo	Pokreće se
abort	onabort	Prekinuto učitavanje slike.
blur	onblur	Element gubi fokus za unos.
change	onchange	Izbor u <code>&lt;select&gt;</code> elementu ili drugim elementima gubi fokus i vrijednost mu se promijeni od kada je dobio fokus.
click	onclick	Klik mišem na element.
dblclick	ondblclick	Dvostruki klik mišem na element.
error	onerror	Došlo je do pogreške prilikom učitavanja slike.
focus	onfocus	Element je dobio fokus za unos.
keydown	onkeydown	Pritisnuta tipka.
keypress	onkeypress	Pritisnuta i otpuštena tipka.
keyup	onkeyup	Tipka je otpuštena.
load	onload	Učitavanje dokumenta je završeno.
mousedown	onmousedown	Tipka miša je pritisnuta (nije još otpuštena).
mousemove	onmousemove	Miš je pomaknut.
mouseout	onmouseout	Miš je pomaknut izvan elementa.
mouseover	onmouseover	Miš je pomaknut na element.
mouseup	onmouseup	Tipka miša je otpuštena.
reset	onreset	Obrazac je postavljena na početne vrijednosti.
resize	onresize	Veličina prozora preglednika se promijenila.
select	onselect	Tekst je odabran (selektiran, zacrnjen).
submit	onsubmit	Obrazac je poslan na obradu.
unload	onunload	Na mjesto trenutnog dokumenta učitava se novi.



## F. Niz znakova

Nizovi znakova usko su vezani uz objekt `String`, iako se rijetko stvaraju eksplicitnim korištenjem objekta. Za objekt `String` definiran je velik broj metoda. Tu opisane najčešće korištene metode:

Metoda	Primjer	Značenje
<code>charAt</code>	<code>sIme.charAt(n)</code>	Vraća znak koji se nalazi na poziciji <code>n</code> .
<code>charCodeAt</code>	<code>sIme.charCodeAt(n)</code>	Vraća <i>Unicode</i> kôd znaka koji se nalazi na poziciji <code>n</code> .
<code>concat</code>	<code>sIme.concat(sPrezime, sAdresa, . . .)</code>	Spaja niz znakova s nizovima znakova koji su proslijeđeni.
<code>indexOf</code>	<code>sIme.indexOf(znak)</code>	Vraća indeks na kojemu se nalazi prvo pojavljivanje znaka <code>znak</code> .
<code>lastIndexOf</code>	<code>sIme.lastIndexOf(znak)</code>	Vraća indeks na kojemu se nalazi zadnje pojavljivanje znaka <code>znak</code> .
<code>split</code>	<code>sIme.split(delimiter)</code>	Dijeli niz znakova na mjestima na kojima se nalazi <code>delimiter</code> i vraća polje kao rezultat.
<code>substr</code>	<code>sIme.substr(indeks, duljina)</code>	Vraća podniz od pozicije <code>indeks</code> u duljini <code>duljina</code> .
<code>substring</code>	<code>sIme.substring(indeksA, indeksB)</code>	Vraća podniz od pozicije <code>indeksA</code> do pozicije <code>indeksB</code> .
<code>toLowerCase</code>	<code>sIme.toLowerCase()</code>	Vraća niz znakova u kojemu su sva slova pretvorena u mala.
<code>toUpperCase</code>	<code>sIme.toUpperCase()</code>	Vraća niz znakova u kojemu su sva slova pretvorena u velika.

Niz znakova ima i svojstvo `length` u koje je pohranjena trenutna duljina niza znakova.

## G. Aplikacija za uređivanje teksta – *Brackets*

Preporučena aplikacija za uređivanje HTML, CSS i JavaScripta datoteka je *Brackets* - <http://brackets.io/>.

Preporučeni dodaci za *Brackets* koji ubrzavaju pisanje i razvoj:

- *JSHint*
- *QuickDocsJS*
- *Emmet*
- *Brackets File Icons*
- *jsbeautifier*
- *CDN Finder*
- *Minifier*
- *Various improvements*

Napomena: *Various improvements* instalirati zadnji.