POGLAVLJE

3

Korišćenje operatora, naredbi i funkcija

U OVOM POGLAVLJU NASTAVLJAMO PREDSTAVLJANJE JAVASCRIPT JEZIKA. OBRADIĆETE SVE OPERATORE KOJE JAVASCRIPT PODRŽAVA I NAUČIĆETE KAKO SE IZRAZI IZRAČUNAVAJU. NAUČIĆETE DA KORISTITE JAVASCRIPT PROGRAMSKE NAREDBE, DA PIŠETE PROSTE SKRIPTOVE KOJI DEMONSTRIRAJU UPOTREBU SVAKE OD TIH NAREDBI I DA KREIRATE I POZIVATE FUNKCIJE. KADA ZAVRŠITE OVO POGLAVLJE MOĆI ĆETE DA NAPIŠETE JAVASCRIPT SKRIPTOVE KOJI KORISTE OPERATORE I NAREDBE JAVASCRIPTA ZA IZVRŠAVANJE RAZLIČITIH VRSTA IZRAČUNAVANJA.

Operatori i izrazi

U prethodnom poglavlju koristili ste neke od osnovnih operatora koje obezbeđuje JavaScript. Ovo uključuje korišćenje + operatora sa stringovima i numeričkim tipovima i operator dodele vrednosti = . U ovom odeljku biće predstavljeni svi operatori koje JavaScript podžava, koji su organizovani po sledećim kategorijama:

- aritmetički operatori
- logički operatori
- operatori poređenja
- operatori stringova
- operatori za manipulaciju bitovima
- operatori dodele
- uslovni operatori.

Započnimo razjašnjavanje terminologije. Operator se koristi za transformisanje jedne, ili više vrednosti u jednu rezultujuću vrednost. Vrednosti na koje se operator primenjuje nazivaju se operandi. Kombinacija operatora i njegovih operanada poznata je kao izraz.

Izrazi se izračunavaju da bi se odredila sama vrednost izraza, koja predstavlja rezultat primene operatora nad operandima. Neki operatori, kao, na primer, operator (dodele) = , kao ishod daju vrednost koja se dodeljuje promenljivoj. Drugi proizvode vrednost koja se može koristiti u drugim izrazima.

NAPOMENA

Za neke operatore, kao što je * operator množenja, redosled operanada nije važan - na primer, `x * y = y * x` predstavlja tačan izraz za sve cele i realne brojeve. Drugi operatori, kao što je + (konkatenacija stringova) operator, daju različite rezultate za različite redoslede svojih operanada. Na primer, `"ab"+"cd"` nije isto što i `"cd"+"ab"`. ■

Unarni operatori se koriste samo sa jednim operandom. Na primer, unarni operator ! se primenjuje na logičku vrednost i vraća logičku not vrednost te vrednosti. Većina JavaScript operatora predstavlja *binarne* operatore, koji imaju po dva operanda. Jedan primer binarnog operatora je operator * (množenja), koji se koristi za izračunavanje proizvoda dva broja. Na primer, izraz 7 * 6 se izračunava kao 42 primenom * operatora nad operandima 7 i 6.

Do sada smo se bavili samo prostim izrazima. Složeniji izrazi mogu da se naprave kombinovanjem prostih unarnih i binarnih izraza. Da biste izračunali složene izraze, morate da ih rasčlanite na njihove komponente unarnih i binarnih izraza, primenjujući pravila redosleda, ili prioriteta (na primer, izračunavanjem grupa pre njihovog sabiranja, ili množenja). Naučićete više o rasčlanjavanju izraza u odeljku "Prioritet operatora".

NAPOMENA

JavaScript podržava i regularne izraze preko RegExp objekta. Naučićete kako da koristite ovaj objekat u online dodatku C, "Regularni izrazi". ■

Aritmetički operatori

Aritmetičke operatore svakodnevno koristimo za izvršavanje jednostavnih matematičkih izračunavanja. Matematički operatori koje podržava JavaScript se nalaze u tabeli 3.1.

Tabela 3.1 Aritmetički operatori

Operator	Opis
+	sabiranje
-	oduzimanje, ili unarna negacija
*	množenje
/	deljenje
%	moduo
++	Inkrementiraj i vrati vrednost (ili vrati vrednost, pa inkrementiraj)
--	Dekrementiraj i vrati vrednost (ili vrati vrednost, pa dekrementiraj)

NAPOMENA

Operator % (moduo) izračunava ostatak deljenja dva cela broja. Na primer, 17 % 3 = 2, zato što je 17/3 = 5 sa ostatkom 2. ■

Logički operatori

Logički operatori se koriste za izvođenje logičkih (Boolean) operacija nad logičkim operandima, kao što su logičko I, logičko ILI i logička negacija. Logički operatori koje podržava JavaScript se nalaze u tabeli 3.2.

Tabela 3.2 Logički operatori

Operator	Opis
&&	logičko I
||	logičko ILI
!	logička negacija

Operatori poređenja

Operatori poređenja se koriste za određivanje kada su dve vrednosti jednake, ili za poređenje numeričkih vrednosti, da bi se ustanovilo koja je vrednost veća od one druge. Operatori poređenja koje podržava JavaScript nalaze se u tabeli 3.3.

Tabela 3.3 Operatori poređenja

Operator	Opis
==	jednako
===	strogo jednako
!=	nije jednako
!==	strogo nije jednako
<	manje
<=	manje, ili jednako
>	veće
>=	veće, ili jednako

NOVO

Operatori jednako (==) i nije jednako (!=) vrše konverziju tipa pre provere jednakosti. Na primer, "5" == 5 se izračunava kao true. Operatori strogo jednako (===) i strogo nije jednako (!==) ne vrše konverziju tipa pre provere jednakosti. Na primer, "5" === 5 se izračunava kao false, a "5" !== 5 vraća true. Operatori strogo jednako (===) i strogo nije jednako (!==) su deo ECMAScript 1 standarda. Oni su uvedeni u Navigatoru samo za JavaScript 1.3 i podržali su ih Navigator 4.06 i kasniji. Takođe su ih podržali Internet Explorer 4 i kasniji. ■

UPOZORENJE

Ako je LANGUAGE atribut <SCRIPT> taga postavljen na "JavaScript1.2", Navigator 4 (i kasniji) tretiraju operator jednakosti (==) kao operator stroge jednakosti. Na primer, "5" == 5 se izračunava kao false. Ovo predstavlja grešku Navigatora. ■

String operatori

String operatori se koriste za izvođenje operacija nad stringovima. JavaScript trenutno podržava samo operator string konkatenacije +. On se koristi za spajanje dva stringa nadovezivanjem. Na primer, "ab" + "cd" proizvodi "abcd."

Operatori za manipulaciju bitovima

Operatori za manipulaciju bitovima sprovode operacije nad vrednošću predstavljenu bitovima, kao što su pomeranje bitova ulevo, ili udesno. Operatori za manipulaciju bitovima koje podržava JavaScript nalaze se u tabeli 3.4.

Tabela 3.4 Operatori za manipulaciju bitovima

Operator	Opis
&	I
|	ILI
^	ekskluzivno ILI

<<	pomeranje ulevo
>>	pomeranje udesno sa čuvanjem znaka
>>>	pomeranje udesno sa punjenjem mesta najveće težine nulama

Operatori dodele

Operatori dodele se koriste za ažuriranje vrednosti promenljive. Neki operatori dodele se kombinuju sa drugim operatorima, da bi sproveli proračun nad vrednošću koja se nalazi u promenljivoj i da, potom, ažuriraju promenljivu novom vrednošću. Operatori dodele koje podržava JavaScript nalaze se u tabeli 3.5.

Tabela 3.5 Operatori dodele

Operator	Opis
=	Postavlja promenljivu sa leve strane operatora = na vrednost izraza sa njegove desne strane.
+=	Uvećava promenljivu sa leve strane operatora += za vrednost izraza sa njegove desne strane. Kada se koristi sa stringovima vrednost sa desne strane se nadovezuje na vrednost promenljive sa leve strane operatora +=.
-=	Umanjuje promenljivu sa leve strane operatora -= za vrednost izraza sa njegove desne strane.
*=	Množi promenljivu sa leve strane operatora *= sa vrednošću izraza sa njegove desne strane.
/=	Deli promenljivu sa leve strane operatora /= sa vrednošću izraza sa njegove desne strane.
%=	Izdvaja moduo promenljive sa leve strane operatora %=, koristeći izraz sa njegove desne strane.
<<=	Pomera ulevo promenljivu sa leve strane operatora <<=, upotrebljavajući vrednost izraza sa njegove desne strane.
>>=	Sprovodi pomeranje udesno sa zadržavanjem znaka promenljive sa leve strane operatora >>=, koristeći vrednost izraza sa njegove desne strane.
>>>=	Sprovodi pomeranje udesno sa punjenjavanjem mesta najveće težine nulama promenljive sa leve strane operatora >>>=, koristeći vrednost izraza sa njegove desne strane.
&=	Sprovodi logičko I nad bitovima promenljive sa leve strane operatora &=, koristeći vrednost izraza sa njegove desne strane.
|=	Sprovodi logičko ILI nad bitovima promenljive sa leve strane operatora |=, koristeći vrednost izraza sa njegove desne strane.
^=	Sprovodi ekskluzivno ILI nad bitovima promenljive sa leve strane operatora ^=, koristeći vrednost izraza sa njegove desne strane.

Ternarni operator uslovnog izraza

JavaScript podržava operator uslovnog izraza ? :, koji se može naći i u Javi, C-u i C++-u. Ovo je *ternarni* operator, budući da prihvata tri operanda - uslov koji treba izračunati i dve alternativne vrednosti, od kojih treba vratiti samo jednu, u zavisnosti da li je uslov ispunjen, ili ne. Format za ovaj uslovni izraz je sledeći:

```
uslov ? vrednost1 : vrednost2
```

NAPOMENA

Uslov je izraz koji kao rezultat daje logičku vrednost - na primer, tačno, ili netačno. ■

Ako je uslov *tačno*, *vrednost1* je rezultat ovog izraza. U suprotnom, *rezultat* je *vrednost2*. Primer upotrebe ovog izraza je sledeći:

```
(x > y) ? 5 : 7
```

Ako je vrednost promenljive `x` veća od vrednosti koja se nalazi u promenljivoj `y`, rezultat izraza je 5. Ako je vrednost x manja od vrednosti `y`, ili su njih dve jednake, rezultat ovog izraza je 7.

Specijalni operatori

JavaScript podržava i brojne specijalne operatore, koji se ne uklapaju ni u jednu od kategorija operatora navedenih u prethodnim odeljcima:

ZAREZ (,) OPERATOR Ovaj operator izračunava dva izraza i vraća vrednost drugog. Razmotrite naredbu a = (5+6) , (2*20). Oba izraza (5+6) i (2*20) se izračunavaju, a vrednost drugog izraza (40) se dodeljuje promenljivoj a.

DELETE OPERATOR Delete operator se koristi za brisanje osobine nekog objekta,ili elemenata niza do navedenog indeksa. Na primer, delete `myArray[5]` briše šest elemenata `myArray` niza. Više ćete naučiti o upotrebi `delete` operatora nad osobinama objekta u Poglavlju 5, "Rad sa objektima". Od JavaScript verzije 1.2 `delete` operator uvek vraća undefined vrednost.

NEW OPERATOR Operator `new` se koristi za kreiranje nove instance nekog tipa objekta. Naučićete kako se koristi u Poglavlju 5.

TYPEOF OPERATOR Typeof operator vraća string vrednost koja identifikuje tip operanda. Razmotrite naredbu a = `typeof` 17. Vrednost dodeljena promenljivoj a je `number`. Pokušajte da upotrebite `typeof` sa različitim izrazima i da vidite vrednosti koje ovaj operator vraća. Možete ga koristiti i sa objektima i funkcijama.

VOID OPERATOR Void operator ne vraća vrednost. On se, obično, koristi sa JavaScript protokolom da vrati URL koji nema vrednost. U Poglavlju 9, "Rad sa linkovima", naći ćete primer za njegovo korišćenje.

Tabela pregleda operatora

Skript prikazan u listingu 3.1 ilustruje upotrebu mnogih JavaScript operatora predstavljenih u prethodnim odeljcima. On generiše HTML tabelu prikazanu na slici 3.1.

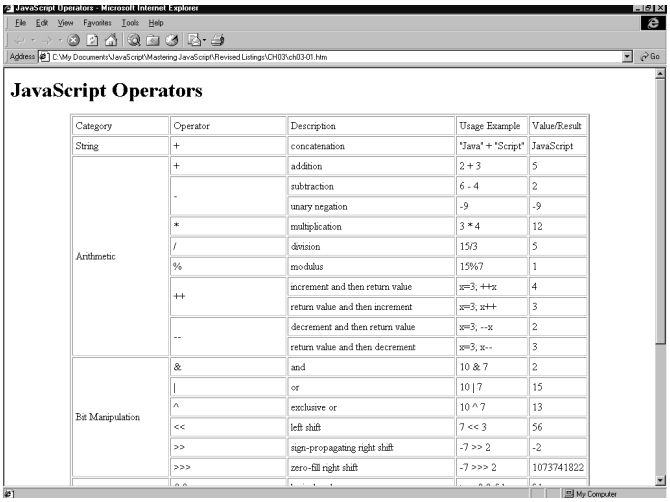
Listing 3.1 JavaScript operatori (ch03-01.htm)

```
<html>
<head>
<title>JavaScript Operatori</title>
</head>
<body>
<h1>JavaScript Operatori</h1>
<table BORDER="2" CELLPADDING="4" ALIGN="CENTER">
<tr><td>Category</td>
<td>Operator</td>
<td>Description</td>
<td>Usage Example</td>
<td>Value/Result</td></tr>
<tr><td>String</td>
<td>+</td>
<td>concatenation</td>
<td>&amp;quot;Java&amp;quot; + &amp;quot;Script&amp;quot;</td>
<td><script><!—
document.write("Java"+"Script")
// —></script>
</td></tr>
<tr><td ROWSPAN="10">Arithmetic</td>
<td>+</td>
<td>addition</td>
<td>2 + 3</td>
<td><script><!—
document.write(2+3)
// —></script>
</td></tr>
<tr><td ROWSPAN="2">-</td>
<td>subtraction</td>
<td>6 - 4</td>
<td><script><!—
document.write(6-4)
// —></script>
</td></tr>
<tr><td>unary negation</td>
<td>-9</td>
<td><script><!—
document.write(-9)
// —></script>
</td></tr>
<tr><td>*</td>
<td>multiplication</td>
<td>3 * 4</td>
<td><script><!—
document.write(3*4)
// —></script>
</td></tr>
<tr><td>/</td>
<td>division</td>
<td>15/3</td>
<td><script><!—
document.write(15/3)
```

```
// —></script>
</td></tr>
<tr><td>%</td>
<td>modulus</td>
<td>15%7</td>
<td><script><!—
document.write(15%7)
// —></script>
</td></tr>
<tr><td ROWSPAN="2">++</td>
<td>increment and then return value</td>
<td>x=3; ++x</td>
<td><script><!—
x=3
document.write(++x)
// —></script>
</td></tr>
<tr><td>return value and then increment</td>
<td>x=3; x++</td>
<td><script><!—
x=3
document.write(x++)
// —></script>
</td></tr>
<tr><td ROWSPAN="2">—</td>
<td>decrement and then return value</td>
<td>x=3; —x</td>
<td><script><!—
x=3
document.write(—x)
// —></script>
</td></tr>
<tr><td>return value and then decrement</td>
<td>x=3; x—</td>
<td><script><!—
x=3
document.write(x—)
// —></script>
</td></tr>
<tr><td ROWSPAN="6">Bit Manipulation</td>
<td>&amp;amp;</td>
<td>and</td>
<td>10 &amp;amp; 7</td>
<td><script><!—
document.write(10&7)
// —></script>
</td></tr>
<tr><td>¦</td>
<td>or</td>
<td>10 ¦ 7</td>
<td><script><!—
document.write(10¦7)
// —></script>
</td></tr>
```

```
<tr><td>^</td>
<td>exclusive or</td>
<td>10 ^ 7</td>
<td><script><!—
document.write(10^7)
// —></script>
</td></tr>
<tr><td>&lt;&lt;</td>
<td>left shift</td>
<td>7 &lt;&lt; 3</td>
<td><script><!—
document.write(7<<3)
// —></script>
</td></tr>
<tr><td>&gt;&gt;</td>
<td>sign-propagating right shift</td>
<td>-7 &gt;&gt; 2</td>
<td><script><!—
document.write(-7>>2)
// —></script>
</td></tr>
<tr><td>&gt;&gt;&gt;</td>
<td>zero-fill right shift</td>
<td>-7 &gt;&gt;&gt; 2</td>
<td><script><!—
document.write(-7>>>2)
// —></script>
</td></tr>
<tr><td ROWSPAN="3">Logical</td>
<td>&amp;&amp;</td>
<td>logical and</td>
<td>true &amp;&amp; false</td>
<td><script><!—
document.write(true&&false)
// —></script>
</td></tr>
<tr><td>¦¦</td>
<td>logical or</td>
<td>true ¦¦ false</td>
<td><script><!—
document.write(true¦¦false)
// —></script>
</td></tr>
<tr><td>!</td>
<td>not</td>
<td>!true</td>
<td><script><!—
document.write(!true)
// —></script>
</td></tr>
<tr><td ROWSPAN="6">Comparison</td>
<td>==</td>
<td>equal</td>
<td>3 == 7</td>
```

```
<td><script><!—
document.write(3==7)
// —></script>
</td></tr>
<tr><td>!=</td>
<td>not equal</td>
<td>3 != 7</td>
<td><script><!—
document.write(3!=7)
// —></script>
</td></tr>
<tr><td>&lt;</td>
<td>less than</td>
<td>3 &lt; 7</td>
<td><script><!—
document.write(3<7)
// —></script>
</td></tr>
<tr><td>&lt;=</td>
<td>less than or equal</td>
<td>3 &lt;= 7</td>
<td><script><!—
document.write(3<=7)
// —></script>
</td></tr>
<tr><td>&gt;</td>
<td>greater than</td>
<td>3 &gt; 7</td>
<td><script><!—
document.write(3>7)
// —></script>
</td></tr>
<tr><td>&gt;=</td>
<td>greater than or equal</td>
<td>3 &gt;= 7</td>
<td><script><!—
document.write(3>7)
// —></script>
</td></tr>
<tr><td>Conditional Izraz</td>
<td>(condition) ? value1 : value2</td>
<td>if condition is true then value1 else value2</td>
<td>true ? 3 : 7</td>
<td><script><!—
document.write(true?3:7)
// —></script>
</td></tr>
</table>
</body>
</html>
```



SLIKA 3.1 *JavaScript reference operatora (listing 3.1)*

Prioritet operatora

Prioritet operatora određuje koja se operacija izvršava pre ostalih tokom rasčlanjavanja i izvršavanja složenih izraza. Na primer, kada izvršite izraz 3 + 4 * 5, treba kao odgovor da dobijete 23, a ne 35, zato što operator množenja * ima veći prioritet od operatora sabiranja +. JavaScript definiše prioritete, a, prema tome, i redosled izvršavanja svih svojih operatora. U tabeli 3.6 dat je pregled prioriteta JavaScript operatora.

Tabela 3.6 Prioritet operatora

Prioritet	Operator
1	zagrade, pozivi funkcija, ili oznake nizova
2	`!`, `~`, `-`, `++`, `--`, `typeof`, `new`, `void`, `delete` (videti napomenu)
3	`*`, `/`, `%` (videti napomenu)
4	`+`, `-` (videti napomenu)
5	`<<`, `>>`, `>>>` (videti napomenu)
6	`<`, `<=`, `>`, `>=` (videti napomenu)
7	`==`, `!=` , `===`. `!==` (videti napomenu)
8	&
9	^
10	|
11	&&
12	||
13	?:
14	`=`, `+=`, `-=`, `*=`, `/=`, `%=`, `<<=`, `>>=`, `>>>=`, `&=`, `^=`, `|=` (videti napomenu)

Napomena: Tamo gde je navedeno više operatora za konkretni nivo svi su istog prioriteta. Ovim ne mislim da kažem da =, na primer,ima neznatno veći prioritet od +=, ili -= na nivou 14 iznad. To pre znači da kako JavaScript čita izraz sleva nadesno na tom nivou prioriteta, tako će i izvršiti bilo koji od tih operatora kada naide na njega.

Da biste videli kako da koristite tabelu 3.6 pri određivanju redosleda izvršavanja, razmotrite sledeći složeni izraz:

```
a = 3 * (9 % 2) - !true >>> 2 - 1
```

Zato što zagrade okružuju izraz 9 % 2 ("devet po modulu od dva") prvo njega izračunavamo, a rezultat je sledeći:

```
a = 3 * 1 - !true >>> 2 - 1
```

NAPOMENA

Kao što sam prethodno pomenuo, % (moduo) operator proračunava ostatak deljenja dva cela broja. Na primer, 9 % 2 = 1, zato što je 9/2 = 4, sa ostatkom 1. ■

Sledeći operator sa najvećim prioritetom je operator negacije !. Nakon izvršavanja !true dobijamo

```
a = 3 * 1 - false >>> 2 - 1
```

Operator množenja * je sledeći koji treba da se izvrši. Daje kao rezultat

```
a = 3 - false >>> 2 - 1
```

Dva operatora - sada imaju najveći prioritet. Logička vrednost false konvertuje se u 0 i prosti izraz 2 - 1 se izračunava u 1, dajući sledeće:

```
a = 3 >>> 1
```

Zato što >>> ima veći prioritet od =, izraz se izračunava u

```
a = 3
```

Konačno, operator dodele = dodeljuje celobrojnu vrednost 3 promenljivoj a.

JavaScript programske naredbe

Naredbe bilo kog programskog jezika predstavljaju instrukcije pomoću kojih je svaki program napisan. Mnogi programski jezici podržavaju zajedničko jezgro skupa naredbi, kao što su naredbe dodele, if naredbe, loop naredbe i druge. Ovi jezici se razlikuju samo u sintaksi koja je upotrebljena za pisanje njihovih naredbi i stepenu do kojeg oni podržavaju modele za razvoj softvera i programske karakteristike, kao što su objektno-orijentisano programiranje, apstraktne definicije podataka, pravila zaključivanja i obrade listi.

JavaScript pruža kompletan opseg osnovnih programskih naredbi. Iako nije jezik objektno-orijentisanog programiranja, on jeste jedan objektno-zasnovan jezik i podržava objekte, osobine objekata i metode. O ovim objektno-orijentisanim osobinama programiranja učićete u Poglavlju 5, "Rad sa objektima".

Naredbe koje omogućava JavaScript su sažete u tabeli 3.7 i detaljnije su objašnjene u odeljcima koji slede.

Tabela 3.7 Pregled JavaScript naredbi

Naredba	Namena	Primer
dodeljivanja	x = y + z	
deklaracije podataka		`card = new Array(52)`
if		`if (x>y) { z = x}`
switch	Selektuje broj između alternativa	`switch (val) { case 1: // Prva alternativa break; case 2: // Druga alternativa break; default: // Podrazumevana akcija}`
while		`while (x!=7) { x %= n —n}`
for		`for(i=0;i<7;++i){ document.write(x[i])}`
do while		`do { // Naredbe} while (i>0)`
label	Povezuje labelu sa naredbom	`labelName : naredba`
break		`if(x>y) break`
continue		`if(x>y) continue`
za pozivanje funkcije	pozivanje funkcije	`x=abs(y)`
return	vraća vrednost iz pozvane funkcije	`return x*y`
with	Identifikuje podrazumevani objekt	`with (Math) { d = PI * 2 * r;}`
for in	petlja za prolazak kroz sva svojstva nekog objekta	`for (prop in employee) { document.write (prop+" ")}`
throw	Aktiviranje izuzetka	throw "Moja greška"
try	Provera pojave izuzetaka	`try { // Nepouzdani kod}catch (Izuzetak) { // Obradi izuzetak}`
catch	hvatanje izuzetaka	(videti iznad)
delete		`delete a[5]`
za pozivanje metoda	poziva metod određenog objekta	`document.write("Hello!")`

NAPOMENA

`Throw`, `try` i `catch` naredbe su podržane samo u Internet Exploreru 5 i kasnijim verzijama. `Import` i `export` naredbe se koriste u označenim skriptovima i podržane su samo u Navigatoru 4 i kasnijim. U Poglavlju 23, "Obezbedite svoje skriptove", objašnjena je upotreby `import` i `export` naredbi. ■

Evo šta treba da imate stalno na umu kada pišete niz naredbi:

- U jednoj liniji teksta može da se pojavi više naredbi, ako obezbedite da su sve odvojene jedna od druge pomoću tačka-zarez karaktera (;). Tačka-zarez je JavaScript oznaka za separator linija.
- Između naredbi koje se pojavljuju u različitim linijama nije potrebno stavljati tačka-zarez karakter.
- Duge JavaScript naredbe mogu da se pišu u više linija teksta. Za ovakve višelinijske naredbe ne zahteva se upotreba identifikatora nastavka linije. Pokazaću Vam, ipak, strelicu nastavljanja, i to samo neznatno u ranim poglavljima kada osetim da to može biti od pomoći pri razjašnjavanju veličine dugih linija naredbi koje mogu da budu nezgodne.

Naredbe dodele

Najosnovnija naredba koja se može naći u gotovo svim programskim jezicima je naredba dodele. Ona ažurira vrednost promenljive zasnovane na operatoru dodele i izrazu (i, opciono, trenutne vrednosti promenljive koju ažuriramo). Videli ste veliki broj primera naredbi dodele u ranijim odeljcima ovog poglavlja i Poglavlja 2, "Upoznavanje JavaScripta i JScripta". Na primer, naredba

```
y = x + 10
```

dodeljuje vrednost promenljive `x` plus 10 promenljivoj `y`.

Deklarisanje podataka

Deklarisanje podataka identifikuje promenljive JavaScript interpretatoru. Do sada ste deklarisali proste promenljive pomoću naredbi dodele. Na primer, naredba

```
a = 25
```

prouzrokuje implicitnu deklaraciju promenljive a kao celog broja i inicijalizuje je na 25. Promenljivu a razmatramo kao *globalnu* promenljivu.Ovo znači da nju mogu da koriste svi skriptovi definisani u HTML dokumentu. Kasnije u ovom poglavlju, u odeljku "Deklarisanje lokalnih promenljivih", naučićete kako da deklarišete promenljive koje su lokalne za definisanu funkciju.

Deklarisanje nizova predstavlja još jedan primer naredbi za deklarisanje podataka. Na primer, sledeće deklarisanje nizova deklariše niz promenljivih:

```
// Declares an array of zero elements
customerNum = new Array()

// Declares an array of 100 null valued elements
productCode = new Array(100)
```

Deklarisanje zbijenih nizova pruža mogućnost deklarisanja i dodele inicijalnih vrednosti svim elementima niza. Slede dva primera ove naredbe deklarisanja:

```
// Declares and initializes a seven-element array
day = new Array("Sun","Mon","Tue","Wed","Thu","Fri","Sat")

// Declares and initializes a four-element array
```

```
name = new Array("Bob","Sybil","Ricky","Mad Dog")
```

Drugačiji tip deklarisanja promenljive obavlja kreiranje instance objekta. O objektima i njihovom kreiranju naučićete u Poglavlju 5, "Rad sa objektima". Instance objekata se kreiraju upotrebom naredbe sledećeg oblika:

```
variableName = new objectConstructor(p1,p2,...,pn)
```

gde je *variableName* ime promenljive koje se dodeljuje novokreiranom objektu, `new` predstavlja kreatora novog objekta, *objectConstructor()* je funkcija koja se koristi za kreiranje objekta, a *p1* do *pn* predstavlja opcionu listu parametara za kreiranje objekta.

If naredba

`If` naredba pruža mogućnost izmene toka izvršavanja programa, u zavisnosti od konkretne logičke vrednosti izraza. Ako je logička vrednost *true* (tačno), određeni skup naredbi se izvršava. Ako je logička vrednost *false* (netačno), skup naredbi se preskače (a, opciono, drugi skup naredbi se izvršava). `If` naredba se koristi u dva oblika. Sintaksa prvog je sledeća:

```
if ( uslov ) {
 naredbe
}
```

Ako je navedeni uslov `true` (tačan), onda se naredbe navedene između zagrada `{` i `}` izvršavaju. Potom se izvršavaju naredbe koje slede if naredbu. Ako je navedeni uslov false (netačan), naredbe koje se nalaze unutar zagrada se preskaču i izvršavanje se nastavlja izvršavanjem naredbi koje slede `if` naredbu. Na primer, sledeća naredba upisuje tekst Good morning u aktuelni web dokument kada je vrednost promenljive hour manja od 12.

```
if(hour<12){
 document.write("Good morning")
}
```

Sintaksa drugog oblika `if` naredbe je slična prvom obliku, izuzev što je dodata else odrednica. Sintaksa drugog oblika je sledeća:

```
if ( uslov ) {
 prvi skup naredbi
} else {
 drugi skup naredbi
}
```

Ako je navedeni uslov `true`, izvršava se prvi skup naredbi. Ako je navedeni uslov `false`, izvršava se drugi skup naredbi. U oba slučaja nastavak je izvršavanje naredbe koja sledi `if` naredbu.

Primer drugog oblika if naredbe je sledeći:

```
if(hour<12){
 document.write("Good morning")
}else{
 document.write("Hello")
}
```

Izvršavanje if naredbe kao rezultat daje ažuriranje aktuelnog dokumenta, tako da ovaj prikazuje tekst *Good morning* ako je vrednost promenljive `hour` manja od 12, ili prikazuje Hello ako je `hour` veća, ili jednaka 12.

NAPOMENA

Zagrade koje uokviruju naredbe u if i `else` odeljcima mogu da se izostave, ukoliko se samo jedna naredba navodi u okviru zagrada. ■

Loop naredbe

Loop naredbe (ili naredbe petlji) služe za ponavljanje izvršavanja skupa naredbi dok je određeni uslov *true*. JavaScript podržava tri tipa naredbi petlji: naredbe `while`, do `while` i for. On obezbeđuje i `label`, `break` i `continue` naredbe. `Break` naredbe se koriste za prekid svih ponavljanja petlje. `Continue` naredba se koristi da izazove hitan prekid jednog ponavljanja petlje i da se nakon toga nastavi sledeće ponavljanje petlje. `Label` naredba označava naredbe koje treba da se koriste sa naredbama `break` i `continue`. Ove naredbe su objašnjene u sledećim odeljcima.

While naredba

`While` naredba je osnovna loop naredba, koja se koristi za ponavljanje izvršavanja skupa naredbi, sve dok je određeni uslov *true*. Sintaksa `while` naredbe je sledeća:

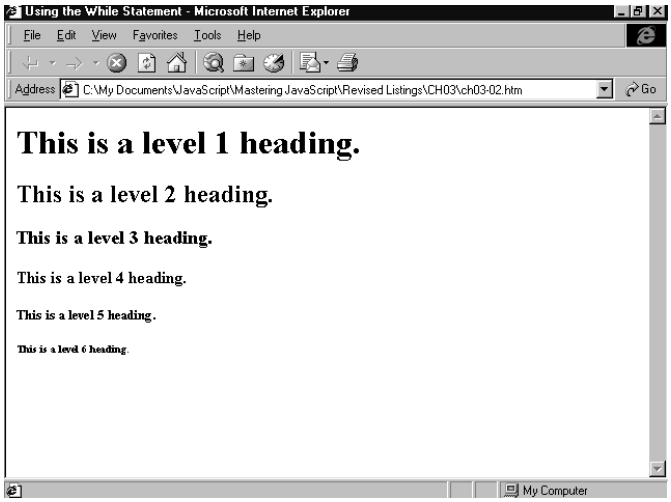
```
while ( uslov ) {
 naredbe
}
```

`While` naredba izračunava svoj uslov. Ako je uslov *true*, izvršava naredbe u okviru zagrada. Kada uslov postane *false*, on prosleđuje kontrolu izvršavanja naredbi koja sledi `while` naredbu.

U listingu 3.2 dat je primer upotrebe `while` naredbe za generisanje sadržaja HTML dokumenta. Slika 3.2 prikazuje kako dokument generisan ovim skriptom izgleda prikazan web browserom. Skript se ponavlja od 1 do 6, generišući šest nivoa HTML zaglavlja. Indeksna promenljiva se uvećava svaki put kada se prođe kroz petlju.

Listing 3.2 While naredba (ch03-02.htm)

```
<HTML>
<HEAD>
<TITLE>Using the While Naredba</TITLE>
</HEAD>
<BODY>
<SCRIPT><!—
i=1
while(i<7){
 document.write("<H"+i+">This is a level "+i+" heading." +"</H"+i+">")
 ++i
}
// —></SCRIPT>
</BODY>
</HTML>
```



f

SLIKA 3.2 *Upotreba while naredbe za generisanje HTML dokumenta (listing 3.2)*

Do *while* naredba

Do `while` naredba, uvedena u JavaScript 1.2, slična je `while` naredbi. Jedina razlika je da se uslov prolaska kroz petlju proračunava (odnosno proverava) na kraju svake petlje, umesto na početku. Ovo omogućava da se naredbe izvršavaju najmanje jednom. Sintaksa `do while` naredbe je sledeća:

```
do {
 naredbe
} while (uslov);
```

Na primer, sledeće naredbe prikazuju cele brojeve od 1 do 10:

```
i = 0
do {
 ++i
 document.writeln(i+"<BR>")
} while (i < 10);
```

For naredba

`For` naredba je slična `while` naredbi u tome što ponavlja izvršavanje skupa naredbi, sve dok je uslov tačan. Razlikuje se od `while` naredbe u tome što je ona napravljena da ažurira promenljivu nakon svakog ponavljanja petlje. Sintaksa for naredbe je

```
for ( initializationNaredba; uslov; updateNaredba ) {
 naredbe
}
```

Naredba inicijalizacije se izvršava samo na početku izvršavanja `for` naredbe. Uslov se testira. Ako je *true*, naredbe navedene u okviru zagrada se izvršavaju. Ako je ulov *false*, petlja se završava i sledeća naredba koja je iza `for` naredbe se izvršava.

Ako se naredbe zatvorene unutar zagrada izvrše, naredba ažuriranja se takođe izvršava i onda se uslov ponovo testira. Uokvirene naredbe i naredba ažuriranja se izvršavaju više puta, sve dok uslov ne postane false.

NAPOMENA

Naredba inicijalizacije, uslov i naredba ažuriranja su opcioni i mogu se izostaviti. ■

Sledi primer for naredbe:

```
a = new Array(2,4,6,8,10)
sum = 0
for (i = 0;i < a.length;++i) {
 sum += a[i]
}
```

Prva naredba u ovom primeru kreira niz od pet elemenata sa vrednostima 2, 4, 6, 8 i 10. Druga naredba postavlja promenljivu `sum` na 0. For naredba počinje postavljanjem promenljive i na 0 i testiranjem dužine niza, da bi se videlo da li je veća od `i`. Zato što je i jednaka 0 i `a.length` je 5, naredba u okviru zagrada se izvršava i vrednost sum se uvećava za `a[i]`, koji je u ovom momentu 2.

Naredba ažuriranja `++i` se izvršava nakon toga. Ovo uzrokuje da se i uvećava na `1`. Uslov se ponovo testira i zato što je i manje od 5 naredba u okviru zagrada se ponovo izvršava. Ovog puta se `sum` uvećava za 4 i njena vrednost postaje 6.

Naredba ažuriranja `++i` se izvršava po drugi put i uslov se ponovo testira. U ovom momentu trebalo bi da možete da pratite ostatak izvršavanja for naredbe. For petlja nastavlja da se ponavlja sve dok uslov `i < a.length` ne postane netačan. Ovo se dešava kad i postane 5. U tom momentu `sum` postaje zbir svih elemenata niza, a njegova vrednost je 30.

U listingu 3.3 ilustrovano je kako skript prikazan u listingu 3.2 može da se ažurira da bi koristio `for` naredbu, umesto `while` naredbe. Web strana koja predstavlja rezultat izvršavanja ovog skripta je ista kao i ona prikazana na slici 3.2.

Listing 3.3 Upotreba for naredbe (ch03-03.htm)

```
<HTML>
<HEAD>
<TITLE>Using the For statement</TITLE>
</HEAD>
<BODY>
<SCRIPT><!—
for(i=1;i<7;++i)
 document.write("<H"+i+">This is a level "+i+" heading." +"</H"+i+">")
// —></SCRIPT>
</BODY>
</HTML>
```

NAPOMENA

Zagrade koje uokviravaju naredbe u `for` i `while` naredbama mogu da se izostave, ako se unutar zagrada nalazi samo jedna naredba. ■

Label naredba

Svaka naredba može da se obeleži postavljanjem labelName : pre same naredbe. Na primer,

```
MyLabel :
 a = 2 * b
```

prouzrokuje da se gornja naredba obeležava labelom `MyLabel`. Labele se koriste za obeležavanje naredbi. One se, obično, koriste za obeležavanje loop, `switch`, ili `if` naredbi. Možete se pozivati na obeležene naredbe preko `break` i `continue` naredbi, kao što je objašnjeno u sledećim odeljcima.

Break naredba

Break naredba se koristi za prekid izvršavanja petlje i za prosleđivanje kontrole izvršavanja na naredbu koja sledi petlju. Ona se sastoji iz reči `break`, koju prati opciona labela. Kada se naiđe na `break` naredbu petlja se momentalno prekida. Sledi primer njene upotrebe:

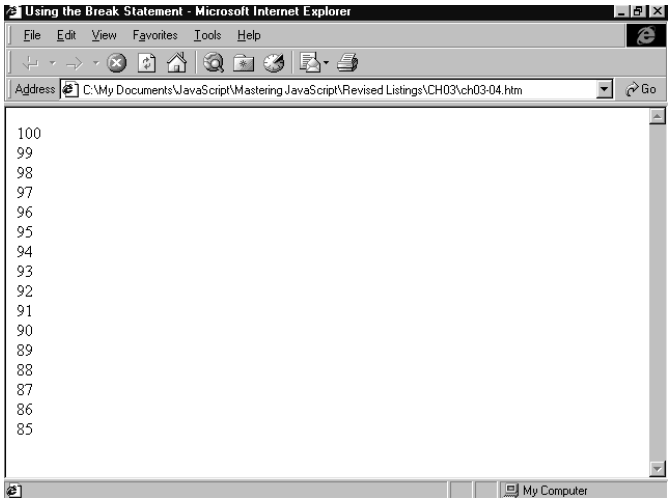
```
a = new Array(5,4,3,2,1)
sum = 0
for (i = 0;i < a.length;++i) {
 if (i == 3) break;
 sum += a[i]
}
```

Prethodna for naredba se izvršava za i koje je jednako 0, 1, 2 i 3. Kada i postane jednako 3, uslov `if` naredbe je ispunjen i `break` naredba se izvršava. Ovo kao rezultat daje momentalni završetak `for` naredbe. Vrednost promenljive `sum` je 12 odmah posle završetka for naredbe.

U listingu 3.4 prikazan je skript koji ilustruje upotrebu break naredbe, a na slici 3.3 web strana generisana ovim skriptom. Skript sadrži petlju koja testira cele brojeve od 100 do 1, sve dok ne naiđe na onaj koji je deljiv bez ostatka sa 17. Kada se nađe takav broj on završava petlju.

Listing 3.4 Break Naredba (ch03-04.htm)

```
<HTML>
<HEAD>
<TITLE>Using the Break Statement</TITLE>
</HEAD>
<BODY>
<SCRIPT><!—
for(i=100;i>0;—i){
 document.write(i+"<BR>")
 if(i%17==0) break
}
// —></SCRIPT>
</BODY>
</HTML>
```



SLIKA 3.3 *Upotreba break naredbe (listing 3.4)*

Ako se break naredba kombinuje sa labelom, kontrola izvršavanja se prebacuje prvoj naredbi koja sledi iza labele. Obeležena naredba ne mora da bude loop naredba. Razmotrite sledeći primer:

```
test1 :
   if (val > 0) {
      document.write("Greater than zero: ")
      test2 :
         if (val == 2) {
            document.write(val)
            break test1
         }
      // Other code
   }
```

Break naredba prebacivanje kontrolu na prvu naredbu koja prati završnu naredbu `if` naredbe (`test1`).

Continue naredba

`Continue` naredba je slična break naredbi u tome što takođe utiče na izvršavanje `for`, `do while`, ili `while` naredbe, ako je u njih uključena. Ona se razlikuje od `break` naredbe u tome što ne završava u potpunosti izvršavanje petlji, nego samo završava izvršavanje naredbi petlje koje se ponavljanju.

Kada se `continue` naredba uoči u `while`, `do while`, ili `for` petlji ostatak naredbi koje su se ponavljale se preskaču i kontrola izvršenja se vraća uslovu petlje.

Razmotrite sledeću `while` petlju kao primer:

```
i = 1
sum = 0
while(i<10) {
 i *= 2;
```

```
 if (i == 4) continue
 sum += i + 1
}
```

`While` petlja se ponavlja za i koje je jednako 1, 2, 4 i 8 na početku svake petlje. Međutim, promenljiva sum se ažurira jedino kada je i jednako 1, 4 i 8. Kada je i jednako 2 na početku petlje, ono se duplira u 4 kao rezultat izvršavanja prve naredbe petlje. Ovo za posledicu ima da je uslov `if` naredbe ispunjen i `continue` naredba se izvršava. Izvršenje continue naredbe izaziva preskok zadnje naredbe u petlji i kontrola se vraća izvršavanju uslova while naredbe. Konačna vrednost promenljive `sum` je 29.

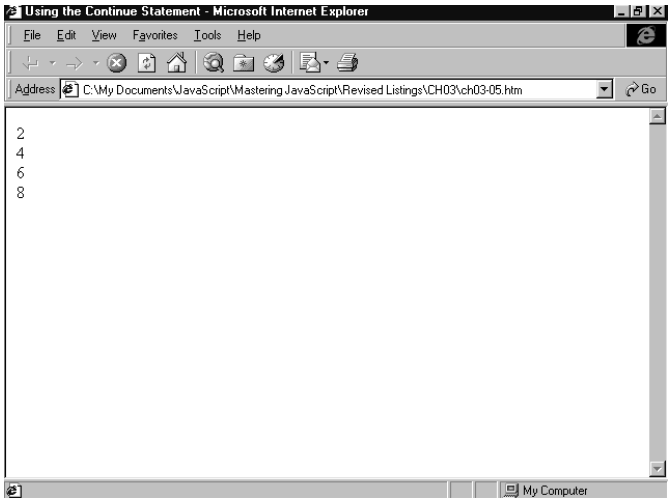
NAPOMENA

`Continue` naredba se može koristiti i sa labelom. Labela određuje naredbu petlje na koju se continue naredba primenjuje. ■

U listingu 3.5 je prikazan skript koji ilustruje upotrebu `continue` naredbe, a na slici 3.4 web strana generisana ovim skriptom. Skript štampa cele brojeve između 1 i 10 (neobuhatajući 10), ali koristi `continue` naredbu da preskoči sve neparne cele brojeve.

Listing 3.5 Continue naredba (ch03-05.htm)

```
<HTML>
<HEAD>
<TITLE>Using the Continue Statement</TITLE>
</HEAD>
<BODY>
<SCRIPT><!—
for(i=1;i<10;++i){
 if(i%2!=0) continue
 document.write(i+"<BR>")
}
//  —></SCRIPT>
</BODY>
</HTML>
```



SLIKA 3.4 *Upotreba continue naredbe (listing 3.5)*

Switch naredba

JavaScriptu 1.2 je dodata i switch naredba. Njena sintaksa je ista kao i u Javi i C++-u:

```
switch (izraz) {
  case vrednost1:
    naredbe
    break
      .
      .
      .
  case vrednostn:
    naredbe
    break
  default:
    naredbe
}
```

`Switch` naredba izvršava izraz i određuje da li je neka od vrednosti (*vrednost1* do *vrednostn*) jednaka vrednosti izraza. Ako je jedna od njih jednaka, naredbe za taj određeni case se izvršavaju. Izvršavanje naredbi se nastavlja posle `switch` naredbe. Ako ne postoji jednaka vrednost, onda se izvršavaju naredbe za podrazumevani case.

`Break` naredbe se mogu izostaviti. Međutim, ako se izostave, izvršenje se nastavlja sledećim caseom (ako postoji).

Sledeća `switch` naredba štampa vrednost stringa pod brojem koji odgovara 1, 2 ili 3, ili, inače, štampa *I don't know* .

```
switch (i) {
  case 1:
    document.writeln("one")
```

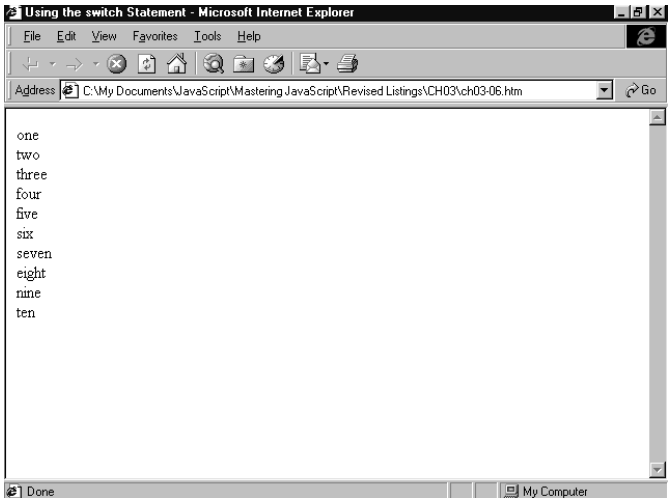
```
    break
  case 2:
    document.writeln("two")
    break
  case 3:
    document.writeln("three")
    break
  default 1:
    document.writeln("I don't know")
}
```

U listingu 3.6 je prikazan skript koji ilustruje upotrebu `switch` naredbe, a na slici 3.5 web strana generisana ovim skriptom. Skript prikazuje imena brojeva od 1 do 10.

Listing 3.6 Switch naredba (ch03-06.htm)

```
<HTML>
<HEAD>
<TITLE>Using the switch Statement</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript"><!—
for(i=1; i<=10; ++i) {
 switch (i) {
  case 1:
   val = "one"
   break;
  case 2:
   val = "two"
   break;
  case 3:
   val = "three"
   break;
  case 4:
   val = "four"
   break;
  case 5:
   val = "five"
   break;
  case 6:
   val = "six"
   break;
  case 7:
   val = "seven"
   break;
  case 8:
   val = "eight"
   break;
  case 9:
   val = "nine"
   break;
  case 10:
   val = "ten"
   break;
  default:
```

```
    val = "unknown"
  }
  document.writeln(val+"<BR>");
}
//  —>
</SCRIPT>
</BODY>
</HTML>
```



SLIKA 3.5 *Upotreba switch naredbe (listing 3.6)*

Naredba za poziv funkcije

Mnogi programski jezici podržavaju pozive funkcija. *Funkcijama* se nazivaju blokovi naredbi koji se pozivaju i izvršavaju kao celina. Podaci neophodni za njihovo izvršavanje mogu im se proslediti kao *parametri*. Funkcije mogu da vraćaju vrednosti, ali to se od njih ne zahteva uvek. Kada funkcija vraća vrednost, onda je poziv funkcije, obično, deo nekog izraza. Na primer, sledeća naredba poziva `factorial()` funkciju, prosleđujući joj celobrojnu vrednost 5 kao parametar:

```
n = factorial(5)
```

U ovom primeru pozvana funkcija `factorial(5)` vraća vrednost koja se dodeljuje promenljivoj `n`. Funkcija `factorial()` je samo zamišljena funkcija i nije definisana u JavaScriptu.

Kada funkcija ne vraća vrednost obično se koristi da izvede operaciju koja ažurira promenljivu, ili neki objekat koji je eksterni za JavaScript. Pozivanje funkcije koja ne vraća vrednost predstavlja izvršnu naredbu - jedino što nije deo većeg izraza. Na primer, razmotrite poziv funkcije u sledećoj naredbi:

```
notifyUser("Product code is invalid")
```

U ovoj naredbi funkcija `notifyUser()` prihvata string *"Product code is invalid"* kao parametar. Nakon toga, ona prikazuje ovaj string korisniku. Funkcija ne vraća vrednost i, zbog toga, nije na desnoj strani neke naredbe dodele.

Definisanje funkcija

Funkcije moraju biti definisane, pre nego što se mogu koristiti. Definicija funkcije se, obično, smešta u zaglavlju HTML dokumenta, iako takav postupak nije obavezan. Smeštanje definicije funkcije u zaglavlju, međutim, obezbeđuje da do definisanja dolazi pre korišćenja funkcije. Sintaksa definisanja funkcije je sledeća:

```
function functionName(p1, p2, ..., pn) {
 naredbe
}
```

Ime funkcije se koristi za navođenje u pozivima funkcije. Parametri su imena promenljivih koje prihvataju prosleđene vrednosti funkciji kada se ona pozove. Parametri koji joj se prosleđuju nazivaju se *argumenti* funkcije. Naredbe u okviru zagrada se izvršavaju kao rezultat poziva funkcije.

Na primer, razmotrite sledeću definiciju funkcije:

```
function display(text) {
 document.write(text)
}
```

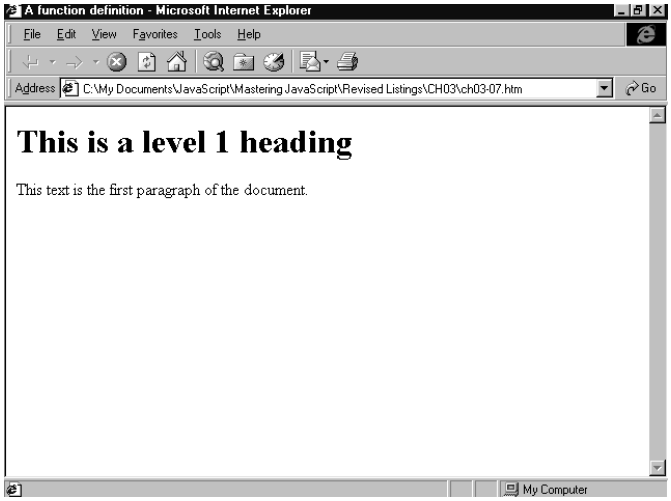
U ovom primeru, ako se funkcija pozove naredbom `display("xyz")`, tekst *xyz* se upisuje u aktuelni web dokument. Poziv funkcije `display("xyz")` je, na ovaj način, ekvivalentan sa naredbom `document.write("xyz")`.

Listing 3.7 pruža primer definisanja funkcije. Na slici 3.6 prikazana je web strana koju ovaj skript generiše.

Listing 3.7 Definisanje funkcije (ch03-07.htm)

```
<HTML>
<HEAD>
<TITLE>A function definition</TITLE>
<SCRIPT LANGUAGE="JavaScript"><!—
function displayTaggedText(tag, text) {
 document.write("<"+tag+">")
 document.write(text)
 document.write("</"+tag+">")
}
// —>
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript"><!—
displayTaggedText("H1","This is a level 1 heading")
displayTaggedText("P","This text is the first paragraph of the
document.")
// —>
</SCRIPT>
</BODY>
</HTML>
```

Skript u zaglavlju dokumenta definiše funkciju `displayTaggedText()`, koja prihvata parametre `tag` i `text`. Konkretne vrednosti za `tag` i `text` se prosleđuju funkciji preko ovih parametara kada se ona pozove. Funkcija se sastoji iz tri naredbe. Prva naredba upisuje levu uglastu zagradu, koju prati vrednost `tag` parametra plus desna uglasta zagrada. Druga `write` naredba upisuje vrednost tekst promenljive u aktuelni dokument. Treća naredba upisuje string </, koji prati vrednost `tag` promenljive i desna uglasta zagrada, u aktuelni dokument. Funkcija `displayTaggedText()` ne vraća vrednost.



SLIKA 3.6 *Definisanje i korišćenje funkcije (listing 3.7)*

Skript u telu dokumenta izvršava dve naredbe poziva funkcije koje pozivaju funkciju `displayTaggedText()`, koja upisuje zaglavlje HTML-a prvog nivoa i pasus teksta u aktuelni dokument. Parametri prosleđeni u prvom pozivu su `"H1"` i `"This is a level 1 heading"`. String `"H1"` se prosleđuje `displayTaggedText()` funkciji preko promenljive `tag`. String `"This is a level 1 heading"` se prosleđuje preko `tekst` promenljive. Drugi poziv `displayTaggedText()` se odvija na isti način. String ″P″ se prosleđuje preko promenljive tag i `"This text is the first paragraph of the document"` se prosleđuje preko promenljive `text`.

Definisanje funkcija sa promenljivim brojem parametara

JavaScript pruža mogućnost definisanja funkcije koja može da prihvati promenljivi broj parametara, koristeći niz `argumenata`. Niz argumenata `automatski` kreira JavaScript pri svakom pozivu funkcije. Pretpostavite da je funkcija `f` pozvana parametrima `"test"`, `true` i 77, kao što je urađeno u sledećoj naredbi:

```
f("test",true,77)
```

Niz f.arguments sadrži vrednosti ovih parametara. U ovom slučaju promenljive su sledeće:

```
f.arguments.length = 3
f.arguments[0] = "test"
f.arguments[1] = true
```

```
f.arguments[2] = 77
```

Sledeća definicija funkcije ilustruje upotrebu niza `argumenata`:

```
function sum() {
 n = sum.arguments.length
 total = 0
 for(i=0;i<n;++i) {
  total += sum.arguments[i]
 }
 return total
}
```

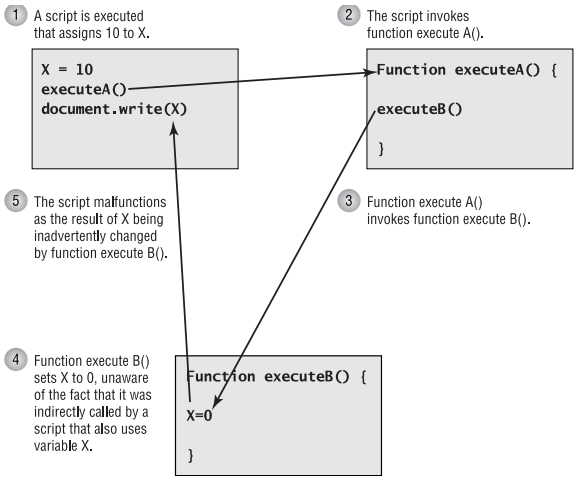
`Sum()` funkcija je kreirana da sabere proizvoljnu listu parametara, promenljivoj n se dodeljuje dužina `sum.arguments` niza, a promenljiva `total` se koristi za sabiranje elemenata `sum.arguments` niza.

NAPOMENA

Ključna reč this se koristi u funkcijama da ukaže na aktuelni objekat. U Poglavlju 5, "Rad sa objektima", naći ćete opis njenog korišćenja. ■

Deklarisanje lokalnih promenljivih

Kada definišite funkciju često je neophodno da definišete i promenljive za smeštanje vrednosti koje ona proračunava. Možete da deklarišete promenljive, koristeći naredbe deklarisanja koje ste proučili u odeljku "Deklarisanje podataka" u ovom poglavlju. Međutim, ovo znači definicija funkcije zavisi od ovih globalnih promenljivih i da je, kao rezultat toga, manje modularna i komplikovanija za otklanjanje grešaka. Mnogo je bolje deklarisati promenljive koje se koriste samo u okviru funkcije. Ove promenljive, koje razmatramo kao lokalne promenljive, dostupne su samo u okviru funkcije u kojoj su i bile deklarisane. Na slici 3.7 ilustrovane su teškoće u određivanju pripadnosti pri korišćenju globalnih promenljivih u okviru funkcija.



SLIKA 3.7 *Upotreba globalne promenljive u okviru funkcije*

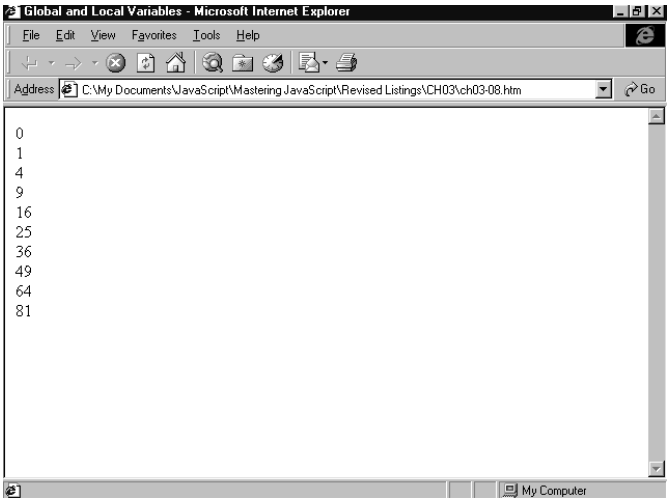
Lokalna promenljiva se deklariše na isti način kao i globalna promenljiva, izuzev što njenom deklarisanju prethodi ključna reč `var`. Sledeći su primeri deklarisanja lokalne promenljive:

```
// Declares temp as a local variable
var temp

// Declares index as a local variable and initializes it to 1
var index = 1

/* Declares the product array with an initial
   capacity of 100 elements */
var product = new Array(100)
```

U slučajevima kada lokalna i globalna promenljiva imaju isto ime sve reference na ime promenljive u okviru funkcije koja definiše lokalne promenljive ukazuju na lokalne promenljive te funkcije, a ne na globalne promenljive. Sve reference na ime promenljive koje se nalaze van funkcije koja definiše lokalnu promenljivu ukazuju na globalnu promenljivu. U listingu 3.8 skript ilustruje upotrebu globalne i lokalne promenljive sa istim imenom. Slika 3.8 prikazuje web stranu koju ovaj skript generiše. Promenljiva `x` je lokalna promenljiva `displaySquared()` funkcije, a globalna promenljiva u skriptu koji se nalazi u okviru tela dokumenta. Obratite pažnju da lokalna i globalna `x` promenljiva mogu da se ažuriraju nezavisno.

Listing 3.8 Korišćenje globalnih i lokalnih promenljivih (ch03-08.htm)

```
<HTML>
<HEAD>
<TITLE>Global and Local Variables</TITLE>
<SCRIPT LANGUAGE="JavaScript"><!—
function displaySquared(y) {
var x = y * y
document.write(x+"<BR>")
}
// —>
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript"><!—
for(x=0;x<10;++x)
 displaySquared(x)
// —>
</SCRIPT>
</BODY>
</HTML>
```



SLIKA 3.8 *Primer upotrebe globalne i lokalne promenljive (listing 3.8)*

Naredba povratka

Naredba povratka (`return`) se koristi za vraćanje vrednosti nastale pri obradi koju izvodi funkcija. Ova vrednost se vraća naredbi koja je pozvala funkciju. Sintaksa naredbe povratka je

```
return izraz
```

Izraz određuje vrednost koju funkcija vraća. Kada se uoči naredba povratka, izraz se izračunava, a vrednost u koju se izraz izračunava funkcija momentalno vraća. Sve naredne naredbe funkcije se ne obrađuju.

Primer korišćenja naredbe povratka je prikazan u sledećoj definiciji funkcije:

```
function factorial(n) {
 var sum = 1
 for(i=1;i<=n;++i)
  sum *= i
 return sum
}
```

U prethodnom primeru sum se izračunava kao proizvod svih celih brojeva od 1 do n. Ova vrednost se vraća naredbom povratka.

Naredbe za pristup objektima

Objekti su strukture podataka JavaScripta - one sadrže i podatke i obezbeđuju funkcije, koje se nazivaju *metodi*, a koriste se za obradu ovih podataka. Lične promenljive koje objekat obuhvata se nazivaju *osobine*. Osobine obezbeđuju pristup podacima u objektu.

NAPOMENA

JavaScript objekti su opisani u Poglavlju 5. U ovom odeljku akcenat je stavljen na programske naredbe koje koriste osobine i metodi objekta. ■

Zato što osobine pružaju pristup vrednostima u promenljivim iz kojih se sastoji objekat, one se, obično, koriste u izrazima koji se pojavljuju ili sa desne, ili sa leve strane naredbe dodele. Na primer, pretpostavite da promenljiva `employee` ukazuje na objekat tipa `employeeRecord` koji ima `employeeID` osobinu. Sledeća naredba dodele uzima vrednost osobine `employeeID` i dodeljuje je promenljivoj `id`.

```
id = employee.employeeID
```

Opšta sintaksa koja se koristi za pristup osobini objekta je

```
variableName.propertyName
```

gde je variableName ime promenljive koja ukazuje na objekat, a propertyName je naziv osobine do koje treba doći.

Kada se referenca osobine pojavi na levoj strani naredbe dodele osobina referenciranog objekta se ažurira. U sledećem primeru osobina `employeeID` objekta referenciranog pomoću promenljive `employee` se ažurira vrednošću koja se nalazi zapamćena u promenljivoj id.

```
employee.employeeID = id
```

Metodi jednog objekta su funkcije koje se koriste za izvođenje operacija nad objektom. Oni se pozivaju na sličan način kao i osobine. Kako god bilo, budući da su metodi funkcije, oni mortaju da uključe i listu parametara metoda. Prazna lista parametara se navodi na isti način kao i za poziv funkcija. Opšta sintaksa poziva metoda je prikazana na sledeći način:

```
variableName.methodName(p1,p2,...,pn)
```

gde je variableName ime promenljive koja ukazuje na objekat, a methodName je naziv metode koja se poziva. Parametri od `p1` do pn predstavljaju opcionu listu parametara metoda.

Neki metodi ne vraćaju vrednost. Njihovo pozivanje predstavlja celu naredbu samu po sebi. Najčešći primer koji ste koristili do sada je `write()` metod `dokument` objekta:

```
document.write("text to be displayed")
```

Neki metodi vraćaju vrednost. U ovim slučajevima pozivi metoda mogu da se pojave kao delovi mnogo većih izraza, kao što je prikazano sledećim primerom:

```
payroll=0
for(i=0;i<employee.length;++i)
 payroll += employee[i].netPay()
```

U prethodnom primeru `employee` predtstavlja niz `employeeRecord` objekata. Metod `netPay()` izračunava osnovnu platu bez odbitaka za svakog zaposlenog, na osnovu podataka u osobinama `employeeRecord` objekata.

With naredba

With naredba pruža mogućnost eliminisanja ponovnog navođenja objekta koji se navodi u nizu referenciranja osobina i poziva metoda. Sintaksa `with` naredbe je sledeća:

```
with(variableName){
 naredbe
}
```

VariableName određuje podrazumevani objekat koji se koristi sa naredbom u okviru zagrada. Sledi primer `with` naredbe:

```
with(document) {
 write("<H1>With It</H1>")
 write("<P>")
 write("Eliminate object name references with with")
 write("</P>")
}
```

U prethodnom primeru potreba za navođenjem prefiksa za svaki poziv `write()` metoda sa objektom `dokumenta` je eliminisana, zbog toga što je `dokument` identifikovan `with` naredbom. Bez `with` naredbe, gornji kod bi trebalo da se napiše kao:

```
document.write("<H1>With It</H1>")
document.write("<P>")
document.write("Eliminate object name references with with")
document.write("</P>")
```

For in naredba

Do sada ste naučili kako se koristi for naredba za ponavljanje skupa naredbi zasnovanih na uslovu petlje i na naredbi za ažuriranje. `For in` naredba je slična sa `for` naredbom u tome što ona više puta izvršava skup naredbi. Međutim, umesto ponavljanja naredbi zasnovanih na uslovu petlje, ona izvršava naredbe za sve osobine koje su definisane za neki objekat.

Sintaksa for in naredbe je sledeća:

```
for (variableName in objectName) {
 naredbe
}
```

`For` in petlja izvršava naredbe koje su u okviru zagrada po jednom za svaku osobinu definisanu za `objectName`. Svaki put kada se naredbe izvrše promenljiva određena sa `variableName` se dodeljuje stringu koji identifikuje trenutni naziv osobine. Na primer, razmotrite izvršavanje sledeće `for in` naredbe u slučaju objekta `employee` koji ima osobine `employeeID`, `employeeName` i `employeeLocation`.

```
for(prop in employee)
    document.write(prop+"<BR>")
```

Prethodna `for in` naredba će prouzrokovati da sledeći tekst bude upisan u aktuelni dokument:

```
employeeID
employeeName
employeeLocation
```

Ovo su tri osobine koje su definisane za `employee` objekat.

Naredbe *throw, try* i *catch*

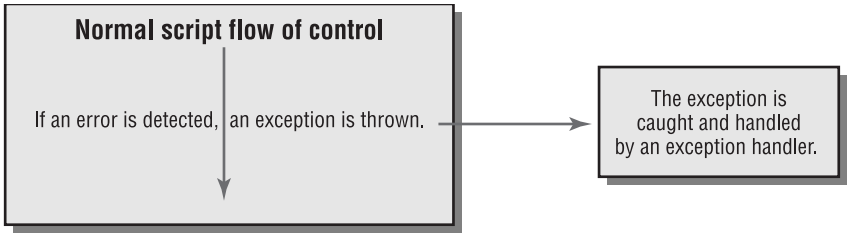
Internet Explorer 5 uvodi JScript 5, koji predstavlja najobimnije okruženje za skriptovanje podržano od bilo kog browsera (do danas). Jedna od najboljih karakteristika JScript 5 je da on podržava naredbe `throw`, `try` i `catch`. Njih tri rade zajedno, da bi omogućile dodatnu kontrolu nad bilo kojom greškom i nad bilo kojim izuzetkom koji se može pojaviti u Vašem kodu. Ako ste Java programer, ove naredbe će Vam biti više nego bliske. JScript koristi istu sintaksu za ove naredbe kao i Java.

JScript 5 uvodi i objekat greške (`error` objekat), koji pruža mogućnost za obradu grešaka, kao što su sintaksne greške i greške u toku izvršavanja programa, koje se mogu pojaviti u Vašim dokumentima. Ovaj objekat greške je mnogo detaljnije obrađen u Poglavlju 4, "Upravljanje događajima", i Poglavlju 5, "Rad sa objektima". Za sada, treba da znate da on postoji i da se koristi sa naredbama `throw`, `try` i `catch`.

NAPOMENA

`Throw`, `try` i `catch` naredbe jedino rade u Internet Exploreru 5. Zbog toga što će ove naredbe postati deo ECMAScripta verzije 2, potpuno je logično očekivati da one budu podržane i u Navigatoru 5. ■

Izuzetak je greška koju skript generiše da bi privukao pažnju na problem koji je otkriven tokom izvršavanja skripta.Vaš skript generiše izuzetak tako da greška može da bude obrađena i otklonjena. Govoreći o načinu generisanja izuzetaka, Vi generišete izuzetak tako što ga aktivirate. Kodovi koji obrađuje izuzetke aktivirane Vašim skriptom (ili JavaScript interpretatorom) poznati su kao obrađivači izuzetaka. Za te obrađivače se kaže da "hvataju" (catch) izuzetke. Slika 3.9 pruža pregled procesa obrade izuzetaka.



SLIKA 3.9 *Kako funkcioniše obrada izuzetaka*

Throw naredba se koristi za aktiviranje izuzetaka. Njena sintaksa je sledeća:

```
throw izraz
```

Vrednost *izraza* se koristi za identifikaciju tipa greške koja se pojavila. Na primer, sledeća naredba aktivira izuzetak pod nazivom BadInputFromUser.

```
throw "BadInputFromUser"
```

SAVET

Iako nije obavezno, dobro je aktivirati izuzetke string vrednostima. Ovo omogućava da se kod lakše razume i da se greške lakše otklanjaju. ■

Naredbe `try` i `catch` rade zajedno kao podrška obradi izuzetaka. Njihova sintaksa je sledeća:

```
try {
 naredba gde izuzetak mo`e biti aktiviran
}
catch(errorVariable) {
 naredba koja obradjuje izuzetak
}
```

`Try` naredba okružuje naredbe za koje treba da se izvrši obrada izuzetaka. Neposredno, zatim, sledi `catch` naredba, koja izvršava obradu izuzetaka (ili "hvatanje" izuzetaka). Promenljiva *errorVariable* se koristi da ukaže na bilo koji izuzetak koji nastane. Njemu se pridružuje instanca objekta greške. Ako se jedan izuzetak aktivira tokom obrade naredbi unutar `try` naredbe, promenljivoj *errorVariable* dodeljuje objekat greške koji identifikuje izuzetak i kontrola se momentalno prebacuje naredbama unutar `catch` naredbe. Ako se ne aktivira ni jedna greška tokom obrade naredbi unutar `try` naredbe, `catch` naredba se preskače i kontrola se prebacuje naredbi koja sledi `catch` naredbu.

U listingu 3.9 prikazana je prosta HTML datoteka koja sadrži dva skripta - jedan u zaglavlju dokumenta, a drugi u telu dokumenta. Skript u telu dokumenta izvršava petlju za i koje se kreće od 0 do 21, prosleđujući ga, kao parametar, funkciji `primeTest()`. Skript u zaglavlju dokumenta definiše `primeTest()` funkciju. Ova funkcija proverava parametar n, da bi odredila da li parametar predstavlja prost broj. Ona sadrži try naredbu u okviru koje se izvodi testiranje prostih brojeva. Prva `if` naredba proverava da li je n između 1 i 20. Ako nije, aktivira se izuzetak *It's out of range* i kontrola se prosleđuje `catch` naredbi. Ako je `n` između 1 i 20, kontrola se prosleđuje `for` naredbi.

Listing 3.9 Primer obrade izuzetaka (ch03-09.htm)

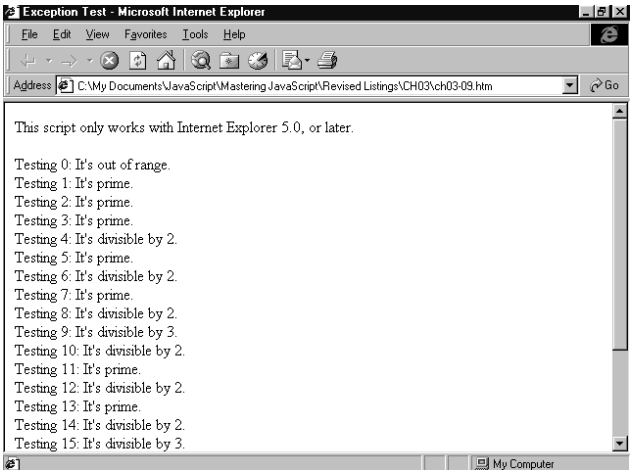
```
<HTML>
<HEAD><TITLE>Exception Test</TITLE></HEAD>
<SCRIPT LANGUAGE="JavaScript"><!—
function primeTest(n) {
 document.write("Testing "+n+": ")
 try {
  if(n < 1 || n > 20) throw "It's out of range"
  for(var i = 2; i < n; ++i)
   if(n % i == 0) throw "It's divisible by " + i
  document.writeln("It's prime.<BR>")
 }
 catch (exception) {
  document.writeln(exception+".<BR>")
 }
}
—><SCRIPT>
<BODY>
<P>This script only works with Internet Explorer 5.0, or later.</P>
<SCRIPT LANGUAGE="JavaScript"><!—
for(i = 0; i <= 21; ++i) {
```

```
 primeTest(i)
}
—></SCRIPT>
</BODY>
</HTML>
```

For naredba izvršava petlju za i koje se kreće od 2 do `n`. Ako je n moduo i nula, onda je `n` prost broj, izuzetak *It's divisible by* se aktivira, a kontrola se prosleđuje `catch` naredbi. U suprotnom, *It's prime* se upisuje u prozor dokumenta i izlazi se iz funkcije.

`Catch` naredba upisuje izuzetak u prozor dokumenta.

Na slici 3.10 prikazan je izlaz korišćenjem funkcije `primeTest()`, kao rezultata obrade skripta.

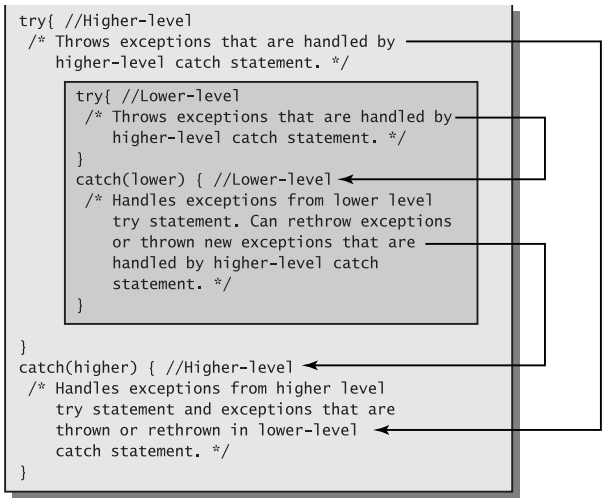


SLIKA 3.10 *Rezultati obrade izuzetaka (listing 3.9)*

Ugnježdene try naredbe i ponovno aktiviranje izuzetaka

JScript 5 dozvoljava da ugnjezdite `try` naredbe jednu u okviru druge. Radeći to, možete da izvedete obradu izuzetaka u više nivoa. Slika 3.11 pruža pregled ovog prilaza. `Try` i `catch` naredbe nižeg nivoa obrade izuzetaka se nalaze u okviru try naredbe višeg nivoa. Ako se izuzetak pojavi u `try` naredbi nižeg nivoa, njega "hvata" `catch` naredba nižeg nivoa. Ako naiđe na grešku pri obradi izuzetaka, `catch` naredba nižeg nivoa može da aktivira novi izuzetak, koji će obraditi `catch` naredba višeg nivoa. `Catch` naredba nižeg nivoa može, takođe, da ponovo aktivira bilo koji izuzetak koji zahteva obradu na višem nivou.

Listing 3.10 pruža primer ugnježdene obrade i ponovnog aktiviranja izuzetaka. On se sastoji iz dva skripta - jednog koji je u zaglavlju dokumenta i drugog koji je u telu dokumenta. Skript u telu dokumenta izvršava petlju za i od 1 do 100 i poziva funkciju `selected()` da odredi da li vrednost promenljive i treba da bude prikazana. Ako funkcija `selected()` vrati *`true`*, onda se i prikazuje.



SLIKA 3.11 *Ugnježdene obrade izuzetaka dozvoljavaju izuzecima da ponovo budu aktivirani.*

Listing 3.10 Ugnježdena obrada izuzetaka (ch03-10.htm)

```
<HTML>
<HEAD><TITLE>Exception Test</TITLE></HEAD>
<SCRIPT LANGUAGE="JavaScript"><!—
function selected(n) {
 try {
  try {
   if (n % 3 == 2) throw "No way"
   if (n % 3 == 1) throw "Try again"
  }
  catch (ex1) {
   if(ex1 == "Try again")
    if (n % 5 == 0) throw "Try again"
   return false
  }
  if (n % 7 == 3) throw "Try again"
  if (n % 7 != 0) throw "No way"
 }
 catch (ex2) {
  if(ex2 != "Try again") return false
  if(n % 11 != 0) return false
 }
 return true
}
—></SCRIPT>
<BODY>
<P>This script only works with Internet Explorer 5.0, or later.</P>
<SCRIPT LANGUAGE="JavaScript"><!—
for(i = 1; i <= 100; ++i) {
 if (selected(i)) document.writeln(i+"<BR>")
}
—></SCRIPT>
```

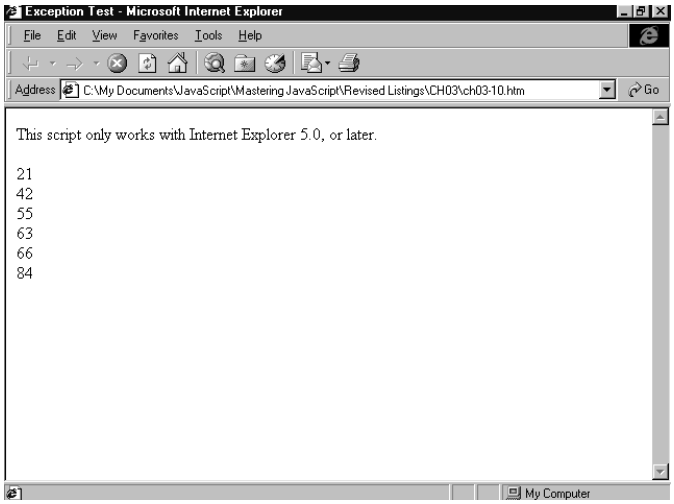
```
</BODY>
</HTML>
```

Obrada koja nas interesuje se sprovodi u funkciji `selected()`. Ova funkcija prihvata parametar n i sprovodi seriju testova nad njim, da bi utvrdila da li funkcija treba da vrati *true*, ili *false*. Funkcija `selected()` se sastoji iz dva skupa `try/catch` naredbi, gde je jedan skup ugnježden u okviru drugog. Unutrašnja `try` naredba proverava vrednost n % 3. Ako je ova vrednost jednaka 2, aktivira se No *way* izuzetak. Ako je 1, aktivira se *Try again* izuzetak. `Catch` naredba obrađuje *Try again* izuzetak na taj način što proverava da li je `n` deljivo sa 5. Ako je deljivo, *Try again* izuzetak se ponovo aktivira. U suprotnom, `catch` naredba vraća vrednost *false*.

Dve `if` naredbe se pojavljuju nakon unutrašnje `try` naredbe, ali pre završetka spoljašnje `try` naredbe. Ako je n % 7 jednako 3, aktivira se *Try again* izuzetak. U suprotnom, ako `n` % 7 nije jednako 0, aktivira se No *way* izuzetak.

Spoljašna `catch` naredba vraća vrednost *false* za No *way* izuzetak i za *Try again* izuzetak kada n nije deljivo sa 11. Ako vrednost n uspe da se probije kroz "gustiš" izuzetaka, funkcija `selected()` vraća vrednost true.

Na slici 3.11 prikazan je izlaz koji generiše listing 3.10. Razlog zbog koga sam stavio na jedno mesto ovakav složeni skup izuzetaka je da možete da istražujete kroz skript kod, da biste videli kako je izlaz na slici 3.12 generisan. Radeći to, steći ćete jasnu predstavu kako ugnježdene obrade izuzetaka i ponovno aktiviranje izuzetaka funkcionišu.



SLIKA 3.12 *Izlaz generisan ugnježdenim obradama izuzetaka (listing 3.10)*

Zaključak

U ovom poglavlju naučili ste sve operatore koje obezbeđuje JavaScript i kako se izrazi izračunavaju korišćenjem prioriteta operatora. Naučili ste kako se koriste JavaScript programske naredbe i

kako se razvijaju prosti skriptovi koji demonstriraju upotrebu ovih naredbi. Takođe ste naučili kako da kreirate i pozivate funkcije. U sledećem poglavlju proširićete JavaScript programersko iskustvo, učeći kako JavaScript podržava obradu korisnički-generisanih događaja.