POGLAVLJE 2
PHP

U ovom poglavlju uvodimo jezik za pisanje skriptova PHP. On je sličan programskim jezicima visokog nivoa, kao što su C, Perl, Pascal, FORTRAN i Java, a programeru koji ima iskustva sa jednim od ovih jezika ne bi trebalo da bude teško da nauči PHP. Namena ovog poglavlja je da posluži kao uvod u PHP; ono nije zamišljeno kao vodič za programiranje. Polazimo od pretpostavke da vam je već poznato programiranje u jednom od jezika visokog nivoa.

U ovom poglavlju obrađene su sledeće teme:

- Osnove PHP-a: struktura skripta, promenljive, podržani tipovi podataka, konstante, izrazi i konverzije tipova podataka
- Iskazi za uslovno izvršavanje i grananje koje PHP podržava, kao što su `if`, `if ... else` i `switch`
- Petlje
- Nizovi i biblioteka funkcija za rad sa nizovima
- Znakovne vrednosti i biblioteka funkcija za rad sa znakovnim vrednostima
- Regularni izrazi
- Funkcije za rad s datumima i vremenom
- Funkcije za rad sa celobrojnim vrednostima i vrednostima sa pokretnim zarezom
- Kako se pišu funkcije, komponente koje se mogu koristiti u različitim aplikacijama, kako se određuje doseg i tip promenljivih
- Uvod u podršku za objektno orijentisano programiranje koju pruža PHP
- Uobičajene greške koje prave početnici u PHP-u i načini da se one izbegnu.

Programerima koji su početnici u PHP-u savetujemo da pročitaju odeljak „Uvod u PHP“, u kome je opisana osnovna struktura PHP skripta i umetanje u HTML kôd, kao i način na koji PHP radi s promenljivama i tipovima podataka. Naredni odeljci, „Uslovno izvršavanje i grananje u kodu“ i „Petlje“, opisuju iskaze za uslovno izvršavanje koda i strukture u obliku petlji , što bi trebalo da vam bude poznato. Zatim ćemo obraditi kratak primer koji objedinjuje mnoge osnovne koncepte PHP-a.

U preostalom delu poglavlja, koji je posvećen naprednijim mogućnostima PHP-a, navodimo odabrane funkcije iz raznih biblioteka i opisujemo neke uobičajene greške koje programeri prave kada uče PHP. Ovaj deo gradiva sad možete površno pročitati, a kasnije, kada budete čitali poglavlja 4 do 13 i dok budete pisali PHP kôd, može vam poslužiti kao referenca. Međutim, programerima koji počinju u PHP-u savetujemo da pročitaju barem početni deo odeljaka o radu s nizovima i znakovnim vrednostima da bi shvatili način na koji PHP podržava te koncepte jer postoje značajne razlike u poređenju s drugim programskim jezicima.

U ovoj knjizi ne pokušavamo da obradimo baš svaku funkciju u svakoj biblioteci koju PHP podržava. Međutim, u dodatku E naći ćete kratak opis podržanih biblioteka. U narednim poglavljima razmatramo specijalizovane funkcije iz biblioteka koje podržavaju koncepte i tehnike opisane u ovom poglavlju.

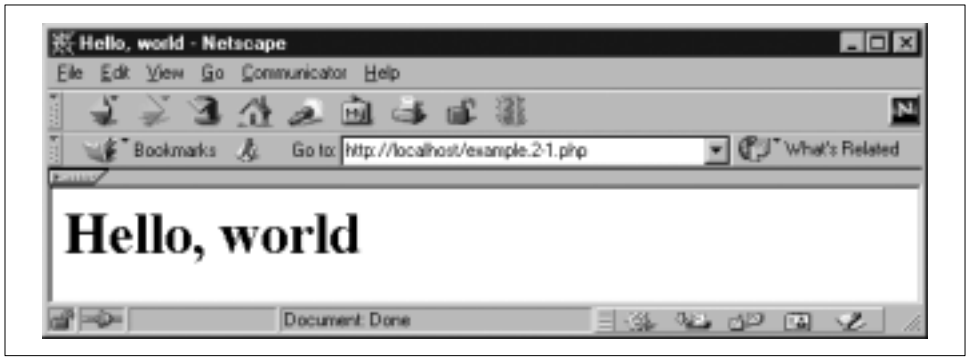
Uvod u PHP

Tekuća verzija PHP-a je PHP4, koju ćemo u celoj knjizi nazivati PHP. Dok smo pisali ovu knjigu, tekuće izdanje bilo je 4.0.6.

PHP je rekurzivna skraćenica za izraz *PHP: Hypertext Preprocessor*; izraz je skovan po ugledu na *GNU*, što je skraćenica za *GNU's Not Unix*, od čega je i započela ta čudna moda izmišljanja skraćenica. Ovo ime ne opisuje najbolje šta je PHP i za koje namene se obično koristi. PHP je jezik za pisanje skriptova čiji se kôd obično ugrađuje u HTML kôd, odnosno kombinuje s njim. Na raspolaganju su i mnogobrojne odlične biblioteke funkcija koje omogućavaju brz i prilagodljiv pristup bazama podataka. To je savršena alatka za razvijanje dela logike aplikacije koja se ugrađuje u srednji sloj troslojne aplikacije.

Osnove PHP-a

Primer 2-1 prikazuje prvi PHP skript u ovoj knjizi, programerima dobro poznati „Hello, world". Kada čitač Weba pošalje zahtev za ovim skriptom, on se izvršava na Web serveru, a rezultujući HTML dokument šalje se nazad čitaču koji ga prikazuje (slika 2-1).



Slika 2-1. Formatiran rezultat izvršavanja primera 2-1, prikazan u čitaču Netscape.

Primer 2-1. Dobro poznati Hello, world napisan u PHP-u

```
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD HTML 4.0 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd" >
<html>
<head>
  <title>Hello, world</title>
</head>
<body bgcolor="#ffffff">
  <h1>
  <?php
    echo "Hello, world";
  ?>
  </h1>
</body>
</html>
```

Primer 2-1 ilustruje osnovne odlike PHP skripta. To je mešavina HTML koda – u ovom primeru, veći deo koda je HTML kôd – i PHP skripta. Skript:

```
<?php
  echo "Hello, world";
?>
```

samo ispisuje reči pozdrava: „Hello, world".

PHP skript prikazan u primeru 2-1 gotovo je beskoristan: isti rezultat mogli smo da postignemo i da smo poruku ugradili direktno u HTML kôd. Budući da se PHP kôd tako dobro kombinuje s HTML kodom, upotreba PHP-a za ispisivanje statičkog teksta znatno je jednostavnija i manje zanimljiva mogućnost od upotrebe drugih programskih jezika visokog nivoa. Međutim, ovaj primer ipak ilustruje više odlika PHP-a:

- Skript počinje oznakom `<?php`, a završava se oznakom `?>`; ili jednostavnije, `<?` i `?>`. Duži oblik početne oznake omogućava da se izbegnu zabune s drugim komandama koje mogu da se upotrebe u HTML kodu. U ovoj knjizi koristimo oba stila oznaka.

  Mogu se podesiti i koristiti i drugi stilovi za početne i završne oznake, kao što je HTML stil koji se koristi za skriptove napisane u JavaScriptu ili drugom skript jeziku: `<script language="PHP">` i `</script>`.
- Beline (razmaci, prelomi redova, uvlake itd.) zanemaruju se i služe samo da programeru olakšaju čitanje koda. Na primer, navedeni skript može da se napiše u sažetom obliku kao `<?php echo "Hello, world";?>` da bi se dobio isti rezultat. Dozvoljena je svaka kombinacija razmaka, znakova za tabulator, preloma redova itd. koju možete upotrebiti da biste razdvojili naredbe koda.
- PHP skript se sastoji od niza iskaza (naredaba), a svaki mora da se završava tačkom i zarezom (;). Naš veoma jednostavan primer sastoji se od samo jednog iskaza: `echo "Hello, world";`.
- PHP skript se može nalaziti na bilo kom mestu unutar datoteke i biti kombinovan s HTML kodom. Primer 2-1 sadrži samo jedan skript, ali jedna datoteka može sadržavati neograničen broj PHP skriptova.
- Kada se izvrši PHP skript, ceo njegov kôd, uključujući i početnu i završnu oznaku, `<?php` i `?>`, zamenjuje se u HTML dokumentu rezultatom izvršavanja PHP koda.

Kada na stranicama knjige prikazujemo više redova koda koji pripadaju dužem skriptu, obično izostavljamo početnu i završnu oznaku za skript.

Mogućnost ugradnje neograničenog broja skriptova u HTML dokument jedna je od najkorisnijih odlika PHP-a. To ilustruje primer 2-2: promenljivoj `$outputString= "Hello, world"` dodeljuje se vrednost pre početka formatiranja HTML dokumenta, a zatim se ta znakovna vrednost ispisuje dvaput, unutar oznaka `<title>` i `<body>`. U nastavku poglavlja biće više reči o promenljivama i načinu na koji se one koriste.

Primer 2-2. Ugrađivanje tri skripta u isti dokument.

```
<?php $outputString = "Hello, world"; ?>
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD HTML 4.0 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd" >
<html>
<head>
  <title><?php echo $outputString; ?></title>
</head>
<body bgcolor="#ffffff">
  <h1><?php echo $outputString; ?></h1>
</body>
</html>
```

Lakoća s kojom se više skriptova može ugraditi u isti HTML dokument može dovesti do koda koji se teško čita i održava. Trebalo bi da dobro razmislite kako ćete rasporediti PHP kôd u HTML dokumentu; u poglavlju 13 opisujemo kako se pomoću šablona može razdvojiti PHP kôd od HTML koda.

Pisanje PHP skriptova

PHP skriptovi mogu se pisati pomoću svakog editora za tekst*, kao što je *joe*, *vi*, *nedit*, *emacs* ili *pico*. Ako PHP snimite u datoteku s nastavkom .php u neki direktorijum podešen kao korenski za dokumente, Apache će izvršiti skript kada primi zahtev za tim resursom. U skladu sa uputstvima za instaliranje datim u dodatku A, korenski direktorijum za dokumente je:

```
/usr/local/apache/htdocs/
```

Pogledajmo šta se događa kada se skript iz primera 2-1 snimi u datoteku:

```
/usr/local/apache/htdocs/example.2-1.php
```

Apache – kada je konfigurisan sa PHP modulom – izvršava skript kada primi zahtev za resursom, *http://localhost/example.2-1.php*, pod pretpostavkom da čitač Weba radi na istom računaru kao i Web server.

Ako su za direktorijum zadata prava koja ne dozvoljavaju formiranje novih datoteka u korenskom direktorijumu za dokumente, mogu se iskoristiti matični direktorijumi

* Iako se mogu upotrebljavati svi znaci u kojima najznačajniji bit ima vrednost 1, u PHP skriptovima se obično koriste samo znaci iz 7-bitnog ASCII skupa.

korisnika sistema. Ako ste primenili uputstva za instaliranje data u dodatku A, svaki korisnik ima pravo da otvara nove direktorijume ispod svog matičnog direktorijuma, a Web server ima pravo čitanja sadržaja tih direktorijuma:

```
mkdir ~/public_html
chmod a+rx~/public_html
```

Kôd primera možete onda upisati u datoteku sa sledećim imenom:

```
~/public_html/example.2-1.php
```

Toj datoteci se zatim može pristupati pomoću URL-a *http://localhost/~user/example.2-1.php*, gde je *user* ime korisnikovog naloga.

Komentari

Komentari se mogu upisivati u kôd primenom uobičajenih stilova u drugim programskim jezicima visokog nivoa. To mogu biti sledeći stilovi:

```
// Ovo je komentar koji se sastoji od jednog reda.

# Ovo je još jedan komentar koji se sastoji od jednog reda.

/* A ovako izgleda komentar
   koji se prostire na
   više redova */
```

Prosleđivanje izlaznih podataka pomoću komandi echo i print

Iskaz `echo` koji smo upotrebili u primerima 2-1 i 2-2 često se koristi i može da prosledi bilo koju vrstu podatka. Za istu svrhu možete koristiti i iskaz `print`. Pogledajmo nekoliko primera:

```
Echo "Hello, world";

// print daje isti rezultat
print "Hello, world";

// Mogu se prosleđivati i brojevi
echo 123;

// kao i vrednosti promenljivih
echo $outputString;
```

Razlika između iskaza `print` i `echo` sastoji se u tome što `echo` prihvata više od jednog argumenta:

```
echo "Hello, ", "world";
```

Za prosleđivanje izlaznih podataka možete upotrebiti i prečicu. Naredni, veoma kratak skript prosleđuje vrednost promenljive `$temp`:

```
<?=$temp; ?>
```

Iskazi `print` i `echo` često se pišu sa zagradama:

```
echo "hello";
// isto je što i
echo ("hello");
```

Upotreba zagrada ne menja ponašanje iskaza `print`. Međutim, kada ih upotrebite u iskazu `echo`, dozvoljen je samo jedan izlazni parametar.

Iskazi `echo` i `print` mogu se upotrebiti u većini slučajeva i mogu proslediti svaku kombinaciju statičkog teksta, brojeva, nizova i drugih tipova podataka opisanih u nastavku ovog poglavlja. Prosleđivanje formatiranih podataka pomoću iskaza `printf` opisano je u odeljku „Znakovne vrednosti“, u nastavku ovog poglavlja.

Znakovni literali

PHP podržava upotrebu znakovnih literala koji se mogu pisati između dvostrukih ili jednostrukih navodnika. Ako znakovna vrednost sadrži dvostruke navodnike, najjednostavnije rešenje je da je uokvirite jednostrukim navodnicima:

```
echo 'Ovo može';
echo 'isto kao i ovo.';

// Nekoliko znakovnih vrednosti koje sadrže navodnike
echo "Ova znakovna vrednost sadrži ': jednostruki navodnik!";
echo 'Ova znakovna vrednost sadrži ": dvostruki navodnik!';
```

Navodnik možete zameniti Escape sekvencama, na sledeći način:

```
echo "Ova znakovna vrednost sadrži \": dvostruki navodnik!";
echo 'Ova znakovna vrednost sadrži \': jednostruki navodnik!';
```

Jedna od korisnih odlika PHP-a jeste mogućnost da se vrednost promenljive umetne u znakovni literal. PHP analizira znakovne vrednosti omeđene dvostrukim navodnicima i zamenjuje imena promenljivih njihovim vrednostima:

```
$kapacitet = 45;
$vozilo = "autobus";
$poruka = "Ovaj $vozilo može da preveze $kapacitet putnika.";

// ispisuje "Ovaj autobus može da preveze 45 putnika."
echo $poruka;
```

Ako u znakovnu vrednost uokvirenu dvostrukim navodnicima treba da umetnete obrnutu kosu crtu ili znak za dolar, možete upotrebiti sekvence znakova `\\` i `\$`. Znakovna vrednost uokvirena jednostrukim navodnicima ne obrađuje se na isti način kao znakovna vrednost napisana između dvostrukih navodnika i može se upotrebiti za ispisivanje teksta nalik sledećem:

```
'znakovna vrednost koja sadrži \ i $'
```

Analiziranje znakovnih literala detaljnije je opisano u odeljku „Znakovne vrednosti“.

Promenljive

Promenljive se u PHP-u označavaju znakom za dolar kojim počinje ime promenljive. One se ne deklarišu i nemaju određen tip podataka dok im se ne dodeli vrednost. U narednom primeru, promenljivoj `$var` dodeljuje se vrednost izraza, u ovom slučaju celobrojna vrednost 15. Zahvaljujući tome, promenljiva `$var` definisana je kao tip integer (celobrojna).

```
$var = 15;
```

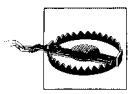
Pošto se promenljiva u ovom primeru koristi tako što joj se dodeljuje vrednost, ona je implicitno deklarisana. Promenljive u PHP-u su jednostavne: na mestu gde ih upotrebite, njihov tip podataka se implicitno definiše – ili ponovo definiše – a promenljiva se implicitno deklariše.

Tip promenljive može da se menja tokom njenog životnog veka. Pogledajmo sledeći primer:

```
$var = 15;
$var = "Maca mačkica";
```

Ovo je sasvim prihvatljivo u PHP-u. Dodeljivanjem druge vrednosti, tip promenljive `$var` menja se od celobrojnog u znakovni tip. To što je PHP-u dopušteno da menja tip promenljive čim se promeni kontekst veoma je zgodno, ali i pomalo opasno.

Budući da PHP pravi razliku između malih i velikih slova u imenima promenljivih, `$Promenljiva`, `$promenljiva`, `$PRomenljiva` i `$PROMENLJIVA` jesu različite promenljive.



Jedan najčešćih uzroka grešaka u PHP-u jeste previđanje da je nenamerno definisano više promenljivih sa – na prvi pogled – jednakim imenima. Fleksibilnost PHP-a je veoma korisna odlika, ali i opasna. U nastavku ovog poglavlja opisano je kako treba podesiti obaveštavanje o greškama u PHP-u da bi se generisale upozoravajuće poruke kada se koriste promenljive kojima prethodno nije dodeljena vrednost.

Tipovi podataka

PHP podržava četiri skalarna (prosta) tipa podataka: logički (engl. *boolean*), celobrojni (engl. *integer*), sa pokretnim zarezom (engl. *float*) i znakovni (engl. *string*) – kao i dva složena: niz (engl. *array*) i objekat (engl. *object*).

U ovoj knjizi, a naročito u ovom poglavlju, predstavljamo prototipove funkcija za koje zadajemo tipove argumenata i povratnih vrednosti. Postoji veliki broj funkcija koje prihvataju argumente i daju povratne vrednosti različitih tipova, koje ćemo nazivati opštim imenom *mešovit_tip*.

Promenljive skalarnih tipova mogu uvek sadržati samo po jednu vrednost. Promenljive složenih tipova – array ili object – sadrže više vrednosti skalarnog ili složenog tipa.

Tipovima podataka array i object posvećeni su posebni odeljci u nastavku ovog poglavlja. Druge aspekte promenljivih, među kojima su globalne promenljive i doseg promenljivih, razmatramo u odeljku o funkcijama koje korisnik definiše.

Promenljive tipa *Boolean* (logičke) ne mogu biti jednostavnije: mogu imati samo vrednost `true` (tačno, istinito) ili `false` (netačno, neistinito). Evo dva primera dodeljivanja vrednosti promenljivama tipa Boolean:

```
$promenljiva = false;
$test = true;
```

Promenljiva tipa *integer* sadrži celobrojnu vrednost, dok promenljiva tipa *float* sadrži vrednost koja se sastoji od celobrojnog i od decimalnog dela. Broj 123,01 je tipa float, kao i 123,0. Broj 123 je ceo broj. Pogledajmo sledeća dva primera:

```
// Ovo je celobrojna vrednost
$var = 6;

// Ovo je vrednost tipa float
$var2 = 6.0;
```

Vrednost tipa float može se predstaviti i u eksponencijalnom obliku:

```
// Ovo je vrednost tipa float jednaka 1120
$var3 = 1.12e3;

// Ovo je takođe vrednost tipa float jednaka 0,02
$var4 = 2e-2;
```

U prethodnom delu ovog poglavlja videli ste primere znakovnih vrednosti kada smo uveli funkcije `echo()` i `print()`. Znakovni literali opisani su u odeljku „Znakovni literali“. Pogledajmo sledeće primere promenljivih znakovnog tipa:

```
$promenljiva = "Ovo je znakovna vrednost.";
$test = "I ovo je znakovna vrednost.";
```

Konstante

Konstanta je ime kome je pridružena prosta, tj. skalarna vrednost. Na primer, vrednosti tipa Boolean `true` i `false` jesu konstante koje imaju vrednost 1, odnosno 0. U skriptovima se često deklarišu konstante. Pogledajte sledeći primer deklaracije konstante:

```
define("pi", 3.14159);

// ova naredba ispisuje vrednost 3,14159
echo pi;
```

Imena konstanti ne počinju znakom $; kada ih definišete, ne možete im menjati vrednost u kodu; dostupne su u svim delovima skripta, bez obzira na mesto na kome su deklarisane i mogu da imaju samo proste, skalarne vrednosti.

Konstante su korisne zato što omogućavaju grupisanje parametara koji su interni za skript. Kada se promeni vrednost jednog parametra – na primer, kada definišete nov maksimalan broj redova po Web stranici – dovoljno je da vrednost parametra definisanog u obliku konstante promenite na samo jednom mestu, a ne na svakom mestu u celom kodu.

Izrazi, operatori i dodeljivanje vrednosti promenljivama

Već smo razmotrili više jednostavnih primera dodeljivanja vrednosti, u kojima se pomoću znaka jednakosti promenljivoj dodeljuje vrednost izraza. Većina operacija dodeljivanja numeričkih vrednosti i izraza koje poznajete iz drugih programskih jezika visokog nivoa ima isti oblik i u PHP-u. Evo nekoliko primera:

```
// Dodeljivanje vrednosti promenljivoj
$var = 1;

// Zbrajanje celobrojnih vrednosti da bi se
// dobio rezultat celobrojnog tipa
$var = 4 + 7;

// Oduzimanje, množenje i deljenje čiji rezultat
// može biti tipa float ili integer, u zavisnosti od
// početne vrednosti promenljive $var
$var = (($var - 5) * 2) / 3;

// Sve tri naredbe povećavaju vrednost $var za 1
$var = $var + 1;
$var += 1;
$var++;

// A sve ove naredbe oduzimaju 1 od $var
$var = $var - 1;
$var -= 1;
$var--;

// Množenje tekuće vrednosti sa 2
$var = $var * 2;
$var *= 2;

// Deljenje tekuće vrednosti sa 2
$var = $var / 2;
$var /= 2;

// Na isti način se radi i s vrednostima tipa float
$var = 123.45 * 28.2;
```

Za složenije operacije možete koristiti mnogobrojne funkcije koje ćete naći u PHP--ovoj biblioteci matematičkih funkcija. Neke od njih su opisane u odeljku „Funkcije za rad s vrednostima tipa integer i float“

Izraz se dodeljuje promenljivama znakovnog tipa na isti način kao i promenljivama numeričkog tipa:

```
// Dodeljivanje promenljivoj vrednosti znakovnog tipa
$var = "ovo je proba";

// Spajanje dva znakovna niza
$var = "ovo je" . " proba";

// Nadovezivanje jedne znakovne
// vrednosti na kraj druge
```

```
$var = "ovo je";
$var = $var . " proba";

// Kraći način nadovezivanja jedne
// znakovne vrednosti na kraj druge
$var .= " proba";
```

Izrazi

Izrazi se u PHP-u formulišu veoma slično kao u drugim programskim jezicima. Svaki izraz je sastavljen od literala (koji mogu biti celi brojevi, znakovne vrednosti, decimalni brojevi, logičke vrednosti, imena memorijskih nizova i objekata), operatora i poziva funkcija koje daju povratne vrednosti. Izraz ima vrednost i tip vrednosti; na primer, izraz `4 + 11` ima vrednost *11*, a njegov tip je *integer* (celobrojni), dok izraz `"Knjiga"` ima vrednost *Knjiga*, a tip je *string* (znakovni). PHP automatski obavlja konverziju tipova kada vrednosti različitih tipova kombinuje unutar izraza. Na primer, izraz `4 + 7.0` sadrži vrednost tipa integer i vrednost tipa float; u ovom slučaju PHP pretvara tip integer u float, a rezultat je takođe tipa float. Rezultat konverzije tipova je u većini slučajeva lako predvidljiv; međutim, postoji i nekoliko zamki koje su opisane u ovom odeljku.

Prioritet operatora

Prioritet operatora u izrazima sličan je prioritetu operatora u drugim jezicima. Množenje i deljenje izvode se pre oduzimanja i sabiranja, itd. Međutim, oslanjanje isključivo na podrazumevani prioritet operatora dovodi do teško razumljivog i zbunjujućeg koda. Umesto da napamet učite pravila, preporučujemo da upotrebom zagrada sastavljate nedvosmislene izraze jer zagrade imaju najviši prioritet pri izračunavanju vrednosti izraza.

Na primer, u narednom bloku koda promenljivoj `$var` dodeljuje se vrednost 32 zbog prioriteta množenja nad sabiranjem:

```
$var = 2 + 5 * 6;
```

Rezultat je lakše razumljiv kada se upotrebe zagrade:

```
$var = 2 + (5 * 6);
```

Konverzija tipova podataka

U PHP-u postoji više mehanizama koji omogućavaju da se postojeći tip promenljive obrađuje kao drugi tip. Tip promenljive može se izričito pretvoriti u drugi tip pomoću sledećih funkcija:

```
string strval(mešovit_tip promenljiva)
integer intval(mešovit_tip promenljiva)
float floatval(mešovit_tip promenljiva)
```

Funkcija `settype(mešovit_tip promenljiva, novi_tip)` izričito menja tekući tip promenljive u *novi_tip*, gde *novi_tip* može da bude *array*, *boolean*, *float*, *integer*, *object* ili *string*.

PHP podržava preslikavanje tipova (engl. *type-casting*), veoma slično jeziku C, kojim se omogućava pretvaranje tipa promenljive unutar izraza. Kada napišete ime novog tipa u zagradama ispred promenljive, PHP pretvara njen tekući tip u zadati:

`(int) $var`	Preslikavanje u tip integer
ili `(integer) $var`	
`(bool) $var`	Preslikavanje u tip Boolean
ili `(boolean) $var`	
`(float) $var, (double) $var`	Preslikavanje u tip float
ili `(real) $ var`	
`(string) $var`	Preslikavanje u tip string
`(array) $var`	Preslikavanje u tip array
`(object) $var`	Preslikavanje u tip object

Većina pravila za konverziju tipova izvedena je na osnovu zdravog razuma, ali neke konverzije možda neće biti jasne na prvi pogled. Tabela 2-1 prikazuje kako se pojedine vrednosti `$var` preslikavaju pomoću operatora `(int)`, `(bool)`, `(string)` i `(float)`.

Tabela 2-1. Primeri konverzije tipova pomoću operatora za preslikavanje

Vrednost $var	(int) $var	(bool) $var	(string) $var	(float) $var
`null`	0	`false`	`""`	0
`true`	1	`true`	`"1"`	1
`false`	0	`false`	`""`	0
0	0	`false`	`"0"`	0
3.8	3	`true`	`"3.8"`	3.8
`"0"`	0	`false`	`"0"`	0
`"10"`	10	`true`	`"10"`	10
`"6 cm"`	6	`true`	`"6 cm"`	6
`"nešto"`	0	`true`	`"nešto"`	0

Automatska konverzija tipova

Automatska konverzija tipova nastaje kada se u istom izrazu zadaju dve promenljive različitih tipova ili kada se promenljiva jednog tipa prosledi kao argument funkciji koja očekuje argument drugog tipa. Kada se promenljiva jednog tipa upotrebljava kao da je drugog tipa, PHP automatski menja njen tip u onaj koji je neophodan.

Pri automatskoj konverziji tipova takođe važe pravila navedena u tabeli 2-1. Nekoliko jednostavnih primera pokazuje šta se događa kada se znakovne vrednosti sabiraju s vrednostima celobrojnog tipa ili kada se brojevi nadovezuju na znakovne vrednosti:

```
// $var dobija vrednost 115 i tip integer
$var = "100" + 15;
```

```
 // $var dobija vrednost 115.0 i tip float
$var = "100" + 15.0;

// $var dobija vrednost "39 stepenica"
$var = 39 . " stepenica";
```

Nisu sve konverzije tipova tako očigledne i mogu da budu uzrok grešaka koje se teško otkrivaju:

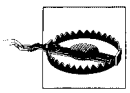
```
// $var dobija vrednost 39 i tip integer
$var = 39 + " stepenica";

 // $var dobija vrednost 42 i tip integer
$var = 40 + "2 siva miša";

// $var dobija tip float, ali šta je to
$var = "test" * 4 + 3.14159;
```

Pri automatskoj konverziji tipova može da se promeni tip promenljive. Pogledajmo sledeće primere:

```
$var = "1";  // $var ima vrednost "1" tipa string
$var += 2;   // $var ima sada vrednost 3 tipa integer
$var /= 2;   // $var ima sada vrednost 1.5 tipa float
$var *= 2;   // $var ima sada vrednost 3 i dalje tipa float
```



Pažljivo razmotrite slučajeve kada se nelogičke vrednosti obrađuju kao logičke (Boolean). Mnoge funkcije iz biblioteka PHP-a daju vrednosti različitog tipa: `false` ukoliko nije mogao da se izračuna ispravan rezultat, ili ispravan rezultat. Ispravna povratna vrednost 0, 0.0, "0", znakovni niz dužine 0, ili prazan niz tumače se kao `false` kada se koriste kao vrednosti logičkog tipa.

Rešenje je da uvek ispituje tip vrednosti pomoću funkcija opisanih u narednom odeljku.

Ispitivanje tipa i vrednosti promenljivih

Budući da je PHP veoma prilagodljiv u pogledu tipova vrednosti, postoje sledeće funkcije koje omogućavaju utvrđivanje tipa promenljive:

```
boolean is_int(mešovit_tip promenljiva)
boolean is_float(mešovit_tip promenljiva)
boolean is_bool(mešovit_tip promenljiva)
boolean is_string(mešovit_tip promenljiva)
boolean is_array(mešovit_tip promenljiva)
boolean is_object(mešovit_tip promenljiva)
```

Sve funkcije vraćaju vrednost tipa Boolean (`true` ili `false`) za argument promenljiva, u zavisnosti od toga da li je njegov tip jednak tipu u imenu funkcije. Kôd u narednom primeru ispisuje 1, što odgovara konstanti `true`:

```
$test = 13.0;
echo is_float($test);  ispisuje 1 za true
```

Otklanjanje grešaka pomoću funkcija print_r() i var_dump()

U PHP-u postoje funkcije `print_r` i `var_dump()` koje ispisuju tip i vrednost datog izraza u obliku koji je razumljiv za korisnika.

```
print_r(mešovit_tip izraz)
var_dump(mešovit_tip izraz [,mešovit_tip izraz ...])
```

Ove funkcije su korisne kada se otklanjaju greške iz skripta, naročito kada su u pitanju nizovi ili objekti. Da biste na određenom mestu u skriptu saznali tip i vrednost promenljive `$var`, možete upotrebiti kôd nalik sledećem:

```
$var = 15;
var_dump($var);
```

Ispisaće se tekst:

```
int(15)
```

Dok funkcija `var_dump()` omogućava da u istom pozivu zadate više promenljivih koje treba ispitati i pruža informacije o veličini sadržaja promenljive, `print_r()` daje sažetiji prikaz sadržaja nizova i objekata. Ove funkcije se mogu pozivati sa ulaznim argumentima bilo kog tipa; u ovom poglavlju ih koristimo za ilustrovanje rezultata naših primera koda.

Ispitivanje, inicijalizovanje i deinicijalizovanje promenljivih

Tokom izvršavanja skripta, promenljiva može biti u neodređenom stanju ili još nedefinisana. Za testiranje stanja promenljive u PHP-u služe funkcije `isset()` i `empty()`:

```
boolean isset(mešovit_tip promenljiva)
boolean empty(mešovit_tip promenljiva)
```

Funkcija `isset()` ispituje da li je promenljiva inicijalizovana bilo kojom vrednošću različitom od Null, a funkcija `empty()` ispituje da li promenljiva ima bilo kakvu vrednost. One daju različite rezultate, što se vidi iz narednih primera:

```
$var = "test";
// ispisuje "Promenljiva ima vrednost"
if (isset($var)) echo "Promenljiva ima vrednost.";

// ne ispisuje ništa
if (empty($var)) echo "Promenljiva je prazna";
```

Promenljiva se može izričito deinicijalizovati pomoću funkcije `unset()`;

```
unset(mešovit_tip promenljiva [, mešovit_tip promenljiva [,  ...]])
```

Posle pozivanja funkcije unset u narednom primeru, promenljiva `$var` više nije definisana:

```
$var = "bilo šta";

// U nastavku skripta
unset($var);

// Ne ispisuje se ništa
if (isset($var)) echo "Promenljiva je definisana";
```

Drugi način da utvrdite da li je određena promenljiva prazna jeste da je preslikate u tip Boolean pomoću operatora (bool). U narednom primeru promenljiva $var se tretira kao da je tipa Boolean, što je ekvivalentno ispitivanju !empty($var):

```
$var = "bilo šta";

// Ispisuju se rezultati komandi iz oba reda
if ((bool)$var))   echo "Promenljiva nije prazna";
if (!empty($var))   echo "Promenljiva nije prazna";
```

Tabela 2-2 prikazuje vrednosti koje daju funkcije isset($var), empty($var) i (bool)$var za različite vrednosti promenljive $var. Neke rezultate možda niste očekivali: kada $var ima vrednost "0", funkcija empty() daje true.

Tabela 2-2. Vrednosti izraza

Vrednost promenljive $var	isset($var)	empty($var)	(bool)$var
$var = null;	false	true	false
$var = 0;	true	true	false
$var = true;	true	false	true
$var = false;	true	true	false
$var = "0";	true	true	false
$var = "";	true	true	false
$var = "nešto";	true	false	true
$var = array();	true	true	false
unset $var;	false	true	false

Uslovno izvršavanje i grananje u kodu

Sintaksa upravljačkih struktura u PHP kodu slična je sintaksi takvih struktura u drugim programskim jezicima visokog nivoa.

Uslovno izvršavanje delova koda skripta omogućava da se u zavisnosti od toga da li određeni izraz ima vrednost true ili false izvršavaju drugačije naredbe. U PHP-u postoje dve vrste naredaba za grananje: if, kojoj se po potrebi može pridružiti neobavezna odredba else, i naredba switch, koja se obično sastoji od dve ili više odredaba case.

Iskaz if ... else

Iskaz if omogućava uslovno izvršavanje blokova koda, a u PHP-u ima istu svrhu kao u bilo kom drugom programskom jeziku. Osnovni oblik iskaza if omogućava da ispitate da li je određeni uslov ispunjen (ima vrednost true) i ako jeste, da izvršite jednu ili više naredaba.

Naredni iskaz `if` izvršava iskaz `echo` koji ispisuje zadati tekst kada uslovni izraz, `$var` veće od 5, ima vrednost `true`:

```
if ($var > 5)
  echo "Vrednost promenljive je veća od 5";
```

U iskazu `if` izvršava se samo iskaz neposredno iza `if`.

Blok od više iskaza može se izvršiti kada se oni uokvire vitičastim zagradama. Ako uslovni izraz ima vrednost `true`, izvršavaju se iskazi unutar vitičastih zagrada. U suprotnom, nijedan od tih iskaza se ne izvršava. Pogledajmo primer u kome se izvršavaju tri iskaza kada uslov ima vrednost `true`:

```
if ($var > 5)
{
  echo "Vrednost promenljive je veća od 5";
  // Sada joj dodeljujemo vrednost 5
  $var = 5;
  echo "A sada, imamo 5.";
}
```

Iskazu `if` može se pridružiti neobavezna odredba `else` ukoliko treba izvršiti naredbu ili blok naredaba kada uslovni izraz ima vrednost `false`. Pogledajmo jedan primer:

```
if ($var > 5)
  echo "Vrednost promenljive je veća od 5";
else
  echo "Vrednost promenljive je jednaka 5 ili manja";
```

Često i odredba `else` sadrži blok iskaza između vitičastih zagrada, kao u narednom primeru:

```
if ($var > 5)
{
  echo "Vrednost promenljive je veća od 5";
  echo " --------------------------------";
}
else
{
  echo "Vrednost promenljive je jednaka 5 ili manja";
  echo "----------------------------------------";
}
```

Više uzastopnih iskaza `if` može dovesti do primera nalik sledećem:

```
if ($var < 5)
  echo "Vrednost je vrlo mala";
else
  if ($var < 10)
    echo "Vrednost je mala";
  else
    if ($var < 20)
      echo "Vrednost je velika;
    else
      if ($var < 30)
        echo "Vrednost je veoma velika";
```

Ako vam treba više uzastopnih iskaza `if`, upotrebite iskaz `elseif`. Izbor metode koju ćete primeniti samo je pitanje ličnog ukusa. Naredni primer je funkcionalno jednak prethodnom:

```
if ($var < 5)
  echo "Vrednost je vrlo mala";
elseif ($var < 10)
  echo "Vrednost je mala";
elseif ($var < 20)
  echo "Vrednost je velika";
elseif ($var < 30)
  echo "Vrednost je veoma velika";
```

Iskaz switch

Iskaz `switch` može se upotrebiti umesto iskaza `if` kada treba izabrati jednu opciju sa liste mogućih:

```
switch ($meni)
{
   case 1:
     echo "Izabrali ste prvu";
     break;
   case 2:
     echo "Izabrali ste drugu";
     break;
   case 3:
     echo "Izabrali ste treću";
     break;
   case 4:
     echo "Izabrali ste četvrtu";
     break;
   default:
     echo "Izabrali ste nešto drugo";
}
```

Ovaj primer se može programirati i pomoću iskaza `if` i `elseif`, ali upotreba iskaza `switch` daje sažet i čitljiv kôd koji se lako piše. Upotreba iskaza `break` je obavezna: on sprečava izvršavanje iskaza koji mu slede unutar bloka `switch` i čini da se izvršavanje nastavlja iza vitičaste zagrade koja označava kraj iskaza `switch`.

Ako iz iskaza `switch` izostavite iskaze `break`, uvodite grešku. Ako korisnik izabere treću opciju, skript će ispisati rečenicu:

```
"Izabrali ste treću"
```

ali i sledeće:

```
"Izabrali ste treću Izabrali ste četvrtu Izabrali ste nešto drugo"
```

Činjenica da su iskazi `break` uvek neophodni ponekad se smatra odlikom koja je uzrok grešaka koje se teže otkrivaju.

Uslovni izrazi

Najčešće poređenje jeste ispitivanje jednakosti dva izraza, a rezultat je vrednost logičkog tipa, koja može biti `true` ili `false`. Jednakost se ispituje pomoću operatora `==`. Pogledajte sledeći primer:

```
$var = 1;

if ($var == 1)
  echo "Jednako je jedan!";
```

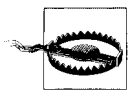
Ako je promenljiva `$var` jednaka vrednosti 1, rezultat poređenja u primeru ima vrednost `true` i ispisuje se poruka. Ako je rezultat poređenja `false`, ništa se ne ispisuje.

Razlika (nejednakost) ispituje se pomoću operatora nejednakosti `!=`.

```
$var = 0;

if ($var != 1)
  echo "Nije jednako jedan!";
```

U ovom primeru, rezultat poređenja je `true` i ispisuje se poruka. Operator `!=` obično se naziva *operator različito od*, zato što znak uzvika negira izraz u kom se zadaje jednakost između obe strane operatora `=`.



Ako ne poznajete dovoljno operator jednakosti `==` i operator dodeljivanja `=` , budite oprezni: lako ih je pobrkati. To je veoma česta greška, koja se teško otkriva.

Pogrešno napisan uslovni izraz (`$var = 1`) uvek ima rezultat `true` jer je operacija dodeljivanja koja se u ovom iskazu zapravo izvodi uvek uspešna i usled toga ceo izraz uvek će imati vrednost `true`.

Znatno se ređe pravi greška upotrebe operatora `==` umesto operatora dodeljivanja. Međutim, i ona se teže otkriva zato što se pogrešno napisan izraz za dodeljivanje vrednosti `$var == 1` u stvari svodi na `true` ili `false`, a vrednost `$var` ostaje nepromenjena.

U izrazima se mogu upotrebljavati zagrade i logički operatori `&&` (and, logičko i) i `||` (or, logičko ili). Na primer, sledeći izraz ima vrednost `true` i ispisuje poruku ako je `$var` jednako 3 ili 7:

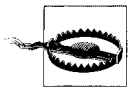
```
if ($var == 3) || ($var == 7)
  echo "Jednako je 3 ili 7";
```

Naredni izraz ima vrednost `true` i ispisuje poruku ako je `$var` jednako 2 i `$var2` je jednako 6:

```
if ($var == 2) && ($var2 == 6)
  echo "Promenljive su jednake 2 i 6";
```

Zanimljiva je činjenica da ukoliko prvi deo izraza (`$var == 2`) ima vrednost `false`, PHP ne ispituje vrednost drugog dela (`$var == 6`), zato što u tom slučaju vrednost celog izraza ne može biti `true`; obe strane operatora `&&` (`and`) moraju imati vrednost `true` da

bi ceo izraz imao vrednost `true`. Ova odlika skraćenog ispitivanja utiče na način pisanja koda; da biste ubrzali izvršavanje koda, izraz koji će najverovatnije imati vrednost `false` postavite tako da on bude prvi sleva, a operacije koje zahtevaju najviše izračunavanja postavite što više udesno.



Nikad ne polazite od prepostavke da će PHP ispitivati vrednosti svih delova izraza kombinovanih pomoću logičkih operatora `&&` i `||`. PHP primenjuje skraćeno ispitivanje kada utvrđuje rezultat izraza logičkog tipa.

Složenji izrazi mogu da se sastavljaju kombinovanjem logičkih operatora i slobodne upotrebe zagrada. Na primer, naredni izraz ima vrednost `true` i ispisuje se poruka ako je istinita jedna od sledećih tvrdnji: `$var` je jednako `6` i `$var2` je jednako `7`, ili `$var` jednako `4` i `$var2` je jednako `1`.

```
if ((($var == 6) && ($var2 == 7)) ||
   (($var == 4) && ($var2 == 1)))
  echo "Izraz ima vrednost true";
```

Kao i u izrazima za dodeljivanje vrednosti, zagrade obezbeđuju da se ispitivanje vrednosti odvija po određenom redosledu.

Jednakost i nejednakost su dve osnovne operacije poređenja, ali se brojevi porede i radi utvrđivanja koji je veći, a koji manji. Pogledajmo sledeće primere:

```
// Daje true ako je $var manje od 5
if ($var < 5)
  echo "Manje od 5";

// Daje true ako je $var jednako ili manje od 5
if ($var <= 5)
  echo "Jednako ili manje od 5";

// Daje true ako je $var veće od 5
if ($var > 5)
  echo "Veće od 5";

// Daje true ako je $var jednako ili veće od 5
if ($var >= 5)
  echo "Jednako ili veće od 5";
```

U PHP4 uveden je nov operator, identično, koji se piše `===`. On ne postoji u drugim programskim jezicima i daje `true` samo kada ceo izraz ima vrednost `true` i svi argumenti su jednakog tipa. Pogledajmo primer:

```
// Daje true jer su na obe strane vrednosti tipa integer
// koje su međusobno jednake
If (5 === 5)
  echo "Jednak tip i vrednost";

// Daje false jer su na obe strane različiti tipovi
// (5.0 je tipa float, a 5 je tipa integer)
If (5.0 === 5)
```

```
  echo "Ovo se nikad neće ispisati!";
// Standardno ispitivanje jednakosti dalo bi rezultat true
if (5.0 == 5)
echo "Ovo se uvek ispisuje";
```

Uslovni izrazi koji su ovde dati mogu da porede i znakovne vrednosti, ali rezultati obično neće biti oni koje biste očekivali. Ako morate da poredite vrednosti znakovnog tipa – što je veoma čest zahtev – upotrebite funkciju `strcmp()` iz PHP-ove biblioteke funkcija za rad sa vrednostima znakovnog tipa.

Funkcija `strcmp()` se u ovoj knjizi koristi za rad sa znakovnim vrednostima, a opisana je u odeljku „Znakovne vrednosti".

Svaki izraz logičkog tipa koji smo dosad upotrebili može se negirati pomoću znaka uzvika `!`, koji je *unarni operator negacije*. Naredna dva izraza su ekvivalentna:

```
if (!($var != 1))
  echo "vrednost je jedan";

if ($var = 1)
  echo "vrednost je jedan";
```

Isto važi i za sledeće izraze:

```
if ($var < 10)
  echo "manje od 10";
if (!($var > =10))
  echo "manje od 10";
```

Verovatno najčešća upotreba unarnog operatora negacije jeste proveravanje da li je izvršavanje pozvane funkcije bilo neuspešno; to često primenjujemo kada pozivamo funkcije za rad sa bazama podataka, u narednim poglavljima.

Petlje

Petlje u PHP-u imaju jednaku sintaksu kao i u drugim programskim jezicima visokog nivoa. Petlje omogućavaju da se pri izvršavanju skripta ponavljaju dati blokovi iskaza dok je ispunjen određeni uslov. U PHP-u postoje četiri iskaza koji omogućavaju izvršavanje koda u petlji: `while`, `do ... while`, `for` i `foreach`. Prva tri su za petlje opšte namene, a `foreach` se koristi isključivo za rad sa nizovima.

Petlja while

Petlja `while` je najjednostavnija struktura petlje, ali ponekad i najmanje sažet oblik koji se može upotrebiti. Petlja `while` ponavlja izvršavanje jednog ili više iskaza – koji čine telo petlje – dok je određeni uslov ispunjen (ima vrednost `true`). Najpre se ispituje vrednost uslova, a zatim se izvršava telo petlje. To znači da se petlja neće izvršiti nijedanput ako već na početku uslov nije ispunjen. Isto kao i u iskazu `if`, ako se telo petlje sastoji od više iskaza, morate ih uokviriti vitičastim zagradama.

Naredni primer, koji ispisuje brojeve od 1 do 10 razdvojene razmakom, ilustruje upotrebu iskaza `while`:

```
$brojac = 1;
while ($brojac < 11)
{
   echo $brojac;
   echo " ";
   // povećavamo $brojac za jedan
   $brojac++;
}
```

Petlja do ... while

Razlika između petlji `while` i `do...while` je u mestu ispitivanja uslova. U petlji do ... `while` vrednost uslova se ispituje posle izvršavanja tela petlje. Dok je uslov ispunjen, ponavlja se telo petlje.

Jednaku funkcionalnost kao u primeru petlje `while` možete postići na sledeći način:

```
$brojac = 1;
do
{
   echo $brojac;
   echo " ";
   $brojac++;
} while ($brojac < 11);
```

Razlika između petlji `while` i `do...while` može se videti u sledećem primeru:

```
$brojac = 100;
do
{
   echo $brojac;
   echo " ";
   $brojac++;
} while ($brojac < 11);
```

U ovom slučaju ispisuje se vrednost 100 zato što se telo petlje izvršava jedanput pre nego što se utvrdi da uslov ima vrednost `false`.

Petlja `do...while` je struktura koja se najređe koristi, verovatno zato što je ređe potrebno da se telo petlje uvek izvrši barem jedanput, čak i kada uslov nije ispunjen.

Petlja for

Petlja `for` je nasloženija struktura petlje, ali daje i najsažetiji kôd.

Pogledajmo sledeći blok koda koji pruža istu funkcionalnost kao primeri kojima smo ilustrovali petlje `while` i `do...while`:

```
for($brojac=1; $brojac<11; $brojac++)
{
   echo $brojac;
   echo " ";
}
```

Iskaz petlje `for` sastoji se od tri dela razdvojena znacima tačka zarez; sva tri dela su neobavezna;

Inicijalizacioni iskazi
: Iskazi koji se izvršavaju jedanput, pre nego što se izvrši telo petlje.

Uslovi izvršavanja petlje
: Uslovni izraz čija se vrednost ispituje pre svakog izvršavanja tela petlje. Ako je rezultat `false` (uslov nije ispunjen), telo petlje se ne izvršava.

Iskazi na kraju petlje
: Iskazi koji se izvršavaju posle svakog izvršavanja tela petlje.

Prethodni blok koda daje iste rezultate kao navedeni primeri petlji `while` i `do... while` koje odbrojavaju od `1` do `10`. Iskaz `$brojac=1` je inicijalizacioni iskaz koji se izvršava samo jedanput pre nego što se izvrši telo petlje. Uslov za izvršavanje petlje je `$brojac<11`, što se uvek proverava pre izvršavanja tela petlje; kada uslov više nije ispunjen, tj. kada `$brojac` dostigne vrednost `11`, izvršavanje petlje se prekida. Iskaz na kraju petlje, `$brojac++`, izvršava se svaki put posle iskaza koji čine telo petlje.

Ovo je tipičan primer petlje `for`. Inicijalizacioni iskaz podešava početnu vrednost brojača, njegova vrednost se ispituje u uslovu petlje, a iskaz na kraju petlje povećava tu vrednost. Većina petlji `for` koje se koriste u PHP skriptovima ima takav format.

Složenost uslova nije ograničena, kao i u iskazima `if`. Osim toga, kada ima više inicijalizacionih iskaza i iskaza na kraju petlje, mogu se razdvojiti zarezima. Evo složenijeg primera:

```
for($x=0,$y=0; $x<10&&$y<$z; $x++,$y+=2)
```

Međutim, previše složene petlje `for` mogu dovesti do teško čitljivog i zbunjujućeg koda.

Petlja foreach

Iskaz `foreach`, koji postoji samo u PHP4, pruža pogodan način za pristupanje elementima memorijskog niza u petlji. Kao i petlja `for`, iskaz `foreach` izvršava telo petlje po jednom za svaki element niza. Naredni blok koda, za svaki element niza, pretvara u inče niz vrednosti izraženih u centimetrima:

```
// Formiramo niz celobrojnih vrednosti
$duzine = array(0, 107, 202, 400, 475);

// Pretvaramo u inče niz dužina izraženih u centimetrima
foreach($duzine as $cm)
{
   $inch = (100 * $cm) / 2.45;
   echo "$cm centimetara = $inch inča\n";
}
```

Petlja `foreach` je veoma korisna i pogodna metoda za rad sa nizovima, što je detaljno opisano u odeljku „Upotreba petlje `foreach` za rad sa nizovima“.

Promena ponašanja petlje

Da biste pre vremena izašli iz petlje – pre nego što uslov za izvršavanje petlje dobije vrednost `false` – upotrebite naredbu `break`. Naredni primer ilustruje kako se to radi:

```
for($x=0; $x<100; $x++)
{
   if ($x < $y)
     break;
   echo $x;
}
```

Uslov za normalno završavanje petlje jeste da `$x` dosegne vrednost `100`. Međutim, ako je `$x` veće od `$y` (ili postane veće od `$y`), petlja se završava prevremeno, a izvršavanje programa se nastavlja na prvom iskazu koji se nalazi iza tela petlje. Iskaz `break` može se koristiti u svim vrstama petlji.

Da biste sa tekućeg iskaza ponovo krenuli od početka petlje, bez izvršavanja preostalih iskaza u telu petlje, upotrebite iskaz `continue`. Pogledajmo sledeći primer:

```
$x = 1;

while($x<100)
{
   echo x$;
   $x++;
   if ($x > $y)
     continue;
   echo $y;
}
```

Kôd u ovom primeru ispisuje i povećava vrednost promenljive `$x` pri svakom izvršavanju tela petlje. Ako je `$x` veće od `$y`, petlja se ponovo izvršava od početka; u suprotnom, ispisuje se vrednost `$y` i petlja se normalno izvršava. Isto kao i iskaz `break`, iskaz `continue` može se zadati u svim vrstama petlji.

Celovit primer

U ovom odeljku primenićemo tehnike koje su dosad opisane da bismo razvili jednostavan, ali potpun primer skripta. Budući da taj skript ne obrađuje podatke koje korisnik unosi, neke od najkorisnijih mogućnosti PHP-a kao jezika za pisanje Web skriptova razmatraćemo u drugim poglavljima u nastavku knjige.

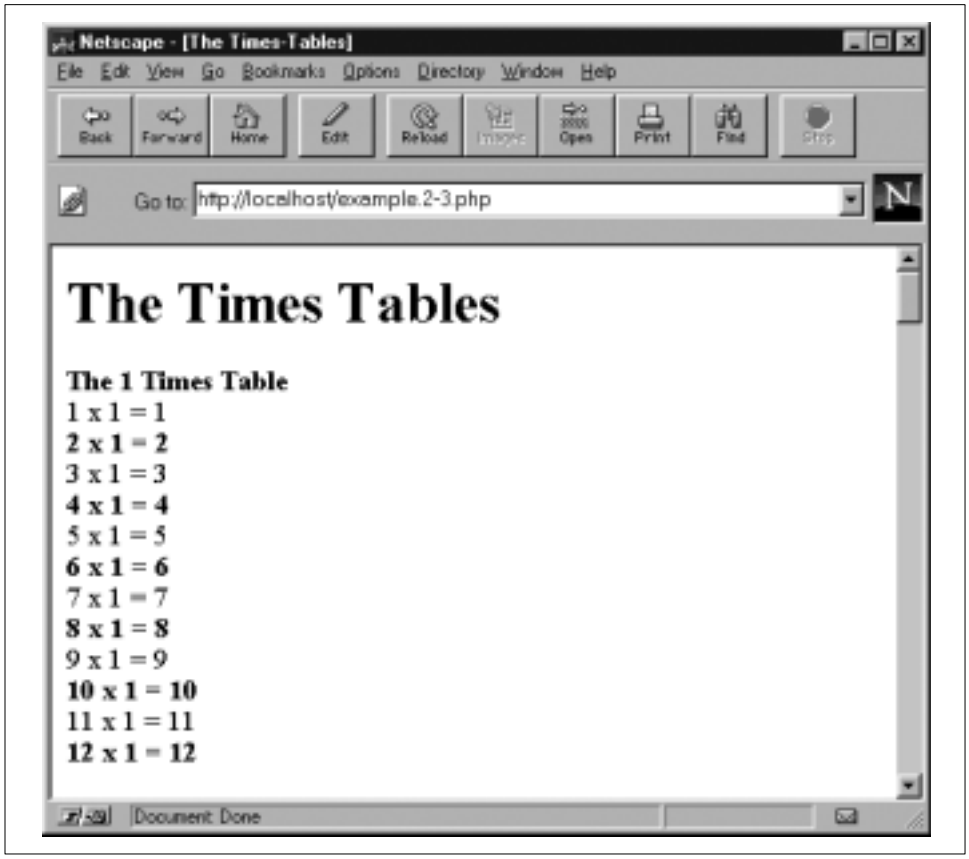
Naš primer je skript koji formira Web stranicu koja prikazuje tablicu množenja. Cilj nam je da prikažemo tablicu množenja za brojeve u opsegu od 1 do 12. Prva tablica je prikazana na slici 2-2, onako kako izgleda u čitaču Netscape.

Prvi korak u razvoju našeg skripta jeste da zadamo kako treba da izgledaju rezultati; dakle, najpre ćemo napisati odgovarajući HTML kôd. Ako se ograničimo na jednostavan HTML kôd, za primer 2-3 treba nam sledećih 12 redova HTML koda:

```
<html>
<head>
  <title>The Times Tables</title>
</head>
```

```
<body bgcolor="#ffffff">
<h1>The Times Tables</h1>
<p><b>The 1 Times Table</b>
<br>1 x 1 = 1
<br><b>2 x 1 = 2</b>
<br>3 x 1 = 3
<br><b>4 x 1 = 4</b>
<br>5 x 1 = 5
```

Naš skript daje sledeće rezultate kombinovanjem HTML koda sa ugrađenim PHP skriptom:



Slika 2-2. Izgled tablice množenja koju skript generiše u čitaču Netscape.

Gotov PHP skript i HTML kôd koji su neophodni za prikazivanje tablice množenja prikazani su u primeru 2-3. Prvih devet redova čini HTML kôd koji se sastoji od komponente `<head>` i zaglavlja `<h1>The Times Tables</h1>` na početku Web stranice. Slično tome, poslednja dva reda čini HTML kôd koji završava dokument: `</body>` i `</html>`.

Između ta dva bloka HTML koda kojim počinje i završava se dokument, nalazi se PHP skript koji generiše tablicu množenja i ugrađen je u HTML. Skript počinje PHP oznakom za početak `<?php`, a završava se oznakom za kraj `?>`.

Primer 2-3. Skript koji generiše tablicu množenja

```
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD HTML 4.0 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd" >
<html>
<head>
  <title>The Times Tables</title>
</head>
<body bgcolor="#ffffff">
<h1>The Times Tables</h1>
<?php

  // Generišemo tablice za brojeve od 1 do 12
  for($table=1; $table<13; $table++)
  {
    echo "<p><b>The " . $table . " Times Table</b>\n";

    // 12 redova za svaku tablicu
    for($counter=1; $counter<13; $counter++)
    {
      $answer = $table * $counter;

      // Da li je vrednost brojača paran broj?
      if ($counter % 2 == 0)
        // Da, sadržaj tekućeg reda se ispisuje
        // polucrnim slovima
        echo "<br><b>$counter x $table = " .
             "$answer</b></br>";

      else
        // Ne, koriste se standardna slova
        echo "<br>$counter x $table = $answer";
    }
  }
?>
</body>
</html>
```

Skript je zamišljen tako da za svaku tablicu množenja generiše zaglavlje i 12 redova podataka. Da bismo to obavili, u skriptu koristimo dve ugnežđene petlje: spoljnu i unutrašnju.

U spoljnoj petlji `for` koristi se celobrojna promenljiva `$table`, čija se vrednost povećava za 1 pri svakom izvršavanju tela petlje, dok ta vrednost ne premaši 12. Telo spoljne petlje ispisuje zaglavlje i izvršava unutrašnju petlju koja pak generiše samu tablicu množenja.

U unutrašnjoj petlji se pomoću celobrojne promenljive `$counter` generišu redovi tekuće tablice množenja. Unutar tela petlje, rezultat tekućeg reda, `$answer`, izračunava se množenjem tekuće vrednosti promenljive `$table` sa tekućom vrednošću promenljive `$counter`.

Svaki drugi red tablice množenja uokviren je oznakama `<b>` i `</b>`, čime se u rezultujućem HTML kodu dobijaju redovi teksta koji se ispisuju naizmenično polucrnim i običnim slovima. Pošto izračunamo vrednost promenljive `$answer`, pomoću iskaza `if` ispitujemo da li red treba ispisati polucrnim ili običnim slovima. U tom

iskazu se pomoću operatora modulo % ispituje da li je vrednost promenljive $counter paran ili neparan broj.

U operaciji modulo vrednost promenljive $counter deli se sa 2 i kao rezultat uzima ostatak deljenja. Na primer, ako $counter ima vrednost 6, rezultat operacije modulo je 0 jer 6 podeljeno sa 2 jeste 3, bez ostatka. Ako $counter ima vrednost 11, rezultat je 1 jer 11 deljeno sa 2 jeste 5, a ostatak je 1. Ako je $counter paran broj, uslovni izraz:

```
($counter %2 == 2)
```

ima vrednost true i dodaju se oznake za polucrni tekst.

Napomene u vezi s primerom 2.3

Primer 2-3 je celovit, ali nije posebno zanimljiv. Bez obzira na to koliko puta se skript izvrši, rezultat će uvek biti ista Web stranica. U stvarnosti, može biti najpogodnije da takav skript izvršite jedanput, da preuzmete rezultate i da ih potom ugradite u statičku HTML datoteku. Ako rezultate snimite u tom obliku, korisnik može da učitava uvek istu stranicu, uz manje opterećenje servera i kraće vreme odziva.

U poglavlju 4 predstavićemo još nekoliko skriptova koji ne prihvataju ulazne podatke od korisnika. Međutim, razlika je u tome što oni komuniciraju sa MySQL-om, kome šalju SQL upite. Rezultat je to da stranica prikazuje druge podatke kada se promeni sadržaj baze podataka. Dakle, za razliku od našeg jednostavnog primera u ovom odeljku, skriptovi u poglavlju ne mogu se tek tako zameniti statičkim HTML stranicama.

Nizovi

Nizovi su u PHP-u složene strukture podataka, prilagodljivije nego u mnogim drugim programskim jezicima visokog nivoa. Niz (engl. *array*) je uređen skup promenljivih, u kome se svaka promenljiva naziva element. Tehnički govoreći, niz može biti numerisan ili asocijativan, što znači da se elementima niza može pristupati pomoću numeričkog indeksa, odnosno pomoću tekstualnih vrednosti.

Niz u PHP-u može sadržati skalarne (proste) vrednosti – cele brojeve, logičke vrednosti, znakovne vrednosti, ili decimalne vrednosti – ili složene vrednosti, kao što su objekti ili čak drugi nizovi, a isti niz može sadržati i vrednosti različitih tipova. U ovom odeljku pokazaćemo kako se formiraju nizovi i uvešćemo više korisnih funkcija iz PHP-ove biblioteke funkcija za rad s nizovima.

Formiranje niza

U PHP-u postoji naredba array() kojom se formiraju nizovi. Naredni primeri pokazuju kako se formiraju nizovi celobrojnih i znakovnih vrednosti koji se zatim dodeljuju promenljivama da bismo ih potom koristili u kodu.

```
$brojevi = array(5, 4, 3, 2, 1)
$pojmovi = array("Web", "aplikacije", "baza", "podataka")
```

```
// Ispisujemo treći element niza
// celobrojnih vrednosti: 3
echo $brojevi[2];

// Ispisujemo prvi element niza
// znakovnih vrednosti: "Web"
echo $pojmovi[0];
```

Podrazumeva se da je indeks prvog elementa niza 0. Vrednosti sadržane u nizu mogu se učitavati i menjati pomoću notacije sa uglastim zagradama []. Naredni primer ilustruje notaciju sa uglastim zagradama za niz znakovnih vrednosti:

```
$novNiz[0] = "Krompir";
$novNiz[1] = "Šargarepa";
$novNiz[2] = "Spanać";

// Ups, greška, moramo da zamenimo
// treći element
$novNiz[2] = "Paradajz";
```

Elementi niza mogu se numerisati tako da indeks počinje od bilo koje vrednosti. Često je pogodno da indeks niza počinje od 1, kao u sledećem primeru:

```
$brojevi = array(1=>"jedan", "dva", "tri", "četiri");
```

Vrednosti indeksa ne moraju da budu uzastopne:

```
$neparniBrojevi = array(1=>"jedan", 3=>"tri", 5=>"pet");
```

Prazan niz se pravi tako što promenljivoj dodeljujete rezultat funkcije `array()` bez parametara. Vrednosti se mogu naknadno dodati pomoću notacije sa uglastim zagradama. Ako ne zadate vrednost indeksa, PHP automatski dodeljuje prvu sledeću vrednost – najveća tekuća vrednost indeksa plus jedan. Pogledajmo naredni primer, u kome se formira prazan niz `$greske`, a zatim na kraju skripta ispituje da li je niz prazan. Opis prve greške koji se dodaje u niz `$greske` ima indeks 0, drugi je element 1 itd.

```
$greske = array();

// dalje u kodu ...
$greske[] = "Došlo je do greške";

// ... još dalje
$greske[] = "Nešto je zaista pošlo naopako";

// Sada ispitujemo da li je bilo grešaka
if (empty($greske))
  // Sve je u redu. Idemo dalje.
  echo " Sve je u redu. Idemo dalje."
else
  echo "Ima grešaka!"
```

Asocijativni nizovi

U asocijativnim nizovima se elementima niza pristupa pomoću tekstualnih vrednosti za indekse koje se zovu ključevi. Asocijativni niz se formira pomoću funkcije `array()`, kao u narednom primeru, gde se formira niz celobrojnih vrednosti:

```
$niz = array("prvi"=>1, "drugi"=>2, "treći"=>3);

// Ispisujemo drugi element: prikazuje se "2"
echo $niz["drugi"];
```

Isti niz celobrojnih vrednosti može se fomirati i pomoću notacije sa uglastim zagradama:

```
$niz["prvi"] = 1;
$niz["drugi"] = 2;
$niz["treći"] = 3;
```

Postoji mala razlika između pristupanja vrednostima u nizu pomoću numeričkih i znakovnih indeksa. Elementima u asocijativnom nizu može se pristupati na oba načina, ali to zbunjuje i treba izbegavati.

Nizovi sa asocijativnim indeksima naročito su korisni za komuniciranje sa slojem baze podataka. Nizovi se obilato koriste u poglavljima 4, 5 i 6, u kojima ćete naći dodatne primere i funkcije za rad s nizovima.

Heterogeni nizovi

Vrednosti sadržane u jednom nizu ne moraju da budu sve istog tipa; PHP nizovi mogu da sadrže *heterogene* vrednosti. Naredni primer prikazuje heterogeni niz `$hetNiz`:

```
$hetNiz = array("mačka", 42, 8.5, false);
```

Funkcija `var_dump()` prikazuje sadržaj niza (dodat je razmak radi čitljivosti):

```
array(4) { [0]=> string(5) "mačka"
           [1]=> int(42)
           [2]=> float(8.5)
           [3]=> bool(false) }
```

Višedimenzionalni nizovi

PHP nizovi mogu da sadrže i druge nizove, čime dobijamo višedimenzionalne nizove. Primer 2-4 prikazuje kako se formiraju višedimenzionalni nizovi.

Primer 2-4. Višedimenzionalni nizovi u PHP-u

```
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD HTML 4.0 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd" >
<html>
<head>
  <title>Višedimenzinalni nizovi</title>
</head>
<body bgcolor="#ffffff">
<h2>Dvodimenzinalni niz</h2>
<?php

  // Dvodimenzionalni niz sa celobrojnim indeksima
  $planete = array(array("Merkur", 0.39, 0.38),
                   array("Venera", 0.72, 0.95),
                   array("Zemlja", 1.0, 1.0),
                   array("Mars", 1.52, 0.53) );
```

Primer 2-4. Višedimenzionalni nizovi u PHP-u (nastavak)

```
  // ispisuje "Zemlja"
  print $planete[2][0]
?>

<h2>Složeniji višedimenzionalni niz</h2>
<?php

  // Složeniji višedimenzionalni niz
  $planete2 = array(
    "Merkur"=> array("razdaljina"=>0.39, "prečnik"=>0.38),
    "Venera"  => array("razdaljina "=>0.39, "prečnik"=>0.95),
    "Zemlja"  => array("razdaljina"=>1.0,  "prečnik"=>1.0,
                 "meseci"=>array("Mesec")),
    "Mars"   => array("razdaljina"=>0.39, "prečnik"=>0.53,
                 "meseci"=>array("Fobos", "Deimos")),
  );

  // ispisuje "Mesec"
  print $planete2["Zemlja"]["meseci"][0];
?>
</body>
</html>
```

Prvi niz formiran u primeru 2-4 dvodimenzionalan je, a njegovim elementima se pristupa pomoću numeričkog indeksa. Niz `$planete` ima četiri elementa, a svaki od njih je i sam niz koji sadrži tri vrednosti: ime planete, razdaljinu od Sunca i relativni prečnik planete u odnosu na Zemlju.

Drugi niz u primeru 2-4 nešto je složeniji: u nizu planete2 se pomoću asocijativnih ključeva pristupa nizu koji sadrži podatke o određenoj planeti. Svakoj planeti pridružen je niz vrednosti koje su asocijativno indeksirane imenima svojstava koje te vrednosti predstavljaju. Planetama koje imaju mesece pridruženo je novo svojstvo čija vrednost je niz sa imenima meseca koji kruže oko planete.

Kao što je napomenuto u uvodnom delu ovog odeljka, PHP nizovi su veoma prilagodljivi. Mnoge strukture podataka – kao što su liste svojstava, stekovi, redovi i hijerarhijska stabla – mogu se napraviti pomoću nizova. Mi ćemo ograničiti upotrebu nizova na jednostavne strukture; proučavanje složenijih struktura podataka izlazi izvan okvira ove knjige.

Upotreba petlje foreach za rad s nizovima

Kao što je već rečeno u prethodnom delu ovog poglavlja, najjednostavniji način da se redom pristupi svakom elementu niza (iteracija) jeste pomoću iskaza `foreach`. Taj iskaz je novina uvedena u PHP4 da bi se olakšao rad s nizovima.

Iskaz `foreach` može da ima dva oblika:

```
foreach(izraz_tipa_niz as $vrednost) iskaz
foreach(izraz_tipa_niz as $kljuc => $vrednost) iskaz
```

U oba slučaja se obrađuje *izraz_tipa_niz* tako što se telo petlje izvršava po jednom za svaki element u nizu. U prvom obliku se vrednost elementa dodeljuje promenljivoj

zadatoj iza rezervisane reči as; u drugom obliku se vrednost ključa i vrednost elementa dodeljuju svaka po jednoj promenljivoj.

U narednom primeru prikazana je upotreba prvog oblika; izraz tipa niz je u ovom slučaju promenljiva $duzine, a svaka njena vrednost se dodeljuje promenljivoj $cm:

```
// Formiramo niz celobrojnih vrednosti
$duzine = array(0, 107, 202, 400, 475);

// Konverzija u inče niza dužina izraženih
// u centimetrima
foreach($duzine as $cm)
{
   $inch = $cm / 2.54
   echo "$cm centimetara = $inch inča\n";
}
```

U ovom primeru elementima niza se pristupa redosledom kojim su oni postavljeni u niz:

```
0 centimetara = 0 inča
107 centimetara = 42.125984251969 inča
202 centimetara = 79.527559055118 inča
400 centimetara = 157.48031496063 inča
475 centimetara = 193.87755102041 inča
```

Prvi oblik iskaza foreach omogućava pristupanje i elementima asocijativnog niza, dok se u drugom obliku vrednosti ključa elementa i samog elementa dodeljuju promenljivama koje su zadate u odredbi as $kljuc = $vrednost. Naredni primer pokazuje kako se vrednost ključa elementa dodeljuje promenljivoj $animal, a vrednost elementa promenljivoj $sound da bi se generisali stihovi pesmice „Old MacDonald“:

```
// Old MacDonald
$sounds = array("cow"=>"moo", "dog"=>"woof",
                       "pig"=>"oink", "duck"=>"quack");

foreach($sounds as $animal => $sound)
{
   echo "<p>Old MacDonald had a farm EIEIO";
   echo "<br>And on that farm he had a $animal EIEIO";
   echo "<br>With a $sound-$sound here";
   echo "<br>And a $sound-$sound there";
   echo "<br>Here a $sound, there a $sound";
   echo "<br>Everywhere a $sound-$sound";
}
```

Ovaj kôd ispisuje po jednu strofu pesmice za svaki par $animal/$sound u nizu $sounds:

```
Old MacDonald had a farm EIEIO
And on that farm he had a cow EIEIO
With a moo-moo here
And a moo-moo there
Here a moo, there a moo
Everywhere a moo-moo
```

```
Old MacDonald had a farm EIEIO
And on that farm he had a dog EIEIO
With a woof-woof here
And a woof-woof there
Here a woof, there a woof
Everywhere a woof-woof
```

Kada se drugi oblik iskaza `foreach` upotrebi sa neasocijativnim nizom, vrednost indeksa se dodeljuje promenljivoj koja predstavlja ključ elementa, a vrednost elementa promenljivoj za vrednost. U narednom primeru se indeks koristi za ispisivanje rednog broja u svakom redu rezultata:

```
// Formiramo niz celobrojnih vrednosti
$duzine = array(0, 107, 202, 400, 475);

// Konverzija u inče niza dužina izraženih
// u centimetrima
foreach($duzine as $indeks => $cm)
{
   $inch = $cm / 2.54
   $item = $index + 1;
   echo $indeks + 1 . ". $cm centimetara = $inch inča\n";
}
```

Iskaz `foreach` koristićemo u poglavljima od 4 do 13.

Upotreba pokazivača u nizovima

Zajedno s ključevima i odgovarajućim vrednostima u nizu, PHP održava i interni pokazivač koji upućuje na tekući element niza.Taj pokazivač koristi i ažurira više funkcija koje omogućavaju pristup elementima niza. Kao ilustraciju kako se taj interni pokazivač može iskoristiti, pogledajmo sledeći primer:

```
$a = array("a", "b", "c", "d", "e", "f");
echo current($a);       // ispisuje "a"

// Niz ( [1]=>  a [value]=> a [0]=> 0 [key]=> 0 )
print_r each($a);

// Niz ( [1]=>  b [value]=> b [0]=> 1 [key]=> 1 )
print_r each($a);

// Niz ( [1]=>  c [value]=> c [0]=> 2 [key]=> 2 )
print_r each($a);

echo current($a);       // ispisuje "d"
```

Kada formirate nov niz, interni pokazivač je podešen tako da upućuje na prvi element niza, a funkcija `current()` daje vrednost na koju upućuje interni pokazivač niza. Kad god je pozovete, funkcija `each()` daje niz koji se sastoji od vrednosti ključa i vrednosti tekućeg elementa, a zatim povećava interni pokazivač tako da upućuje na naredni element niza. Niz koji funkcija `each()` vraća sastoji se od četiri elementa: dva koja

sadrže ključ kome se pristupa numeričkim indeksom 0 i/ili asocijativnim ključem `key`, i druga dva elementa koja sadrže vrednost, kojoj se pristupa numeričkim indeksom1 i/ili asocijativnim ključem `value`.

Druge funkcije koje koriste interni pokazivač niza su `end()`, `next()`, `prev()`, `reset()` i `key()`.

Pre nego što je iskaz `foreach` uveden u jezik PHP, uobičajen način da se u petlji pristupa pojedinačnim elementima niza bila je upotreba petlje `while` u kojoj su se pomoću funkcije `each()` učitavali parovi ključ/vrednost za svaki element niza, a onda se pomoću funkcije `list()` tako učitana vrednost dodeljivala nekoj promenljivoj. Naredni primer pokazuje kako bi se taj metod primenio na niz `$sounds` radi generisanja stihova pesmice „Old MacDonald“:

```
$sounds = array("pig"=>"oink", "cow"=>"moo",
                        "duck"=>"quack", "dog"=>"woof");

while(list($animal, $sound) = each($sounds))
{
   echo "<p>Old MacDonald had a farm EIEIO";
   echo "<br>And on that farm he had a $animal EIEIO";
   echo "<br>With a $sound-$sound here";
   echo "<br>And a $sound-$sound there";
   echo "<br>Here a $sound, there a $sound ";
   echo "<br>Everywhere a $sound-$sound ";
}
```

Iskaz `foreach` je razumljiviji i trebalo bi da ga koristite u većini slučajeva. Ovde smo dali i primer s petljom `while` zato što se u mnogim postojećim skriptovima koristi ta struktura za pristupanje elementima asocijativnog niza.

Funkcija `list()` zapravo i nije funkcija, već konstrukcija jezika PHP koja omogućava da vrednosti elemenata izraza tipa niz dodelite pojedinačnim promenljivama.

```
list($promenljiva1, $promenljiva2, ...) = izraz_tipa_niz
```

`list()` se pojavljuje na levoj strani operatora dodeljivanja, a izraz tipa niz na desnoj. Argumenti zadati u `list()` moraju biti promenljive. Prvoj promenljivoj dodeljuje se vrednost prvog elementa niza, drugoj promenljivoj vrednost drugog elementa niza itd. Mi izbegavamo upotrebu konstrukcije `list()` zato što se mora unapred znati broj elemenata u nizu. Iskaz `foreach` čini izlišnom upotrebu konstrukcije `list()` radi pristupanja parovima ključ/vrednost koje vraća funkcija `each()`.

Osnovne funkcije za rad s nizovima

U ovom odeljku uvodimo nekoliko osnovnih funkcija iz PHP-ovih biblioteka za rad sa nizovima.

Prebrojavanje elemenata u nizu

Funkcija `count()` daje ukupan broj elemenata u nizu `niz`:

```
integer = count(mešovit_tip niz)
```

Naredni primer ispisuje vrednost 7, kao što bi se i očekivalo:

```
$dani = array("pon", "uto", "sre", "čet",
              "pet", "sub", "ned");
echo count($dani);     // 7
```

Funkcija `count()` može se upotrebiti s bilo kojim tipom promenljive i daje 0 kada je niz prazan ili kada se ispituje promenljiva koja nije definisana. Ako postoji i takva mogućnost, trebalo bi najpre da ispitate promenljivu pomoću funkcija `isset()` i `is_array()`.

Utvrđivanje najmanje i najveće vrednosti u nizu

Najmanja i najveća vrednost u nizu `brojevi` mogu se pronaći pomoću funkcije `min()`, odnosno `max()`:

```
broj max(niz brojevi)
broj min(niz brojevi)
```

Ako pretražujete niz vrednosti tipa integer, rezultat funkcije biće takođe integer, a kada ispitujete niz vrednosti tipa float, funkcije `min()` i `max()` daju rezultat tipa float:

```
$var = array(10, 5, 37, 42, 1, -56);
echo max($var);     // ispisuje 42
echo min($var);     // ispisuje -56
```

Obe funkcije, `min()` i `max()`, mogu se pozivati s listom argumenata tipa integer ili float:

```
broj max(broj arg1, broj arg2, broj arg3, ...)
broj min(broj arg1, broj arg2, broj arg3, ...)
```

Funkcije `min()` i `max()` rade i sa argumentima znakovnog tipa, ali rezultati možda neće uvek biti ono što biste očekivali.

Pronalaženje date vrednosti u nizu pomoću funkcija in_array() i array_search()

Funkcija `in_array()` daje rezultat `true` ako niz *plast_sena* sadrži vrednost zadatu argumentom *igla*:

```
boolean in_array(mešovit_tip igla, array plast_sena [, boolean  striktno])
```

U narednom primeru, u nizu celobrojnih vrednosti `$maliProsti` traži se vrednost 19:

```
$maliProsti = array(3, 5, 7, 11, 13, 17, 19, 23, 29);

if in_array(19, $maliProsti))
  echo "19 je prost broj";  // Ovo se uvek ispisuje
```

Funkciji se može proslediti i treći, neobavezan argument kojim se zahteva striktna jednakost kada se porede elementi niza sa traženom vrednošću `igla`. U narednom primeru, funkcija `in_array()` daje uvek `true`; međutim, kada se zahteva striktna jednakost, znakovna vrednost `"19"` nije isto što i broj 19 u nizu i zato je rezultat funkcije `false`:

```
$maliProsti = array(3, 5, 7, 11, 13, 17, 19, 23, 29);

if in_array("19", $maliProsti, true))
  echo "19 je prost broj";  // Ovo se NE ispisuje
```

Funkcija `array_search()` – uvedena u verziji PHP 4.0.5 – deluje na isti način kao i funkcija `in_array()`, osim što je njen rezultat ključ tražene vrednosti *igla*, a ne logička vrednost `true`.

```
mešovit_tip array_search(mešovit_tip igla, array plast_sena [, boolean
striktno])
```

Međutim, ako tražene vrednosti nema u nizu, funkcija `array_search` vraća vrednost `false`. Naredni primer koda prikazuje kako funkcija radi sa asocijativnim i indeksiranim nizovima:

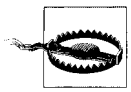
```
$razdaljina = array("centimetar"=>1, "metar"=>12, "kilometar"=>36);

// ispisuje "metar"
echo array_search(12, $razdaljina);

$jedMere = array("inch", "centimetar", "hvat", "milja");

// ispisuje 2
echo array_search("centimetar", $jedMere);
```

Budući da funkcija `array_search()` vraća rezultat mešovitog tipa – logičku vrednost `false` kada tražena vrednost nije nađena, ili ključ elementa kada tražena vrednost postoji u nizu – pojavljuje se problem kada je tražena vrednost jednaka prvom elementu niza. Pri automatskoj konverziji tipova vrednosti, PHP tretira vrednost 0 – indeks prvog elementa niza – kao vrednost `false` u izrazu logičkog tipa.



Budite oprezni kada koristite funkcije, kao što je array_search, koje vraćaju vrednost `false` kako bi pokazale da se rezultat ne može utvrditi. Ako se povratni rezultat funkcije tumači kao vrednost logičkog tipa – u nekom izrazu, ili kao argument tipa Boolean u pozivu druge funkcije – može se dogoditi da se ispravan rezultat automatski pretvori u `false`. Ako takva funkcija vraća 0, 0.0, `"c"`, znakovnu vrednost dužine 0 ili prazan niz, PHP-ova automatska konverzija tipova vrednosti pretvara rezultat u `false` kada je potrebna vrednost tipa Boolean.

Pravilan način ispitivanja rezultata jeste pomoću operatora identičnosti `===`, kao što je prikazano u narednom primeru:

```
$indeks = array_search("inch", $jedMere);

if ($indeks === false)
  echo "Nepoznata jedinica mere: inch";
else
  // Vrednost promenljive $indeks je OK
  echo "Indeks = $indeks";
```

Menjanje redosleda elemenata niza pomoću funkcije array_reverse()

Često je korisno da se obrne redosled elemenata u nizu. Funkcija `array_reverse()` formira nov niz u kome su elementi niza *izvorni*, postavljeni obrnutim redosledom:

```
array array_reverse(array izvorni [, bool sačuvati_ključeve])
```

Naredni primer pokazuje kako se obrće redosled elemenata indeksiranog niza znakovnih vrednosti:

```
$brojanje = array("jedan", "dva", "tri", "četiri");

$obrnuto = array_reverse($brojanje);
```

Kada neobaveznom argumentu *sačuvati_ključeve* zadate vrednost `true`, redosled elemenata niza se obrće, ali ostaju sačuvane veze između ključeva i elemenata kao u izvornom nizu. Za niz sa numeričkim indeksima to znači da se redosled elemenata obrće, ali vrednosti indeksa elemenata ostaju nepromenjene. To vam možda izgleda malo neobično, ali naredni primer pokazuje šta se zapravo događa:

```
$brojanje = array("nula,", "jedan", "dva", "tri");
$obrnuto = array_reverse($brojanje, true);
print_r($obrnuto);
```

Ovaj kôd ispisuje sledeće rezultate:

```
Array ([3] => tri [2] = > dva [1] = > jedan [0] = > nula)
```

Sortiranje sadržaja niza

U prethodnom odeljku pokazali smo kako se obrće redosled elemenata niza. U ovom odeljku pokazaćemo kako se sortira sadržaj niza. Za razliku od funkcije `array_reverse()` koja pravi kopiju izvornog niza, ali sa elementima u obrnutom redosledu, funkcije za sortiranje sadržaja niza menjaju redosled elemenata samog izvornog niza. Zbog takvog ponašanja, funkcijama za sortiranja sadržaja niza mora se proslediti promenljiva, a ne izraz.

Sortiranje pomoću funkcija sort() i rsort()

Najjednostavnije funkcije za sortiranje sadržaja nizova su `sort()` i `rsort()`, koje elemente niza `za_sortiranje` ređaju u zavisnosti od vrednosti svakog elementa po rastućem, odnosno opadajućem redosledu:

```
sort(array za_sortiranje [, integer način_sortiranja])
rsort(array za_sortiranje [, integer način_sortiranja])
```

Obe funkcije sortiraju elemente niza *za_sortiranje* u zavisnosti od vrednosti svakog elementa. Naredni primer prikazuje primenu funkcije `sort()` na niz vrednosti tipa integer:

```
$brojevi = array(24, 19, 3, 16, 56, 8, 171);
sort($brojevi);

foreach($brojevi as $n)
  echo $n . " ";
```

Rezultat primera su elementi niza `$brojevi` sortirani po vrednosti:

```
3 8 16 19 24 56 171
```

Vrednosti sortiranog niza mogu se prikazati na drugi način pomoću funkcije `print_r()`, opisane u odeljku „Ispitivanje tipa i vrednosti promenljivih". Rezultati iskaza `print_r($brojevi)` prikazuju sortirane vrednosti elemenata niza sa odgovarajućim indeksima:

```
Array ( [0] => 3
        [1] => 8
        [2] => 16
        [3] => 19
        [4] => 24
        [5] => 56
        [6] => 171 )
```

Naredni primer prikazuje funkciju `rsort()` upotrebljenu sa istim nizom:

```
$brojevi = array(24, 19, 3, 16, 56, 8, 171);
rsort($brojevi);
print_r($brojevi);
```

Ovog puta rezultat su elementi niza sortirani opadajućim redosledom vrednosti:

```
Array ( [0] => 171
        [1] => 56
        [2] => 24
        [3] => 19
        [4] => 16
        [5] => 8
        [6] => 3 )
```

Ako ne zadate drugačije, PHP sortira alfanumeričke i numeričke vrednosti numeričkim redosledom. Funkcijama za sortiranje može se proslediti neobavezan parametar, `način_sortiranja`, kojim se izričito zahteva sortiranje po pravilima za znakovne ili za numeričke vrednosti. U narednom primeru, PHP-ova konstanta `SORT_STRING` nalaže da se brojevi sortiraju kao da su znakovne vrednosti:

```
$brojevi = array(24, 19, 3, 16, 56, 8, 171);
sort($brojevi, SORT_STRING);
print_r($brojevi);
```

Rezultati ovog primera su sledeći:

```
Array ( [0] => 16
        [1] => 171
        [2] => 19
        [3] => 24
        [4] => 3
        [5] => 56
        [6] => 8 )
```

Mnoge funkcije za sortiranje nizova prihvataju parametar `način_sortiranja`. Druge vrednosti tog parametra mogu biti `SORT_REGULAR`, kojom se zahteva da se pri sortiranju vrednosti elemenata niza porede na uobičajeni način, ili `SORT_NUMERIC`, kada se elementi niza porede kao brojevi.

Funkcije sort() i rsort() mogu se koristiti sa asocijativnim nizovima, ali se onda gube ključevi. Rezultat je niz koji sadrži samo vrednosti u sortiranom redosledu. Pogledajte sledeći primer:

```
$map =
   array("o"=>"kk", "e"=>"zz", "z"=>"hh", "a"=>"rr");

sort($map);
print_r($map);
```

Rezultati funkcije print_r() prikazuju izmenjen niz bez vrednosti ključeva:

```
Array ( [0] => hh [1] => kk [2] => rr [3] => zz )
```

Sortiranje asocijativnih nizova

Često je poželjno da pri sortiranju asocijativnih nizova ostanu očuvani parovi ključ/vrednost. To omogućavaju funkcije asort() i arsort():

```
asort(array za_sortiranje [, integer način_sortiranja])
arsort(array za_sortiranje [, integer način_sortiranja])
```

Isto kao i sort() i rsort(), ove funkcije menjaju redosled elemenata u nizu za_sortiranje od najmanjeg ka najvećem, odnosno od najvećeg ka najmanjem. Naredni primer pokazuje jednostavan niz sortiran pomoću funkcije asort():

```
$map =
   array("o"=>"kk", "e"=>"zz", "z"=>"hh", "a"=>"rr");

asort($map);
print_r($map);
```

Funkcija print_r() prikazuje strukturu sortiranog niza:

```
Array ( [z] => hh
        [o] => kk
        [a] => rr
        [e] => zz )
```

Kada se funkcije asort() i rsort() koriste sa neasocijativnim nizovima, elementi se ređaju po redosledu sortiranja, ali se ne menjaju vrednosti indeksa pojedinačnih elemenata. To možda izgleda pomalo neobično; vrednosti indeksa iz izvornog niza tretiraju se u rezultujućem nizu kao asocijativni ključevi elemenata novog, sortiranog niza. Naredni primer pokazuje šta se događa:

```
$brojevi = array(24, 19, 3, 16, 56, 8, 171);
asort($brojevi);
print_r($brojevi);
Rezultati su sledeći:
Array ( [2] => 3
        [5] => 8
        [3] => 16
        [1] => 19
        [0] => 24
        [4] => 56
        [6] => 171 )
```

Sortiranje prema vrednostima ključeva

Umesto po vrednostima elemenata niza, funkcije `ksort()` i `krsort()` rađaju elemente niza po vrednostima njihovih ključeva ili indeksa:

```
integer ksort(array za_sortiranje [, integer način_sortiranja])
integer krsort(array za_sortiranje [, integer način_sortiranja])
```

Funkcija `ksort()` sortira elemente niza od najmanje vrednosti ključa ka najvećoj, a funkcija `krsort()` sortira elemente niza obrnutim redosledom. Naredni primer pokazuje kako radi funkcija `ksort()`:

```
$map =
   array("o"=>"kk", "e"=>"zz", "z"=>"hh", "a"=>"rr");

ksort($map);
print_r($map);
```

Sortiran niz `$map` sada izgleda ovako:

```
Array ( [a] => rr
        [e] => zz
        [o] => kk
        [z] => hh )
```

Nema mnogo smisla koristiti funkciju `ksort()` za niz sa numeričkim indeksima jer su ključevi već sortirani. Kada se funkcija `krsort()` upotrebi za indeksiran niz, ona obrće redosled elemenata.

Sortiranje po redosledu koji definiše korisnik

Funkcije za sortiranje koje su dosad opisane u ovom odeljku sortiraju elemente niza po alfabetskom, numeričkom ili alfanumeričkom redosledu. U PHP-u postoje tri funkcije za sortiranje po proizvoljnom redosledu koji korisnik definiše:

```
usort(array za_sortiranje, string za_poređenje)
uasort(array za_sortiranje, string za_poređenje)
uksort(array za_sortiranje, string za_poređenje)
```

Funkcija `usort()` sortira niz *za_sortiranje* po vrednostima pojedinačnih elemenata, `uasort()` zadržava asocijacije ključ/vrednost, isto kao prethodno opisana funkcija `asort()`, a funkcija `uksort()` ređa elemente niza po vrednostima ključeva elemenata. Kada ove funkcije sortiraju elemente niza *za_sortiranje*, poziva se funkcija *za_poređenje*, koju korisnik treba da definiše, da bi se utvrdilo da li je element veći od, manji od ili jednak drugom elementu. Funkcija *za_poređenje* može se napisati za bilo koji naročit redosled sortiranja, ali mora imati sledeći oblik:

```
integer poredi_ovako(mešovit_tip a, mešovit_tip b)
```

Pisanje funkcija detaljnije je objašnjeno u odeljku „Funkcije koje korisnik piše“. Funkcija *za_poređenje* ima dva argumenta, a i b, a daje -1, kada je a manje od b, 1, kada je a veće od b i 0, kada je a jednako b. Kako funkcija utvrđuje da li je jedna vrednost, manja, veća ili jednaka drugoj zavisi od redosleda sortiranja koji se želi obezbediti. Naredni primer pokazuje kako funkcija `usort()` sortira niz znakovnih vrednosti na osnovu dužine svakog elementa niza:

```
// Poredi dve znakovne vrednosti na osnovu dužine
function poredi_duzinu($a, $b)
{
   if (strlen($a) < strlen($b)) return -1;
   if (strlen($a) > strlen($b)) return 1;

   // dužine znakovnih vrednosti su jednake
   return 0;
}

$zivotinje =
   array("krava", "vo", "antilopa", "žirafa");

usort($zivotinje, "poredi_duzinu");

print_r($zivotinje)
```

Ispisuje se sadržaj niza `$zivotinje`:

```
Array ([0]=>vo [1]=>krava [2]=>žirafa [3]=>antilopa)
```

U ovom primeru, funkcija `poredi_duzinu()` zadata je kao funkcija `za_poređenje`, ali nju skript ne poziva direktno. Ime funkcije, `"poredi_duzinu"`, prosleđuje se kao argument funkciji `usort()`, koja koristi funkciju `poredi_duzinu()` kao sastavni deo algoritma za sortiranje. Funkcije koje korisnik definiše upotrebljene na ovaj način često se nazivaju *povratnim* (engl. *callback*) funkcijama.

U bibliotekama PHP-a postoji više funkcija koje omogućavaju da korisnik zada određeno ponašanje pomoću povratnih funkcija koje sam definiše. Funkcije `array_map()` i `array_walk()` omogućavaju da se proizvoljna funkcija primeni na elemente niza. Još jedan takav primer nalazi se u dodatku D u kome implementiramo upravljanje sesijom na način koji korisnik definiše.

Znakovne vrednosti

Znakovna vrednost – vrednost znakovnog tipa – verovatno je najčešće upotrebljavan tip podataka u skriptovima. U PHP-u postoji obimna biblioteka funkcija za rad sa znakovnim vrednostima koje omogućavaju da menjate ili na druge načine upravljate vrednostima znakovnog tipa. Rad sa znakovnim vrednostima uveli smo već na početku poglavlja, u odeljku „Osnove PHP-a". Ovde ćemo podrobnije proučiti znakovne literale i opisati neke korisne PHP-ove funkcije za rad sa znakovnim vrednostima.

Znakovni literali

Kao što ste već videli u prethodnim primerima, kada uokvirite grupu znakova između jednostrukih ili dvostrukih navodnika, nastaje znakovni literal. Znakovne vrednosti uokvirene jednostrukim navodnicima najjednostavniji su oblik znakovnog literala; znakovne vrednosti napisane između dvostrukih navodnika PHP analizira da bi u njima zamenio imena promenljivih tekućim vrednostima i omogućio zadavanje kodiranih znakova pomoću specijalnih grupa znakova. U znakovnim vrednostima koje napišete

između jednostrukih navodnika nisu dozvoljene specijalne grupe znakova, osim \', koja označava jednostruki navodnik, i \\, koja označava obrnutu kosu crtu.

U znakovnu vrednost uokvirenu dvostrukim navodnicima mogu se umetnuti znaci za tabulator, nov red i početak reda; to su sekvence: \t, \n, odnosno \r. Da biste u znakovnu vrednost uokvirenu dvostrukim navodnicima umetnuli obrnutu kosu crtu, znak za dolar, ili dvostruki navodnik, upotrebite sekvence \\, \$ ili \".

Drugi upravljački znaci i znaci u kojima najznačajniji (osmi) bit ima vrednost 1 mogu se zadavati pomoću specijalnih grupa oktalnih ili heksadecimalnih znakova. Na primer, da biste umetnuli o sa umlautom (ö), upotrebite oktalnu sekvencu \366 ili heksadecimalnu \xf6:

```
// Ispisuje tekst koji sadrži
// malo o sa umlautom
echo "Vidimo se na festivalu u G\xf6teborgu";
```

U znakovnim vrednostima PHP koristi 8-bitne vrednosti, što znači da se mogu zadati znaci u opsegu od \000 do \377, u oktalnoj notaciji, ili od \x00 do \xff, u heksadecimalnoj notaciji.

Za razliku od mnogih drugih jezika, PHP dopušta da se znak za nov red umeće neposredno u znakovni literal. Naredni primer pokazuje kao se promenljivoj $var dodeljuje znakovna vrednost koja sadrži znak za nov red:

```
// Ovo je u redu, $var sadrži znak za nov red
$var = 'The quick brown fox
       jumps over the lazy dog';
```

Ova mogućnost se koristi u narednim poglavljima za formiranje SQL iskaza koji su čitljivi u izvornom kodu; na primer:

```
$upit = "SELECT max(order_id)
         FROM orders
         WHERE cust_id = $custID";
```

Drugi upravljački znaci, kao što su znak za tabulator i znak za početak reda, kao i znaci čiji najznačajniji bit ima vrednost 1 – oni u opsegu od \x80 do \xff – mogu se takođe upisivati direktno u znakovni literal. Međutim, preporučujemo da koristite specijalne sekvence radi bolje čitljivosti i prenosivosti datoteka sa izvornim kodom.

Zamena promenljivih tekućim vrednostima

Zamena promenljivih tekućim vrednostima je praktičan način za ispisivanje vrednosti promenljivih koje umetnete u znakovne literale. Kada PHP analizira znakovnu vrednost uokvirenu dvostrukim navodnicima, imena promenljivih prepoznaje po znaku $. Kad god naiđe na jednu od njih, zamenjuje je tekućom vrednošću promenljive. U prethodnom delu ovog poglavlja već smo prikazali primere slične sledećem:

```
$cm = 127;
$inch = $cm / 2.54;

// ispisuje "127 centimetara = 50 inča"
echo "$cm centimetara = $inch inča";
```

Kada ime promenljive nije lako prepoznatljivo, možete upotrebiti vitičaste zagrade {} da biste ga izdvojili od ostalog teksta, kao u narednom primeru:

```
$memorija = 256;

// Pogrešno: ne postoji promenljiva memorijaMB
$poruka = "Moj računar ima $memorijaMB RAM-a";

// Ispravno: ime promenljive se izdvaja pomoću
// vitičastih zagrada
$poruka = "Moj računar ima {$memorija}MB RAM-a";

// Ovo je takođe ispravno
$poruka = "Moj računar ima ${memorija}MB RAM-a";
```

Vitičaste zagrade se koriste i za složene promenljive, kao što su višedimenzionalni nizovi i objekti:

```
echo "Veličina Marsovog prečnika je {$planete['Mars']['prečnik']}
Zemljinog";

echo "Kupio je {$faktura->boca} zelenih boca ...";
```

Primer 2-4 pokazuje kako se dodeljuju vrednosti elementima višedimenzionalnog niza `$planete`, a objekti i operator za pristup elementima objekata `->` opisani su u odeljku „Objekti".

Dužina znakovnog niza

Dužina znakovnog niza utvrđuje se pomoću funkcije `strlen()`, koja daje ukupan broj 8-bitnih znakova u znakovnoj vrednosti `znakovna_vrednost`:

```
integer strlen(string znakovna_vrednost)
```

Naredni primer ispisuje vrednost 19:

```
print strlen("Ovo je znakovni niz");    // daje 19
```

Ispisivanje i formatiranje znakovnih vrednosti

U prethodnom delu ovog poglavlja opisali smo najjednostavnije načine za ispisivanje teksta – pomoću metoda `echo` i `print` – kao i funkcije `print_r()` i `var_dump()` koje daju sadržaj promenljivih, a upotrebljavaju se za otkrivanje grešaka.

U PHP-u postoji više drugih funkcija koje omogućavaju složenije formatiranje znakovnih vrednosti.

Ispisivanje formatiranog teksta pomoću funkcija sprintf() i printf()

Ponekad je potrebnije složenije formatiranje ispisane vrednosti od onog što omogućavaju metode `echo` i `print`. Na primer, kada vrednost sa pokretnim zarezom, kao što je 3,14159, treba prikazati kao 3,14. Kada je neophodno složenije formatiranje, korisne su funkcije `sprintf()` ili `printf()`:

```
string sprintf (string format [, mešoviti argumenti...])
integer printf (string format [, mešoviti argumenti...])
```

Ove funkcije rade na isti način kao funkcije sa jednakim imenima u programskom jeziku C: obe prihvataju prvi argument tipa string sa neobaveznim uputstvom za konverziju, a tom argumentu slede promenljive ili vrednosti koje se formatiraju. Razlika između `sprintf()` i `printf()` sastoji se u tome što se rezultat funkcije `printf()` prenosi direktno u izlazni bafer koji PHP koristi za formiranje HTTP odgovora, dok funkcija `sprintf()` vraća vrednost znakovnog tipa.

Pogledajmo primer iskaza `printf`:

```
printf("Rezultat je: %.2f\n", $promenljiva);
```

Znakovna vrednost koja opisuje format `Rezultat je: %.2f\n` je prvi parametar iskaza `printf`. Znakovni nizovi, kao što je `Rezultat je:`, ispisuju se u istom obliku kao pomoću iskaza `echo` ili `print`. Komponenta `%.2f` je uputstvo za konverziju:

- Svako uputstvo za konverziju počinje znakom `%`.
- Znak `f` opisuje kako treba tumačiti vrednost koja se formatira. Znak `f` znači da je u pitanju vrednost s pokretnim zarezom, kao što je 3,14159 ili 128,23765. Druge mogućnosti su `b`, `c`, `d` i `s`, gde se `b` koristi za binarne vrednosti, `s` za pojedinačne znakove, `d` za celobrojne (integer) vrednosti, a `s` za znakovne nizove.
- `.2` je neobavezni specifikator dužine. U ovom primeru, `.2` znači dva decimalna mesta; to znači da se kao konačan rezultat uputstva `%.2f` ispisuje broj sa pokretnim zarezom koji ima dva decimalna mesta. Specifikator `%5.3` znači da levo od decimalnog zareza (ili tačke) treba da bude najmanje pet cifara (ako ih ima manje od pet, rezultat se popunjava sleva razmacima), a desno od decimalnog zareza prikazuju se tri cifre (ako ih ima manje od tri, dopunjavaju se nulama zdesna).

U navedenom primeru, vrednost koja se formatira pomoću znakovnog niza za formatiranje `%.2f` zadata je kao drugi parametar funkcije `printf` – to je promenljiva `$promenljiva`.

Kao ilustraciju upotrebe funkcije `printf()` pogledajte sledećih nekoliko primera:

Primer 2-5. Upotreba funkcije printf za ispisivanje formatiranih podataka

```
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD HTML 4.0 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd" >
<html>
<head>
  <title>Primeri upotrebe funkcije printf()</title>
</head>
<body bgcolor="#ffffff">
<h1> Primeri upotrebe funkcije printf()</h1>
<pre>
<?php
  // Ispisuje "3.14"
  printf("%.2f\n", 3.14159);

  // Ispisuje "      3.14"
  printf("%10.2f\n", 3.14159);

  // Ispisuje "3.1415900000"
  printf("%.10f\n", 3.14159);
```

Primer 2-5. Upotreba funkcije printf za ispisivanje formatiranih podataka (nastavak)

```
  // Ispisuje "poluznak"
  printf("%.9s\n", "polaznakovnogniza");

  // Ispisuje "    3.14 3.141590       3.142"
  printf("%5.2f %f %7.3f\n", 3.14159, 3.14159, 3.14159);

  // Ispisuje "1111011 123 123.000000 test"
  printf("%b %d %f %s\n", 123, 123, 123, "test");
?>
</pre>
</body>
</html>
```

Dopunjavanje znakovnih vrednosti

Jednostavan način dopunjavanja znakovne vrednosti razmacima do zadate dužine jeste pomoću funkcije `str_pad()`:

```
string str_pad(string znak_vred, int dužina [, string znak_dopuna [, int
način_dopune]]))
```

Ulazni parametar `znak_vred` dopunjava se zadatim znakom tako da se dobije znakovna vrednost dužine *dužina*. Naredni primer prikazuje najjednostavniji oblik upotrebe funkcije `str_pad()` u kome se ulazna znakovna vrednost dopunjava razmacima:

```
// Ispisuje "PHP" iza čega slede tri razmaka
echo str_pad("PHP", 6);
```

Pomoću neobaveznog argumenta znakovnog tipa `znak_dopuna` zadaje se znak kojim se vrednost prvog argumenta dopunjava ukoliko to nije razmak. Ako ne zadate treći argument, `način_dopune`, vrednost prvog argumenta dopunjava se zdesna. Ako tom neobaveznom argumentu zadate vrednost `STR_PAD_LEFT` ili `STR_PAD_BOTH`, vrednost prvog argumenta se dopunjava sleva, odnosno na oba kraja. Naredni primer pokazuje kako se pomoću funkcije `str_pad()` može prikazati poravnat sadržaj:

```
$likovi =
   array("DUNCAN, kralj Škotske"=>"Laza",
         "MALCOLM, kraljević"=>"Cane",
         "MACBETH"=>"Mika",
         "MACDUFF"=>"Rade");

echo "<pre>";

// Ispisujemo zaglavlje
echo str_pad("Likovi ove drame", 50, " ", STR_PAD_BOTH) . "\n";

// Ispisujemo po jedan red za svaki element niza
foreach($likovi as $uloga=$glumac)
  echo str_pad($uloga, 30, "."
       . str_pad($glumac, 20, ".", STR_PAD_LEFT) . "\n";

echo "</pre>";
```

Rezultat ovog primera je sledeći:

```
                    Likovi ove drame
DUNCAN, kralj Škotske.....................Laza
MALCOLM, kraljević........................Cane
MACBETH...................................Mika
MACDUFF...................................Rade
```

Menjanje veličine slova

Naredne PHP funkcije daju kopiju ulazne vrednosti `znak_vred` u kojoj je promenjena veličina slova:

```
string strtolower(string znak_vred)
string strtoupper(string znak_vred)
string ucfirst(string znak_vred)
string ucwords(string znak_vred)
```

Naredni primeri pokazuju kako svaka od ovih funkcija radi:

```
print strtolower("PHP i MySQL");      // php i mysql
print strtoupper("PHP i MySQL");      // PHP I MYSQL
print ucfirst("sad je pravo vreme");  // Sad je pravo vreme
print ucwords("sad je pravo vreme");  // Sad Je Pravo Vreme
```

Odsecanje belina

U PHP-u postoje tri funkcije koje iz znakovnih vrednosti uklanjaju početne ili završne beline, odnosno znakove null, tabulator, vertikalni tabulator, nov red, početak reda i razmak:

```
string ltrim(string znak_vred)
string rtrim(string znak_vred)
string trim(string znak_vred)
```

Ove tri funkcije vraćaju kopiju ulaznog parametra `znak_vred`: funkcija `trim()` uklanja beline na početku i na kraju, `ltrim()` uklanja beline na početku, a `rtrim()` na kraju ulaznog znakovnog niza. Naredni primeri pokazuju delovanje ovih funkcija:

```
$var = trim(" Tiger Land\n");   // "Tiger Land"
$var = ltrim(" Tiger Land\n");   // "Tiger Land"
$var = rtrim(" Tiger Land\n");   // " Tiger Land"
```

Započinjanje novog reda pomoću oznake

Beline obično nemaju nikakvog značenja u HTML kodu, ali je često korisno da se očuvaju počeci redova teksta kada se stranica formira za prikazivanje. Funkcija `nl2br()` generiše znakovnu vrednost tako što u argumentu *izvor* umeće po jedan HTML element `<br />`* ispred svakog znaka za nov red:

```
string nl2br(string izvor)
```

* Od verzije PHP-a 4.0.5 nadalje, nl2br umeće oznaku
, prihvatljivu u XHTML-u, koja sadrži i prečicu za zatvaranje praznog elementa. U starijim verzijama PHP-a funkcija nl2br umetala je oznaku
, koja nije prihvatljiva u XML-u.

Naredni primer pokazuje kako radi funkcija `nl2br`:

```
// Kratka pesmica
$stihovi = "Isn't it funny\n";
$stihovi .= "That a bear likes honey.\n";
$stihovi .= "I wonder why he does?\n";
$stihovi .= "Buzz, buzz, buzz.\n";

// Sva četiri stiha ispisuju se kao jedan red
echo $stihovi;

// HTML kôd ispisuje stihove pesmice
// u četiri reda, kako i treba
echo nl2br($stihovi);
```

Poređenje znakovnih vrednosti

U PHP-u postoje funkcije za poređenje znakovnih vrednosti, `strcmp()` i `strncmp()`, koje tačno porede dve znakovne vrednosti, *str1* i *str2*:

```
integer strcmp(string str1, string str2)
integer strncmp(string str1, string str2, integer dužina)
```

Kada se dve znakovne vrednosti porede pomoću operatora jednakosti `==`, rezultati nisu uvek saglasni sa očekivanim ukoliko znakovne vrednosti sadrže znakove u kojima najznačajniji bit ima vrednost 1 (ANSI znakovi iznad 127). Obe funkcije, `strcmp()` i `strncmp()` prihvataju dva argumenta znakovnog tipa, *str1* i *str2*, i vraćaju nulu ako su vrednosti jednake, 1 kada je *str1* manje od *str2*, odnosno -1 ako je *str1* veće od *str2*. Naredni primeri prikazuju rezultate nekoliko poređenja:

```
print strcmp("aardvark", "zebra");          // -1
print strcmp("zebra", "aardvark");          // 1
print strcmp("mouse", "mouse");             // 0
print strncmp("aardvark", "aardwolf", 4);   // 0
print strncmp("aardvark", "aardwolf", 5);   // -1
```

Funkcije `strcasecmp()` i `strncasecmp()` su verzije funkcija `strcmp()` i `strncmp()` koje ne prave razliku između malih i velikih slova.

Funkcije `strncmp`, `strcasecmp()` ili `strncasecmp()` mogu se upotrebiti kao povratne funkcije kada se sortiraju nizovi pomoću funkcije `usort()`.

Pronalaženje i izdvajanje podvrednosti

PHP ima više jedostavnih i efikasnih funkcija koje mogu da pronađu i izdvoje zadatu podvrednost iz znakovne vrednosti.

Izdvajanje podvrednosti iz znakovne vrednosti

Funkcija `substr()` izdvaja zadatu podvrednost iz znakovne vrednosti izvor:

```
string substr(string izvor, integer početak [, integer dužina])
```

Kada je pozvana sa dva argumenta, funkcija `substr()` vraća sve znakove iz znakovne vrednosti *izvor* koji se nalaze između pozicije početak – počev od nule – i kraja znakovne vrednosti *izvor*. Ako je zadat i neobavezni argument *dužina*, funkcija vraća najviše dužina znakova. Naredni primeri pokazuju kako radi funkcija `substr()`:

```
$var = "abcdefgh";

print substr($var, 2);        // "cdefgh"
print substr($var, 2, 3);     // "cde"
print substr($var, 4, 10);    // "efgh"
```

Ako zadate negativnu početnu poziciju *početak*, pretražvanje počinje od desnog kraja znakovnog niza *izvor*. Ako je parametar *dužina* negativna vrednost, tumači se kao indeks, a podvrednost koji funkcija daje završava se na poziciji dužina od desnog kraja znakovnog niza *izvor*. Naredni primeri pokazuju kako se mogu koristiti negativni indeksi:

```
$var = "abcdefgh";

print substr($var, -1);       // "h"
print substr($var, -3);       // "fgh"
print substr($var, -5, 2);    // "de"
print substr($var, -5, -2);   // "def"
```

Pronalaženje početne pozicije podvrednosti

Funkcija `strpos()` daje prvu poziciju unutar znakovnog niza *plast_sena* na kojoj počinje podvrednost `igla`:

```
integer strpos(string plast_sena, string igla [, integer pomak])
```

Kada se ova funkcija pozove s dva argumenta, traženje podvrednosti `igla` počinje od početka znakovnog niza `plast_sena`, od pozicije nula. Kada se funkcija `str_pos` pozove s tri argumenta, traženje počinje od pozicije pomak u znakovnom nizu `plast_sena`. Naredni primeri pokazuju kako radi funkcija `strpos()`:

```
$var = "Biti ili ne biti, pitanje je sad";

print strpos($var, "B");         // 0
print strpos($var, "ne");        // 9

// Traženje počinje od petog znaka u $var
print strpos($var, "it", 4);     // 13
```

Funkcija `strrpos()` daje indeks (poziciju) poslednjeg pojavljivanja znaka `igla` u znakovnom nizu `plast_sena`:

```
integer strrpos(string plast_sena, string igla)
```

Za razliku od funkcije `strpos()`, funkcija `strrpos()` traži samo jedan znak, a uzima se u obzir samo prvi znak argumenta `igla`. Naredni primeri pokazuju kako radi funkcija `strrpos()`:

```
$var = "Biti ili ne biti, pitanje je sad";

// Ispisuje 27: poslednje pojavljivanje znaka "e"
print strrpos($var, "e");
```

```
// Ispisuje 0: traži samo znak "B"
// koji se nalazi na poziciji nula
print strrpos($var, "Bobi");

// False: "Z" nije sadržano u $var
print strrpos($var, "Zoo");
```

Ako funkcije `strpos()` i `strrpos()` ne pronađu podvrednost `igla`, obe funkcije daju rezultat `false`. Preporučujemo da pomoću operatora identičnosti `====` ispitate da li je povratna vrednost ovih funkcija jednaka konstanti `false`. Ako se podvrednost `igla` nalazi na samom početku znakovnog niza `plast_sena`, obe funkcije daju rezultat nula, koja se tumači kao `false` kada se koristi kao vrednost logičkog tipa.

Izdvajanje zadatog dela znakovnog niza

Funkcije `strstr()` i `stristr()` traže podvrednost `igla` u znakovnom nizu `plast_sena` i vraćaju njegov deo od pozicije na kojoj se prvi put pojavljuje podvrednost `igla` do kraja znakovnog niza `plast_sena`:

```
string strstr(string plast_sena, string igla)
string stristr(string plast_sena, string igla)
```

Pri traženju, funkcija `strstr()` pravi razliku između malih i velikih slova, dok funkcija `stristr()` to ne čini. Ako `igla` nije nađena u `plastu_sena`, obe funkcije daju rezultat `false`. Naredni primeri pokazuju kako ove funkcije rade:

```
$var = "Biti ili ne biti, pitanje je sad";

print strstr($var, "biti");   // biti, pitanje je sad
print stristr($var, "biti");  // Biti ili ne biti, pitanje je sad
print stristr($var, "oz");    // false
```

Funkcija `strrchr()` daje deo znakovne vrednosti `plast_sena` u kome traži prvi znak podvrednosti `igla`; međutim, `strrchar()` daje poziciju poslednjeg pojavljivanja podvrednosti `igla`:

```
string strrchr(string plast_sena, string igla)
```

Za razliku od funkcija `strstr()` i `stristr()`, `strrchr()` traži samo jedan znak, a uzima u obzir samo prvi znak koji zadate u parametru `igla`. Naredni primeri pokazuju kako radi funkcija `strrchr()`:

```
$var = "Biti ili ne biti, pitanje je sad";

// Ispisuje "je sad"
print strrchr($var, "j");

// Ispisuje "e sad"; traži samo "e"
// koje se poslednje nalazi na poziciji 27
print strrchr($var, "eh");
```

Izdvajanje više vrednosti iz jedne znakovne vrednosti

U PHP-u postoje funkcije `explode()` i `implode()` koje znakovne vrednosti pretvaraju u nizove i obrnuto:

```
array explode(string separator, string izvor [, granica])
string implode(string lepak, array delovi)
```

Funkcija `explode()` da je niz znakovnih vrednosti koji nastaje „seckanjem" znakovnog niza *izvor* na mestima gde se pojavljuje znak *separator*. Neobaveznim celobrojnim *parametrom* zadaje se maksimalan broj elemenata u rezultujućem nizu; kada je granica dosegnuta, poslednji element niza sadrži preostali deo znakovne vrednosti *izvor*. Funkcija `implode()` daje znakovnu vrednost formiranu nadovezivanjem svih elemenata niza *delovi*, pri čemu se između njih umeće znakovna vrednost *lepak*. Naredni primer prikazuje rad funkcija `implode()` i `explode()`:

```
$gostiLista = "Sale Mira Svetlana Brana Jelena Mika Aca";
$ime = "Mile";

// Proveravamo da li se $ime nalazi u $gostiLista
if (strpos($gostiLista, $ime) === false)
{
   $gostiNiz = explode(" ", $gostiLista);
   sort($gostiNiz);
   echo "Izvinjavamo se, $ime nije na listi gostiju.\n";
   echo "Lista gostiju: " . implode(", ", gostiNiz)
}
```

Kada funkcija `strpos()` ne nađe sadržaj znakovne promenljive `$ime` u promenljivoj `$gostiLista`, kôd ispisuje poruku o tome. Poruka sadrži sortiranu listu imena, razdvojenih zarezima; pomoću funkcije `explode()` formira se niz imena gostiju, a zatim pomoću funkcije `implode()` taj niz pretvara ponovo u znakovnu vrednost u kojoj su imena razdvojena zarezom i razmakom. Rezultat izvršavanja primera je sledeći ispis:

Zamenjivanje znakova i podvrednosti

PHP ima više funkcija koje omogućavaju zamenjivanje podvrednosti ili pojedinačnih znakova unutra date znakovne vrednosti. U narednom odeljku opisane su moćne alatke za pronalaženje i zamenjivanje složenih uzoraka znakova. Ovde opisane funkcije su moćnije od regularnih izraza i često su bolje rešenje kada treba pronalaziti i zamenjivati znakovne nizove.

Zamenjivanje podvrednosti

Funkcija `substr_replace()` zamenjuje zadatom vrednošću podvrednost na zadatoj poziciji (indeksu):

```
string substr_replace(string izvor, string zamena, int početak [, int
ukupan_broj])
```

Funkcija daje kopiju znakovne vrednosti izvor u kojoj su znaci od pozicije *početak* zamenjeni znakovnom vrednošću *zamena*. Ako zadate i neobavezni parametar *ukupan_broj*, zamenjuje se samo *ukupan_broj* znakova. Naredni primeri pokazuju kako radi funkcija `substr_replace()`:

```
$var = "abcdefghij";

// ispisuje "abcDEF";
echo substr_replace($var, "DEF", 3);

// ispisuje "abcDEFghij"
echo substr_replace($var, "DEF", 3, 3);

// ispisuje "abcDEFdefghij"
echo substr_replace($var, "DEF", 3, 0);
```

Funkcija `str_replace()` daje znakovnu vrednost koju formira tako što u argumentu izvor zamenjuje vrednošću *zamena* vrednost *traži*, na svim mestima gde se ona pojavljuje.

```
mešovit_tip str_replace(mešovit_tip traži, mešovit_tip zamena, mešovit_tip
izvor)
```

U narednom primeru, ispisuje se znakovna vrednost *izvor* u kojoj su sve reči `"staro"` zamenjene rečju `"novo"`:

```
$var = "staro shvatanje za staro vreme";

echo str_replace("staro", "novo", $var);
```

Rezultat je:

```
novo shvatanje za novo vreme
```

Od verzije PHP-a 4.0.5 nadalje, funkcija `str_replace()` prihvata da se kao parametri zadaju niz vrednosti za traženje i niz odgovarajućih vrednosti za zamenu. Naredni primer pokazuje kako se mogu popuniti polja kratkog dopisa:

```
// Kratko cirkularno pismo dužnicima
$opomena = "Poštovan#pol #titula #ime, iznos vašeg duga je: #iznos.";

// Formiramo niz od četiri elementa koje ćemo
// zamenjivati u cirkularnom pismu odgovarajućim
// vrednostima
$polja = array("#pol", "#titula"; "#ime", "#iznos");

// Niz dužnika. Svaki njegov element je niz
// koji sadrži vrednosti koje ćemo umetnuti u
// cirkularno pismo
$duznici = array(
    array("i", "gospodine", "Kostiću", "146,00"),
    array("a", "gospođo", "Mladenović", "1.662,00"),
    array("i", "profesore", "Mitiću", "84,75"));

foreach($duznici as $duznik)
  echo "<p>" . str_replace($polja, $duznik, $opomena);
```

Rezultat izvršavanja ovog skripta je sledeći:

```
Poštovani gospodine Kostiću, iznos vašeg duga je 146,00.
Poštovana gospođo Mladenović, iznos vašeg duga je 1.662,00.
Poštovani profesore Mitiću, iznos vašeg duga je 84,75.
```

Ako je niz vrednosti koje treba da zamene postojeće kraći od niza s vrednostima za traženje, vrednosti za koje nisu date zamene zamenjuju se praznim znakovnim nizovima.

Prevođenje pojedinačnih znakova i podvrednosti

Funkcija `strtr()` prevodi znakove ili podvrednost znakovne vrednosti *izvor*:

```
string strtr(string izvor, string staro, string novo)
string strtr(string izvor, array preslikati)
```

Kada se funkcija `strtr()` pozove sa tri argumenta, ona prevodi *znakove argumenta izvor* zadate argumentom *staro* u znakove zadate argumentom *novo*. Kada se funkcija pozove sa dva argumenta, znakovnom vrednošću *izvor* i nizom *preslikati*, na svim mestima gde se u izvoru pojavljuju ključevi iz niza *preslikati* zamenjuju se odgovarajućim vrednostima iz tog niza.

U narednom primeru se pomoću funkcije `strtr()` zamenjuju svi samoglasnici odgovarajućim znacima sa umlautom:

```
$bezveze = strtr("command.com", "aeiou", "äëïöü");
print $bezveze;  // ispisuje cömmänd.cöm
```

Kada se funkciji prosledi asocijativan niz kao argument za preslikavanje, ona zamenjuje podvrednosti umesto pojedinačnih znakova. Naredni primer pokazuje kako se pomoću funkcije `strtr()` može ispisati značenje akronima:

```
// kratka lista akronima koji se koriste
// u korespondenciji e-poštom
$recnik = array("BTW"=>"by the way",
                "IMHO"="in my humble opinion",
                "IOW"="in other words",
                "OTOH"="on the other hand")
// sada će možda postati razumljivije
print strtr($skraceno, $recnik);
```

Regularni izrazi

U ovom odeljku, objasnićemo kako se pomoću *regularnih izraza* (engl. *regular expressions*) može utvrđivati poklapanje sa složenijim uzorcima podataka radi pronalaženja, izdvajanja ili čak zamene složenijih podvrednosti unutar date znakovne vrednosti.

Iako regularni izrazi pružaju više mogućnosti od izraza opisanih u prethodnom odeljku, utvrđivanje poklapanja sa složenim uzorcima podataka nije tako efikasno kao prosto poređenje znakovnih vrednosti. Funkcije opisane u prethodnom odeljku efikasnije su od onih koje koriste regularne izraze i trebalo bi da se koriste kada nije potrebno utvrđivanje poklapanja sa složenim uzorcima podataka.

Ovaj odeljak počinje kratkim opisom POSIX-ove sintakse za regularne izraze. Ovde nije dat potpun opis svih mogućnosti, ali naći ćete dovoljno gradiva da sami formirate prilično moćne regularne izraze. U drugoj polovini odeljka opisane su funkcije koje koriste POSIX-ove regularne izraze. Primeri regularnih izraza nalaze se u ovom odeljku i u poglavlju 7.

Sintaksa regularnih izraza

Regularan izraz sledi stroga pravila sintakse koja opisuju uzorke znakova. PHP ima dve grupe funkcija koje koriste regularne izraze: jedna grupa podržava PCRE sintaksu (Perl Compatible Regular Expressions – regularni izrazi kompatiblilni s Perlom), dok druga podržava POSIX-ovu proširenu sintaksu za regularne izraze. U ovoj knjizi koristimo POSIX funkcije.

Da bismo objasnili sintaksu regularnih izraza, upotrebićemo funkciju `ereg()`:

```
boolean ereg(string uzorak, string izvor [, array var])
```

Funkcija `ereg()` daje rezultat `true` ukoliko se regularan izraz *uzorak* nalazi unutar znakovne vrednosti *izvor*. U nastavku ovog odeljka razmotrićemo kako funkcija `ereg()` može da izdvoji vrednosti u opcioni niz *var*.

Sledeći, izuzetno jednostavan, primer pokazuje kako se funkcija `ereg()` poziva da kako bi otkrila uzorak `"cat"` u izvornoj vrednosti `"raining cats and dogs"`:

```
// ispisuje "Nađeno je 'cat'"
if (ereg("cat", raining cats and dogs"))
  echo "Nađeno je 'cat'";
```

Regularan izraz `"cat"` daje poklapanje u znakovnoj vrednosti *izvor*, a kôd ispisuje „Nađeno je `'cat'`“.

Znaci i džokerski znaci

Da biste u uzorku predstavili bilo koji znak, upotrebite tačku kao džokerski znak. Uzorak `"c.."` daje poklapanje za svaku znakovnu vrednost od tri znaka koja počinje malim slovom `"c"`; na primer, `"cat"`, `"cow"`, `"cop"` itd. Da biste zadali uzorak koji sadrži tačku, ispred nje umetnite obrnutu kosu crtu `\` – na primer, `"\.com"` daje poklapanje za `".com"`, ali ne i za `"\.xcom"`.

Upotreba obrnute kose crte u regularnom izrazu može dovesti do zabune. Da biste umetnuli obrnutu kosu crtu u znakovni niz omeđen dvostrukim navodnicima, treba da promenite značenje obrnute kose crte tako što ćete ispred nje umetnuti još jednu takvu crtu. Naredni primer pokazuje kako se zadaje regularni izraz `"\.com"`:

```
// $found dobija vrednost true
$found = ereg("\\.com", "www.ora.com");
```

Da biste izbegli zabunu kada kao regularan izraz zadajete znakovnu vrednost, bolje je da je napišete između jednostrukih navodnika:

```
$found = ereg('\.com", "www.ora.com');
```

Liste znakova

Umesto džokerskog znaka koji znači poklapanje s bilo kojim znakom, u uzorku možete zadati listu znakova, omeđenu uglastim zagradama. Na primer, da biste zadali poklapanje znakovnih vrednosti od tri znaka, koje počinju i završavaju se slovom `"p"`, a srednje slovo je samoglasnik, možete upotrebiti sledeći izraz:

```
ereg("p[aeiou]p"), $var);
```

Rezultat će biti true za svaku znakovnu vrednost iz sledećeg skupa: "pap", "pep", "pip", "pop" ili pup". Možete zadati i opseg znakova; na primer: "[0-9]" zadaje sve cifre od 0 do 9:

```
// Poklapanje za "A1", "A2", "A3", "B1", ...
$found = ereg("[ABC][123]", "A3 format");   // true

// Poklapanje za "00" do "39"
$found = ereg("[0-3] [0-9]", "27");   // true
```

Listu znakova za koje se ne zahteva poklapanje možete zadati pomoću operatora negacije ^ koji ćete napisati kao prvi znak unutar zagrada. Uzorak "[^123]" daje poklapanje za sve znakove, osim za 1, 2 i 3. Naredni primeri prikazuju nekoliko regularnih izraza u kojima lista znakova sadrži operator negacije:

```
// Daje true za "pap", "pbp", "pcp" itd., ali ne i za "php"
$found = ereg("p[^h]p"), $var);

// Daje true ako $var ne sadrži
// alfanumeričke znakove
$found = ereg("[^0-9a-zA-Z]", $var);
```

Ako želite da se znak ^ tumači kao običan znak, postavite ga na bilo koje mesto, osim na prvo, unutra skupa znakova između uglastih zagrada. Na primer, regularan izraz "[0-9^]" daje poklapanje za cifre od 0 do 9 i znak ^. Za znak - možete zadati poklapanje ako ga postavite na početak ili na kraj liste; na primer, "[-123]" daje poklapanje za znakove -, 1, 2 i 3.

Sidra

Pomoću *sidra* u regularnom izrazu može se zadati da uzorak treba da se nalazi na početku ili na kraju vrednosti koja se ispituje. Znak ^ sidri uzorak na početak, a znak $ na kraj znakovne vrednosti. Na primer, izraz:

```
ereg("^php", $var);
```

daje poklapanje samo za znakovne vrednosti koje počinju slovima "php". Naredni primeri pokazuju upotrebu obe vrste sidra:

```
$var = "biti ili ne biti, pitanje je sad";
$poklapase = ereg("^biti", $var);   // true
$poklapase = ereg("sad$", $var);   // true
$poklapase = ereg("^ili", $var);   // false
```

Obe vrste sidra mogu se upotrebiti u regularnom izrazu kojim se zahteva potpuno poklapanje. To ilustruje naredni primer:

```
// Mora da bude tačno "Da"
$poklapase = ereg('^Da$', "Da");              // true
$poklapase = ereg('^Da$', "Da, slažem se");   // false
```

Neobavezni znaci i znaci koji mogu da se ponavljaju

Ako u regularnom izrazu iza nekog znaka napišete operator ?, * ili +, time zadajete poklapanje nula ili jednog, nula ili više, odnosno jednog ili više znakova.

Operatorom ? zadaje se nula ili jedno pojavljivanje određenog znaka; dakle, izraz:

```
ereg("pe?p", $var);
```

daje poklapanje za vrednosti "pep" ili "pp", ali ne i za "peep". Operatorom * zadaje se nula ili više pojavljivanja slova "o" u narednom izrazu:

```
ereg("po*p", $var);
```

U ovom slučaju imamo poklapanje za vrednosti "pp", "pop", "poop", "pooop" itd. I najzad, operatorom + zadaje se jedno ili više pojavljivanja slova "b" u narednom izrazu:

```
ereg("ab+a", $var);
```

U ovom slučaju imamo poklapanje za vrednosti "aba", "abba", "abbba", ali ne i za "aa".

Operatori ?, * i + mogu se koristiti i u kombinacijama sa džokerskim znacima ili sa listama znakova. Naredni primeri pokazuju kako se to radi:

```
$var = "www.rmit.edu.au";

// Daje true za znakovne vrednosti koje
// počinju sa "www", a završavaju se sa "au"
$poklapase = ereg('^www.*aus$', $var);   // true

$hexString = "x01ff";

// Daje true za znakovne vrednosti koje počinju
// znakom 'x' kome sledi bar jedna heksadecimalna
// cifra
$poklapase = ereg('x[0-9a-fA-F]+$', $hexString);   // true
```

U prvom primeru imaćemo poklapanje za svaku vrednost koja počinje znacima "www" i završava se sa "au"; uzorak ".*" daje poklapanje za grupu bilo kojih znakova, uključujući i prazan znakovni niz. U drugom primeru imaćemo poklapanje za svaku vrednost koja počinje znakom "x", kome sledi jedan ili više znakova sa liste [0-9a-fA-F].

Fiksni broj pojavljivanja tražene podvrednosti može se zadati između vitičastih zagrada. Na primer, uzorak "[0-7]{3}" daje poklapanje za sve trocifrene brojeve koji sadrže cifre od 0 do 7:

```
$prihvatljivo = ereg("[0-7]{3}", "075");   // true
$prihvatljivo = ereg("[0-7]{3}", "75");    // false
```

Pomoću sintakse sa vitičastim zagradama mogu se zadati minimalan i maksimalan broj pojavljivanja, kao što ilustruje naredni primer:

```
$val = "58273";

// Daje true ako $val sadrži isključivo cifre
// a broj cifara je u opsegu od 4 do 6
$prihvatljivo = ereg('^[0-9]{4,6}$', $val);    // true

$val = "5827003";
$prihvatljivo = ereg('^[0-9]{4,6}$', $val);    // false

// Bez sidra na početku i kraju uzorka,
```

```
// imamo poklapanje za uzorak "582768"
$val = "582768986456245003";
$prihvatljivo = ereg("[0-9]{4,6}", $val);    // true
```

Grupe

Poduzorci u regularnom izrazu mogu se grupisati tako što se uokvire zagradama. To omogućava da se operatori za neobaveznost i ponavljanje primene na cele grupe umesto na pojedinačne znakove. Na primer, izraz:

```
ereg("(123)+", $var);
```

daje poklapanje za `"123"`, `"123123"`, `"123123123"` itd. Grupisanjem znakova mogu se predstaviti složeni uzorci, kao u sledećem primeru u kome se kontroliše format URL-a:

```
// Jednostavan, ali nepotpun regularan izraz za HTTP URL,
// koji ne prihvata cifre

$uzorak = '^(http://)?[a-zA-Z]+(\.[a-zA-Z]+)+$';

$prihvatljivo = ereg($uzorak, "www.ora.com");   // true
```

Regularan izraz dodeljen promenljivoj `$uzorak` sadrži i početno i završno sidro, ^ i $, što znači da *znakovna vrednost* koja se ispituje, `"www.ora.com"`, mora da se u celini poklapa sa uzorkom. Uzorak počinje neobaveznom grupom znakova `"(http://)?"`. Taj deo u našem primeru ne daje poklapanje, ali ga ni ne isključuje, zato što uzorak `"http://"` nije obavezan. Zatim se uzorkom `"[a-zA-Z]+"` zadaje jedan ili više alfabetskih znakova, što nam daje poklapanje za `"www"` u *znakovnoj vrednosti* koju ispitujemo. Naredni uzorak je grupa `"(\.[a-zA-Z]+)"`. Ovaj uzorak počinje tačkom – obrnuta kosa crta menja značenje tačke, koja u ovom slučaju nije džokerski znak – kome sledi jedan ili više alfabetskih znakova. Uzorku u ovoj grupi sledi operator +, što znači da se uzorak mora pojaviti barem jedanput, a može se ponoviti neograničen broj puta. U ovom primeru, prvo pojavljivanje je `"ora"`, a drugo `"com"`.

Pomoću grupa mogu se definisati i poduzorci, što funkciji `ereg()` omogućava da izdvaja vrednosti u niz. Postupak ćemo objasniti u nastavku ovog odeljka.

Alternativni uzorci

Alternativni iuzorci se zadaju pomoću operatora |; na primer, uzorak `"cat|bat|rat"` daje poklapanje za `"cat"`, `"bat"` ili `"rat"`. Operator | ima najniži prioritet od svih operatora u regularnim izrazima i tumači uzorak zadat između navodnika kao sastavljen od više alternativnih uzoraka. Da biste na drugi način utvrdili poklapanje za `"cat"`, `"bat"` ili `"rat"`, možete upotrebiti naredni izraz:

```
$var = "bat";
$prihvatljivo = ereg("(c|b|r)at", $var);   // true
```

Sledeći primer pokazuje kako se zadaju alternativni počeci uzorka:

```
// Daje poklapanje za neke URL-ove
$uzorak = '(^ftp|^http|^gopher)://';

$prihvatljivo = ereg($uzorak, "http://www.ora.com");   //true
```

Promena značenja posebnih znakova

Već smo razmatrali potrebu da se nekim posebnim znacima ponekad promeni značenje operatora u regularnim izrazima. Međutim, kada to treba učiniti, zavisi od načina na koji se znak koristi. Posebno značenje određenog znaka isključuje se pomoću obrnute kose crte, kao u izrazu `"2\+3"`, što daje poklapanje za znakovnu vrednost `"2+3"`. Kada bi se znak + tumačio sa standardnim značenjem operatora, ovaj uzorak bi dao poklapanje za jedno ili više pojavljivanja znaka 2, kome sledi znak 3. Drugi način da se napiše ekvivalentan uzorak jeste da se + stavi na listu znakova, kao `"2[+]3"`. Pošto + nema značenje operatora kada je na listi znakova, u tom kontekstu nema potrebe da mu se izričito menja značenje. Upotreba lista znakova na ovaj način može da poboljša čitljivost. Naredni primeri pokazuju kako se može izvesti i izbeći promena značenja:

```
// Ovde moramo promeniti značenje operatora ( i )
$telefon = "(03) 9429 5555";
$prihvatljivo = ereg("^\([0-9]{2,3}\)", $telefon);   // true

// Znacima (*.+?) ne treba menjati značenje
// ukoliko se nalaze između zagrada
$posebni = "Posebni znaci su (, ), *, +, ?, |";
prihvatljivo = ereg("[(*.+?)]", $posebni);   // true

// Obrnuta kosa crta mora se zadati između
// jednostrukih navodnika da bi se dobilo poklapanje
$obrCrta = 'Znak obrnuta kosa crta \';
$prihvatljivo = ereg('^[a-zA-Z \\]*$', $obrCrta);   // true

// Kada se tačka nalazi između zagrada, ne treba
// joj izričito menjati značenje
$domen = "www.ora.com";
$prihvatljivo = ereg("[.]com", $domen);   // true
```

Uzrok još jednog problema jeste činjenica da se regularan izraz prosleđuje u obliku znakovne vrednosti funkcijama za rad sa regularnim izrazima. U PHP-u, znakovne vrednosti mogu sadržati obrnutu kosu crtu kojom se menja značenje jednostrukog navodnika (apostrofa) ili kodiraju znaci tabulator, nov red itd. Pogledajte sledeći primer, koji daje poklapanje za obrnutu kosu crtu:

```
// Znakovna vrednost omeđena jednostrukim navodnicima
// koja sadrži obrnutu kosu crtu
$obrCrta = '\ obrnuta kosa crta';

// Daje vrednost true
$prihvatljivo = ereg("^\\\\ obrnuta kosa crta\$", $obrCrta);
```

Ovaj regularan izraz izgleda prilično neubičajeno: da bi otkrio poklapanje sa obrnutom kosom crtom, regularan izraz mora najpre da joj promeni značenje, ali pošto je uzorak zadat između dvostrukih navodnika, svakoj obrnutoj kosoj crti treba promeniti značenje. Poslednji problem je to što PHP tumači znak $ kao početak imena promenljive, pa zato i tom znaku moramo da promenimo značenje. Upotreba jednostrukih navodnika može da olakša čitanje i pisanje regularnih izraza.

Metaznaci

U regularnim izrazima mogu se upotrebljavati i metaznaci. Na primer, znak tabulator predstavlja se kao `\t`, a znak za nov red kao `\n`. Postoje i skraćenice: `\d` znači bilo koja cifra, a `\s` znači bilo koja belina. Naredni primer daje rezultat `$true` jer je znak tabulator, `\t`, sadržan u znakovnoj vrednost `$izvor`:

```
$izvor = "fast\tfood";

$rezultat = ereg('\s', $izvor);   // true
```

Funkcije za rad s regularnim izrazima

PHP ima više funkcija koje na osnovu regularnih izraza zadatih u POSIX sintaksi pronalaze i izdvajaju podvrednosti znakova, zamenjuju podvrednosti znakova datim vrednostima ili razdvajaju znakovnu vrednost na elemente niza. Funkcije koje obavljuju navedene poslove postoje u dve varijante: jedna koja pravi razliku između malih i velikih slova, i druga koja to ne čini. Iako se mogu pisati regularni izrazi u kojima se pravi razlika između malih i velikih slova, verzije ovih funkcija koje ne prave razliku između malih i velikih slova omogućavaju pisanje kraćih regularnih izraza.

Pronalaženje i izdvajanje vrednosti

Funkcija `ereg()` i njena verzija `eregi()`, koja ne pravi razliku između malih i velikih slova, definišu se na sledeći način:

```
boolean ereg(string uzorak, string izvor [, array var])
boolean eregi(string uzorak, string izvor [, array var])
```

Obe funkcije daju rezultat `true` ako je regularan izraz uzorak pronađen unutar znakovne vrednosti izvor. Kao treći, neobavezan argument može se zadati promenljiva var; ona se popunjava delovima argumenta *izvor* za koje je otkriveno poklapanje sa jednim do devet grupisanih podizraza zadatih u argumentu uzorak. Obe funkcije daju rezultat `false` ukoliko uzorak nije pronađen u argumentu *izvor*.

Da biste izdvojili delove iz znakovne vrednosti u niz, uzorke u regularnom izrazu možete grupisati pomoću zagrada. Naredni primer pokazuje kako se komponente datuma – godina, mesec i dan – mogu izdvojiti u niz:

```
$delovi = array();
$vrednost = "2001-09-07";
$uzorak = '^([0-9]{4})-([0-9]{2})-([0-9]{2})$';

ereg($uzorak, $vrednost, $delovi);

// Niz: [0]=2001-09-07 [1]=2001 [2]=09 [3]=07
print_r($delovi);
```

Izraz:

```
'^([0-9]{4})-([0-9]{2})-([0-9]{2})$'
```

daje poklapanje za datume u formatu GGGG-MM-DD. Pošto je pozivana funkcija `ereg()`, elementu delovi[0] dodeljuje se deo izvorne vrednosti koji se u potpunosti

poklapa s regularnim izrazom – u ovom slučaju, to je cela vrednost `2001-09-07`. Delovi datuma koji se poklapaju s pojedinim grupama u izrazu dodeljuju se sledećim elementima niza: `$delovi[1]` sadrži godinu, izdvojenu pomoću grupe `([0-9]{4})`, `$delovi[2]` sadrži mesec, izdvojen pomoću grupe `([0-9]{2})`, a `$delovi[3]` sadrži dan, izdvojen pomoću druge grupe `"([0-9]{2})"`.

Zamena delova izvorne vrednosti

Naredne funkcije formiraju nove znakovne vrednosti tako što u izvornim zamenju zadate delove:

```
string ereg_replace(string uzorak, string zamena, string izvor)
string eregi_replace(string uzorak, string zamena, string izvor)
```

Ove funkcije formiraju nove znakovne vrednosti tako što delove argumenta *izvor* za koje regularni izraz uzorak otkrije poklapanje zamenjuju vrednošću argumenta *zamena*. Ove funkcije su slične funkciji `str_replace()`, opisane u prethodnom delu ovog poglavlja u odeljku „Znakovne vrednosti“, s tom razlikom što se delovi za zamenu pronalaze pomoću regularnog izraza. Pogledajmo nekoliko primera:

```
$izvor = "The  quick\tbrown\n\tfox jumps";

// ispisuje "The quick brown fox"
echo ereg_replace("[ \t\n]+", " ", $izvor);

$izvor = "\xf6 The   quick\brown\n\tfox jumps\x88";

// zamenjuje razmakom sve znakove koji nisu prikazivi
echo ereg_replace("[^ -~]+", " ", $izvor);
```

Drugom primeru se pomoću regularnog izraza `"[^ -~]+"` utvrđuje poklapanje za sve znakove, osim za one koji se u ASCII tabeli nalaze između znaka za razmak i znaka tilde. To su gotovo svi prikazivi 7-bitni znaci.

Razdvajanje izvorne vrednosti na elemente niza

Naredne dve funkcije razdvajaju vrednosti svojih ulaznih argumenata:

```
array split(string uzorak, string izvor [, integer granica])
array spliti(string uzorak, string izvor [, integer granica])
```

One od ulaznog argumenta *izvor* formiraju elemente novog niza tako što vrednost ulaznog argumenta „seckaju“ na mestima gde se otkrije poklapanje sa vrednošću zadatom u argumentu *uzorak*. Ove funkcije obavljaju sličan zadatak kao funkcija `explode`, opisana u prethodnom delu ovog poglavlja. Kao i za funkciju `explode()`, neobaveznim argumentom `granica` zadaje se maksimalan broj elemenata u rezultujućem nizu.

Naredni primer pokazuje kako se rečenica razdvaja na niz „reči“, tako što se svaka grupa nealfabetskih znakova tumači kao separator:

```
$recenica = "I wonder why does\nBuzz, buzz, buzz!";
$reci = split("[^a-zA-Z]+", $recenica);
```

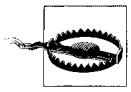
Kada za razdvajanje znakovne vrednosti na elemente niza nisu potrebni složeni uzorci, funkcija `explode()` je bolji izbor.

Funkcije za rad s datumima i vremenom

U bibliotekama PHP-a postoji više funkcija za rad s datumima i vremenom.Većina daje rezultat u Unixovom formatu timestamp ili formatira podatak tipa timestamp u čitljiv oblik.

Generisanje rezultata u formatu Timestamp

Datumi i vreme obično se predstavljaju u Unixovom formatu timestamp: broj sekundi od 1. januara 1970, 00:00:00, griničko vreme. Na većini sistema, format timestamp se predstavlja pomoću 32-bitne vrednosti tipa integer s predznakom, što omogućava predstavljanje datuma od 13. decembra 1901, do 19. januara 2038. Iako je format timestamp pogodan za upotrebu u skriptovima, treba biti pažljiv da bi se izbegle greške prekoračenja kapaciteta za tip integer. Česta greška je poređenje dva podataka tipa timestamp u kojima je opseg datuma veći od najveće pozitivne vrednosti tipa integer, a to je za 32--bitnu vrednost tipa integer opseg koji malo premašuje 68 godina.



PHP daje neočekivane rezultate kada se porede dve vrednosti tipa integer između kojih je razlika veća od najveće pozitivne vrednosti za tip integer, obično 2^{31}-1. Bezbedan način poređenja dve vrednosti tipa integer jeste da se one preslikaju u brojeve s pokretnim zarezom. Naredni primer ilustruje tu mogućnost:

```
$var1 = -2106036000;   // 16.08.1902
$var2 = 502808400;     // 24.08.1984

// $rezultat dobija vrednost false
$rezultat = $var1 < var2;

// $rezultat dobija vrednost true,
// kao što bi se i očekivalo
$rezultat = (float) $var1 < (float) var2;
```

Tekuće vreme

PHP ima više funkcija koje daju rezultate u Unixovom formatu timestamp. Najjednostavnija:

```
integer time()
```

daje tekući datum i vreme u timestamp formatu, kao u sledećem primeru:

```
// ispisuje timestamp oblik tekućeg datuma i vremena
// na primer: 1008553254
echo time();
```

Generisanje podataka u timestamp formatu pomoću funkcija mktime() i gmmktime()

Za generisanje podataka u timestamp formatu za prošli ili budući datum, u opsegu od 13. decembra 1901. do 19. januara 2038, definisane su funkcije `mktime` i `gmmktime()`:

```
int mktime(int sati, int minuti, int sekunde, int mesec, int dan, int godina
     [, int is_dst])
int gmmktime(int sati, int minuti, int sekunde, int mesec, int dan, int
godina
     [, int is_dst])
```

Obe funkcije daju rezultate u timestamp formatu za komponente koje im prosledite; razlika je u tome što parametri funkcije `gmmktime()` predstavljaju griničko vreme, dok parametri funkcije `mktime()` predstavljaju lokalno vreme. Naredni primer generiše timestamp rezultat za 9:30, 18. juna 1998.:

```
$datum = mktime(9, 30, 0, 6, 18, 1998);
```

Obe funkcije prihvataju, u razumnim granicama, nule kao vrednosti ulaznih parametara. Osim toga, obe pravilno obrađuju vrednosti izvan prihvatljivih opsega, što omogućava da se one koriste u skriptovima za dodavanje određenog vremenskog intervala bez potrebe da se proverava ispravnost opsega. Ako su komponente datuma izvan opsega za koji je funkcija definisana, rezultat je -1. Naredni primer pokazuje kako se datumu i vremenu dodaje 30 dana:

```
$rokZaUplatu = 30;   // dana

// Generiše timestamp za 26. jun 2002. tako
// što dodaje 30 dana datumu 27. maj 2002.
$uplatitiDo =
mktime(0, 0, 0, 5, 27 + $rokZaUplatu, 2002);

// Druga mogućnost je da se doda odgovarajući broj
// sekundi timestamp obliku datuma 27. maj 2002.
$uplatitiDo = mktime(0, 0, 0, 5, 27, 2002)
  + ($rokZaUplatu * 24 * 3600);
```

Obe funkcije uzimaju u obzir i razliku pri prelasku na letnje/zimsko računanje vremena ako indikatoru `is_dst` dodelite vrednost 1.

Redosled argumenata ovih funkcija je neuobičajen i lako dovodi do zabune. Iako su funkcije `mktime()` i `gmmktime()` slične Unixovoj funkciji `mktime()`, argumenti nisu u jednakom redosledu.

Pretvaranje znakovne vrednosti u format timestamp

Funkcija `strtotime()` daje rezultat u timestamp formatu na osnovu datuma i vremena, od 13. decembra 1901. do 19. januara 2038, koje joj zadate u tekstualnom obliku argumentom *vreme*:

```
integer strtotime(string vreme)
```

Ova funkcija prihvata više standardnih formata datuma:

```
// Apsolutni datumi i vreme
$var = strtotime("25 December 2002");
$var = strtotime("14/5/1955");
$var = strtotime("Fr1, 7 Sep 2001 10:28:07 -1000");
```

```
// Tekuće vreme: ekvivalentno pozivanju funkcije time()
$var = strtotime("now");

// Relativno vreme
echo strtotime("+1 day");
echo strtotime("-2 weeks");
echo strtotime("+2 hours 2 seconds");
```

Budite oprezni kada funkciji strtotime prosleđujete datum koji je korisnik uneo. Preporučujemo da upotrebu funkcije `strtotime()` ograničite na slučajeve kada ispravnost ulaznog podatka može prethodno da se proveri u skriptu; na primer, kada se ispituje minimalna obavezna starost izračunavanjem relativnog datuma:

```
// datum rođenja: timestamp za 16. avgust 1983.
$datum = mktime(0, 0, 0, 16, 8, 1982);

// Sada proveramo da li osoba ima više od 18 godina
if (float)$datum < (float)strtotime("-18 years"))
  echo "Ima pravo na vozačku dozvolu."
```

Obratite pažnju na to da se, pre poređenja, oba podatka iz timestamp formata preslikavaju u tip float da bi se izbegli problemi usled prekoračenja kapaciteta tipa integer, koji su opisani u prethodnom delu ovog poglavlja. U poglavlju 7 predstavljeno je drugačije rešenje ovog problema.

Vremenski intervali kraći od jedne sekunde

Iako Unixov format timestamp predstavlja datume i vremena sa tačnošću od jedne sekunde, u mnogim aplikacijama neophodno je predstavljanje vremenskih intervala kraćih od sekunde. U PHP-u postoji funkcija:

```
string microtime()
```

Ova funkcija daje znakovnu vrednost koja se sastoji od podatka u Unixovom formatu timestamp i komponente koja predstavlja mikrosekunde. Rezultujuća znakovna vrednost počinje komponentom koja predstavlja mikrosekunde, kojoj sledi celobrojni timestamp:

```
// ispisuje tekuće vreme u formatu "mikrosekunde sekunde"
// na primer: 0.34783800 1008553410
echo microtime();
```

Funkcija `microtime()` se često upotrebljava za generisanje početnog zrna za generator nasumičnih brojeva:

```
// Generišemo zrno.
$zrno = (float)microtime() * 100000000;

srand($zrno);
```

Pošto se komponenta koja predstavlja mikrosekunde nalazi u početnom delu rezultata funkcije `microtime()`, ta vrednost se može preslikati u tip float pomoću operatora `(float)`. Množenjem tako dobijenog rezultata obezbeđujete da funkcija za inicijalizovanje generatora, `srand()`, dobije uvek različitu vrednost zrna. Generisanje nasumičnih brojeva detaljnije je opisano u odeljku „Funkcije za rad s vrednostima tipa Integer i Float".

Formatiranje datuma

Iako je Unixov format tiemstamp koristan za programiranje, on nije pogodan za prikazivanje datuma. Funkcije `date()` i `gmdate()` daju datum i vreme u formatu koji je čitljiv krajnjem korisniku:

```
string date(string format [, integer timestamp])
string gmdate(string format [, integer timestamp])
```

Format rezultujuće znakovne vrednosti zavisi od argumenta *format*. Datum koji želite da formatirate možete da prosledite funkcijama neobaveznim argumentom *timestamp*. Ako taj argument ne zadate, obe funkcije formatiraju tekuće vreme. Za argument format koriste se znaci za formatiranje navedeni u tabeli 2-3, koji omogućavaju prikazivanje raznih komponenata ili karakteristika ulaznog podatka zadatog kao *timestamp*. Da bi se u prikazu datuma pojavio i znak iz tabele, postavite ispred njega obrnutu kosu crtu. Naredni primeri pokazuju neke od mogućih kombinacija:

```
// Formiramo timestamp za 08:15 24. avgusta 1964.
$var = mktime(8, 15, 25, 8, 24, 1964);

// "24/08/1964"
echo date('d/m/y', $var);

// "08/24/1964"
echo date('m/d/y', $var);

// Born on Thursday 24th of August"
echo date('\B\o\r\n \o\n l jS \of F', $var);
```

Tabela 2-3. Znaci za formatiranje koji predstavljaju komponente datuma i vremena

Znak za formatiranje	Značenje
a, A	"am" ili "pm"; "AM" ili "PM" (za 12-časovni format prikazivanja vremena)
S	Sufiks od dva znaka koji opisuje redosled (na engleskom jeziku): "st", "nd", "rd", "th"
d, j	Dan u mesecu; s početnom nulom: "01"; bez nje: "1"
D, l	Ime dana u sedmici (na engleskom jeziku); predstavljeno s tri slova: "Mon", ili s punim imenom: "Monday"
M, F	Ime meseca (na engleskom jeziku); predstavljeno s tri slova: "Jan", ili s punim imenom: "January"
m, n	Mesec: s početnom nulom: "01" – "12"; bez nje: "1" – "12"
h, g	Sati, 12-časovni format: s početnom nulom: "09"; bez nje: "9"
H, G	Sati, 24-časovni format: s početnom nulom: "01"; bez nje: "1"
i	Minuti: "00" do "59"
s	Sekunde: "00" do "59"
Y, y	Godine: četvoricifrene "2002"; dvocifrene "02"

Tabela 2-3. Znaci za formatiranje koji predstavljaju komponente datuma i vremena (nastavak)

Znak za formatiranje	Značenje
r	Vreme u RFC-2822 formatu: npr.: "Tue, 29 Jan 2002 09:15:33 + 1000" (dodato u PHP-u 4.0.4)
w	Dan sedmice predstavljen kao redni broj: "0" (nedelja) do "6" (subota)
t	Ukupan broj dana u mesecu: "28" do "31"
z	Dan predstavljen kao redni broj u godini: "0" do "365"
B	Vreme se predstavlja u formatu Swatch Internet Time
L	Prestupna godina: "0" obična godina; "1" prestupna godina
I	"0" standardno vreme; "1" uzima se u obzir prelazak na letnje/zimsko računanje vremena
O	Razlika u odnosu na griničko vreme, u satima: "+0200"
T	Vremenska zona na koju je podešen lokalni računar
Z	Razlika vremenske zone, u sekundama: "-43200" do "43200"
U	Broj sekundi koji je protekao od početka računanja vremena: 00:00:001/1/1970

U PHP-u postoje i ekvivalentne funkcije:

```
string strftime(string format [, integer timestamp])
string gmstrftime(string format [, integer timestamp])
```

U znakovnom argumentu *format* koriste se iste grupe znakova za formatiranje kao u funkciji `strftime()` u bibliteci jezika C.

Proveravanje ispravnosti datuma

Funkcija `checkdate()` daje rezultat `true` ukoliko zadati mesec, dan i godina čine ispravan gregorijanski datum:

```
boolean chekdate(integer mesec, integer dan, integer godina)
```

Budući da ova funkcija ne radi s podacima u timestamp formatu, može da prihvati širi opseg datuma; otprilike sve datume s godinama od 1 do 32767. Ona automatski uzima u obzir prestupne godine.

```
// Radi sa širokim opsegom datuma
$prihvatljivo = checkdate(1, 1, 1066);    // true
$prihvatljivo = checkdate(1, 1, 2929);    // true

// Tačno otkriva neprihvatljive datume
$prihvatljivo = checkdate(13, 1, 1996);   // false
$prihvatljivo = checkdate(4, 31, 1066);   // false

// Tačno obrađuje prestupne godine
$prihvatljivo = checkdate(2, 29, 1996);   // true
$prihvatljivo = checkdate(2, 29, 2001);   // false
```

Funkcije za rad s vrednostima tipa Integer i Float

Osim osnovnih operatora +, -, /, * i %, u PHP-u postoji i biblioteka uobičajenih matematičkih funkcija. U ovom odeljku opisujemo neke od funkcija iz biblioteke koje se koriste za rad s vrednostima tipa Integer i Float.

Apsolutna vrednost

Apsolutna vrednost argumenta tipa integer ili float dobija se pomoću funkcije `abs()`:

```
integer abs(integer broj)
float abs(float broj)
```

Naredni primeri pokazuju rezultate dobijene primenom funkcije `abs()` na celobrojne i decimalne vrednosti:

```
echo abs(-1);          // ispisuje 1
echo abs(1);           // ispisuje 1
echo abs(-145.89);     // ispisuje 145.89
echo abs(145.89);      // ispisuje 145.89
```

Najviša i najniža celobrojna vrednost

Funkcije `ceil()` i `floor()` daju celobrojne vrednosti neposredno iznad, odnosno ispod vrednosti decimalnog argumenta *vrednost*:

```
float ceil(float vrednost)
float floor(float vrednost)
```

Rezultat je tipa float zato što tip integer možda neće biti dovoljan ukoliko se kao ulazni argument prosledi veoma velika vrednost. Pogledajte naredni primer:

```
echo ceil(27.3);     // ispisuje 28
echo floor(27.3);    // ispisuje 27
```

Zaokruživanje brojeva

Funkcija `round()` primenjuje pravilo 4/5 za zaokruživanje nagore ili nadole ulaznog argumenta *vrednost*, uz zadatu *preciznost*:

```
float round(float vrednost [, integer preciznost])
```

Podrazumeva se zaokruživanje na nula decimalnih mesta, ali se preciznost zaokruživanja može zadati pomoću neobaveznog argumenta *preciznost*. Pravilo zaokruživanja 4/5 određuje da li se broj zaokružuje nagore ili nadole, u zavisnosti od cifara koje se odbacuju zbog zadate preciznosti zaokruživanja. Na primer, 10.4 se zaokružuje nadole, na 10, a 10.5 nagore, na 11. Naredni primeri pokazuju rezultat zaokruživanja s nekoliko različitih preciznosti:

```
echo round(10.4);            // ispisuje 10
echo round(10.5);            // ispisuje 11
echo round(2.40964, 3);      // ispisuje 2.410
echo round(567234.56, -3);   // ispisuje 567000
echo round(567234.56, -4);   // ispisuje 570000
```

Numerički sistemi

PHP ima sledeće funkcije za konverziju celih brojeva iz decimalnog sistema u uobičajeni binarni, oktalni i heksadecimalni sistem:

```
string decbin(integer broj)
integer bindec(string binarnistring)
string dechex(integer broj)
string hexdec(string hexstring)
string decoct(integer broj)
integer octdec(string oktalnistring)
```

Decimalni brojevi se uvek obrađuju kao celobrojne vrednosti, a brojevi iz drugih numeričkih sistema tumače se kao znakovne vrednosti. Pri konverziji u decimalni sistem, vodite računa da izvorni broj ne bude veći od maksimalnog koji je prihvatljiv za tip integer. Evo nekoliko primera:

```
echo decbin(45);         // ispisuje "101101"
echo bindec("1001011");  // ispisuje 75
echo dechex(45)          // ispisuje "20"
echo hexdec("5a7b");     // ispisuje 23163
echo decoct(45);         // ispisuje "55"
echo octdec("777");      // ispisuje 511
```

Osnovne trigonometrijske funkcije

PHP podržava osnovi skup trigonometrijskih funkcija koje su navedene u tabeli 2-4.

Tabela 2-4. Trigonometrijske funkcije koje PHP podržava

Funkcija	Opis
`float sin(float arg)`	Sinus argumenta `arg` zadatog u radijanima
`float cos(float arg)`	Cosinus argumenta `arg` zadatog u radijanima
`float tan(float arg)`	Tangens argumenta `arg` zadatog u radijanima
`float asin(float arg)`	Arkus sinus argumenta `arg` zadatog u radijanima
`float acos(float arg)`	Arkus kosinus argumenta `arg` zadatog u radijanima
`float atan(float arg)`	Arkus tangens argumenta `arg` zadatog u radijanima
`float atan2(float x, float y)`	Arkus tangens x/y gde predznak oba argumenta određuje kvadrant rezultata
`float pi()`	Daje vrednost `3.1415926535898`
`float deg2rad(float arg)`	Preračunava `arg` stepeni u radijane
`float rad2deg(float arg)`	Preračunava `arg` radijana u stepene

Stepenovanje i logaritmi

Matematička biblioteka PHP-a sadrži eksponencijalne i logaritamske funkcije navedene u tabeli 2-5.

Tabela 2-5. Eksponencijalne i logaritamske funkcije

Funkcija	Opis
`float exp(float arg)`	Diže e na stepen `arg`
`float pow(float osnova, number stepen)`	Diže izraz `osnova` na stepen `stepen`
`float sqrt(float arg)`	Daje kvadratni koren od `arg`
`float log(float arg)`	Daje prirodni logaritam od `arg`
`float log10(float arg)`	Daje dekadni logaritam od `arg`

Generisanje nasumičnih vrednosti

PHP ima funkciju `rand()` koja daje vrednosti iz generisanog niza *pseudonasumičnih* brojeva. Dobro poznati algoritmi generišu nizove brojeva koji na prvi pogled izgledaju nasumično izabrani, ali oni to nisu u potpunosti. Funkcija `srand()` obezbeđuje početnu vrednost algoritmu i mora se pozvati pre prve upotrebe funkcije `rand()` u skriptu. Ako to ne učinite, funkcija `rand()` će uvek davati isti niz brojeva kad god pozovete skript u kome se ona nalazi. Šabloni funkcija su sledeći:

```
void srand(integer zrno)
integer rand()
integer rand(integer min, integer max)
```

Pri pozivanju, funkciji `srand()` treba proslediti argument *zrno* tipa integer, čija se vrednost obično generiše na osnovu tekućeg vremena. Kada je pozvana bez argumenata, funkcija `rand()` daje nasumičan broj u opsegu od 0 do vrednosti koju daje funkcija `getrandmax()`. Kada se funkcija `rand()` pozove sa dva argumenta – vrednosti *min* i *max* – rezultat je nasumičan broj između *min* i *max*. Pogledajmo primer:

```
// Generišemo početno zrno
$zrno = (float) microtime() * 100000000;

// Inicijalizujemo generator pseudo nasumičnih brojeva
srand($zrno);

// Generišemo dva nasumična broja
print rand();        // između 0 i getmaxrand()
print rand(1, 6);    // između 1 i 6 (uključivo)
```

Funkcije koje korisnik definiše

Funkcije omogućavaju grupisanje programskih komandi u smislen blok. Kada kôd koji pišete treba da se upotrebi na više mesta, upotrebom funkcija izbegavate dupliranje komandi, a održavanje koda postaje znatno lakše.

U ovom poglavlju smo dosad predstavili mnogobrojne primere pozivanja funkcija. Kada napišete funkciju, pozivate je na isti način kao bilo koju funkciju iz PHP-ovih biblioteka. Sledi primer jednostavne funkcije koju je korisnik definisao (primer 2-6).

Primer 2-6. Namenska funkcija za ispisivanje teksta polucrnim slovima

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.0 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Jednostavan primer pozivanja funkcije</title>
</head>
<body bgcolor="#ffffff">
<?php

function bold($string)
{
  echo "<b>" . $string . "</b>\n";
}

// Prvi primer pozivanja funkcije
// (sa statičkom znakovnom vrednošću)
echo "ovo se ispisuje običnim slovima\n";
bold("ovo se ispisuje polucrnim slovima");
echo "ovo se ponovo ispisuje običnim slovima \n";

// Drugi primer pozivanja funkcije
// (kojoj se kao parametar prosleđuje promenljiva)
$myString = " ovo se ispisuje polucrnim slovima ";
bold($myString);
?>
</body></html>
```

U ovom skriptu definisana je funkcija `bold()`, koja prihvata jedan ulazni parametar, `$string`, i ispisuje ga tako što ga uokviruje između HTML oznaka `<b>` i `</b>`. Funkcija `bold()`, kao što je ovde definisana, može se upotrebiti za izraz znakovnog tipa ili za promenljivu.

Funkcije mogu da daju i povratne vrednosti. Na primer, pogledajte sledeći blok koda u kome je deklarisana i pozvana funkcija `heading()` koja pomoću iskaza `return` vraća rezultat znakovnog tipa:

```
function heading($text, $headingLevel)
{
   switch ($headingLevel)
   case 1:
      $result = "<h1>" . ucwords($text) . "</h1>";
      break;

   case 2:
      $result = "<h2>" . ucwords($text) . "</h2>";
      break;

   case 3:
      $result = "<h3>" . ucfirst($text) . "</h3>";
      break;

   default:
      $result = "<p><b>" . ucfirst($text) . "</b>";
```

```
    return($result);
}

$test = "user defined functions";
echo heading($test, 2);
```

Funkcija `heading()` prihvata dva parametra: tekst zaglavlja i nivo HTML zaglavlja kojim se ispisuje tekst. U zavisnosti od vrednosti argumenta `$headingLevel`, funkcija formira odgovarajući HTML kôd za prikazivanje teksta zaglavlja, pri čemu podešava odgovarajuću veličinu slova. Rezultat izvršavanja prethodnog bloka koda je znakovna vrednost:

```
<h2>User Defined Functions</h2>
```

Povratna vrednost funkcije koja se prosleđuje pomoću naredbe `return` može se napisati i između zagrada: `return($result)` isto je što i `return $result`.

Tipovi podataka argumenata i povratne vrednosti funkcije

Tipovi podataka argumenata i povratne vrednosti funkcije nisu zadati u definiciji funkcije. PHP dopušta da funkciji prosledite argument bilo kog tipa, a kao i za promenljive, tip podataka povratne vrednosti određuje se tek u trenutku kada se prosleđuje rezultat izvršavanja funkcije. Pogledajmo jednostavnu funkciju koja deli jedan broj s drugim:

```
function podeli($a, $b)
{
     return ($a/$b);
}

$c = podeli(4, 2);       // rezultat = 2, tipa integer
$c = podeli(3, 2);       // rezultat = 1.5, tipa float
$c = podeli(4.0, 2.0);   // rezultat = 2.0, tipa float
```

Ako od tipova vrednosti ulaznih argumenata funkcije zavisi tip rezultata, onda bi trebalo da ih ispitujete kao što je opisano u odeljku „Tipovi podataka“, u prethodnom delu ovog poglavlja.

Doseg promenljivih

Promenljive koje se koriste unutar funkcije razlikuju se od onih izvan funkcije. *Doseg* (ili *vidljivost, tj. oblast važenja*, engl. *scope*) prvih promenljivih ograničen je na samu funkciju (postoje izuzeci iz ovog pravila, opisani u nastavku ovog odeljka). Pogledajmo primer koji ilustruje doseg promenljivih:

```
function (doublevalue($var)
{
   $temp = $var * 2;
}

$var = 5;
doublevalue($var);
echo "\$temp je: $temp";
```

Rezultat primera je znakovna vrednost:

```
$temp je:
```

bez bilo kakve vrednosti za `$temp`. Doseg promenljive `$temp` je lokalan za funkciju `$doublevalue` i njena vrednost je nepoznata po izlasku iz funkcije.

PHP-ova mašina za izvršavanje skriptova neće vas upozoriti na to da upotrebljavate nedefinisanu promenljivu. Međutim, upotrebu nedefinisane promenljive možete otkriti pomoću podešavanja za obaveštavanje o greškama, što je opisano u odeljku „Česte greške“, u nastavku poglavlja.

Ako vrednost koja je lokalna u određenoj funkciji želite da upotrebite na nekom drugom mestu u skriptu, najjednostavnije je da je prosledite kao povratnu vrednost funkcije, pomoću naredbe `return`. To se radi na sledeći način:

```
function (doublevalue($var)
{
   $returnVar = $var * 2;
   return($returnVar);
}

$var = 5;
doublevalue($var);
$temp = doublevalue($var);
echo "\$temp je: $temp";
```

Rezultat ovog primera je znakovna vrednost:

```
$temp je: 10
```

Promenljvu `$temp` mogli ste da upotrebite i unutar funkcije `doublevalue()`. Međutim, promenljiva `$temp` unutar funkcije razlikuje se od istoimene promenljive izvan funkcije. Opšte pravilo glasi da su promenljive koje se koriste isključivo unutar funkcije lokalne za funkciju, bez obzira na to da li možda negde izvan funkcije postoji promenljiva s jednakim imenom. Ovo opšte pravilo ima dva izuzetka: promenljive koje se prosleđuju po referenci i one koje su unutar funkcije deklarisane kao `globalne`; u tom slučaju, nisu lokalne za funkciju.

Globalne promenljive

Ako istu promenljivu želite da koristite na bilo kom mestu u kodu, uključujući i mesto unutar funkcija, to će vam omogućiti iskaz `global`. Tim iskazom se promenljiva deklariše unutar funkcije kao da je to u stvari promenljiva sa istim imenom koja već postoji izvan funkcije. Pogledajmo primer:

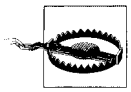
```
function doublevalue()
{
   global $temp;
   $temp = $temp * 2;
}

$temp = 5;
doublevalue();
echo "\$temp je: $temp";
```

Pošto je unutar funkcije promenljiva `$temp` deklarisana kao globalna, ona je dosegljiva i izvan funkcije. Usled toga, skript ispisuje:

```
$temp je: 10
```

Upozorenje: izbegavajte preteranu upotrebu globalnih promenljivih jer se tako dobija teže razumljiv kôd.



Deklaracija globalne promenljive može biti zamka.

U nekim drugim programskim jezicima globalne promenljive se obično deklarišu kao globalne izvan funkcija, a zatim koriste unutar funkcija.

U PHP-u dešava se suprotno: da biste globalnu promenljivu mogli da koristite unutar funkcije, treba da je deklarišete kao globalnu unutar funkcije.

Alternativa upotrebi globalnih promenljivih jeste da funkcija vraća više od jedne promenljive, tako što će vraćati niz vrednosti. Bolje rešenje problema je da se funkciji prosleđuju parametri po referenci umesto po vrednosti. Ovaj drugi pristup opisan je u narednom odeljku.

Kako se promenljive prosleđuju funkcijama

Promenljive funkcije se standardno prosleđuju po vrednosti, a ne po referenci. Naredni primer:

```
function doublevalue($var)
{
   $var = $var * 2;
}

$var = 5;
doublevalue($var);
echo "\$var je: $var";
```

daje rezultat:

```
$var je: 5
```

Vrednost parametra `$var` koji se prosleđuje funkciji `doublevalue()` ne menja se unutar funkcije. Ono što se u stvari odvija jeste da se funkciji prosleđuje vrednost 5, koja množenjem sa dva postaje 10, a onda se taj rezultat zauvek gubi! Funkciji se prosleđuje vrednost parametra, a ne sam parametar.

Prosleđivanje argumenata po referenci

Umesto vraćanja rezultata ili upotrebe globalne promenljive, druga mogućnost je prosleđivanje reference na promenljivu kao argument funkcije. To znači da će se svaka izmena vrednosti promenljive unutar funkcije preneti na izvornu promenljivu. Pogledajmo primer:

```
function doublevalue(&$var)
{
   $var = $var * 2;
}

$var = 5;
doublevalue($var);
echo "\$var je: $var";
```

daje rezultat:

```
$var je: 10
```

Jedina razlika između ovog primera i prethodnog jeste u tome što je parametru $var u deklaraciji funkcije `doublevalue()` dodat prefiks `&`, pa smo dobili `&$var`. Znak `&` (ampersend) znači da se kao parametar prosleđuje referenca na izvornu promenljivu, a ne samo njena vrednost. Rezultat je to da se izmene vrednosti promenljive `$var`, načinjene unutar funkcije, prenose u izvornu promenljivu `$var` izvan funkcije.

Funkcije čiji su argumenti definisani kao reference na promenljive ne mogu se pozivati u literalnim izrazima jer funkcija očekuje promenljivu čiju će vrednost menjati. U takvim slučajevima PHP javlja grešku.

Dodeljivanje vrednosti putem reference

Referenciranje pomoću znaka `&` može se primeniti i pri dodeljivanju vrednosti promenljivama, što omogućava da bloku memorije koji sadrži određenu vrednost pristupa više promenljivih. Naredni primer ilustruje princip:

```
$x = 10;
$y = &$x;
$y++;
echo $x;
echo $y;
```

Rezultati su sledeći:

```
11
11
```

Pošto je `$y` referenca na `$x`, svaka izmena nad `$y` prenosi se i na `$x` jer u suštini, u pitanju je ista promenljiva. Dakle, dodavanjem 1 promenljivoj `$y`, za isto toliko povećava se i promenljiva `$x` i obe postaju jednake 11.

Referenca `$y` može se ukloniti pomoću naredbe:

```
unset($y);
```

koja ne utiče na promenljivu `$x`, niti na njenu vrednost.

Podrazumevane vrednosti argumenata

PHP omogućava da se definišu podrazumevane vrednosti argumenata funkcija. Podrazumevana vrednost argumenta zadaje se pomoću znaka jednakosti. Pogledajmo izmenjenu varijantu funkcije `heading()`, opisane u prethodnom delu ovog poglavlja:

```
function heading($text, $headingLevel = 2)
{
   switch ($headingLevel)
   case 1:
      $result = "<h1>" . ucwords($text) . "</h1>";
      break;

   case 2:
      $result = "<h2>" . ucwords($text) . "</h2>";
      break;

   case 3:
      $result = "<h3>" . ucfirst($text) . "</h3>";
      break;

   default:
      $result = "<p><b>" . ucfirst($text) . "</b>";

   return($result);
}
$test = "user defined functions";
echo heading ($test);
```

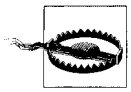
Kada se poziva funkcija `heading()`, može se izostaviti drugi argument; u tom slučaju će promenljivoj `$headingLevel` biti dodeljena vrednost 2.

Višekratna upotreba istih funkcija pomoću datoteka za umetanje

Da bi iste funkcije mogle da se koriste u više skriptova, PHP podržava naredbu `include` i direktivu `require`.

Ako se opredelite da funkciju `bold()` iz primera 2-6 koristite u više skriptova, njen kôd možete smestiti u datoteku koju ćete umetnuti u skript pomoću naredbe `include`. Na primer, možete da formirate datoteku *funkcije.inc* u koju ćete uneti kôd funkcije bold:

```
<?php
function bold($string)
{
   echo "<b>" . $string . "</b>\n";
}
?>
```



Svaki blok PHP koda u datoteci čiji sadržaj umećete u skript pomoću naredbe `include` mora da bude uokviren između početne i završne oznake za PHP skript. Ako to ne uradite, PHP-ova mašina za izvršavanje skriptova obrađivaće takav PHP kôd kao HTML kôd.

Zatim u skriptu upotrebite iskaz `include` da biste omogućili pristup funkciji `bold()`:

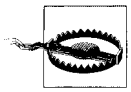
```
<html>
<head>
  <title>Jednostavan primer pozivanja funkcije</title>
</head>
```

```
<body bgcolor="#ffffff">
<?
include "funkcije.inc";

// Prvi primer pozivanja funkcije
// (sa statičkom znakovnom vrednošću)
echo "ovo se ispisuje običnim slovima\n";
bold("ovo se ispisuje polucrnim slovima");
echo "ovo se ponovo ispisuje običnim slovima \n";

// Drugi primer pozivanja funkcije
// (kojoj se kao parametar prosleđuje promenljiva)
$myString = " ovo se ispisuje polucrnim slovima ";
bold($myString);
?>
</body></html>
```

Ovaj skript radi isto kao i prethodna verzija, ali se funkcija `bold()` od sada može koristiti i u drugim skriptovima tako što se pomoću iskaza `include` umetne sadržaj datoteke *funkcije.inc*.



Budite oprezni pri upotrebi iskaza `include`. Ako umetnete istu datoteku dva puta, ili u skriptu deklarišete funkciju koja već postoji u datoteci čiji ste sadržaj umetnuli u skript, PHP će prijaviti grešku – ponovljena definicija funkcije.

Iskaz `include` tretira se na isti način kao svaki drugi iskaz. Na primer, sadržaje različitih datoteka možete uslovno umetati pomoću bloka koda nalik na sledeći:

```
If ($netscape == true)
{
   include "netscape.inc";
}
else
   include "ostali.inc";
}
```

Sadržaj datoteke se umeće u skript samo kada se iskaz `include` izvršava unutar skripta. U ovom primeru, vitičaste zagrade su obavezne: kada biste ih izostavili, primer ne bi radio u skladu sa očekivanjima.

Ukoliko određena datoteka treba uvek da bude umetnuta u skript, umesto iskaza `include` treba upotrebiti direktivu `require`. Ta direktiva se obrađuje pre skripta, a sadržaj datoteke zadate tom direktivom umeće se u skript. To je korisno kada isti HTML kôd treba da se koristi u više HTML dokumenata. Na primer, ako želite da prikažete isto zaglavlje ili podnožje na svakoj stranici Web lokacije – bez obzira na greške ili druge probleme – direktiva `require` to pojednostavljuje i olakšava održavanje koda.

Pogledajte naredni blok HTML koda:

```
<hr><br>(c) 2001 Hugh E. Williams and David Lane
```

Ako želite da se ovaj blok koda prikazuje u dnu svake stranice, možete ga uneti u datoteku *footer.inc*, a zatim pomoću direktive `require` uključiti u svaki skript koji napišete:

```
require "footer.inc";
```

Prednost ovog pristupa je u tome što ukoliko poželite da izmenite sadržaj HTML podnožja, to ćete morati da uradite na samo jednom mestu.

Objekti

PHP pruža ograničenu podršku za objektno orijentisano programiranje tako što omogućava programerima da definišu sopstvene klase i da zatim prave objektne instance tih klasa. U ovoj knjizi malo koristimo objekte, a ovaj odeljak je uvod u mogućnosti koje PHP pruža za objektno orijentisano programiranje. Tema objektno orijentisanog programiranja je veoma obimna i ne možemo je u potpunosti obraditi u ovoj knjizi.

Klase i objekti

Klasa je definicija složene strukture podataka, koja se sastoji od pripadajućih promenljivih i skupa funkcija – poznatih kao metode klase – koje deluju na promenljive klase. Primer 2-7 prikazuje kako je u PHP-u definisana klasa *Counter* (brojač). Ta klasa ima dve pripadajuće promenljive – `$count` i `$startPoint`, obe tipa integer – i četiri funkcije koje rade s tim promenljivama. Promenljive i funkcije se nazivaju zajedničkim imenom *članovi* (engl. *members*) klase *Counter*.

Primer 2-7. Definicija jednostavne klase Counter

```
<?php
  // Klasa koja definiše brojač.
  class Counter
  {
    // Pripadajuće promenljive
    var $count = 0;
    var $startPoint = 0;

    // Metode
    function startCountAt($i)
    {
      $this->count = $i;
      $this->startPoint = $i;
    }

    function increment()
    {
      $this->count++;
    }

    function reset()
    {
      $this->count = $this->startPoint;
    }
```

Primer 2-7. Definicija jednostavne klase Counter (nastavak)

```
    function showvalue()
    {
      print $this->count;
    }
  }
?>
```

Da bi mogle da se koriste strukture podataka i funkcije definisane u klasi, neophodno je da se formira instanca (primerak) klase. Kao i vrednosti drugih tipova – integer, string, array itd. – objekti su sadržani u promenljivama. Međutim, za razliku od promenljivih, objekti se formiraju pomoću operatora `new`. Objekat klase *Counter* može se napraviti i dodeliti promenljivoj na sledeći način:

```
$aCounter = new Counter;
```

Čim formirate promenljivu `$aCounter`, možete koristiti pripadajuće promenljive i funkcije novog objekta. Članovima objekta, promenljivama i funkcijama, pristupa se pomoću operatora `->`. Pogledajte naredni primer:

```
echo $aCounter->count;    // ispisuje 0
$aCounter->increment();
echo $aCounter->count;    // ispisuje 1

// Zaobilaženje funkcije koja ažurira vrednost
// pripadajuće promenljive count
$aCounter->count = 101;
```

U definiciji klase, u kodu koji definiše funkcije klase promenljivama klase može se pristupati pomoću promenljive `$this`, kao u klasi *Counter*, u primeru 2-7. Promenljiva `$this` ima specijalno značenje i služi kao zamena dok se ne formira objekat klase. Na primer, kada pozovete funkciju `$aCounter->increment()`, promenljiva `$this` igra ulogu objekta `$aCounter`.

Ako kôd prikazan u primeru 2-7 unesete u datoteku *counter.inc*, klasu *Counter* možete koristiti i u drugim skriptovima da biste pravili nove objekte, kao što je prikazano u primeru 2-8.

Primer 2-8. Pravljenje i upotreba objekata klase Counter

```
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD HTML 4.0 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>Counter</title></head>
<body>
<?php
  include "counter.inc";

  // Formiramo nov objekat tipa "counter"
  $temp = new Counter;

  // Promenljivoj counter zadajemo vrednost 10
  $temp->startCountAt(10);
```

Primer 2-8. Pravljenje i upotreba objekata klase Counter (nastavak)

```
  // Povećavamo vrednost promenljive counter
  $temp->increment();
  $temp->increment();
  $temp->increment();

  // Ispisujemo vrednost promenljive counter
  echo "<p>Counter is now: ";
  $temp->showvalue();

  // Promenljivu counter vraćamo na početnu vrednost
  $temp->reset();

  // Ispisujemo vrednost promenljive counter
  echo "<p>Counter is now: ";
  $temp->showvalue();
?>
</body></html>
```

Možete napraviti neograničen broj objekata date klase. Na primer, pomoću koda nalik sledećem možete napraviti tri objekta koje ćete dodeliti trima promenljivama:

```
$a = new Counter;
$b = new Counter;
$c = new Counter;
```

Promenljive `$a->count`, `$b->count` `$c->count` međusobno se razlikuju. Svaka je tipa object i referencira objekat klase *Counter*, ali ti objekti ne zavise jedan od drugog.

Nasleđivanje

Jedan od moćnih koncepata objektno orijentisanog programiranja je nasleđivanje. *Nasleđivanje* omogućava da se definiše nova klasa koja proširuje mogućnosti postojeće osnovne klase. Pomoću rezervisane reči `extends`, PHP omogućava da se definiše nova klasa koja proširuje postojeću klasu. Primer 2-9 pokazuje kako se klasa *Counter* proširuje da bi se od nje dobila nova klasa *BottleCounter*, koja izračunava broj gajbi vina koje treba isporučiti.

Primer 2-9. Nova klasa BottleCounter definisana proširivanjem osnovne klase Counter

```
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD HTML 4.0 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>Bottle Counter</title></head>
<body>
<?php
  include "counter.inc";

  class BottleCounter extends Counter
  {
    // Dodajemo 12 boca ukupnom broju
    function addCase()
    {
      $this->count += 12;
    }
```

Primer 2-9. Nova klasa BottleCounter definisana proširivanjem osnovne klase Counter (nastavak)

```
    // Vraća ukupan broj gajbi koje treba isporučiti
    function caseCount()
    {
      return ceil($this->count / 12);
    }

    // Konstruktor klase koji zadaje početnu vrednost
    // broja gajbi za isporuku
    function BottleCounter($startCount)
    {
      $this->count = $startCount;
    }
  }

  // Formiramo nov objekat tipa "BottleCounter"
  // kome prosleđujemo inicijalnu vrednost 12
  $temp = new BottleCounter(12);

  // Povećanje ukupnog broja boca
  $temp->increment();

  // Dodajemo još jednu gajbu
  $temp->addCase();

  // Ispisujemo ukupan broj boca: 24
  echo "<p>Counter is now: ";
  $temp->showvalue();

  // Ispisujemo ukupan broj gajbi
  $cases = $temp->caseCount();
  echo "<p>The number of cases to ship: $cases";
?>
</body></html>
```

Nova klasa *BottleCounter* ne dodaje nijednu novu pripadajuću promenljivu, ali dodaje tri nove funkcije. Funkcije klase *BottleCounter* koriste promenljive osnovne klase *Counter* na način koji odgovara klasi *BottleCounter*. Funkcija `addCase()` povećava vrednost promenljive `$count` za 12, a funkcija `caseCount()` daje ukupan broj gajbi za isporuku, uključujući i nepotpune gajbe.

Poslednja funkcija, `BottleCounter()`, konstruktor je klase *BottleCounter*. Funkcije klase koje imaju jednako ime kao sama klasa tretiraju se drugačije od ostalih funkcija. PHP koristi te funkcije kao konstruktore koji se pozivaju svaki put kada se formira nov objekat određene klase. Konstruktorska funkcija može da ima argumente pomoću kojih se zadaju početne vrednosti promenljivama klase kada se pravi nov objekat. Primer 2-9 pokazuje kako se pravi nov objekat klase *BottleCounter*:

```
// Formiramo nov objekat tipa "BottleCounter"
// kome prosleđujemo inicijalnu vrednost 12
$temp = new BottleCounter(12);
```

Snaga nasleđivanja nije u činjenici da omogućava samo višekratnu upotrebu istog koda. Objekti napravljeni od proširene klase mogu se koristiti i kao da su napravljeni od osnovne klase. Mogućnost upotrebe objekta kao da je instanca osnovne klase poznata je pod imenom *polimorfizam*. Klasu *Counter* možete upotrebiti kao osnovu za druge

nove klase, kao što bi bila *CanCounter*, u kojoj gajba sadrži 24 konzerve pića, a ne 12 boca. U kodu koji radi sa objektom klase *Counter* potom se mogu koristiti objekti tipa *BottleCounter* ili *CanCounter*. Pogledajte naredni primer, u kome je definisana funkcija `volumeDiscount()` (količinski popust), koja vraća koeficijent popusta u zavisnosti od sadržaja objekta *Counter*:

```
// Daje koeficijent popusta koji zavisi od
// vrednosti promenljive $var koja
// predstavlja objekat klase Counter
function volumeDiscount($var)
{
   // $var se koristi kao Counter
   if ($var->count > 24)
     return 0.95;
   else
     return 1.0;
}

$bottles = new BottleCounter(10);
$cans = new CanCounter(24);
$bottleDiscountFactor = volumeDiscount($bottles);
$canDiscountFactor = volumeDiscount($can);
```

Ako su klase *BottleCounter* i *CanCounter* definisane kao proširenja klase *Counter*, onda se funkcija `volumeDiscount()` može pozivati za objekte sve tri klase.

Česte greške

Dok početnici uče PHP, prave nekoliko veoma čestih grešaka, koje se omaknu čak i iskusnijim programerima. U ovom kratkom odeljku opisujemo nekoliko tih grešaka i dajemo osnovna uputstva kako da ih ispravite.

Stranica se u čitaču Weba prikazuje delimično ili se čak ništa ne prikazuje

Jedan od najuobičajenijih problema čiji uzrok treba otkriti prilikom otklanjanja grešaka iz PHP skripta jeste sledeći:

- Čitač ne prikazuje ništa
- Pojavljuje se iskačući okvir za dijalog s porukom „Document Contains No Data“
- Stranica se samo delimično prikazuje u čitaču.

Uzrok većine ovih problema nije greška u logici ugrađenoj u kôd skripta, već u pogrešnom HTML kodu koji je skript generisao. Na primer, ako su izostavljene završne oznake `</table>`, `</form>` ili `</frame>`, može se dogoditi da se stranica uopšte i ne prikaže u čitaču.

Ako je uzrok problema HTML kôd, to obično možete otkriti ako pregledate izvorni HTML kôd koji skript generiše. U Netscapeu, ceo HTML kôd tekuće stranice može se videti ako se izabere prikaz page-source.

U slučaju višestrukih ili teško uočljivih grešaka u HTML kodu, može vam pomoći sistem za ispitivanje ispravnosti HTML koda, koji organizacija W3C stavlja na raspolaganje na adresi `http://validator.w3.org`. Sistem učitava stranicu, ispituje ispravnost HTML koda i prikazuje izveštaj. To je odlična alatka za otkrivanje grešaka i proveravanje usklađenosti pre isporuke aplikacije kupcu.

Ako je problem i dalje nerešiv, pokušajte da iza iskaza `echo`, `print` ili `printf` dodate pozive funkcije `flush()`. Funkcija `flush()` prazni izlazni bafer koji održava PHP-ova mašina, a tekući sadržaj bafera šalje Web serveru. Ova funkcija ne deluje na bafer Web servera, niti čitača Weba, ali obezbeđuje da svi podaci koje je skript baferisao budu na raspolaganju Web serveru, koji ih pak, prosleđuje čitaču radi prikazivanja. Nemojte zaboraviti da pošto otklonite greške, uklonite i pozive funkciji `flush()` jer nepotrebno pražnjenje bafera može da spreči efikasno baferisanje izlaznih podataka koje daje PHP-ova mašina za izvršavanje skriptova.

Čest problem, koji ne treba brkati s prethodnim, jeste kada izostaje bilo kakav odziv Web servera i pojavljuje se poruka o grešci `"no response"`. Ovaj problem je simptom grešaka opisanih u narednom odeljku, a može se odvojiti od problema opisanih u ovom odeljku ako posmatrate ponašanje čitača Weba. Većina popularnih grafičkih čitača Weba pokazuje da čeka odgovor, tako što prikazuje animiran logotip u gornjem desnom uglu. Ako se pojave HTML problemi opisani u ovom odeljku, postupak učitavanja stranice biće završen, animacija logotipa će se prekinuti, a izvorni HTML kôd stranice možete pogledati pomoću neke od opcija menija čitača Weba.

Problemi u vezi s promenljivama

U ovom odeljku razmatraćemo probleme zbog kojih se može dogoditi da se stranica uopšte ne prosleđuje čitaču Weba ili da se cele stranice pojavljuju bez vrednosti promenljivih.

Imena promenljivih

Greška čiji je uzrok ime promenljive može ponekad dovesti do beskonačne petlje. Rezultat beskonačne petlje je taj da na strani čitača Weba istekne vreme čekanja na odgovor. U tom slučaju čitač Weba upozorava korisnika da se Web server ne odaziva na poslati HTTP zahtev.

Naredna petlja se nikada ne završava i ne šalju se nikakvi izlazni podaci:

```
for($brojac=0; $brojac<10; $Brojac++)
   nekaFunkcija();
```

Vrednost promenljive `$brojac` se ne povećava, veće se to događa za drugu promenljivu, `$Brojac`, a `$brojac` ostaje uvek na vrednosti 10. Ovo je česta posledica neznatnog menjanja imena promenljivih izmenom vrste slova, dodavanjem ili uklanjanjem znaka za podvučeno ili usled običnih grešaka pri kucanju.

Beskonačne petlje mogu da budu i uzrok neočekivanih rezultata. Naredna petlja može da za veoma kratko vreme ispiše hiljade pozdravnih poruka u prozoru čitača Weba:

```
for($brojac=0; $Brojac<10; $brojac++)
   echo "<br>zdravo";
```

Ova vrsta grešaka ponekad se može otkriti ako se PHP-ov nivo obaveštavanja o greškama podesi na veću osetljivost. Ako naredni blok koda dodate na početak svakog PHP skripta (ili u datoteku koju priključujete skriptu pomoću naredbe `include`), bićete obavešteni u slučaju upotrebe nedefinisane promenljive:

```
error_reporting(E_ALL);
```

Posle ove naredbe, svaka promenljiva mora biti deklarisana zadavanjem vrednosti da bi mogla da se upotrebljava. Pogledajmo sledeći primer:

```
error_reporting(E_ALL);
for($brojac=0; $Brojac<10; $brojac++)
   echo "<br>zdravo";
```

Rezultat će biti beskrajan niz poruka o greškama nalik na sledeću:

```
Warning: Undefined variable: Brojac in /var/lib/apache/htdocs/winestore/a.
php on line 2
```

Skript će se izvršavati jer je u pitanju samo upozorenje. Upotrebom funkcije `set_error_handler()` možete ugraditi namenski kôd za obradu grešaka koji će prekinuti izvršavanje skripta u slučaju upozorenja ili greške. Obrada grešaka je opisana u poglavlju 10.

Promenljive bez sadržaja

Promenljiva koja nije inicijalizovana ne sadrži ništa. Uzrok problema je često očigledan, ali se ponekad teško otkriva ukoliko je u pitanju mala greška u kucanju. Pogledajte primer u kome je promenjena veličina slova u imenu promenljive:

```
$testprom = "zdravo";
echo "Vrednost promenljive je $testProm";
```

Rezultat je sledeći tekst:

```
Vrednost promenljive je
```

Problem može da bude znatno teže uočljiv vizuelnim isptivanjem koda ukoliko je promenljiva deo složene operacije, npr. kada se koristi kao indeks elemenata niza ili se njena vrednost ugrađuje iza oznake `<table>`, ili se promenljiva koristi kao parametar upita koji se šalje bazi podataka.

Ukoliko rezultujuća stranica nije u potpunosti ono što očekujete, uzrok može da bude promenljiva koju niste inicijalizovali. Tu vrstu greške najjednostavnije ćete otkriti ako na početak skripta postavite naredbu `error_reporting(E_ALL)`, kao što je opisano u prethodnom odeljku.

Pomoću funkcije `isset()` može se takođe upravljati izvršavanjem koda i otkrivati greške u njemu jer ona daje rezultat `true` ako promenljiva postoji (čak i kad joj je dodeljena vrednost `NULL`), odnosno `false` ukoliko promenljiva nije inicijalizovana.

Još jedan srodan problem se vidi po tome što se u rezultujućoj stranici pojavljuju imena promenljivih umesto odgovarajućih vrednosti. Uzrok je najčešće jednostavan i lako se otklanja: izostavljen je znak za dolar na početku imena promenljive. Na primer:

```
echo "Vrednost promenljive test je test";
```

a trebalo bi da bude:

```
echo "Vrednost promenljive test je $test";
```

Ako je znak za dolar izostavljen u naredbi za dodeljivanje vrednosti ili za ispitivanje uslova, PHP-ov interpretator će javiti sintaksnu grešku sa podrazumevanim nivoom obaveštavanja o greškama.

Sličan problem može se pojaviti i kada umesto dvostrukih navodnika upotrebite jednostruke jer se vrednosti napisane između jednostrukih navodnika uvek prosleđuju direktno, bez analiziranja i tumačenja sadržaja kao u slučaju dvostrukih navodnika. Na primer:

```
echo 'Vrednost promenljive test je $test';
```

Rezultat je tekst:

```
Vrednost promenljive test je $test
```

Vrednost promenljive `$test` se ne prikazuje.

Greške u HTTP zaglavljima

U ovom poglavlju nismo pominjali funkcije `header()` i `setcookie()`. Obe služe za formiranje HTTP zaglavlja koje Web server šalje čitaču Weba i često se koriste u dinamičkim Web aplikacijama. Opise ovih funkcija i načine upotrebe naći ćete u poglavljima 5, 6, 8 i 9.

Čest problem pri formiranju HTTP zaglavlja u PHP kodu jeste poruka o grešci koja počinje sa:

```
Warning: Cannot add header information - headers already sent ...
```

Zaglavlja se mogu slati samo pre slanja bilo kakvog HTML koda, a to važi čak i za beline na početku datoteke. Na primer, ako ste pre oznake kojom započinje skript `<?php` poslali prazan red ili samo znak za razmak, to je već HTML kôd – mada ne baš naročito koristan – i pozivanjem funkcije `header()` ili `setcookie()` dobijate poruku o grešci.

Problemi u vezi sa slanjem zaglavlja mogu se izbeći ako pomoću funkcija za upravljanje slanjem rezultata skripta drugačije podesite način na koji PHP baferiše podatke. Opis tih funkcija izlazi izvan okvira ove knjige.

Drugi česti problemi

Tri kategorije problema koje smo dosad opisali su najčešće greške koje prave programeri u PHP-u. Navodimo i nekoliko manje uobičajenih i manje specifičnih za PHP.

Izostavljen znak tačka zarez na kraju iskaza obično se lako otkriva. PHP-ov interpretator nastavlja da analizira skript i kada pređe nivo „zbunjenosti“ koji može da prihvati, ili kada premaši maksimalnu dužinu iskaza, obaveštava vas da je naišao na grešku jedan ili više redova iza stvarnog mesta na kom nedostaje tačka zarez. U većini slučajeva, takva greška se lako ispravlja jer se u poruci pojavljuje i broj reda u kome je greška.

U nekim slučajevima nedostajuća tačka zarez se jednako teško otkriva kao i nedostajuća završna vitičasta zagrada ili navodnik. U narednom kodu greška je u tome što nedostaje završna vitičasta zagrada:

```
<?
for($x=0; $x<100 ;$x++)
{
   for($y=0; $y<100; $y++) {
     echo "test1";
     for($z=0; $z<100; $z++)
       echo "test2";
}
?>
```

PHP javlja sledeću grešku:

```
Parse error: parse error in bug.php on line 9
```

Budući da je deveti red poslednji red skripta, priroda i uzrok greške nisu odmah jasni. Međutim, uzrok grešaka pri analiziranju sintakse koje nisu odmah uočljive u navedenom redu koda obično se nalazi u prethodnom redu, ili nedostaje vitičasta zagrada ili navodnik.

U ovom primeru je za otkrivanje nedostajuće vitičaste zagrade dovoljan minut ili malo više, ali ako su funkcije složenije, može biti neophodno znatno više vremena da se otkrije greška. To samo ističe važnost uvlačenja delova koda i štetnost loše navike pisanja početne vitičaste zagrade na kraju reda. Trebalo bi da se vitičaste zagrade uvek pišu u zasebnim redovima.

Nedostajuća početna ili završna oznaka za skript može da bude uzrok sličnih problema, ali se oni znatno lakše otklanjaju. Kada nedostaje početna oznaka za skript, to je sasvim očigledno jer se deo stranice – ili čak cela stranica – ne prikazuje u čitaču Weba. Nedostajuća završna oznaka obično je uzrok greške u sintaksi (parse error) zato što se PHP-ova mašina za izvršavanje skriptova zbuni kada pokuša da raščlani HTML kôd i tumači ga kao PHP kôd.

Ako se izvorni kôd skripta samo prikazuje u čitaču Weba a ne izvršava se na serveru, onda je verovatno Apache pogrešno konfigurisan. Razlog je verovatno u tome što je tokom postupka instaliranja Apachea direktiva `AddType` za obradu skriptova ostala u obliku komentara; izgleda da je to podrazumevana opcija u novijim distribucijama RedHat Linuxa.

Drugi mogući uzrok prikazivanja izvornog koda skripta umesto njegovog izvršavanja jeste da PHP skriptovi nisu snimljeni u datoteke sa nastavkom *.php*. Ovaj problem se često pojavljuje kada se radi sa starijim PHP3 kodom, budući da su datoteke PHP3 skriptova imale nastavak *.php3*. Problem se može otkloniti tako što će se preimenovati datoteke skriptova tako da imaju nastavak *.php*, ili dodati još jedna direktiva `AddType` u Apacheovu datoteku *httpd.conf*:

```
AddType application/x-httpd-php .php3
```

U retkim slučajevima, PHP3 skriptove treba neznatno izmeniti da bi pod PHP4 mogli da rade.