

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika - praktična matematika (VSŠ)

Jasna Kobal

Regularni izrazi in jezik PHP

Diplomska naloga

Ljubljana, 2009

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

Zahvala

Zahvalila bi se vsem, ki so kakorkoli pripomogli pri nastanku moje diplomske naloge. Predvsem bi se zahvalila za vse popravke, pomoč in nasvete mentorju mag. Matiji Lokarju ter somentorju Gregorju Jeršetu. Prav tako bi se zahvalila družini, ki mi je stala ob strani tako v času pisanja diplomske naloge, kot tudi skozi cel študij. Posebej bi se zahvalila še Mitju, Andreji in Bojanu ter VSEM ostalim, ki so me spodbujali in mi ta čas stali ob strani.

Diplomsko nalogo posvečam svojim staršem.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

Program dela

V diplomski nalogi prikažite, kaj so to regularni izrazi in kako jih uporabljamo pri programiranju v programskem jeziku PHP.

Mentor:

mag. Matija Lokar

Somentor:

Gregor Jerše

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

Povzetek

V diplomski nalogi bo govora o regularnih izrazih v povezavi z jezikom PHP. Kombinacijo regularni izrazi/PHP sem izbrala zato, ker ima jezik PHP dobro podporo za delo z regularnimi izrazi.

Diplomska naloga je zasnovana za bralca, študenta Praktične matematike, ki pozna osnove jezika java oziroma za bralca, ki že pozna osnove nekega programskega jezika. Pogledali si bomo osnovne funkcije za delo z nizi in datotekami. Večino jezika PHP bomo spoznali pri konkretnih primerih. Bralec bo v diplomski nalogi izvedel kaj regularni izrazi sploh so, pa tudi nekaj o uporabi regularnih izrazov v PHP-ju. Bralec bo spoznal tudi najzmogljivejše funkcije za delo z regularnimi izrazi, ki jih vsebuje jezik PHP.

Math.Subj.Class(2000): 68N15, 68N15, 94A15

Computer review Class.system(1998): D.3.0, D.3.1, D.3.3, E.4

Ključne besede: PHP, PCRE, regularni izrazi, vzorec, ujemanje, niz

Key words: PHP, PCRE, regular expressions, pattern, matching, string

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Kazalo

1 PHP	6
1.1 UVOD	6
1.2 UPORABA	6
1.3 ZGODOVINA	10
1.4 HITRI PREGLED PHP-JA	11
1.4.1 Spremenljivke in podatkovni tipi	11
1.4.2 Funkcije za delo z nizi	17
1.4.3 Delo z datotekami	27
2 REGULARNI IZRAZI	31
2.1 UVOD	31
2.2 ZNAKOVNI RAZRED IN SKUPINE ZNAKOV	32
2.2.1 Znakovni razred	33
2.2.2 Skupine znakov	34
2.3 NADOMESTNI ZNAKI	35
2.4 DOLOČEVALCI KOLIČINE	36
3 PODPORA REGULARNIM IZRAZOM V PHP	37
3.1 PREG_GREP	37
3.2 PREG_LAST_ERROR	38
3.3 PREG_MATCH	39
3.4 PREG_MATCH_ALL	41
3.5 PREG_QUOTE	44
3.6 PREG_REPLACE	44
3.7 PREG_REPLACE_CALLBACK	46
3.8 PREG_SPLIT	47
4 PRIMERI	49
4.1 ZAMENJAVA DELA BESEDILA Z DRUGIM	50
4.2 ODSTRANJEVANJE INTERNETNEGA POŠTNEGA NASLOVA	50
4.3 PREVERJANJE VELJAVNOSTI INTERNETNEGA POŠTNEGA NASLOVA	51
4.4 IZRAČUN POVPREČNE STAROSTI ZAPOSLENIH	55
4.5 ODSTRANJEVANJE KOMENTARJEV PHP	57
4.6 ODSTRANJEVANJE KOMENTARJEV HTML	ERROR! BOOKMARK NOT DEFINED.
4.7 ODSTRANJEVANJE PODVOJENIH PRESLEDKOV	58
4.8 PREVERJANJE VELJAVNOSTI VNOSOV V OBRAZCE	58
4.8.1 Preverjanje polja ime in priimek	59
4.8.2 Preverjanje polja poštna številka	60
4.8.3 Preverjanje polja kraj pošte	61
4.8.4 Preverjanje polja naslov	61
4.8.5 Preverjanje polja telefonska številka	62
4.8.6 Preverjanje polja rojstni datum	63
4.8.7 Preverjanje polja internetni poštni naslov	65
4.8.8 Preverjanje polja uporabniško ime	65
4.8.9 Preverjanje polja geslo	66
5 ZAKLJUČEK	67

1 PHP

1.1 Uvod

Kratica PHP je svojo pot začela kot akronim za Personal Home Page Tools, tekom časa pa je spremenila svoj pomen v PHP Hypertext Preprocessor. PHP je sedaj rekurzivni akronim, saj se kratica PHP uporablja tudi v svoji definiciji.

Je skriptni programski jezik, torej jezik, ki se interpretira in vsebuje številne funkcije, ki znajo izvesti kompleksne naloge. Prerasel je svoj namen – pisanje domačih spletnih strani in se sedaj uporablja predvsem za razvoj spletnih aplikacij. Zelo je razširjen, saj se lahko izvaja na večini operacijskih sistemov in strežnikov ter omogoča delo z relacijskimi sistemi zbirk podatkov (RDBS). Zato tudi ni presenetljivo, da je v jeziku PHP napisanih več kot 20 milijonov spletnih strani, ki tečejo na približno milijon strežnikov.¹

Ko uporabnik spletnemu strežniku pošlje zahtevek za spletno stran, ki vsebuje tudi kodo PHP, strežnik kodo PHP izlušči iz datoteke HTML ter jo pošlje pogonu PHP. Pogon to kodo obdela in jo pošlje strežniku nazaj. Strežnik vrne kot rezultat kodo HTML, ki jo lahko vidi uporabnik. Spodaj vidimo, kako poteka komunikacija med uporabnikom in strežnikom.



Slika 1: Komunikacija uporabnik - strežnik

1.2 Uporaba

Veliko dinamičnih spletnih strani je napisanih v jeziku PHP. Osnovna značilnost dinamične spletne strani je, da se njena vsebina in/ali zgled spreminja v odvisnosti od podatkov, uporabnikovih vnosov in akcij. Dinamična spletna stran je pogosto povezana z bazo podatkov. Primeri dinamičnih strani so portali, spletni časopisi, forumi, spletne strani z anketami ..., skratka vse spletne strani, kjer se vsebina

¹ Vir: <http://en.wikipedia.org/wiki/PHP>

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

lahko prilagaja posameznemu uporabniku glede na zahtevo uporabnika in/ali zahtevajo njegovo akcijo. Poleg dinamičnih spletnih strani obstajajo tudi statične spletne strani, katerih vsebina je vedno enako prikazana.

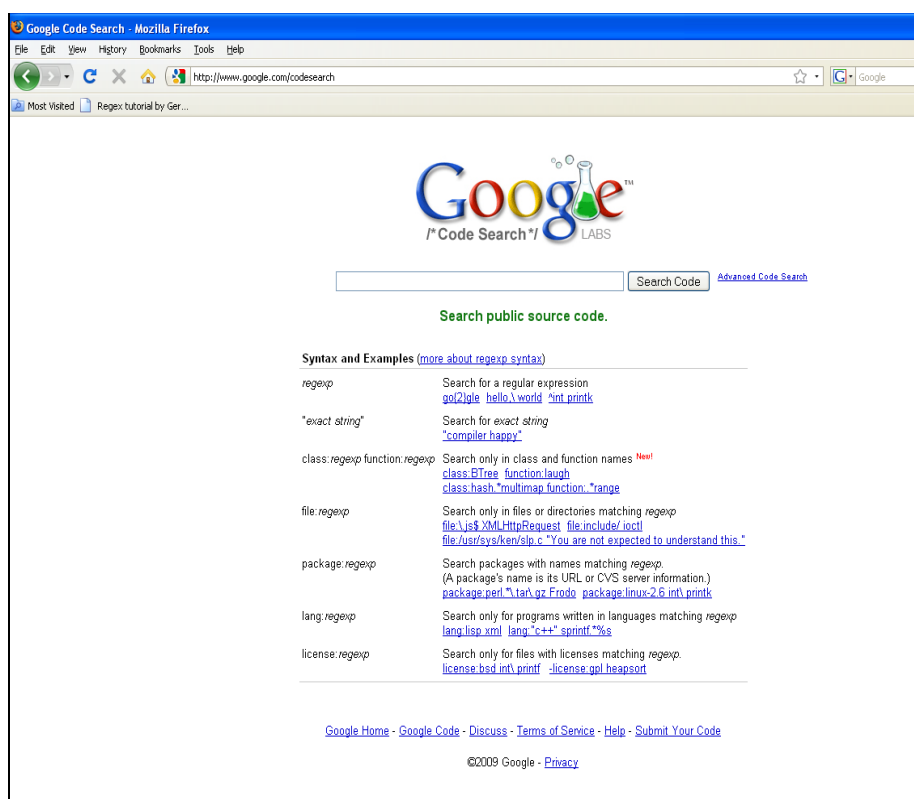
Dinamična spletna stran, kjer uporabljamo jezik PHP, je običajno sestavljena iz dveh delov, ki se med seboj prepletata in dopolnjujeta. Prvi del je napisan v jeziku HTML, drugi del pa zajema kodo PHP. Ločuje ju oznaka za začetek oz. konec kode PHP. Začetek kode PHP ponavadi označujemo z značko '`<?php`', konec pa z oznako '`?>`'. Vse ostalo, napisano izven teh oznak, je običajno koda HTML.

V diplomski nalogi nas jezik PHP zanima predvsem zato, ker vsebuje vrsto metod, ki omogočajo enostavno delo z regularnimi izrazi. Zaradi močne podpore pri delu z regularnimi izrazi se na spletnih straneh, kjer je potrebna uporaba regularnih izrazov, pogosto uporablja jezik PHP.

Kaj regularni izrazi so, si bomo natančneje ogledali v nadaljevanju (razdelek 2.1). Zaenkrat povejmo le, da so regularni izrazi nekakšni vzorci, s katerimi lahko opišemo, kako je videti neko besedilo, število ...

Razlog, da so regularni izrazi tako pogosti na spletnih straneh, je v tem, da funkcionalnost spletne strani lahko precej izboljšamo, kadar uporabimo regularne izraze. Ti nam namreč omogočajo hitrejšo in bolj učinkovito delo z nizi: iskanje, zamenjavo, preverjanje, ali je niz sestavljen na določen način, delitev nizov na podnize in podobno. Med drugim imajo zelo pomembno vlogo pri varnem pisanju kode, saj lahko s preverjanjem vhodnih podatkov preprečimo marsikateri napad hekerjev.

Najpogostejše uporabljene funkcije pri delu z regularnimi izrazi sta iskanje in zamenjava. Funkcija iskanja je sicer že vgrajena v internetni brskalnik, ki pa omogoča le osnovno iskanje niza na prikazani strani. Številne spletne strani želijo uporabniku ponuditi tudi možnost bolj naprednega iskanja, ki med drugim lahko išče gesla tudi na podstraneh, išče "podobne" nize in drugo.



Slika 2: Google Code Search

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Primer take strani je Google Code Search. Poleg običajnega iskalnika ima ta stran možnost iskanja s pomočjo regularnih izrazov.

Oglejmo si primer preproste spletne strani, napisane v jeziku PHP. S pomočjo te spletne strani smo želeli izvesti določeno anketo. Ko uporabnik klikne na gumb 'Potrdi', želimo, da se uporabniku prikažejo tekoči rezultati. V ta namen smo pripravili datoteko anketa_DELA.php. Ko jo zahtevamo v brskalniku, vidimo:

Kje ste bili letos na morju?

☐ Na Slovenski obali.

☐ Na Hrvaškem.

☐ Drugod.

☐ Nismo bili na morju.

Potrdi

Na Slovenski obali. 0 glasov
Na Hrvaškem. 0 glasov
Drugod. 0 glasov
Nismo bili na morju. 0 glasov

Na Slovenski obali. Procenti: 0 %
Na Hrvaškem. Procenti: 0 %
Drugod. Procenti: 0 %
Nismo bili na morju. Procenti: 0 %

Slika 3: Primer preproste spletne strani v PHP-ju

Na strežniku je vsebina datoteke sledeča:

```
<html>
<head>
  <title>AnketaODopustu</title>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
  <link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
  <?php
    session_start();
    if (isset($_POST['Potrdi'])){
      for ($i = 1; $i<=4; $i++){
        if (isset($_POST['odgovor'.$i])){
          $_SESSION['tabela'][$i]++;
        }
      }
    }
    $tabela[1] = "Na Slovenski obali.";
    $tabela[2] = "Na Hrvaškem.";
    $tabela[3] = "Drugod.";
    $tabela[4] = "Nismo bili na morju.";
    if (!isset($_SESSION['tabela'])){
      $_SESSION['tabela'] = array();
      $_SESSION['tabela'][1] = 0;
      $_SESSION['tabela'][2] = 0;
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
$_SESSION['tabela'][3] = 0;
$_SESSION['tabela'][4] = 0;
}
echo "Kje ste bili letos na morju?<br><br>";
echo '<form name="anketa" action="anketa_DELA.php" method="post">';

    foreach ($tabela as $moznost => $vrednost) {
        echo '<input name="odgovor" type="radio" value="'.$moznost.'" /> '.$vrednost.'<br>';
    }
echo '<br>';
echo '<input type="submit" value="Potrdi" name="Potrdi">';
echo '</form>';
$stOdg=0;
if (isset($_SESSION['tabela'])){
    for ($i = 1; $i<=4; $i++){
        echo $tabela[$i].' &nbsp; ';
        echo $_SESSION['tabela'][$i].' glasov <br>';
        $stOdg = $stOdg+$_SESSION['tabela'][$i];
    }
    echo "<br><br>";
    for ($i=1; $i<=4; $i++){
        echo $tabela[$i].' &nbsp; ';
        if ($stOdg == 0){
            echo "Procenti: 0 %<br>";
        }
        else{
            echo "Procenti: ".round(($_SESSION['tabela'][$i]/$stOdg)*100) ."%<br>";
        }
    }
}
?>
</body>
</html>
```

Vidimo, da je celotna vsebina datoteke sestavljena iz ukazov v jeziku PHP, saj se značka '<?php' pojavi takoj na začetku telesa dokumenta (za <body>) in se zaključí ('?>') šele tik ob koncu datoteke. Ko uporabnik v brskalniku to datoteko zahteva od spletnega strežnika, le-ta poskrbi, da se datoteka dopust.php na strežniku prevede in izvede. Rezultat izvajanja je koda HTML. Strežnik pošlje uporabniku ta rezultat (datoteko s kodo HTML). Uporabnik lahko v brskalniku to kodo tudi vidi. Če torej uporabnik v brskalniku pogleda "izvorno" kodo, vidi:

```
<html>
<head>
    <title>AnketaODopustu</title>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
    <link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
    Kje ste bili letos na morju?<br><br>
    <form name="anketa" action="anketa_DELA.php" method="post">
        <input name="odgovor" type="radio" value="1" /> Na Slovenski obali.<br>
        <input name="odgovor" type="radio" value="2" /> Na Hrvaskem.<br>
        <input name="odgovor" type="radio" value="3" /> Drugod.<br>
        <input name="odgovor" type="radio" value="4" /> Nismo bili na morju.<br><br>
        <input type="submit" value="Potrdi" name="Potrdi">
    </form>
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
Na Slovenski obali. &nbsp; 0 glasov </br>
Na Hrvaskem. &nbsp; 0 glasov </br>
Drugod. &nbsp; 0 glasov </br>
Nismo bili na morju. &nbsp; 0 glasov </br>
<br><br>Na Slovenski obali. &nbsp;
Procenti: 0 %<br>Na Hrvaskem. &nbsp;
Procenti: 0 %<br>Drugod. &nbsp;
Procenti: 0 %<br>Nismo bili na morju. &nbsp;
Procenti: 0 %<br>
</body>
</html>
```

Na strežniku se je torej koda PHP predelala v ustrezne ukaze v jeziku HTML, ki se potem prikažejo na zaslону.

Ko bo uporabnik kliknil na gumb Potrdi, se bo na strežniku zopet izvedel program na datoteki anketa_DELA.php, saj v kodi HTML piše:

```
<form name="anketa" action="anketa_DELA.php" method="post">
```

Ukaz action pri znački FORM namreč pomeni, da se s klikom na gumb izvrši akcija forme. V našem primeru to pomeni, da se na strežniku izvede program na datoteki anketa_DELA.php. Tako se nam odpre nova stran, ki vrne najnovejše rezultate ankete.

1.3 Zgodovina

PHP je bil prvotno namenjen izdelovanju in objavi preprostih spletnih strani. Spodaj si lahko pogledamo kratko zgodovino razvoja jezika PHP, kjer je razvoj jezika povzet in strnjen v glavne verzije.

PHP/FI (Personal Home Page / Forms Interpreter)

Avtor jezika PHP je Rasmus Lerdorf. PHP je nasledil starejšo verzijo jezika, imenovano PHP/FI, ki je nastala leta 1995. PHP/FI je bil na začetku napisan v jeziku Perl. Jezik je bil prvotno namenjen zgolj pripravi skripte, ki bi štela število obiskov Rasmusove internetne strani. Sčasoma je jezik PHP postal uporaben za izdelavo preprostih dinamičnih spletnih strani. Že takrat je PHP imel omogočeno vključevanje v jezik HTML. Celotna sintaksa je bila podobna sintaksi jezika Pearl.

PHP 3

Verzijo PHP 3 so objavili leta 1997 in to je prva verzija, ki spominja na današnji PHP. Ta verzija je že omogočala delo z relacijskimi bazami podatkov. Pomembna značilnost te verzije je podpora objektno usmerjenemu programiranju. Ime so spremenili v rekurzivni akronimom PHP Hypertext Preprocessor.

PHP 4

Da bi omogočili učinkovitejše izvajanje skriptnih jezikov, jim pogosto dodajo tako imenovane pogone. Pogon je virtualni računalnik, ki skrbi za izvajanje skript.

Pogon z imenom Zend se je prvič pojavil v verziji PHP 4, ki je bila objavljena leta 2000. PHP 4 vsebuje tudi nekatere druge nove izboljšave: podpora več strežnikom, večja varnost pri vnosu vhodnih podatkov ter nekaj novih jezikovnih konstruktov.

PHP 5

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Julija 2004 je izšla prva verzija jezika PHP 5. Prinesla je kar nekaj izboljšav. Najpomembnejši sta: možnost uporabe imenskih prostorov ter pojav razredov XMLWriter in XMLReader. Slednja dva omogočata branje in pisanje datotek, sestavljenih v jeziku XML. V razvoju nove verzije so predvsem poskrbeli za boljšo varnost in odpravili nekatere napake v kodi. V času pisanja diplomske naloge je aktualna verzija 5.2.9.

1.4 Hitri pregled PHP-ja

Diplomska naloga je zasnovana za bralca, ki že pozna osnove programiranja v javi ali katerem drugem programskem jeziku. Tako v tem razdelku ne bo govora o zanki for ali while, stavku if in podobnem. Predpostavljamo, da bralec pozna osnovno strukturo nekega programskega jezika, da ve, kaj so objekti in podobno. V tem razdelku bomo predstavili nekatere posebnosti jezika PHP, ki jih bomo potrebovali pri delu z regularnimi izrazi.

Kot smo povedali že v uvodu, v spletni strani kodo zapisano v jeziku PHP navedemo znotraj značk

```
<?php  
?  
>
```

S tem jo ločimo od običajnega teksta oziroma od značk HTML.

1.4.1 Spremenljivke in podatkovni tipi

Ena večjih razlik med jezikom PHP in nekaterimi drugimi jeziki (na primer jezikom java) je v tem, da PHP iz vrednosti spremenljivke "prebere" za katero vrsto spremenljivke gre. Če na primer v spremenljivko shranimo niz, bo spremenljivka tipa niz. Če bi kasneje prepisali vrednost iste spremenljivke in bi vanjo shranili logično vrednost, bi spremenljivka postala tipa bool. Spremenljivke lahko pred prvo uporabo napovemo (deklariramo), ni pa to potrebno.

Kljub temu, da je tip spremenljivke določen z njeno vrednostjo, pa spremenljivki v PHP-ju ni potrebno takoj na začetku prirediti vrednosti. Šele ko jo potrebujemo, jo navedemo in vanjo shranimo neko vrednost. Tudi kadar jo deklariramo vnaprej, ni potrebno določiti, katerega tipa je.

Spremenljivke v PHP-ju označujemo tako, da njeno ime začnemo z znakom '\$':

```
$ime_spremenljivke;
```

Ime spremenljivke lahko vsebuje črke, števke in podčrtaj, vendar pa nikjer v imenu spremenljivke ne sme biti presledka. Prvi znak v imenu spremenljivke ne sme biti števka.

V PHP-ju poznamo štiri osnovne podatkovne tipe: niz, celo število, decimalno število ter logično vrednost.

Osnovni podatkovni tipi:

Vrsta	Pomen	Primer
integer	celo število	13
double	decimalno število	13.1
string	niz	trinajst

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

bool	logična vrednost	false
------	------------------	-------

Poleg tega poznamo tudi sestavljene tipe. To so: tabela, objekt in vir (resource). Obstaja tudi tip null. Tega uporabimo, če želimo imeti spremenljivko, a ta naj še ne bi imela vrednosti. Če torej napišemo:

```
<?php
$prazna_spremenljivka = null;
?>
```

smo s tem napovedali spremenljivko `$prazna_spremenljivka`, ki pa še nima vrednosti in je tipa null.

Spremenljivka tipa vir vsebuje sklic na nek zunanji vir, kot je na primer datoteka, povezava na bazo in podobno. Tako na primer po izvedbi stavka:

```
$imeDatoteke = fopen("datoteka.php","r")
```

spremenljivka `$imeDatoteke` vsebuje sklic na datoteko in je tipa vir. Pogosto kot ime take spremenljivke zasledimo ime `$handle`. Vire najpogosteje uporabljamo za povezavo z datotekami in pri povezavah na bazo.

Viri so različnih vrst (tipov). Katerega tipa je določen vir, lahko preverimo s klicem funkcije **`get_resource_type`**.

```
<?php
$spVir = fopen("zaposleni.csv","w");
print get_resource_type($spVir);
?>
```

Funkcija vrne vrsto tipa:

```
stream
```

Spremenljivka `$spVir` je torej vir, ki je podatkovni tok.

Kadar nas zanima, ali spremenljivka vsebuje vrednost določenega tipa, to lahko preverimo s klicem ene izmed funkcij:

```
is_int
is_double
is_string
is_bool
is_array
is_object
is_null
is_resource.
```

```
<?php
$dat = fopen("zaposleni.csv", "w");
print is_resource($dat);
?>
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Na zaslon dobimo izpisano število:

1

Število '1' (kot tudi katerokoli drugo, od nič različno število) predstavlja logično vrednost true. V našem primeru to pomeni, da je spremenljivka dat res tipa vir.

Če se izvede koda

```
<?php
print "Je 'abc' tipa niz? ";
if (is_string('abc')){
    print "Da.";
}
else print "Ne.";
print "<br>";
print "Je \"23\" tipa niz? ";
if (is_string("23")){
    print "Da.";
}
else print "Ne.";
print "<br>";
print "Je 23.5 tipa niz? ";
if (is_string(23.5)){
    print "Da.";
}
else print "Ne.";
print "<br>";
print "Je true tipa niz? ";
if (is_string(true)){
    print "Da.";
}
else print "Ne.";
?>
```

se to v brskalniku prikaže v obliki izpisa na zaslonu:

```
Je 'abc' tipa niz? Da.
Je "23" tipa niz? Da.
Je 23.5 tipa niz? Ne.
Je true tipa niz? Ne.
```

S funkcijo **is_string** smo preverili, katere spremenljivke so tipa string.

V zgledu smo uporabili tudi ukaz **'print "
"**. Kot smo omenili, je rezultat programa v PHP običajno koda HTML. Zato smo s **'print "
"** v izhod dodali ukaz jezika HTML za prehod v novo vrsto.

Omenjene funkcije, s katerimi preverjamo vrsto podatka v spremenljivki, so zelo uporabne, kadar preverjamo vhodne podatke.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Omenili smo že, da kadar nas zanima, ali je spremenljivka določenega tipa, uporabimo funkcijo oblike '**is_imeTipa**>', kot na primer **is_bool**, **is_string**. Če pa želimo izvedeti katerega tipa je določena spremenljivka, uporabimo funkcijo **gettype**.

```
<?php
    $test;
    print "Prazna spremenljivka: ".gettype($test);
    print "<br/>";
    $test = 18;
    print "18: ".gettype($test);
    print "<br/>";
    $test = 18.1;
    print "18.1: ".gettype($test);
    print "<br/>";
    $test = "osemnajst";
    print "osemnajst: ".gettype($test);
    print "<br/>";
    $test = true;
    print "true: ".gettype($test);
?>
```

V kodi zgoraj smo uporabili združevanje niza in spremenljivke (oziroma kode PHP). To storimo tako, da vmes dodamo piko '!'. Tak način združevanja se uporablja, kadar želimo združiti dva niza ali pa niz in spremenljivko.

Ko poženemo program, se na zaslon izpiše:

```
Prazna spremenljivka: NULL
18: integer
18.1: double
osemnajst: string
true: boolean
```

Funkcija **gettype** vrne tip spremenljivke. S prvim klicem te funkcije želimo preveriti, katerega tipa je spremenljivka, ki smo jo samo napovedali, nismo pa ji dodelili vrednosti. Tako nam prvi izpis **print gettype(\$test)**; na zaslon izpiše null. Spremenljivka `$test`, ki smo jo le deklarirali, je tipa null. Ostali klici te funkcije pa nam pokažejo, da se tip spremenljivke res "prilagaja" vsebini.

1.4.1.1 Pretvarjanje med tipi

Kadar želimo spremenljivki določiti nov tip, uporabimo funkcijo **settype**. Funkcija **settype** sprejme kot parametra spremenljivko in tip. Tako lahko spremenljivki poljubno spremenimo tip. Ob tem se seveda ustrezno spremeni tudi vrednost spremenljivke.

1. Pretvorba tipa int v double

```
<?php
    $test = 5;
    print $test;
    print "<br/>";
    settype($test, 'double');
    print $test;
    print "<br/>";
    print is_double($test);
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
?>
```

Ko število 5 hranimo v spremenljivko, je ta tipa `int`. Ko pa jo s funkcijo **`settype`** spremenimo v tip `double`, njena vrednost ni več celo število 5, ampak decimalno število 5.0. Ker pa se decimalna števila, katerih decimalni del je 0, izpišejo enako kot cela števila, to ni povsem očitno. Zato na koncu še preverimo, če je spremenljivka `$test` sedaj res tipa `double`. Zato uporabimo funkcijo **`is_double`**. Število 1 nam predstavlja logično vrednost `true`.

```
5
5
1
```

2. Pretvorba tipa `double` v `int`

Bolj zanimiv primer je pretvorba med `double` in `int`. Decimalnemu številu ob pretvorbi v obliko `int` funkcija `settype` odreže decimalna mesta. Tako se decimalno število 7.8 pretvori v celo število 7.

```
<?php
    $test = 7.8;
    print $test;
    print "<br/>";
    settype($test, 'int');
    print $test;
?>
```

```
7.8
7
```

3. Pretvorba tipa `string` v `int`

```
<?php
    $test = "abeceda";
    print $test;
    print "<br/>";
    settype($test, 'int');
    print $test;
?>
```

```
abeceda
0
```

Če niz pretvorimo v celo število in niz ne predstavlja zapisa celega števila, tedaj se niz pretvori v število 0.

```
<?php
    $test1 = "23.8";
    $test2 = "28";

    settype($test1, 'int');
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
print $test1;  
print "<br/>";  
settype ($test2, 'int');  
print $test2;  
?>
```

Če pa niz vsebuje število, se ta niz pretvori v celo število. Če je število slučajno decimalno, se shrani v spremenljivko `$test1` vrednost 23, to je celo število, decimalna mesta pa se zanemari (kot pri pretvorbi iz double v int). Tako dobimo izpis:

```
23  
28
```

4. Pretvorba tipa bool v string

```
<?php  
$test = true;  
print $test;  
print "<br/>";  
settype ($test, 'string');  
print $test;  
print "<br/>";  
print is_string($test);  
?>
```

```
1  
1  
1
```

Spremenljivki `$test`, tipa bool, ki ima vrednost true, smo spremenili tip v string. Vrednost se je spremenila v niz "1". Ker pa se tudi vrednost true izpiše kot 1, dobimo v obeh stavkih print izpisani enaki vrednosti. Zato smo želeli preveriti, ali je nova spremenljivka res tipa string. Ponovno dobimo izpis 1 (predstavlja logično vrednost true). Če pa bi vrstico

```
$test = true;
```

zamenjali z:

```
$test = false;
```

bi spremenljivka `$test` po pretvorbi iz tipa bool v string vsebovala prazen niz. Spremenljivka `$test` bi se tudi v tem primeru uspešno pretvorila v tip string.

5. Pretvorba med string in bool

Poglejmo si pretvorbo spremenljivke tipa string v tip bool.

```
<?php  
$test = "test";  
print $test;  
print "<br/>";
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
settype ($test, 'bool');  
print $test;  
?>
```

Če spreminjamo prazen niz ali pa niz "0" v tip bool, dobimo vrednost false, v vseh drugih primerih pa true. Denimo, da imamo v nizu shranjeno vrednost "test". Ker je to neprazen niz, je funkcija **settype** vrednost spremenljivke \$test pretvorila v vrednost true.

```
test  
1
```

Poglejmo si še primer, kjer funkcija spremeni tip spremenljivke iz tipa string v tip bool in s tem tudi nastavi vrednost spremenljivke, v našem primeru dobimo vrednost false.

```
<?php  
$test = "0";  
print $test;  
print "<br/>";  
settype ($test, 'bool');  
print $test;  
?>
```

```
0  
0
```

1.4.2 Funkcije za delo z nizi

V tem razdelku si bomo pogledali najpomembnejše funkcije za delo z nizi. Sem sodijo funkcije, ki jih bomo uporabljali v poglavju o regularnih izrazih ter v poglavju, kjer bomo PHP in regularne izraze uporabili na konkretnih primerih. S pomočjo funkcij za delo z nizi lahko rešujemo tudi probleme, ki bi jih sicer rešili s funkcijami za regularne izraze. Vendar bomo kasneje opazili, da te iste probleme z uporabo regularnih izrazov rešimo bolj elegantno. Funkcije so podane po abecednem vrstnem redu.

1.4.2.1 Echo

Funkcija **echo** izpiše niz, ki ga uporabimo kot parameter. Namesto te funkcije lahko uporabimo funkcijo **print**, ki smo jo za izpisovanje uporabljali do sedaj, saj obe delujeta podobno. Določene razlike so, vendar pa za nas niso bistvenega pomena. Zato jih tukaj ne bomo omenjali. Kot smo že povedali na strani 14 uporabljamo operator '.' za stikanje nizov, pa tudi niza in spremenljivke.

```
<?php  
$st_strani = 100;  
print "Diplomska naloga ima ".$st_strani." strani."  
?>
```

```
<?php  
$stevilo = 13;  
echo $stevilo." je moja srečna številka."  
?>
```

Na zaslon se izpiše:

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Diplomska naloga ima 100 strani.

13 je moja srečna številka.

1.4.2.2 Explode

Funkcija razbije niz v tabelo podnizov. Vhodni parametri funkcije so: ločilo, niz in število podnizov. Tretji parameter je izbirni in ga torej lahko uporabimo ali pa ne. Pri razbitju niza uporabimo ločilo, ki pove, na katerem mestu moramo niz razbiti. Ločilo lahko poljubno definiramo: presledek, vejica, pika. Če ločilo, ki smo ga določili funkciji, v nizu ne obstaja, dobimo tabelo z enim elementom, ki je kar kopija originalnega niza. V splošnem pa dobimo tabelo nizov z več elementi. Prvoten niz ostane nespremenjen. Če je niz, ki ga želimo razbiti, prazen, funkcija vrne tabelo z enim elementom, ki je prazen niz.

```
<?php
$niz = "Niz razbijemo na posamezne besede.";
$besede = explode(" ", $niz);
foreach ($besede as $izpis_besede){
    print $izpis_besede;
    print "<br/>";
}
?>
```

```
Niz
razbijemo
na
posamezne
besede.
```

V zgledu smo kot ločilo uporabili presledek. Tako funkcija niz razbije na mestih, kjer naleti na presledek. Posamezne besede so elementi tabele. Izpisali smo jih s pomočjo zanke foreach. Zapis

```
foreach ($besede as $izpis_besede)
```

pomeni, da se na vsakem koraku vrednost trenutnega elementa tabele \$besede zapiše v spremenljivko \$izpis_besede. Seveda bi lahko napisali program tudi z zanko while ali zanko for.

Če je v nizu med besedami več zaporednih presledkov (torej več tistih znakov, ki smo ga uporabili kot ločilo), vmes med podnizi dobimo tudi prazne podnize. Presledek je v tem primeru ločilo, loči pa tudi več presledkov skupaj in s tem vrne prazne podnize.

Poglejmo si primer, kjer med "Niz" in "razbijemo" naredimo 3 presledke:

```
<?php
$niz = "Niz  razbijemo na posamezne besede.";
$besede = explode(" ", $niz);
foreach ($besede as $izpis_besede){
    print $izpis_besede;
    print "<br/>";
}
?>
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
Niz  
  
razbijemo  
na  
posamezne  
besede.
```

Prvi presledek je torej ločil "niz" in prazen podniz, drugi presledek je ločil oba prazna podniza, tretji presledek pa je ločil drugi prazni podniz in "razbijemo".

Če bi uporabili le dva presledka, torej "Niz razbijemo na posamezne besede.", bi dobili samo en prazen niz. Če pa bi bilo presledkov denimo 7, bi dobili tabelo velikosti 11, kjer bi bili vsi elementi od drugega do sedmega niza v tabeli prazni podnizi.

Izbirni parameter število podnizov določa število elementov, ki jih tabela vsebuje. Če zamenjamo del kode

```
$besede = explode(" ", $niz);
```

ter uporabimo izbirni parameter število podnizov:

```
$besede = explode(" ", $niz, 2);
```

bi bil izpis programa

```
Niz  
razbijemo na posamezne besede.
```

S parametrom število podnizov (2) smo določili, da želimo dobiti tabelo velikosti 2. Tako funkcija razbije niz samo enkrat, na mestu, kjer se prvič pojavi ločilo. Prvi podniz je torej niz do prvega presledka, v drugi podniz pa se shrani preostali del niza.

Če pa bi napisali (v primeru, da imamo med besedami samo po en presledek):

```
$besede = explode(" ", $niz, 4);
```

bi dobili

```
Niz  
razbijemo  
na  
posamezne besede.
```

Če pa vpišemo pozitivno število, ki je večje od največjega števila podnizov, na primer

```
$besede = explode(" ", $niz, 40);
```

nam funkcija vrne:

```
Niz  
razbijemo  
na
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
posamezne  
besede.
```

Torej funkcija razbije niz, ko naleti na ločilo. V našem primeru je največje število podnizov, ki jih funkcija lahko ustvari, 40. Funkcija za vsak presledek, na katerega naleti, razbije preostanek niza na nov podniz. Ko naleti funkcija na konec niza, kjer ni več presledkov, konča z izvajanjem.

Če kot število podnizov navedemo negativno število, s tem povemo, da zadnjih nekaj elementov tabele ne potrebujemo. Poglejmo si zgled:

```
$besede = explode(" ", $niz, -3);
```

V našem primeru to pomeni, da zadnje tri podnize "zavržemo". Dobimo torej izpis:

```
Niz  
razbijemo
```

Kadar uporabimo negativno število in navedemo večje število, kot je število vseh podnizov v prvotnem nizu, torej na primer

```
$besede = explode(" ", $niz, -10);
```

funkcija vrne prazno tabelo.

1.4.2.3 Print_r

Funkcija **print_r** se uporablja predvsem za izpis tabel ali objektov. Tabele izpiše pregledno. Kadar izpisujemo tabele, dobimo izpisano številko polja ter njeno vsebino. Vhodni parameter funkcije je niz, tabela, lahko tudi objekt. Seveda lahko izpisujemo tudi int, double, bool ...

Poglejmo si primer izpisa z uporabo funkcije **print_r**.

```
<pre>  
<?php  
$tabela = array ('a', 'b', 'c', 'd');  
print_r ($tabela);  
?>  
</pre>
```

Oznaka '**<pre>**' je značka v jeziku HTML. Ta nam ohranja obliko izpisa (npr. presledek, skok v novo vrsto). Uporabna je pri izpisu tabel, saj je tako izpis preglednejši.

Tako dobimo

```
Array  
(  
    [0] => a  
    [1] => b  
    [2] => c  
    [3] => d  
)
```

Ker ima tabela \$tabela, ki smo jo izpisali, 4 polja, dobimo 4 vrstice oblike [i] => vrednost. Tu je i ustrezen indeks, vrednost pa vrednost tega polja tabele.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

S funkcijo **print_r** pa lahko izpisujemo tudi bolj zapletene primere tabel, kot so na primer dvo- ali večdimenzionalne tabele. V jeziku PHP je posamezni element tabele lahko spet tabela. Prav tako ni potrebno, da so vsi elementi tabele podatki istega tipa.

```
<pre>
<?php
$tabela = array ('a' => '1', 'b' => '2', 'c' => array ('3.1', '3.2', '3.3'));
print_r ($tabela);
?>
</pre>
```

```
Array
(
    [a] => 1
    [b] => 2
    [c] => Array
        (
            [0] => 3.1
            [1] => 3.2
            [2] => 3.3
        )
)
```

Tu smo uporabili tudi možnost, da kot indekse uporabimo poljubne vrednosti. V našem primeru so bili to znaki 'a', 'b' in 'c'. Vidimo tudi, da sta vrednosti prvih dveh polj števili, vrednost tretjega polja pa je tabela (kjer smo uporabili običajne indekse).

Primer izpisa tabele brez značke <pre>:

```
Array ( [a] => 1 [b] => 2 [c] => Array ( [0] => 3.1 [1] => 3.2 [2] => 3.3 ) )
```

1.4.2.4 Sprintf

Funkcija vrne niz. Ta niz na ustreznih mestih vsebuje vrednosti izrazov, ki smo jim določili izgled (npr. celo število spremenimo v obliko za odstotek). Prvi parameter določa, kako bo videti niz, ki je rezultat funkcije. V niz med "običajno" besedilo vstavimo še oznake '%'. Na ta mesta vstavimo ustrezne vrednosti drugega, tretjega, oz. n-tega parametra funkcije (kjer je n število parametrov funkcije). Pri tem se vrednost parametra oblikuje glede na formatna določila, ki sledijo znaku za odstotek '%'. Nekaj najpogostejše uporabljenih formatnih določil si lahko pogledamo v spodnji tabeli:

Tip	Pomen
%.%	odstotek
%.b	binarno število
%.d	celo število
%.u	absolutna vrednost celega števila
%.f	decimalno število s plavajočo vejico
%.s	niz
%.x	heksadecimalno število (male črke)
%.X	heksadecimalno število (velike črke)

Poglejmo zgled.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
<?php
    $niz = "Pozdravljeni";
    $stevalo = 123.23;
    $izpis = sprintf("%s obiskovalci. Vaša srečna številka je: %d.", $niz, $stevalo);
    echo $izpis;
?>
```

```
Pozdravljeni obiskovalci. Vaša srečna številka je: 123.
```

Podani imamo dve spremenljivki. V nizu na prvo mesto vstavimo spremenljivko, ki jo oblikujemo v niz z določilom '%s', na drugo mesto pa spremenljivko, ki jo oblikujemo v celo število z '%d'. Tako spremenljivka `$niz` dobi obliko niza ter spremenljivka `$stevalka` obliko celega števila.

Če pa bi spremenili `$izpis` v

```
$izpis = sprintf("%s obiskovalci. Vaša srečna številka je: %f.", $niz, $stevalo);
```

bi bil izpis:

```
Pozdravljeni obiskovalci. Vaša srečna številka je: 123.230000.
```

Decimalno število 123.23 smo preoblikovali v število s plavajočo vejico. To število se vedno izpiše na šest decimalnih mest (razen če sami določimo drugače).

Pri zamenjavi dela kode

```
$izpis = sprintf("%s obiskovalci. Vaša srečna številka je: %s.", $niz, $stevalo);
```

pa bi bil izpis:

```
Pozdravljeni obiskovalci. Vaša srečna številka je: 123.23.
```

V našem primeru funkcija vrne število 123.23 oblikovano v niz.

Če v prvotnem nizu uporabimo več formatnih določil, kot pa imamo spremenljivk, moramo za znak '%' dodati oznako mesta, ki pove, katera spremenljivka spada na določeno mesto. Kadar na ta način določimo, da imamo spremenjen vrstni red izpisa spremenljivk, tudi formatna določila zapišemo v drugačni obliki. Tako v tem primeru ne pišemo '%d', kot je običajno za celoštevilsko obliko spremenljivke, ampak napišemo '\$.d' takoj za oznako mesta.

Poglejmo si primer za spremenjeni vrstni red izpisa števil:

```
<?php
    $stevalo1 = 111;
    $stevalo2 = 222;
    $stevalo3 = 333;
    $besedilo2 = sprintf("<br/>Obrnjen vrstni red števil: %3$.d, %2$.d, %1$.d", $stevalo1,
        $stevalo2, $stevalo3);
    echo $besedilo2;
?>
```

Zapis '%3' pomeni, da se prva spremenljivka pojavi na tretjem mestu. Ker pa moramo še določiti, v kakšni obliki bi radi imeli spremenljivko, napišemo za znakoma '%3' še '\$.d', ki pove, da bo spremenljivka dobila obliko celega števila.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Obrnjen vrstni red števil: 333, 222, 111

Poglejmo si, kako lahko uporabimo formatno določilo %f. Ta nam služi za izpis števila s plavajočo vejico:

```
<?php
$steveno1 = 111.367;
$steveno2 = 222.367;
$steveno3 = 333.367;
$besedilo2 = sprintf("<br/>Izpis števil s plavajočo vejico: %.0f, %.2f, %.5f", $steveno1,
    $steveno2, $steveno3);
echo $besedilo2;
?>
```

Izpis števil s plavajočo vejico: 111, 222.37, 333.36700

Število 111.367 zapišemo kot celo število. Ko število 222.367 zapišemo na dve decimalni mesti, funkcija zaokroži zadnje decimalno mesto. Kadar pa ima vrednost spremenljivke manj decimalnih mest, kot pa smo jih določili v formatnem določilu, spremenljivki doda na konec ustrezno število ničel.

Če bi v kodi napisali

```
$besedilo2 = sprintf("Izpis števila s plavajočo vejico: %.f", $steveno);
```

bi nam funkcija vrnila število napisano na šest decimalnih mest, ne glede na to, kako je bilo število prvotno napisano (celo število ali pa decimalno število, ki ima več kot sedem mest). To pomeni, da je %f enako kot %.6f .

Izpis števila s plavajočo vejico: 123.110000

Poglejmo še naslednji zgled

```
<?php
$steveno = 123;
$niz = "naslednjo";
$izpis = sprintf("Vaša srečna številka je: %x. Sedaj izberite %%", $steveno, $niz);
echo $izpis;
?>
```

Kadar danega parametra ne moremo preoblikovati tako, kot določa formatno določilo (npr. niza ne moremo preoblikovati v obliko za odstotek), funkcija na tem mestu vstavi prazen niz (znak za odstotek pa ostane).

Tako nam v našem zgledu v prvem primeru vrednost spremenljivke \$steveno pretvori v heksadecimalni zapis števila 123, to je 7b. Vrednost spremenljivke tipa niz smo hoteli zapisati v obliki za odstotke (%%). Ker pa to ne gre, je funkcija **sprintf** na tem mestu za znakom % napisala prazen niz. Kadar spremenljivko tipa bool preoblikujemo v niz in le-ta vsebuje vrednost false, se vrednost false izpiše v obliki praznega niza. Kadar pa spremenljivka tipa bool vsebuje vrednost false in jo preoblikujemo v tip int, se vrednost false izpiše kot število 0.

Vaša srečna številka je: 7b. Sedaj izberite %

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

1.4.2.5 Str_replace

Podan imamo niz. Funkcija **str_replace** v nizu zamenja podane podnize z novimi vrednostmi in vrne spremenjeni niz. Prvotni niz ostane nespremenjen. Funkcija sprejme tri obvezne parametre. To so vzorec, zamenjava in niz. Vzorec nam določa del niza, ki ga želimo zamenjati. V postopku zamenjave potrebujemo novo vrednost, ki jo bomo vnesli na mesto stare. Imenujemo jo zamenjava. Zadnji parameter pri klicu funkcije je niz. Ta nam določa niz, v katerem menjamo vzorec z novo vrednostjo. Denimo, da bi radi niz "Dragi obiskovalci!" spremenili v "Dragi gostje!". Vzorec "obiskovalci" moramo torej zamenjati z zamenjavo "gostje".

```
<?php
    print str_replace("obiskovalci","gostje","Dragi obiskovalci!");
?>
```

```
Dragi gostje!
```

Oglejmo si še en zgled.

```
<?php
    $orig = "13120011229901";
    $novi = str_replace("0","9",$orig);
    print "Novi niz: ".$novi;
    print "Prvotni : ".$orig;
?>
```

```
Novi niz: 13129911229991
Prvotni : 13120011229901
```

V nizu 13120011229901 smo vse številke 0 zamenjali s števkami 9.

Funkcija lahko sprejme tudi četrti, izbirni parameter. Ta parameter mora biti obvezno ime spremenljivke. Vanjo funkcija shrani število zamenjav. Če bi torej določili tudi izbirni parameter število zamenjav, nam funkcija **str_replace** poleg tega, da vrne nov spremenjen niz, pove tudi število zamenjav.

```
<?php
    $orig = "13120011229901";
    $novi = str_replace("0","9", $orig, $zam);
    print "0 smo z 9 zamenjali ".$zam." krat!";
?>
```

```
0 smo z 9 zamenjali 3 krat!
```

Če imamo več nizov, v katerih želimo zamenjati vzorec z zamenjavo, za tretji parameter uporabimo tabelo nizov!

```
<pre>
<?php
    $orig = array("13120011229901", "687502", "87890007987");
    $novi = str_replace("0","9", $orig, $zam);
    print_r ($orig);
    print "<br>";
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
foreach ($novi as $zamenjava){
    print $zamenjava;
    print " ";
}
print "<br>";
print "0 smo z 9 zamenjali ".$zam." krat!";
?>
<pre>
```

Sedaj nam `$zam` prešteje skupno število zamenjav v vseh nizih. Najprej smo izpisali vhodni podatek - tabelo. Nato pa smo izpisali še spremenjene nize, ki jih funkcija vrne.

```
Array
(
    [0] => 13120011229901
    [1] => 687502
    [2] => 87890007987
)
13129911229991 687592 87899997987
0 smo z 9 zamenjali 7 krat!
```

Kadar sta parametra `zamenjava` in `vzorec` tabeli, se v splošnem dogaja naslednje: v nizu (enem ali na vseh nizih v tabeli) vse podnize `vzorec[0]` zamenjamo s podnizi `zamenjava[0]`, vse podnize `vzorec[1]` s podnizi `zamenjava[1]`, vse podnize `vzorec[n-1]` s podnizi `zamenjava[n-1]`, kjer je `n` število elementov tabele `vzorec`.

Poglejmo si še primer, kjer imamo podano tabelo zamenjav.

```
<?php
$tekst = "Zaradi problemov s šumniki bomo zamenjali v besedilu vse
         črke, ki imajo strešice.";
$sumniki = array("č", "š", "ž", "Č", "Š", "Ž");
$nesumniki = array("c", "s", "z", "C", "S", "Z");
$stZamenjav;
$nov_tekst = str_replace( $sumniki, $nesumniki, $tekst, $stZamenjav);
echo "Originalno besedilo:\n".$tekst."\n";
echo "<br/>";
echo "Zamenjano besedilo:\n".$nov_tekst."\n";
echo "<br/>";
echo "Šumniki so v besedilu nastopali ".$stZamenjav."-krat";
?>
```

```
Originalno besedilo:"Zaradi problemov s šumniki bomo zamenjali v
besedilu vse črke, ki imajo strešice."
Zamenjano besedilo:"Zaradi problemov s sumniki bomo zamenjali v
besedilu vse crke, ki imajo stresice."
Šumniki so v besedilu nastopali 3-krat
```

Če se zgodi, da je število polj v tabeli `zamenjava` manjše kot v tabeli `vzorec`, namesto manjkajočih elementov zamenjave uporabimo prazen niz.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Poglejmo si zgled. Denimo, da imamo recept, v katerem imamo količine podane z besedami. Radi bi jih zamenjali s števili. Le tam, kjer imamo količino podano kot besedo "en", bi to kar izpustili:

```
<?php
$recept = "Biskvit: pet jajc, pet žlic tople vode, deset žlic
          sladkorja, osem žlic moke, en pecilni prašek, en
          vanilijev sladkor.";
$besede = array("pet", "deset", "osem", "en");
$stevilke = array("5", "10", "8");
$nov_recept = str_replace( $besede, $stevilke, $recept);
echo $nov_recept;
?>
```

```
Biskvit: 5 jajc, 5 žlic tople vode, 10 žlic sladkorja, 8 žlic moke,
pecilni prašek, vanilijev sladkor.
```

1.4.2.6 Strpos

Funkcija vrne prvo mesto, kjer se v danem nizu pojavi nek podniz. Sprejme tri parametre: niz, v katerem iščemo, iskani niz ter dodatni izbirni parameter zamik. Če funkcija iskanega niza ne najde, vrne vrednost false.

Primer:

```
<?php
$niz = "V pravokotnem trikotniku je kvadrat hipotenuze enak vsoti kvadratov obeh katet.";
$isci = "kvadrat";
$indeks = strpos($niz, $isci);

if ($indeks == false) {
    print "Iskanega niza \"".$isci."\" ni bilo mogoče najti v nizu \"".$niz."\"";
}
else {
    print "Našli smo iskani niz \"".$isci."\" v nizu: \"".$niz."\"";
    print " Nahaja se na mestu z indeksom ".$indeks.".";
}
?>
```

Izpis:

```
Našli smo iskani niz "kvadrat" v nizu: "V pravokotnem trikotniku je
kvadrat hipotenuze enak vsoti kvadratov obeh katet." Nahaja se na
mestu z indeksom 28.
```

Če bi želeli, da funkcija podniz išče od določenega mesta naprej, to možnost določimo s parametrom zamik. Zamik nam določi, pri katerem znaku v nizu naj funkcija začne iskanje. Če bi torej v prejšnjem zgledu želeli najti še drugo besedo kvadrat, bi torej morali napisati:

```
$indeks = strpos($niz, $isci, 29);
```

oziroma namesto 29 uporabiti poljubno število med 29 in 58 (58 je mesto, kjer se iskani niz "kvadrat" drugič pojavi v nizu "V pravokotnem trikotniku je kvadrat hipotenuze enak vsoti kvadratov obeh katet".)

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Poglejmo še en zgled:

```
<?php
    $niz = "V pravokotnem trikotniku je kvadrat hipotenuze enak vsoti kvadratov obeh katet.";
    $isci = "pravokotnem";
    $mesto = "35";
    $indeks = strpos($niz, $isci, $mesto);

    if ($indeks == false){
        print "Iskanega niza '". $isci.'" ni bilo mogoče najti v nizu '". $niz.'"';
        print " Iskali smo od indeksa ". $mesto." dalje.";
    }
    else{
        print "Našli smo iskani niz '". $isci.'" v nizu: '". $niz.'"';
        print " Nahaja se na mestu z indeksom ". $indeks.".";
        print " Iskali smo od indeksa ". $mesto." dalje.";
    }
?>
```

Vidimo, da smo kot tretji parameter (zamik) uporabili kar spremenljivko tipa string. Ker pa je v nizu zapisano celo število, PHP to mirno prenese in upošteva ustrezno vrednost. Zgled bi se torej izvedel enako tudi, če bi napisali:

```
$mesto = 35;
```

V opisanem zgledu funkcija ne bo našla nič, saj se iskani podniz nahaja pred indeksom zamika, ki smo ga "nerodno" določili.

```
Iskanega niza "pravokotnem" ni bilo mogoče najti v nizu "V pravokotnem
trikotniku je kvadrat hipotenuze enak vsoti kvadratov obeh katet."
Iskali smo od indeksa 35 dalje.
```

1.4.3 Delo z datotekami

Najpogosteje uporabljene funkcije za delo z datotekami so branje datoteke ter pisanje in dodajanje v datoteko. Pri delu z regularnimi izrazi bomo potrebovali le funkciji, ki omogočata branje in pisanje v datoteko.

Karkoli počnemo z datoteko, jo moramo najprej odpreti. Za to uporabimo funkcijo **fopen**. Pri tem kot prvi parameter navedemo ime datoteke, ki jo odpiramo. Ta parameter je niz, v katerem je ime datoteke v obliki, kot je poimenovana v operacijskem sistemu.

Razlikujemo tri načine odpiranja. Katerega uporabljamo, je odvisno od tega, kaj bomo z datoteko počeli. Ustrezen način označimo z vrednostjo drugega parametra. Če želimo na datoteko pisati, to označimo z oznako 'w' (write). Kadar želimo datoteko odpreti za branje, to označimo z 'r' (read). Poznamo tudi tretji način, 'a'(append), ki se uporablja za dodajanje v datoteko.

Funkcija **fopen** vrne vir, ki zna brati podatke iz datoteke, oziroma pisati v datoteko (odvisno od načina v katerem odpremo datoteko). Tako je datoteka odprta in pripravljena za obdelovanje (branje, pisanje, dodajanje).

1.4.3.1 Branje iz datoteke

Če želimo z datoteke brati, jo moramo najprej odpreti v načinu za branje.

```
<?php
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
//odpiranje datoteke za branje
$datoteka = "test.txt";
$branje = fopen($datoteka, 'r');
?>
```

Kot smo že omenili, to storimo s pomočjo funkcije **fopen**. Sklic na datoteko je spremenljivka tipa vir. V zgledih kode, ki jih najdemo na spletu, jo največkrat zasledimo pod imenom `$handle`.

Poglejmo si zgled, kjer smo napačno vnesli ime datoteke:

```
$datoteka = "test5.txt";
```

Če je v kodi napačno ime datoteke ali pa pot do nje ni pravilna, nam funkcija vrne `false`. Če pa bi izpisovali vsebino, bi funkcija izpisala prazen niz ter napako.

```
Warning: fopen(test5.txt) [function.fopen]: failed to open stream: No
such file or directory in C:\wamp\www\test2.php on line 4
```

S tem pove, da na mestu, ki smo ga navedli, ne najde ustrezne datoteke oz. je ne uspe odpreti.

Ko je datoteka uspešno odprta v načinu za branje, podatke z nje lahko beremo s funkcijama **file** ali **file_get_contents**. Prva datoteko vrne v obliki tabele. Vsako vrstico datoteke zapiše v svoje polje tabele. Funkcija **file_get_contents** vrne vsebino datoteke v obliki enega niza.

Omejili se bomo na funkcijo **file**, saj je vsebino datoteke, prebrano s to funkcijo, po branju lažje obdelovati. Funkcija **file** torej vrne tabelo, ki vsako vrstico iz datoteke vsebuje v svojem polju.

Poglejmo si primer uporabe te funkcije.

Naj bo vsebina datoteke `test.txt` naslednja:

```
aaa
bbb
ccc
ddd
eee
fff
ggg
```

Denimo, da imamo naslednjo kodo

```
<pre>
<?php
    $lines = file('test.txt');
    $zadnja = count($lines)-1;
    $srednja = $zadnja/2;

    print 'Prva vrstica: '.$lines[0];
    print 'Srednja vrstica: '.$lines[$srednja];
    print 'Zadnja vrstica: '.$lines[$zadnja];
?>
</pre>
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Funkcija **file** shrani vsako vrstico v datoteki v svoje polje v tabeli. S primerom izpisa prve, srednje in zadnje vrstice je to tudi lepo vidno. Ker pa imamo liho število vrstic, pa dobimo pri računanju številko srednje vrstice navzdol zaokroženo. Polja v tabeli se začnejo s številko 0, zato je tudi številka zadnje vrstice za eno manjša od števila vseh vrstic.

```
Prva vrstica: aaa
Srednja vrstica: ddd
Zadnja vrstica: ggg
```

1.4.3.2 Zapisovanje v datoteko

Tako kot pri branju, moramo tudi pri zapisovanju v datoteko le to najprej odpreti v ustreznem načinu. Kot smo že omenili, ko datoteko odpiramo za pisanje, uporabimo oznako 'w'. Kadar se odločimo za pisanje v tem načinu in datoteka, v katero pišemo, ni prazna, se stara vsebina zbriše in pišemo v prazno datoteko.

```
<?php,
//odpiranje datoteke za pisanje
$datoteka = "http://localhost/test.txt";
$handle = fopen($datoteka, 'w');
?>
```

Če bi pri odpiranju datoteke uporabili oznako 'a', bi novo vsebino samo dodali na konec že obstoječe vsebine datoteke. Sklic na datoteko zapišemo v neko spremenljivko, v našem primeru `$handle`. To spremenljivko potem uporabimo v funkcijah, s katerimi delamo (pišemo, beremo, zapiramo ...). Postopek odpiranja datoteke v načinu za pisanje si bomo pogledali na konkretnem primeru.

```
<?php
$handle = fopen("http://localhost/test.txt", "w");
fputs($handle, "Pozdravljeni!");
fclose($handle);
?>
```

V prvi vrstici datoteko `test.txt` odpremo za pisanje. Funkcija nam vrne sklic na datoteko (vir) ki ga zapišemo v datotečno spremenljivko z imenom `$handle`. V drugi vrstici uporabimo funkcijo **fputs**. Tej posredujemo vir in niz, ki ga želimo na vir zapisati. S tem v datoteko zapišemo niz "Pozdravljeni!". Vir na koncu zapremo (prenehamo obdelovati) s klicem funkcije **fclose**. S tem se vsebina, ki smo jo zapisali na vir, tudi dokončno zapiše v datoteko.

1.4.3.3 Fgetcsv

Funkcija **fgetcsv** je posebna vrsta funkcije za branje iz datoteke. Kadar so v vsaki vrstici podatki ločeni z ločilom (običajno z vejico), lahko to funkcijo uporabimo za lažje branje posameznih podatkov. Okrajšava CSV pomeni comma-separated values (z vejico ločene vrednosti). Funkcija prebere tekočo vrstico ter pri tem glede na ločilo razdeli vrstico (niz) na ustrezne podnize in jih shrani v tabelo.

Če je kakšna vrstica v datoteki prazna, funkcija vrne tabelo, ki ima le eno polje, ki vsebuje prazen niz. Če je bilo branje datoteke uspešno, nam funkcija vrne tabelo, sicer pa vrne `false`.

Funkcija ima dva obvezna vhodna parametra. Parameter datoteka določa vir podatkov (torej kazalec na datoteko CSV, ki smo ga dobili z odpiranjem datoteke v načinu za branje).

V splošnem funkcija vrne novo polje v tabeli, ko naleti v datoteki CSV na ločilo. Drugače pa je, če določimo izbirni parameter dolžina. V tem primeru dobimo preštete vse znake iz prvotne datoteke CSV, poleg znakov v nizu tudi vsa ločila ter prehod v novo vrsto. Število znakov v eni vrstici mora

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

biti manjše ali enako dolžini. Če je število znakov vsaj v eni vrstici daljše od dolžine se predolge vrstice prelomijo in se prelomljen del od preloma do konca vrstice obravnava kot ločena vrstica.

Poleg obveznih parametrov ima funkcija tudi izbirni parameter ločilo. To je znak, ki ločuje posamezna polja. Privzeta vrednost ločila je vejica. Ločilo lahko določimo tudi sami, vendar pa mora biti obvezno samo en znak.

Poglejmo si primer, kjer imamo podatke o zaposlenih shranjene v datoteki CSV. Podatki iz datoteke:

IME,PRIIMEK,STAROST,POKLIC

Janez,Novak,28,direktor

Marija,Novak,30,tajnica

Ana,Lampe,33,prodajalka

Martin,Jurko,25,natakar

Monika,Petric,40,delavka

Cene,Rutar,35,blagajnik

Petra,Drole,22,programer

Jure,Balog,32,natakar

Igor,Serjun,24,prodajalec

Maja,Kikelj,35,prodajalka

```
<?php
$branje = fopen("zaposleni2.csv", "r");
while (($podatki = fgetcsv($branje, 1000, ",")) !== false){
    $stevilo = count($podatki);
    for ($i = 0; $i < $stevilo; $i++){
        echo $podatki [$i]. "<br/>\n";
    }
    echo "-----";
    echo "<br/>\n";
}
fclose($branje);
?>
```

Ko datoteko zaposleni2.csv odpremo v načinu za branje, zapišemo sklic na datoteko vir v datotečno spremenljivko `$handle`. Zanka `while` prebira vrstice datoteke in jih shranjuje v tabelo `$podatki`, zanka `for` pa izpiše elemente te tabele. Polja v vrstici datoteke so med sabo ločena z vejico. Funkciji **`fgetcsv`** povemo, da je ločilo vejica. S celim številom 1000, ki pomeni število znakov v določeni vrstici omejimo dolžino najdaljše vrstice. Dolžino najdaljše vrstice pa bi lahko tudi izpustili, vendar pa s tem upočasnimo delovanje funkcije **`fgetcsv`**. Datoteko beremo po vrsticah, vse do konca datoteke, torej, ko v spremenljivko `$podatki` funkcija `fgetcsv` da vrednost `false`.

Program vsako vrstico vhodne datoteke (ki je sedaj v tabeli) zapiše v več vrstic na standardni izhod, ki so med seboj ločene z vrstico "----".

Ko zanka obdela posamezno vrstico vhodne datoteke, napiše na standardni izhod še vrstico "----".

Poglejmo si začetek izpisa, ki ga dobimo:

```
IME
PRIIMEK
STAROST
POKLIC
-----
Janez
Novak
28
direktor
```

Marija
Novak
30
tajnica

2 Regularni izrazi

2.1 Uvod

Regularni izraz je posebna oblika niza, ki opisuje izgled določenega niza. Sestavljen je iz točno določenega vrstnega reda znakov (črk, števk in simbolov). Ti znaki opisujejo niz oz. del niza. Regularni izrazi se uporabljajo v urejevalnikih besedil, v programskih jezikih za iskanje in spreminjanje podatkov, pa tudi v drugih primerih, kadar potrebujemo nize. Uporabo regularnih izrazov v grobem ločimo na dva dela: preverjanje veljavnosti vhodnih podatkov in iskanje med pravnimi podatki.

Najprej si pogledjmo dva pojma, ki ju bomo pogosto srečevali pri regularnih izrazih. To sta ujemati se in ustrezati.

Ujemati se: o tem, da se regularni izraz ujema z nizom govorimo, če v nizu najdemo podniz, ki ga opisuje regularni izraz. Tako se regularni izraz '**a**' ujema s poljubnim nizom, ki vsebuje 'a' (npr. "abc", "danes", "aaa"), regularni izraz '**danes**' se ujema s poljubnim nizom, ki vsebuje danes, npr. nizom "danes je lep dan", "ne danes". Regularni izraz '**ab***' se ujema s poljubnim nizom, ki vsebuje podniz, sestavljen iz znaka a, ki mu sledi poljubno število b-jev, torej tudi nič. Torej se ta regularni izraz ujema npr. z nizi "danes", "abeceda", "Daaabbb", ne pa npr. z nizi "kolo", "hribi", "bbb". Ujema se torej z vsakim nizom, ki vsebuje vsaj en a.

Ustrezati: O ustrezanju govorimo, če niz lahko sestavimo z ustreznim opisom iz regularnega izraza. Tako regularnima izrazoma '**a**' in '**^a\$**' ustreza le niz "a", regularnemu izrazu '**^danes\$**' pa ustreza le niz "danes". Regularnemu izrazu '**ab***' ustrezajo npr. nizi "a", "ab", "abbbb", torej vsi nizi, ki se začno z a-jem, ki mu sledi poljubno število b-jev.

Ustrezati je torej "strožji" pojem. Z regularnim izrazom se torej ujemajo vsi nizi, ki kot kak podniz vsebujejo niz, ki ustreza regularnemu izrazu.

Regularni izraz je sestavljen iz več znakov. Te delimo na običajne in nadomestne znake. Oglejmo si najprej nekaj primerov regularnih izrazov. Podrobneje bomo te regularne izraze razložili v nadaljevanju. Regularne izraze bomo v besedilu vedno navajali v enojnih narekovajih (npr. '**a***'), ki pa niso del samega regularnega izraza. Pisali jih bomo tudi odebeljeno, da se bodo lažje ločili od ostalega besedila. Nize, ki ustrezajo danim regularnim izrazom ali pa tiste nize, ki se z regularnim izrazom ujemajo, bomo v besedilu vedno zapisali v dvojnih narekovajih (npr. "aaa"), ki zopet niso del ustreznega niza.

Najenostavnejši so regularni izrazi, ki so sestavljeni iz samih običajnih znakov. Tem regularnim izrazom ustrezajo kar enaki nizi, ujemajo pa se z vsemi tistimi nizi, ki dani niz vsebujejo kot podniz. Tako se na primer regularni izraz '**a**' ustreza le nizu "a" in se ujema s poljubnim nizom, ki vsebuje črko a (na primer z nizom "anja", "danes", "aaa", "bedarija", "diploma" ...), regularni izraz '**dan**' pa se ujema s katerikoli nizom, ki vsebuje podniz "dan" ("danes", "dante", "dandanašnji", "adan", "osmi dan", "rojstni dan", "xdany" ...).

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Seveda so regularni izrazi bolj uporabni takrat, ko vsebujejo posebne znake. Tak posebni znak je tudi '*'. Regularnemu izrazu '**a***' ustrezajo nizi, ki so sestavljeni iz poljubnega števila a-jev. Temu regularnemu izrazu ustrezajo nizi "a", "aa", "aaa" in podobni. S tem regularnim izrazom se torej ujemajo vsi nizi, ki te kombinacije a-jev vsebujejo. Znak a je v tem regularnem izrazu običajni znak, '*' pa nadomestni znak.

Ker pa '**a***' pomeni tudi nič a-jev in mu torej ustreza tudi prazen niz, se dejansko z regularnim izrazom '**a***' ujemajo vsi nizi.

Ker '*' kot nadomestni znak pogosto srečamo pri različnih iskanjih, moramo biti pozorni, saj je tu raba drugačna. V iskalnikih običajno zapis '**a***' pomeni, da iščemo nize, ki se začenejo z a (npr "abeceda", "anakonda"). Iskalnik torej najde besede, sestavljene iz korena besede, ki mu sledi poljubno število različnih znakov. Pri regularnih izrazih pa znak '*' pomeni, da se zadnji znak pred zvezdico lahko poljubno-krat ponovi.

Regularnemu izrazu '**a*b**' ustrezajo vsi nizi, kjer zaporedju a-jev sledi črka b. Taki nizi so npr. "ab", "aaaaab", "aab", pa tudi "b", saj '*' pomeni tudi, da se določen znak lahko tudi ne pojavi. Z regularnim izrazom pa se ne ustrezajo nizi "bba", "Ab", "aaacbb" in podobni. Pač pa se niz "aaacbb" s tem regularnim izrazom ujema, ker vsebuje podniz b.

Seveda pa so regularni izrazi lahko bistveno bolj zapleteni, saj '*' ni edini nadomestni znak. Tako sta regularna izraza tudi '**[A-Z]{3}[0-9]{3}**' in '**^/[a-z]+\.[a-z]+@(student\.)?fmf.uni-lj.si\$**'.

Regularni izraz nam torej pove, kakšne oblike naj bo niz, ki ga iščemo. Npr. za niz "ABC123" bi bil eden izmed veljavnih regularnih izrazov (torej regularni izraz, ki mu ustreza niz "ABC123") tak: '**[A-Z]{3}[0-9]{3}**'.

Oglejmo si še en primer zapletenejšega regularnega izraza z nadomestnimi znaki. Regularni izraz '**^/[a-z]+\.[a-z]+@(student\.)?fmf.uni-lj.si\$**' opisuje internetne poštne naslove študentov Fakultete za matematiko in fiziko, ki so oblike ime.priimek@student.fmf.uni-lj.si. Tako niz "jasna.kobalstudent.fmf.uni-lj.si" ne ustreza temu regularnemu izrazu, ker mu manjka znak '@'.

Pri sestavljanju regularnih izrazov moramo biti previdni. Večkrat nam namreč ni dovolj le to, da se ujema oblika, ustrezati mora tudi vsebina. Recimo, da bi želeli napisati regularni izraz, ki bi ustrezal IP naslovom (ti so sestavljeni iz štirih eno- do trimestrnih števil med 1 in 255, ki so med seboj ločena s pikami). Recimo, da bi v ta namen napisali regularni izraz '**b\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}b**'. S tem smo povedali, da je IP naslov sestavljen iz štirih skupin, ki jih sestavljajo ena do tri števke. Obliko IP naslova smo s tem sicer "ujeli". A temu regularnemu izrazu ustreza na primer tudi naslov "111.99.987.333", ki pa ni veljaven IP naslov.

2.2 Znakovni razred in skupine znakov

Kot smo povedali že prej, regularne izraze zapišemo s pomočjo običajnih in nadomestnih znakov. Običajni znaki so tisti, ki imajo dobeseden pomen. V regularnem izrazu so to tisti znaki, ki ustrezajo sami sebi. Tak primer je npr. črka '**a**'. Nadomestni znaki pa imajo v izrazu poseben pomen. Tako npr. pika '.' nadomesti katerikoli znak. Tako regularni izraz '**...**' ustreza poljubnemu nizu, sestavljenemu iz točno treh znakov. Regularni izraz '**J...A**' ustreza poljubnemu nizu petih znakov, ki se začnejo z J in se z A končajo, denimo "JASNA", "JELKA", "JJjA", "J!2aA" ...

Znak '*', kot smo spoznali že prej, ustreza večkratni ali nobeni ponovitvi znaka, ki nastopa pred njim. Regularni izraz '**.***' ustreza katerikoli nizu, ki vsebuje enega ali nič znakov, ki se lahko pojavijo večkrat ali nikoli ("a", "", "bbbb", "XX" ...). Pika '.' pomeni katerikoli znak, '*' pa poljubno (tudi nič) dodatnih znakov. Tako se niz "razred" seveda ujema z regularnim izrazom '**.***'. S tem regularnim izrazom se ujema poljubni niz, saj vsak niz vsebuje prazen niz, ki ustreza temu izrazu.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Omeniti velja še ubežne znake, ki izničijo pomen nadomestnih znakov. Najpogostejši ubežni znak je poševnica nazaj ' $\backslash$ '.

Če bi želeli napisati regularni izraz, ki bi ustrezal le nizu, sestavljenemu iz znaka pika in zvezdica " $.*$ ", bi to napisali z regularnim izrazom ' $\backslash.*$ '. Tako smo izničili posebni pomen znakov pika (ujemanje kateregakoli znaka) in zvezdica (znak, ki se ponovi nič ali večkrat). Pomen in uporabo nadomestnih znakov bomo podrobneje spoznali v razdelku znakovni razred in znakovna skupina.

2.2.1 Znakovni razred

Včasih bi v regularnem izrazu želeli, da se na določenem mestu pojavi poljuben znak iz skupine znakov. V ta namen uporabimo znakovni razred.

Znakovni razred navedemo tako, da razredu pripadajoče znake zapišemo med oglate oklepaje. Predstavlja nam katerikoli znak med tistimi, ki so v oglatih oklepajih. Tako npr. v regularnem izrazu zapis **[abc]** pomeni, da se na tistem mestu lahko pojavi bodisi znak a, bodisi znak b, bodisi znak c. Regularnemu izrazu '**[J]asna**' torej ustreza tako niz "jasna" kot tudi "Jasna".

Najbolj razširjena znakovna razreda sta razred števk in razred črk. Razred števk bi lahko napisali kot **[0123456789]**. Podobno bi razred črk lahko napisali tako, da bi navedli celotno abecedo. Na srečo (predvsem zaradi preglednosti) ju lahko zapišemo tudi kot serijo znakov **[0-9]** oz. za črke angleške abecede **[a-z]** ali **[A-Z]**, odvisno od velikosti črk. Razrede lahko tudi poljubno združujemo. Tako znakovni razred '**[0-9a-zA-Z]**' pomeni, da se na tem mestu lahko pojavi katerakoli črka ali števka.

Poleg teh dveh vnaprej definiranih razredov lahko sestavljamo tudi svoje. Kadar želimo na primer določiti razred, ki ustreza ločilom: ' $.,?!$ ', zapišemo '**[.,?!]**'. Omenjeni zgled je zanimiv tudi zaradi tega, ker smo prej omenili, da pika ' $.$ ' pomeni nadomestni znak. Vendar kadar se znak pojavi znotraj oglatih oklepajev, vedno pomeni običajni znak. Torej bi regularni izraz, ki ustreza nizu " $.*$ ", lahko napisali tudi kot '**[.][*]**'. Seveda gre tu za "zlorabo" razreda, zato tako običajno ne pišemo. Opozorimo še, da se regularna izraza '**[.][*]**' in '**[.]***' razlikujeta. Prvemu ustreza le niz " $.*$ ", drugemu pa niza " $.$ " in " $.*$ ".

Znakovni razred '**[*+]**' ustreza trem osnovnim matematičnim operatorjem, čeprav v njem nastopa ' $*$ '. Kot smo ravno povedali, so znotraj '**[]**' vsi znaki običajni in ne nadomestni. No, kot pa smo že videli, tudi znotraj '**[]**' lahko nastopajo nadomestni znaki. Enega, namreč ' $-$ ', smo že uporabili, ko smo zapisali serijo znakov. Če bi torej v omenjeni razred želeli dodati še ' $-$ ', bi to morali napisati drugače. Kako, bomo pojasnili v nadaljevanju.

Če bi želeli napisati razred, ki bi se ujemal s črkami slovenske abecede (ne smemo pozabiti na šumnike), bi naš razred napisali kot **[a-žčšž]**. Vendar pa so v tem razredu tudi angleške črke q, y, w in x.

Razred lahko tudi zanikamo. To možnost uporabimo takrat, kadar bi radi povedali, da se določeni znaki na tem mestu ne smejo pojaviti. Tako se regularni izraz '**[^A-Z]**' ujema z vsemi znaki, ki niso velike črke (poleg malih črk in števk se s tem znakovnim razredom ujemajo tudi vsi ostali znaki).

Poglejmo še en zgled uporabe zanikanja. Razred '**[^0-9]**' se ujema z vsemi znaki, ki niso števke.

Če torej v znakovni razred takoj za oglatim oklepajem napišemo znak ' $^$ ', to pomeni, da se znaki, ki sledijo, ne smejo pojaviti. Tako zapis '**[^čšž]**' pomeni, da na tem mestu ne sme biti šumnika. Če pa se znak ' $^$ ' pojavi v "notranjosti" razreda, ima povsem običajni pomen. Tako '**[*%^]**' pomeni, da je na tem mestu bodisi znak ' $*$ ', bodisi znak ' $%$ ' ali pa znak ' $^$ '.

Znakovni razred '**[^^]**' se torej ujema z vsakim znakom, ki ni ' $^$ '.

Poleg nadomestnih znakov ' $-$ ' in ' $^$ ' imata v oglatih oklepajih poseben pomen tudi znaka ' $\backslash$ ' in ' $|$ '. Znak ' $\backslash$ ' je dovoljen, saj le prvi zaklepaj pomeni začetek razreda. Če se pojavi še kakšen oglati oklepaj, pa le-ta predstavlja sam znak ' $\backslash$ '. Poševnica nazaj, ki pomeni ubežni znak, ima tudi v razredu drug

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

pomen. Naznanja namreč začetek bližnjice za razrede. Zaklepaj pa je znak za konec znakovnega razreda.

Pri zapisu znakovnih razredov so torej nadomestni znaki tile:

- \ ubežni znak, nakazuje začetek bližnjice razreda,
- ^ (če je prvi znak za prvim '[') zanikan razred,
- nakazuje serijo znakov ter
-] konec znakovnega razreda.

Prej smo omenili bližnjice. Poznamo nekaj bližnjic za zapis znakovnih razredov. Tako lahko npr. razred števk namesto **[0-9]** zapišemo krajše z **\d** (digit). Navedimo najpogostejše bližnjice:

Bližnjica	Pomen
\w	znak črka in številka, ali znak '_'
\W	vsi znaki, ki niso črke, številke ali znak '_' (gre torej za "zanikanje" razreda \w)
\s	katerikoli nevidni znak, kot so presledek, tabulator, prehod v novo vrsto
\S	vsi znaki, razen nevidni znaki
\d	katerakoli številka
\D	katerikoli znak, ki ni številka
\i	poljubna velika ali mala črka

2.2.2 Skupine znakov

Skupina znakov, zapisana med okroglimi oklepaji, nam predstavlja zaporedje znakov. Kadar imamo podano skupino znakov, regularnemu izrazu ustreza celotno zaporedje znakov v oklepaju. Tako regularni izraz **'(pas)'** ustreza nizu "pas", ujema pa se na primer z nizom "pasma" ali pa "napasti".

Skupine znakov svoj pravi pomen dobijo v kombinaciji z nadomestnimi znaki. Tako regularni izraz **'(ab)*'** ustreza tistim nizom, ki so sestavljeni iz parov znakov a in b, torej "ab", "abab", "ababababababab" in podobno. Ustreza mu tudi prazni niz, saj znak '*' pomeni ujemanje nič ali večkrat.

Tudi v skupini znakov lahko uporabljamo nadomestne znake. Eden izmed nadomestnih znakov znotraj skupine je navpična črta, ki se uporablja za izbiro. Obnaša se kot logični operator ali. Denimo, da želimo poiskati ime tekstovne datoteke, ki ima končnico bodisi doc, bodisi txt. Zapišimo regularni izraz: **'[a-zA-Z]+\.(doc|txt)'**. Nadomestni znak '+' pomeni, da se znak pred njim pojavi enkrat ali večkrat. Skupina znakov **[a-zA-Z]** predstavlja ujemanje katerekoli črke angleške abecede, kombinacija razreda črk in znaka '+'. **[a-zA-Z]+** pa pomeni, da se mora pojaviti vsaj ena črka, lahko pa je črk (ne nujno enakih, seveda) tudi več. Temu zaporedju znakov mora slediti najprej pika '.', nato pa bodisi doc bodisi txt. S tem regularnim izrazom se ujemajo imena datotek "diplma.doc", "kazalo.txt", "aaa.txt", ne pa tudi "vaja.csv". Ujema pa se tudi niz "aa.txt1" ali pa "bla.docx". Seveda pa ta dva niza ne ustrezata regularnemu izrazu.

Zanimiv primer iskanja s pomočjo skupine znakov je iskanje korena besede. V določenem besedilu bi želeli poiskati besedo funkcija v vseh sklonih. Če bi v iskalniku uporabili regularni izraz **'(funkcij)(a|e|i|o)'**, bi ta našel besede "funkcija", "funkcije", "funkciji" in "funkcijo". Skupini znakov funkcij namreč lahko sledi katerikoli izmed znakov a, e, i in o.

Poglejmo si še en primer: v datoteki HTML želimo poiskati sklice na vse slike v dokumentu. Dovoljene končnice datotek s slikami v dokumentu so: jpg, bmp, gif, png. Regularni izraz oblike **'+\.(jpg|bmp|gif|png)'** se bo ujema z imeni slik "avto2.jpg", "hiša.bmp", ne pa tudi z "ladja.jpeg". Kot

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

smo povedali že prej, znak '+' določa količino pojavitve znaka. Znak pred njim se lahko pojavi enkrat ali večkrat. V nasprotju z '*' se mora ob uporabi '+' znak torej obvezno pojaviti vsaj enkrat.

2.3 Nadomestni znaki

Kot smo že opazili, se nadomestni znaki razlikujejo v odvisnosti od položaja, kjer se nahajajo. Znotraj znakovnega razreda uporabljamo drugačne nadomestne znake kot pa znotraj skupine znakov. Izven znakovnega razreda in skupine znakov pa uporabljamo tudi druge nadomestne znake, ki jih bomo podrobneje spoznali v nadaljevanju.

Z oklepaji smo se pri tvorbi regularnih izrazov že srečali. Še enkrat omenimo, da oglati oklepaji omejujejo znakovni razred, okrogli pa skupino znakov. Zavite oklepaje uporabljamo za označevanje določevalcev količine, ki jih bomo spoznali v naslednjem razdelku.

Pozorni moramo biti na znak strešico '^', ki nam znotraj znakovnega razreda predstavlja zanikan razred, zunaj razreda pa deluje kot sidro. Sidra so pomembna pri ujemanju nizov z regularnim izrazom. Sidro v regularnem izrazu nam določa, kje v nizu naj določen znak nastopa. Tako znak '^' pomeni, da se mora niz, ki se želi ujemati z regularnim izrazom, začeti tako, kot je navedeno v nadaljevanju (nadaljevanje regularnega izraza mora torej ustrezati začetku niza). Podobno znak '\$' pomeni konec niza. Kadar sta ta dva znaka (torej '^' in '\$') napisana na kakšnem drugem mestu, izgubita svoj posebni pomen in sta "običajna" znaka.

Dober primer za prikaz delovanja sidra je regularni izraz 'je'. Ta se ujema s poljubnim nizom, ki vsebuje "je" kjerkoli v nizu. Torej, kadar je besedica samostojna, na začetku, na koncu besede ali pa v sredini. Ujema se z besedo "jesen", "morje", "je". Regularni izraz '^je\$' pa se ujema le z nizom "je". Torej do ujemanja pride le, kadar je beseda samostojna in ni del večjega niza.

Poglejmo še en zgled. Regularni izraz '^j.*e\$' se ujema z nizi "jablane", "jokanje", "je", "jure", torej s poljubnim nizom, ki se začne z j in konča z e. Ne ujema pa se z nobenim nizom, ki se ne konča z e in ne začne z j.

Regularni izraz '^**(je)**\$' se ujema le z besedico "je". '^' na začetku regularnega izraza namreč pomeni, da se bo regularni izraz ujema le z nizom, ki se bo začel z zaporedjem črk "je". '\$' pa pravi, da se mora niz končati z zaporedjem črk "je". Vmes pa ne sme biti nobenega znaka, torej je ujema se niz le samostojni niz "je".

Regularni izraz '^**je.**+' se ujema z besedo "jezero". Obrnjena situacija bi bila, če bi želeli najti ujemanje dela besede "je" na koncu niza. Regularni izraz '**.+je\$**' bi se ujema z besedo "morje" ali "veselje".

Kadar želimo določiti začetek in konec niza, vmes pa je poljubno mnogo različnih znakov, zapišemo regularni izraz '^**je.*je\$**', ujema se z besedico "jemanje" ali pa z nizom "je26543je", ne pa tudi z besedo "listje".

Spomnimo se še enkrat, da znak pika '.' nadomesti katerikoli znak. .

Pogosto pa bomo uporabljali tudi znak '/'. Ta nam označuje začetek in konec regularnega izraza. Ko uporabljamo regularni izraz znotraj funkcije, moramo obvezno z znakom '/' označiti začetek in konec regularnega izraza. Tako bi regularni izraz '^**je.*je\$**', ki bi ga uporabili v funkciji zapisali med dva znaka '/': '/^**je.*je\$**/'. Poglejmo pregled nadomestnih znakov, ki nastopajo zunaj znakovnega razreda in znakovne skupine.

Nadomestni znak	Pomen
\	začetek bližnjice razreda
/	začetek in konec regularnega izraza
^	sidro, napoveduje začetek niza
\$	sidro, napoveduje konec niza
.	nadomesti katerikoli znak, razen poševnice nazaj
[	napoveduje začetek znakovnega razreda

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

]	napoveduje konec znakovnega razreda
	napoveduje možnost izbire
(	napoveduje začetek skupine znakov
)	napoveduje konec skupine znakov
{	naznanja začetek določevalcev količine
}	naznanja konec določevalcev količine

Določevalci količine nam določajo kolikokrat se določen znak, znakovni razred ali skupina znakov ponovi. Več o tem si bomo pogledali v naslednjem poglavju.

2.4 Določevalci količine

Določevalci količine nam določajo, kolikokrat se posamezni znak ali skupina znakov ponovi. Označimo jih z oklepaji '{}', ki omejujejo določevalce količine. Tako na primer '**a{3}**' pomeni, da se mora znak a ponoviti natanko trikrat.

Poleg takega zapisa za označevalce količine uporabljamo tudi posebne znake. Kot smo že omenili, znak '*', ki sledi znaku, pomeni nič ali več ponovitev znaka, skupine znakov ali razreda znakov. Zelo pogost določevalec količine je tudi nadomestni znak '+'. Ta je podoben '*', le da pomeni eno ali več ponovitev.

Poglejmo si nekaj primerov.

Regularni izraz '**[a-z]+**' ustreza eni ali več malim črkam. Niz "Evropa" se s tem regularnim izrazom seveda ujema. Pri tem regularni izraz opiše (ustreza) podniz "vropa".

Če bi želeli preveriti, ali v določenem nizu nastopa vsaj ena številka, bi preverili ujemanje niza z regularnim izrazom oblike: '**[0-9]+**' ali '**1d+**'. Nizi, ki se ujemajo s tem regularnim izrazom, so: "F11A88AD", "LAS5JM", "8542362". Niza "LKJAF" ali "MTQXD" pa se ne ujemata, ker nimata nobene števke.

Že prej smo povedali, da skupina '**(ab)***' ustreza nizom, ki so sestavljeni iz parov znakov a in b, torej "ab", "abab" pa tudi s praznim nizom ". Denimo, da namesto znaka zvezdica napišemo znak '+'. Regularni izraz '**(ab)+**' se ujema z nizom, ko imamo v nizu vsaj eno zaporedje znakov "ab", denimo z nizom "labradorec". Niz "acaaaa" pa se ne ujema s tem regularnim izrazom. A ta niz ("acaaaa") se z regularnim izrazom '**(ab)***' ujema, saj '*' pomeni, da se skupina ab pojavi nikoli ali večkrat.

Znak	Opis
*	nič ali več pojavitev
+	en ali več pojavitev
?	nič ali en pojavitev
{n}	n pojavitev
{, n}	do n pojavitev (med 0 in n)
{n, }	vsaj n pojavitev (n ali več)
{n1, n2}	število pojavitev mora biti med (vključno) n1 in (vključno) n2

Poglejmo si še kakšen primer z uporabo določevalcev količine. Preveriti želimo, ali niz, ki je podan, ustreza obliki zapisa za poštno številko v Sloveniji (imeti mora štiri števke). Regularni izraz '**1d{4}**' ustreza vsem nizom, ki imajo štiri števke. Seveda pa s tem nismo preverili, če taka poštna številka res obstaja. Tudi niz "0000" ustreza regularnemu izrazu, ki pa ni poštna številka v republiki Sloveniji.

V nizu naslov želimo poiskati del niza, ki ustreza obliki za hišno številko. Regularni izraz '**1d{1,3}**' ustreza vsem hišnim številkam, ki so ena-, dva- ali trimestne. Tudi niza "000" in "0" ustrezata temu

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

regularnemu izrazu, vendar pa to nista podatka, ki bi ustrezala hišni številki. Bolj pravilen zapis bi bil **'[1-9][0-9]{0,2}'** tako izključimo vse primere števil, ki imajo na prvem mestu števko nič. Temu regularnemu izrazu pa ustrezajo naslednji nizi: "2", "30", "500", "999" ...

Regularni izraz lahko pogosto napišemo na več načinov. Poglejmo si še kak zgled, kjer bomo napisali različne oblike regularnih izrazov, ki pa bodo ustrezali istim nizom in pri tem uporabili tudi označevalce količine.

Denimo, da želimo preveriti, ali določeni nizi predstavljajo datume. Predpostavimo, da morata biti dan in mesec obvezno dvomestni števili, leto pa štirimestno število. Ločilo med deli datuma je znak '-'. Želimo torej napisati regularni izraz, ki bi ustrezal datumom oblike: "dd-mm-llll", kjer d predstavlja števko za dan v mesecu, m števko za mesec v letu, ter y številke za obliko zapisa leta. Števke lahko nadomestimo z razredom števk **'[0-9]'**.

Če nas zanima le, ali niz vsebuje dovoljene znake, lahko zapišemo: **'[0-9-]{3}[0-9-]{3}[0-9]{4}'**, vendar temu regularnemu izrazu ustrezata tudi niza "--01232009" in "20--2-2009". Prav tako bi lahko napisali kar **'[0-9-]{10}'**. Zato bo potrebno regularni izraz napisati tako, da natančno določimo položaja vezajev. Napišemo ga lahko z bližnjico za razred števk, predvsem pa določimo mesto, kjer se lahko pojavi vezaj: **'\d\d\d-\d\d\d-\d\d\d\d'**, ali pa napišemo regularni izraz bolj enostavno: **'\d{2}\d{2}\d{4}'**. Regularni izraz lahko napišemo tudi drugače. Vendar pa to ne bo več isti regularni izraz, saj tukaj spremenimo obliko datuma. Denimo, da dovolimo zapis dneva in meseca kot enomestni ali dvomestni števili. Regularni izraz je sedaj oblike: **'\d{1,2}\d{1,2}\d{4}'**.

Če imamo podan niz oblike "25-12-2008", ustreza našemu regularnemu izrazu, datum oblike "25/12/2008" pa ne. S temi regularnimi izrazi seveda še ne zagotavljamo pravilnost "vsebine" datuma. Ustreza oblika, ne pa nujno vsebina. Tak primer bi bil niz "22-22-2222", ki ustreza gornjemu regularnemu izrazu, čeprav ta niz ni datum.

Poleg zgoraj naštetih določevalcev količine, ki jim pravimo tudi požrešni, poznamo tudi lene določevalce količine. Ker pa teh ne bomo rabili, jih na tem mestu samo omenimo. Bralec pa si lahko sam prebere kaj več o razliki med lenimi in požrešnimi določevalci količine.²

3 Podpora regularnim izrazom v PHP

Pogledali si bomo najpogostejše uporabljene funkcije za delo z regularnimi izrazi. Delimo jih v dva sklopa. V prvem so funkcije po standardu POSIX, v drugem pa funkcije PCRE.

Standard POSIX za regularne izraze je standard iz leta 1986. Od verzije 5.3.0 dalje so te funkcije v PHP-ju uradno označene kot "deprecated", kar pomeni, da naj se ne bi uporabljale več.

V drugi sklop spadajo funkcije PCRE (Perl Compatible Regular Expressions). To je kratica za regularne izraze združljive s Perl-om. Podporo za PCRE najdemo tudi v programskih jezikih Perl, Delphi in še kje drugje. Funkcije v knjižnici PCRE so podane po abecedi.

V funkcijah regularnega izraza bomo uporabili znak '/', ki napoveduje začetek ali konec regularnega izraza. Pišemo ga na začetku za narekovaji, na koncu pa pred narekovaji, kot na primer v **'/abc/'**. Zaradi boljše preglednosti regularnih izrazov prej tega znaka nismo pisali. Pri pisanju funkcij PCRE je znak '/' obvezen, saj brez tega znaka program ne bo zaznal začetka in konca regularnega izraza.

3.1 Preg_grep

² Vir: <http://www.softec.st/en/OpenSource/DevelopersCorner/RegularExpressions/Quantifiers.html#greedy>

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Funkcija **preg_grep** vrne tabelo tistih nizov, ki se ujemajo z danim regularnim izrazom. Funkcija ima dva obvezna vhodna parametra, in sicer regularni izraz in tabelo. V tabeli imamo shranjene nize, za katere ugotavljamo, ali se ujemajo z danim regularnim izrazom.

Poglejmo si primer. Med nizi "AB584", "QM684", "A2G45", "E259", "AN54" in "yyAB584xxxx" želimo poiskati tiste nize, ki vsebujejo podniz dve črki, ki jima sledijo tri števke. V ta namen sestavimo regularni izraz. Nize zložimo v tabelo. Če napišemo

```
<?php
$gesla = array("AB584 ", "A2G45", "E259", "AN54", "QM684", "yyAB584xxxx");
$zadetki = preg_grep("/[A-Z]{2}[0-9]{3}/", $gesla);
print_r($zadetki);
?>
```

dobimo izpis:

```
Array ( [0] => AB584 [4] => QM684 [5] => yyAB584xxxx )
```

ki pove, da je take oblike poleg nizov "AB584" in "QM684" tudi niz "yyAB584xxxx". Funkcija **preg_grep** vrne tabelo, ki smo jo izpisali s pomočjo funkcije **print_r**. Funkcija vrne indekse iz stare tabele.

Funkcija ima poleg obveznih parametrov lahko tudi izbirni parameter. Izbirni parameter zapišemo v obliki konstante **PREG_GREP_INVERT**. V tem primeru funkcija kot rezultat vrne tabelo tistih nizov, ki se ne ujemajo z regularnim izrazom.

```
<?php
$gesla = array("AB584 ", "A2G45", "E259", "AN54", "QM684", "yyAB584xxxx");
$zadetki = preg_grep("/[A-Z]{2}[0-9]{3}/", $gesla, PREG_GREP_INVERT);
print_r($zadetki);
?>
```

Kot smo omenili, funkcija vrne indekse iz stare tabele.

```
Array ( [1] => A2G45 [2] => E259 [3] => AN54 )
```

Kadar nimamo regularnega izraza in želimo poiskati nize, ki se ne ujemajo z določenim vzorcem, je včasih lažje sestaviti regularni izraz, ki opisuje vzorec in uporabiti izbirni parameter **PREG_GREP_INVERT**. Pogosto bi bilo težje sestaviti regularni izraz, ki opisuje nize, ki se ne ujemajo z danim vzorcem.

3.2 Preg_last_error

Funkcija **preg_last_error** vrne kodo napake zadnje funkcije PCRE, ki smo jo izvedli.

Kadar pri izvajanju katere izmed funkcij PCRE ni prišlo do napake, funkcija **preg_last_error** vrne konstanto **PREG_NO_ERROR**. S tem vemo, da je bila ta funkcija PCRE uspešno izvedena.

Poglejmo si, kako lahko to spremenljivko uporabimo. S pomočjo funkcije **preg_grep** smo želeli poiskati nize, ki zadostijo našim kriterijem. Hkrati pa nas zanima, ali se je funkcija izvedla brez napake.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Z stavkom `if` preverimo ali je prišlo do napake v zadnji funkciji `PCRE`. Najpogosteje to pomeni, da je v regularnem izrazu nekaj narobe. Preverili smo pravilnost funkcije **`preg_grep`** ter sintakso regularnega izraza, ki smo ga uporabili v funkciji.

V našem primeru nas zanima ali so nizi v tabeli zapisani v skladu s kodo UTF-8. V regularnem izrazu imamo zapisano oznako `u`, ki pomeni, da morajo biti nizi napisani v skladu s kodo UTF-8.

```
<?php
$stabela=array('~~~~~', 'abc');
preg_grep('/[a-z]/u',$stabela );

if (preg_last_error() == PREG_BAD_UTF8_ERROR) {
    print 'Napaka: PREG_BAD_UTF8_ERROR!';
}
else{
    print 'Nizi v tabeli so kodirani po standardu UTF-8!';
}
?>
```

Na zaslonu se nam izpiše ime napake, saj niz ni napisan v kodi UTF-8.

Napaka: PREG BAD UTF8 ERROR!

Funkcija **preg_last_error** vrne kodo ene izmed naslednjih napak:

PREG NO ERROR

Napake ni bilo, funkcija se je izvedla pravilno.

PREG BACKTRACK LIMIT ERROR

Prekoračeno je maksimalno število izvajanj povratnih korakov algoritma (backtracking). Backtracking v regularnem izrazu pomeni ponovni poskus ali ustreza del niza z regularnim izrazom pri naslednji možni izbiri. Do večjega števila izbir pride zaradi določevalcev količine ali pa ob uporabi skupin z izbirno možnostjo (znak "|").

PREG RECURSION LIMIT ERROR

Prekoračeno je maksimalno število klicev rekurzije (ko imamo v kodi klic rekurzije, na primer pri funkciji `preg_replace_callback`).

PREG BAD UTF8 ERROR

Niz ni napisan v kodi UTF-8.

PREG BAD UTF8 OFFSET ERROR

Tudi v tem primeru niz ni napisan v kodi UTF-8. Vendar pa bralca ne bomo obremenjevali s podrobnosti, saj za naše primere to ni bistvenega pomena.

PREG INTERNAL ERROR

Pri izvajanju je prišlo do težav, a napaka ni ena izmed zgoraj definiranih. To je privzeta vrednost, kadar pride do napake.

3.3 Preg match

Funkcija **preg_match** sprejme tri obvezne parametre: regularni izraz, niz in tabelo. Ugotovi, če se dani niz ujema z regularnim izrazom. Funkcija vrne vrednost tipa int in sicer število zadetkov. Če je niz, ki ustreza regularnemu izrazu, podniz danega niza, funkcija vrne 1, sicer pa 0. Funkcija išče samo do prvega podniza, ki ustreza regularnemu izrazu, tudi če je takih podnizov več, vrne število 1. Če pa takega podniza ni, vrne število 0. V tabelo pa funkcija shrani najden zadetek. Funkcija se izvaja tako,

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

da brž ko najde prvi ustrezen podniz, ustavi iskanje, ne glede na to, kolikokrat regularni izraz ustreza podnizom danega niza. Funkcija začne iskanje na mestu z indeksom 0, razen, če določimo drugače.

Poglejmo si zgled. V nizu, ki ga hranimo v spremenljivki `$pesem`, želimo preveriti, ali se kje pojavi črka 'd', ki ji sledi katerakoli mala črka slovenske ali angleške abecede. Najprej sestavimo regularni izraz: `'d[a-zčšž]'`.

Funkcija v tretji parameter, spremenljivko `$tabela` zapiše zadetek, ki ustreza danemu regularnemu izrazu. Prvi element tabele je prvi podniz, ki ustreza regularnemu izrazu, drugi pa indeks mesta, kjer se ta podniz začne.

```
<?php
    $pesem ="Naša četica koraka, strumno in veselo,
            drug za drugim v ravni vrsti, zdaj gremo na delo...";
    print preg_match('/d[a-zčšž]/', $pesem, $tabela);
    print "<br>";
    print_r ($tabela);
?>
```

Ker v danem nizu obstaja vsaj en podniz, ki ustreza regularnemu izrazu, funkcija vrne 1, izpisali pa smo tudi zadetek, ki ga je funkcija shranila v spremenljivko `$tabela`.

```
1
Array ( [0] => dr )
```

V nizu sedaj želimo preveriti še, ali se kje pojavi črka 'z'. To storimo z regularnim izrazom: `'/z/'`.

```
<?php
    $maja ="Nekoč, že dolgo je tega, v deželi, res
            ne vemo kje, živela je čebelica, imela lepo je ime...";
    print preg_match('/z/', $maja, $tabela);
?>
```

Tokrat pa funkcija vrne 0, ker črke 'z' v nizu pesem ne najde.

```
0
```

Kadar v izbirnem parametru uporabimo vrednost **PREG_OFFSET_CAPTURE**, začne funkcija iskanje na mestu, ki smo ga določili. Navedemo ga za besedo **PREG_OFFSET_CAPTURE**.

V nizu želimo poiskati črko 'd', ki ji sledi katerakoli črka angleške ali slovenske abecede. S pomočjo izbirnega parametra **PREG_OFFSET_CAPTURE** smo določili, da funkcija začne iskanje v nizu šele na mestu z indeksom 70.

```
<pre>
<?php
    $pesem ="Naša četica koraka, strumno in veselo,
            drug za drugim v ravni vrsti, zdaj gremo na delo...";
    print preg_match('/d[a-zčšž]+/', $pesem, $tabela, PREG_OFFSET_CAPTURE, 70);
    print "<br>";
    print_r ($tabela);
?>
</pre>
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Pri izpisu vrednosti funkcije dobimo izpis števila 1, saj se v nizu špesem desno od določenega zamika 70 nahaja vsaj en podniz, ki ustreza regularnemu izrazu.

```
1
Array
(
    [0] => Array
        (
            [0] => daj
            [1] => 86
        )
)
```

Ker je določevalc količine '+' grabežljiv funkcija vrne celoten podniz "daj", ki ustreza regularnemu izrazu.

3.4 Preg_match_all

Za razliko od funkcije **preg_match** nam ta funkcija vrne število vseh podnizov, ki ustrezajo regularnemu izrazu. Funkcija **preg_match_all** sprejme regularni izraz, niz, ki ga pregledujemo, in spremenljivko tipa tabela, v katero shrani zadetke.

Zanima nas, kolikokrat se v nizu pojavi datum. Oblika datuma je v tem primeru točno določena kot: "dd.mm.llll". Deli datuma (dan, mesec in leto) so med sabo ločeni z znakom pika, tako dan kot mesec pa nastopata izključno v obliki dvomestnega števila. Podniz, ki ustreza obliki za datum poiščemo s pomočjo regularnega izraza: `'/(d{2}\.d{2}\.)(d{4})/'`. Ker sta podniza, ki nas zanimata dva, en podniz je sestavljen iz dneva in meseca, drugi podniz pa je leto, zapremo podnize v posamezne skupine. Tako bo funkcija shranila posamezne podnize glede na skupine, ki smo jih določili v regularnem izrazu.

```
<pre>
<?php
    $prazniki = "Novo leto: 01.01.2008, kulturni praznik: 08.02.2008,
        Velika noč: 22.4.2008, praznik dela: 1.5.2008";
    $vzorec = "/(d{2}\.d{2}\.)(d{4})/";
    $stevaloDatumov = preg_match_all($vzorec, $prazniki, $datumi);
    print "Regularni izraz se v nizu prazniki ujema: $stevaloDatumov -krat.";
    print "<br>";
    foreach ($datumi as $zadetek){
        print_r ($zadetek);
    }
?>
</pre>
```

Funkcija **preg_match_all** ima tudi četrti, izbirni, parameter - način. Z njim določimo, v kakšnem zaporedju bo funkcija vrnila podnize, ki ustrezajo regularnemu izrazu. Izbirni parameter ima lahko naslednje vrednosti: **PREG_PATTERN_ORDER**, **PREG_SET_ORDER** ali **PREG_OFFSET_CAPTURE**.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

V našem zgledu dobimo izpisane vse zadetke, ki jih funkcija **preg_match_all** najde. Če namreč načina ne navedemo, se se upošteva privzeta vrednost izbirnega parametra, ki je **PREG_PATTERN_ORDER**

```
Regularni izraz se v nizu prazniki ujema: 2 -krat.  
Array  
(  
    [0] => 01.01.2008  
    [1] => 08.02.2008  
)  
Array  
(  
    [0] => 01.01.  
    [1] => 08.02.  
)  
Array  
(  
    [0] => 2008  
    [1] => 2008  
)
```

Parameter **PREG_PATTERN_ORDER** shrani celoten niz v eno tabelo ter tudi vsako skupino podnizov v svojo tabelo. Skupino podnizov smo določili v regularnem izrazu, ko smo jih izolirali vsak del v svojo skupino. Tako funkcija shrani v prvo tabelo celoten niz, v drugo tabelo prvi podniz (iz prve skupine), v tretjo tabelo shrani drugi podniz (drugo skupino) ...

Če skupin ne bi določili in uporabili

```
$vzorec = "/d{2}\\d{2}\\d{4}/";
```

pa bi dobili naslednji izpis.

```
Regularni izraz se v nizu prazniki ujema: 2 -krat.  
Array  
(  
    [0] => 01.01.2008  
    [1] => 08.02.2008  
)
```

Tako funkcija vrne samo niz, ki ustreza celotnemu regularnemu izrazu, ne pa tudi podnizov (ki smo jih ločili v skupine).

PREG_OFFSET_CAPTURE

Kadar ima parameter način vrednost nastavljeno na **PREG_OFFSET_CAPTURE**, funkcija **preg_match_all** poleg zadetka vrne tudi mesto, kjer se le-ta nahaja.

```
$steviloDatumov = preg_match_all($vzorec, $prazniki, $datumi, PREG_OFFSET_CAPTURE);
```

Kadar določimo izbirnemu parametru način vrednost **PREG_OFFSET_CAPTURE**, le-ta shrani najprej zadetek, poleg pa mesto, na katerem je bil zadetek najden. Vrstni red shranjevanja je enak kot pri **PREG_PATTERN_ORDER**, le da imamo tukaj tudi mesto, kjer se podniz nahaja v prvotnem nizu.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
Regularni izraz se v nizu prazniki ujema: 2 -krat.
Array
(
    [0] => Array
        (
            [0] => 01.01.2008
            [1] => 11
        )
    [1] => Array
        (
            [0] => 08.02.2008
            [1] => 41
        )
)
Array
(
    [0] => Array
        (
            [0] => 01.01.
            [1] => 11
        )
    [1] => Array
        (
            [0] => 08.02.
            [1] => 41
        )
)
Array
(
    [0] => Array
        (
            [0] => 2008
            [1] => 17
        )
    [1] => Array
        (
            [0] => 2008
            [1] => 47
        )
)
```

PREG_SET_ORDER

Kadar izbirnemu parametru način določimo vrednost **PREG_SET_ORDER**, pa funkcija shranjuje zadetke v drugačnem vrstnem redu. V prvi tabeli so shranjeni nizi in podnizi prvega zadetka, v drugi tabeli so shranjeni vsi nizi in podnizi drugega zadetka ...

```
$steviloDatumov = preg_match_all($vzorec, $prazniki, $datumi, PREG_SET_ORDER);
```

```
Regularni izraz se v nizu prazniki ujema: 2 -krat.
Array
(
    [0] => 01.01.2008
    [1] => 01.01.
    [2] => 2008
)
Array
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
(  
    [0] => 08.02.2008  
    [1] => 08.02.  
    [2] => 2008  
)
```

Parameter način ima lahko vrednosti **PREG_PATTERN_ORDER**, **PREG_SET_ORDER** in **PREG_OFFSET_CAPTURE**. Lahko pa je tudi kombinacija vrednosti **PREG_SET_ORDER** in **PREG_OFFSET_CAPTURE**.

Načini so uporabni kadar želimo podnize grupirati po skupinah.

Na primer da imamo podatke o določenih izdelkih (naziv, količina in cena). Če bi želeli gupirati po izdelkih bi uporabili način **PREG_SET_ORDER**. Tako bi dobili najprej vse podatke o izdelku 1, nato vse podatke o izdelku 2 ... Če pa bi želeli grupirati po posameznih kategorijah: naziv izdelka 1, naziv izdelka 2 ..., količina izdelka 1, količina izdelka 2 ..., cena izdelka 1, cena izdelka 2 ... bi uporabili način **PREG_PATTERN_ORDER**.

3.5 Preg_quote

Denimo, da imamo nek niz. V njem se pojavljajo tudi znaki, ki v sklopu regularnega izraza pomenijo nadomestne znake. Da nam jih ni potrebno "ročno" pretvarjati (prednje vrinemo ubežni znak), uporabimo funkcijo **preg_quote**. Ta sprejme nek niz in vrne drug niz, v katerem je pomen nadomestnih znakov izničen. Funkcija nam pride prav, kadar želimo dobljeni niz uporabiti kot regularni izraz ali del regularnega izraza v neki drugi funkciji. Nadomestni znaki, ki jih funkcija **preg_quote** prepozna, so: `.\+ : * ? [ ^ ] $ ( ) { } = ! < > | '`. Vendar pa nekaterih nadomestnih znakov nismo spoznali.

Izbirni parameter funkcije **preg_quote** je ubežni znak - ločilo. Funkcija ta znak vstavi pred vsak nadomestni znak v regularnem izrazu. Ta znak poskrbi, da izgine posebni pomen nadomestnega znaka. Privzeta vrednost ločila je `'\'`.

```
<?php  
$url_naslov = "http://www.domena.com?id=43";  
$citac = preg_quote($url_naslov, '/');  
echo "Citiran url naslov: $citac\n";  
?>
```

Funkcija **preg_quote** izniči pomen nadomestnih znakov. S funkcijo **echo** pa izpišemo rezultat funkcije:

```
Citiran url naslov: http:\/\/www\.domena\.com\?id\=43
```

3.6 Preg_replace

Funkcija **preg_replace** omogoča, da v nizu poiščemo podniz, ki ustreza regularnemu izrazu. Tega nato zamenjamo z novim podnizom.

Zaradi lažjega razumevanja bomo pri funkciji **preg_replace** regularni izraz poimenovali vzorec.

Niz, ki bo nadomestil vzorec, pa bomo imenovali zamenjava. Funkcija tako sprejme tri parametre:

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

vzorec (regularni izraz), zamenjavo in niz, v katerem iščemo. Funkcija vrne nov, spremenjen niz, v katerem zamenja vse zadetke, ki jih je našla.

Poglejmo si preprost primer. V nizu želimo zamenjati besedo obiskovalci z besedo gostje.

```
<?php
    $niz = 'Dragi obiskovalci!';
    $vzorec = '/obiskovalci/';
    $zamenjava = 'gostje';
    print preg_replace($vzorec, $zamenjava, $niz);
?>
```

Na ekranu se nam pojavi izpis:

```
Dragi gostje!
```

Lahko pa imamo namesto niza vzorec in niza zamenjava tudi tabelo (tabela vzorec, tabela zamenjava). V tem primeru delamo naenkrat več zamenjav, i-ti niz v tabeli vzorec zamenjamo z i-tim znakom v tabeli zamenjava.

Podan imamo niz recept. V njem so števila napisana s črkami, želimo pa jih zapisati s števki.

```
<?php
    $recept = "Biskvit: pet jajc, pet žlic tople vode, deset žlic
               sladkorja, osem žlic moke, en pecilni prašek, en
               vaniljev sladkor";

    $besede[0] = '/pet/';
    $besede[1] = '/deset/';
    $besede[2] = '/osem/';
    $besede[3] = '/en/';
    $stevilke[0] = '5';
    $stevilke[1] = '10';
    $stevilke[2] = '8';
    $stevilke[3] = '1';
    echo preg_replace($besede, $stevilke, $recept);
?>
```

Tako dobimo nov izpis niza recept.

```
Biskvit: 5 jajc, 5 žlic tople vode, 10 žlic sladkorja, 8 žlic moke, 1
pecilni prašek, 1 vaniljev sladkor.
```

Seveda nima smisla, da bi funkcijo uporabljali na ta način. Hitreje bi bilo, če bi uporabili kar funkcijo **str_replace**, ki smo jo spoznali v razdelku 2.4.2.

Pravo uporabnost dobi funkcija takrat, ko več podnizov ustreza regularnemu izrazu. Poglejmo si primer, ko želimo v nizu \$stariDatumi zamenjati vsakršno pojavitev leta v 20. stoletju kot tudi leta od 2000 do 2008 z vrednostjo letošnjega leta (2009).

```
<?php
    $stariDatumi = '15.04.2000, 18.06.2008, 12.12.1995, 04.12.2004';
    $vzorec = '/19[0-9]{2}|200[0-8]/';
    $zamenjava = '2009';
    print "Stari datumi: ".$stariDatumi;
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
$noviDatumi = preg_replace($vzorec, $zamenjava, $stariDatumi);  
print "<br>";  
print "Novi datumi: ".$noviDatumi;  
?>
```

Vsi štirje datumi ustrezajo regularnemu izrazu, zato dobimo kar štiri zamenjave.

```
Stari datumi: 15.04.2000, 18.06.2008, 12.12.1995, 04.12.2004  
Novi datumi: 15.04.2009, 18.06.2009, 12.12.2009, 04.12.2009
```

Kadar funkcija ne najde zadetkov, vrne nespremenjeno kopijo niza. Funkcija **preg_replace** pa ima tudi izbirni parameter. Ta parameter navzgor omeji število zamenjav, ki naj jih izvede funkcija. Če je število vzorcev večje, kot je dovoljeno število zamenjav, vzorci, ki so odveč, ostanejo nespremenjeni. Če bi pri izvajanju funkcije prišlo do napake, npr. če bi bila nepravilna sintaksa regularnega izraza, bi funkcija vrnila null.

Če bi vrstico spremenili in dodali parameter število zamenjav

```
$noviDatumi = preg_replace($vzorec, $zamenjava, $stariDatumi, 2);
```

bi dobili samo dve zamenjavi (kot smo to določili z zadnjim parametrom):

```
Stari datumi: 15.04.2000, 18.06.2008, 12.12.1995, 04.12.2004  
Novi datumi: 15.04.2009, 18.06.2009, 12.12.1995, 04.12.2004
```

3.7 Preg_replace_callback

Funkcija **preg_replace_callback** je nadgradnja funkcije **preg_replace**. Prav tako tudi ta funkcija najden podniz zamenja z novim nizom. A v nasprotju z "običajno" funkcijo, kjer povemo, kateri niz naj nadomesti najden podniz, tu nov niz dobimo kot rezultat klica neke druge funkcije, katere ime podamo kot parameter funkcije **preg_replace_callback**.

Argumenti, ki jih funkcija sprejme, so torej regularni izraz, funkcija ter niz. V nizu s pomočjo regularnega izraza iščemo ustrezne podnize, ki jih s pomočjo funkcije nato dodatno "predelamo". Tako kot **preg_replace** lahko tudi tu kot zadnji parameter uporabimo tabelo nizov. Funkcija vrne niz oz. tabelo, odvisno kaj ima funkcija **preg_replace_callback** za vhodni parameter. Kadar zadetka ne najde, nam funkcija vrne nespremenjen niz.

Ko regularni izraz najde ustrezen podniz, se pokliče funkcija, ki jo podamo kot parameter. Ta kot argument sprejme tabelo, v kateri so zapisani podatki o nizu, ki ga spreminjamo.

Prvi element tabele ustreza celotnemu regularnemu izrazu. Drugi element tabele ustreza prvi skupini v regularnem izrazu, tretji element ustreza drugi skupini v regularnem izrazu ... Če v regularnem izrazu nimamo skupin, funkcija vrne prazno tabelo.

Poglejmo si zgled. V nekem besedilu imamo na primer datume v obliki dan.mesec.leto. Besedilo želimo preoblikovati tako, da bomo leto v datumu povečali za ena.

Funkcija **preg_replace_callback** v nizu \$prazniki zamenja ustrezen podniz s pomočjo regularnega izraza in funkcije **prihodnjeLeto**.

Funkcija **prihodnjeLeto** dobi kot parameter tabelo nizov. Kot smo omenili zgoraj je niz na mestu ena v tabeli, ki jo funkcija **prihodnjeLeto** dobi kot argument dan, niz na mestu dva je mesec, na mestu tri pa se nahaja leto. Niz na tretjem mestu spremenimo tako, da nanj gledamo kot na število in ga povečamo za ena.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
<pre>
<?php
    $prazniki = "Zadnji kulturni praznik je bil: 08.02.2008.
    Naslednji dan osamosvojitve bo: 25.06.2009.";
    function prihodnjeLeto($zadetki){
        return $zadetki[1].$zadetki[2].($zadetki[3]+1);
    }
    echo preg_replace_callback('/(\d{2}\.)(\d{2}\.)(\d{4})/', 'prihodnjeLeto', $prazniki);
?>
</pre>
```

Izpis:

```
Zadnji kulturni praznik je bil: 08.02.2009.
Naslednji dan osamosvojitve bo: 25.06.2010.
```

3.8 Preg_split

Funkcija **preg_split** s pomočjo regularnega izraza razbije izvorni niz na podnize. Funkcija vrne kot izhodni podatek tabelo. Tabela vsebuje podnize, ki so deli izvornega niza, razdeljeni glede na ločilo, ki smo ga določili v regularnem izrazu. Funkcija **preg_split** sprejme dva vhodna parametra: regularni izraz, ki bo uporabljen kot ločilo in izvorni niz. Regularni izraz vsebuje vsa ločila, ki jih želimo uporabiti za razbitje niza (lahko imamo tudi več kot eno ločilo).

Oglejmo si primer. Imamo internetni poštni naslov, ki ga želimo razbiti na sestavne dele. Zanima nas lokalni del in domena ter sestavni deli lokalnega dela in domene. Lokalni del internetnega poštnega naslova so znaki, ki se nahajajo pred znakom '@', domena pa stoji za tem znakom. V znakovnem razredu določimo ločila, kjer bomo izvorni niz razbili na podnize. V internetnem poštnem naslovu sta ti dve ločili znak '.' in znak '@'.

```
<pre>
<?php
    $niz = 'mici.novak@enmail.com';
    $znaki = preg_split('/[.@]/', $niz);
    print_r($znaki);
?>
</pre>
```

Funkcija nam vrne sestavne dele elektronskega poštnega naslova.

```
Array
(
    [0] => mici
    [1] => novak
    [2] => enmail
    [3] => com
)
```

Tretji izbirni parameter, meja, nam omogoča, da omejimo število zadetkov. Kadar je meja določena, funkcija vrne samo toliko podnizov, kot smo jih določili v parametru meja.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Če bi torej v prejšnjem zgledu napisali:

```
$znaki = preg_split('/[. @]/', $niz, 2);
```

bi bil izpis:

```
Array
(
    [0] => mici
    [1] => novak@enmail.com
)
```

Kadar je meja nastavljena na vrednost -1 , funkcija vrne vse zadetke.

PREG_SPLIT_OFFSET_CAPTURE

Kadar imamo določen izbirni parameter **PREG_SPLIT_OFFSET_CAPTURE**, funkcija vrne tabelo tabel, ki vsebuje podnize in indekse. Ta podniz je izvorni niz razdeljen glede na ločilo (ločila), ki smo ga določili v regularnem izrazu. Vsak zadek posebej je tabela, ki vsebuje v enem polju del niza, v drugem pa indeks. Indeks pa pove, kje se je zadek nahajal v prvotnem nizu.

Poglejmo primer, kjer funkcije lahko poleg zadetkov pove tudi indeks mesta, kjer so bili zadetki najdeni.

```
<pre>
<?php
    $niz = 'mici.novak@enmail.com';
    $znaki = preg_split('/\.[ @]/', $niz, -1, PREG_SPLIT_OFFSET_CAPTURE);
    print_r($znaki);
?>
</pre>
```

```
Array
(
    [0] => Array
        (
            [0] => mici
            [1] => 0
        )
    [1] => Array
        (
            [0] => novak
            [1] => 5
        )
    [2] => Array
        (
            [0] => enmail
            [1] => 11
        )
    [3] => Array
        (
            [0] => com
            [1] => 18
        )
)
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

)

Na kratko povzemimo funkcije PCRE:

Funkcija	Uporaba
Preg_grep	Poišče in vrne zadetke, ki se ujemajo z regularnim izrazom.
Preg_grep s parametrom PREG_GREP_INVERT	Vrne zadetke, ki ne se ne ujemajo z regularnim izrazom.
Preg_last_error	vrne kodo zadnje napake, ki je nastala pri klicih funkcij iz PCRE knjižnice.
Preg_match	Če najde zadetek vrne 1, sicer 0.
Preg_match_all	Vrne vse najdene zadetke.
Preg_match_all s parametrom PREG_OFFSET_CAPTURE	Vrne vse zadetke in indeks mesta, kjer se zadetki nahajajo.
Preg_quote	Vrne regularni izraz, v katerem izničimo pomen nadomestnih znakov.
Preg_replace	Zamenja del niza z novim nizom.
Preg_replace_callback	S pomočjo druge funkcije zamenja del niza v obstoječem nizu z novim delom niza
Preg_split	Razdeli niz na več podnizov.
Preg_split s parametrom PREG_SPLIT_OFFSET_CAPTURE	Vrne razdeljene nize in indeks mesta, kjer so se ti posamezni nizi nahajali.

4 Primeri

V tem razdelku si bomo ogledali uporabo regularnih izrazov na konkretnih primerih. Ogledali si bomo, kako se preverja internetni poštni naslov, preverjanje podatkov v vnosnih poljih, zamenjavo dela besedila z drugim, odstranjevanje komentarjev, pa tudi odstranjevanje podvojenih presledkov. Presledke bomo v besedilu označevali z znakom '_', v kodi pa samo s praznim prostorom.

Najpogostejša uporaba regularnih izrazov je ujemanje (iskanje) dela niza v večjem nizu, pogosto pa tudi ustrežanje veljavnosti določenega niza (npr. če niz zadošča kriterijem za internetni poštni naslov, kar si bomo kot primer ogledali kasneje). Vendar pa moramo biti pri sestavljanju regularnega izraza pozorni. Pogosto nek niz sicer ustreza regularnemu izrazu, a ne predstavlja veljavnega podatka. Poglejmo si preprost primer, ko niz sicer ustreza regularnemu izrazu, vendar pa "vsebina" niza ni pravilna. Išče niz števk, ki ustrezajo dnevom v mesecu. Veljavna so števila med 1 in 31 vključno z mejami. Zapis lahko predstavlja enomestno (za dneve do desetega dne v mesecu) ali dvomestno število. Dan, ki je lahko zapisan kot enomestno število, lahko zapišemo kot dvomestno število, če dodamo spredaj številko nič: 02,05,07. Recimo, da sestavimo regularni izraz kar z bližnjico za številke 'd' ter določevalcem količine '{2}', torej '**d{2}**'. Vendar temu regularnemu izrazu ustreza poljubno dvomestno število, tudi na primer 65, 91 in 00. Dnevi v mesecu morajo biti večji od nič in hkrati manjši ali enaki 31. Zato bo pravilni regularni izraz bolj zapleten. Uporabili bomo kombinacijo znakovnega razreda in skupine znakov: '**(0?[1-9] | [12][0-9] | 3 [01])**'. Ta regularni izraz sedaj ustreza vsem številom med 1 in 31.

4.1 Zamenjava dela besedila z drugim

V besedilu se pogosto uporablja beseda direktorij, ki jo je pisec večkrat napisal tudi slovnično nepravilno (direktori, Direktori). Ker želimo popraviti zmedo v besedilu in tujko direktorij zamenjati z lepo slovensko besedo imenik, si bomo pri tem pomagali s funkcijo **preg_replace**.

Poglejmo si поблиžje regularni izraz: `'/[dD]irektori(j)?/'`. Skupina `[dD]` najde ustrezno črko, tako malo, kot veliko d. Zadnji del regularnega izraza `(j)?` pa pomeni, da se črka j pojavi nikoli ali pa enkrat. Tako nam regularni izraz najde katerokoli izmed spodnjih oblik zapisa besede direktorij: direktori, Direktori, direktorij in Direktorij.

```
<?php
$ niz = 'Na lokalnem disku C naredimo nov direktorij, ki ga
      poimenujemo vaje. Nato v ta direktori shranimo vaje, ki smo jih
      napisali danes, Direktori vaje nato kopiramo na disk U.';
$ vzorec = '/[dD]irektori(j)?/';
$ zamenjava = 'imenik';
echo preg_replace($ vzorec, $ zamenjava, $ niz);
?>
```

Sedaj se namesto vsake izmed oblik besede direktorij pojavi beseda imenik:

```
Na lokalnem disku C naredimo nov imenik, ki ga poimenujemo vaje. Nato
v ta imenik shranimo vaje, ki smo jih napisali danes, imenik vaje nato
kopiramo na disk U.
```

To pa bi se seveda dalo rešiti tudi z osnovnimi funkcijami za delo z nizi (**str_replace**).

4.2 Odstranjevanje internetnega poštnega naslova

Denimo, da imamo neko besedilo, ki lahko vsebuje tudi poštni naslov. Iz besedila želimo ta internetni poštni naslov odstraniti. Predpostavimo, da se v besedilu nahajajo samo veljavni internetni poštni naslovi. Zato bo regularni izraz za iskanje internetnih poštnih naslovov bolj preprost kot je običajno. Iskali bomo naslove, ki imajo obliko podobno kot: "imeprimek@enmail.com" ali pa "vijavaja2@gmail.com". Napišimo najprej pravila, ki definirajo za nas pravičen poštni naslov.

- Vsak poštni naslov je sestavljen iz lokalnega dela in domene. Vmes je znak '@'. Naslov je torej oblike lokalniDel@domena.
- Presledek ni dovoljen znak ne v lokalnem delu in ne v domeni.
- V lokalnem delu so dovoljeni naslednji znaki: velike in male črke in števke.
- Znak '.' ne sme biti nikoli na začetku ali koncu internetnega poštnega naslova, prav tako pa ne sme biti takoj pred in takoj za znakom '@'.
- Dolžina lokalnega dela je od 2 do 63 znakov.
- V domeni moramo imeti natanko en znak '.'. Ostali dovoljeni znaki so še male in velike črke.
- Prvi del domene smatramo tisti del, ki se pojavi takoj za znakom '@' do znaka '.', drugi del pa stoji za znakom '.' ter se konča na koncu domene.
- Dolžino domene pa smo določili: prvi del od 2 do 30 znakov, drugi del pa od 2 do 8 znakov.

```
<?php
$ niz = "Iz varnostnih razlogov želimo iz besedila odstraniti internetni poštni naslov.
      Uporabnik nam je posredoval naslednji
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
naslov: nekdosfmfja@email.com. Izbrisali smo ga s pomočjo funkcije
preg_replace.";
$svzorec = '/[a-zA-Z0-9]{2,63}@[a-zA-Z]{2,30})\.[a-zA-Z]{2,8})/';
$zamenjava = "";
echo preg_replace($svzorec, $zamenjava, $niz);
?>
```

Z delom regularnega izraza pred znakom '@': `'/[a-zA-Z0-9]{2,63}'` smo poskrbeli za ujemanje dovoljenih znakov. V lokalnem delu imamo lahko samo črke angleške abecede in števke. Drugi del regularnega izraza pa nam dovoljuje samo črke, ki jih mora obvezno ločevati znak '.' (`'([a-zA-Z]{2,30})\.[a-zA-Z]{2,8})'`). Ker iščemo samo najbolj pogoste oblike internetnih poštne naslovov, v prvem delu domene dovolimo najmanj dve ter največ trideset črk angleške abecede, v drugem delu pa od dve do osem črk angleške abecede. Med njima mora seveda biti znak '.'.

Iz besedila želimo odstraniti internetni poštne naslov iz varnostnih razlogov. Uporabnik nam je posredoval naslednji naslov: Izbrisali smo ga s pomočjo funkcije `preg_replace`.

Če pa bi namesto zgornjega niza v internetni poštne naslov napisali dve zaporedni piki, bi funkcija vrnila drugačen rezultat.

```
$niz = "Iz varnostnih razlogov želimo iz besedila odstraniti internetni poštne naslov.
Uporabnik nam je posredoval naslednji
naslov: nekdo..sfmfja@email.com. Izbrisali smo ga s pomočjo funkcije
preg_replace.";
```

V tem primeru pa dobimo izpis:

Iz varnostnih razlogov želimo iz besedila odstraniti internetni poštne naslov. Uporabnik nam je posredoval naslednji naslov: nekdo... Izbrisali smo ga s pomočjo funkcije `preg_replace`.

4.3 Preverjanje veljavnosti internetnega poštne naslova

Internetni poštne naslov je lokacija, kamor lahko pošljemo elektronsko sporočilo (email). Kot smo že omenili v razdelku 4.1, je sestavljen iz treh glavnih delov: lokalni del in domeno loči znak '@'.

Kadar pišemo regularne izraze, kjer se ukvarjamo z internetnimi poštne naslovi, v grobem ločimo dva primera.

Imamo internetne poštne naslove, za katere vemo, da so točno določene oblike, npr. "ime.priimek@student.fmf.uni-lj.si", ali pa "test@test.si". V tem primeru se nam ni potrebno ukvarjati s poljubno možno obliko internetnega poštne naslova, ampak samo z neko podmnožico internetnih poštne naslovov.

Drugače pa je, če želimo napisati regularni izraz, ki dovoljuje poljuben internetni poštne naslov, le da ta ustreza standardom za internetni poštne naslov.

Preverjanje veljavnosti poljubnega naslova je bolj zapleteno. Regularni izraz bomo zapisali drugače, saj mora internetni poštne naslov zadostiti določenim pogojem.

Primer internetnega poštne naslova:

Internetni poštne naslov "jasna.kobal@student.fmf.uni-lj.si" je sestavljen iz lokalnega dela (jasna.kobal), znaka '@' ter domene (student.fmf.uni-lj.si). V domeni mora biti vsaj ena pika.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Poglejmo si še dovoljene znake. V lokalnem delu so dovoljeni znaki: `-_a-z0-9.\'+*${^&%=~!}{}` V domeni pa so dovoljeni znaki samo: `a-z0-9.-`. Pogoji, ki je skupen domeni in lokalnemu delu je, da presledek ni dovoljen znak. To pomeni, da je naslov skupina znakov, ki se morajo držati skupaj.

Poglejmo si glavne pogoje za veljavnost internetnega poštnega naslova:

- naslov mora imeti natanko en znak '@'
- dolžini lokalnega dela in domene morata biti znotraj mej (lokalni del: 1-64, domena: 1-255)
- dovoljeni znaki v lokalnem delu so: `-_a-z0-9.\'+*${^&%=~!}{}`
- dovoljeni znaki v domeni so: `a-z0-9.-`
- v domeni mora biti vsaj en znak '.'.
- tako v lokalnem delu kot tudi v domeni ne smemo imeti več zaporednih pojavitev znaka '.'.

Splošno pravilo za internetni poštni naslov je zelo zapleteno³. Zgoraj smo navedli samo nekaj glavnih pravil. Ob upoštevanju teh pravil, lahko sestavimo internetni poštni naslov, ki ni povsem pravilen, ampak je teh izjem zelo malo. Da ne bi preveč zapletali, jih ne bomo navajali (in s tem tudi ne upoštevali pri sestavljanju regularnega izraza).

Zavedati se moramo, da nam noben regularni izraz ne bo preveril, ali tak naslov res obstaja ("afna@afna.afna" bi bil veljaven naslov, ki pa ne obstaja). Prav tako bo lahko veljaven naslov tisti, ki ima pravilno domeno, vendar pa je lokalno ime izmišljeno.

Ali nek niz predstavlja veljaven (glede na zgoraj navedena pravila) internetni poštni naslov, bomo preverjali s programom, v njem pa bomo uporabili funkcijo za delo z regularnimi izrazi, to je funkcija **preg_match**. Kot smo že omenili, je lokalni del internetnega poštnega naslova dolžine med 1 in 64 znakov. Domena pa je lahko dolga največ 255 znakov.

Najprej preverimo, da dolžina lokalnega dela ter domene zadostita pogojem. Istočasno preverimo tudi, da imamo natanko en znak '@'. Poštni naslov razbijemo na dva dela, domeno in lokalni del ločimo z znakom '@'. Denimo, da imamo internetni poštni naslov "jasna.kobal@student.fmf.uni-lj.si". Tako dobimo v lokalnem delu na primer "jasna.kobal", v domeni pa "student.fmf.uni-lj.si". Naslednji korak je razbitje lokalnega dela na več nizov, ki smo jih ločili z znakom pika. Sedaj napišemo razred, v katerem so shranjeni vsi znaki, ki so dovoljeni v lokalnem delu: `'[A-Za-z0-9!#$%&*+\/=?^_`{}~\']'`, torej male in velike črke angleške abecede, števke in znaki `'!#$%&*+\/=?^_`{}~\']'`. Za vsak del v lokalnem delu posebej preverimo ali ustreza regularnemu izrazu.

Nato pa v vsakem posameznem podnizu preverimo dovoljene znake (med njimi ne sme biti praznega niza). Če je v nizu več zaporednih pik, dobimo pri razbitju prazne nize. Ko smo preverili vse podnize v lokalnem delu, se prepričamo, da tudi domena ustreza pogojem, ki smo jih določili v regularnem izrazu. Tako kot lokalni del tudi domeno ločimo na več nizov, ki jih ločuje znak '.'. Število nizov v domeni mora biti vsaj dva ali več. Z znakovnim razredom `[A-Za-z0-9-]` preverimo nize v delu, kjer se nahaja domena. Kadarkoli del internetnega poštnega naslova ne ustreza regularnemu izrazu, vrne funkcija logično vrednost false. Funkcija na koraku, kjer del niza ne ustreza regularnemu izrazu, konča z izvajanjem.

```
<?php
function preveri_email_naslov ($email) {
    // preverimo, da imamo samo en znak @, dolžina mora biti znotraj dovoljene meje
    $tabela1;
    if (!preg_match('/^[^@]{1,64}@[^@]{1,255}$/', $email, $tabela1)) {
        // email naslov je neveljaven, ker ni dolžina podnizov pravilna ali pa je število znaka @
        // različno od 1
        return false;
    }
}
```

³Vir: <http://www.faqs.org/rfcs/rfc2822.html>

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
// email razdelimo na manjše nize
$email_tabela = explode("@", $email);
$lokalniDel = explode(".", $email_tabela[0]);
for ($i = 0; $i < sizeof($lokalniDel); $i++) {
    if (!preg_match('/^([A-Za-z0-9!#$%&*+\\/=/?^_`{|}~-][A-Za-z0-9!#$%&*+\\/=/?^_`{|}~\\.\\-]{2,63})$/', $lokalniDel[$i], $tabela2)){
        return false;
    }
}

//domeno razbijemo na podnize
if (!preg_match('/^[?0-9\\.]\\+\\)?$/', $email_tabela[1])) {
    $domena = explode(".", $email_tabela[1]);
    //če je število podnizov v domeni premajhno
    if (sizeof($domena) < 2){
        return false;
    }
    for ($i = 0; $i < sizeof($domena); $i++){
        if (!preg_match('/^([A-Za-z0-9][A-Za-z0-9-]{2,61}[A-Za-z0-9])([A-Za-z0-9]+)$/', $domena[$i], $tabela3)){
            return false;
        }
    }
}
return true;
}

//v spremenljivko email shranimo niz (email), ki ga želimo preveriti
$email = "test@test.com";
if (preveri_email_naslov ($email)) {
    echo $email .' je veljaven email naslov.';
}
else{
    echo $email .' ni veljaven email naslov.';
}
?>
```

Funkcija je napisana tako, da vrne logično vrednost bool. Če naslov zadosti našim pogojem se nam na zaslon izpiše:

```
test@test.com je veljaven email naslov.
```

Če internetni poštni naslov ne zadosti kateremkoli izmed teh pogojev se na zaslon izpiše:

```
test@test...com ni veljaven email naslov.
```

Spodaj je seznam nekaterih internetnih naslovov, ki ne zadostijo programu:

Nepravilni naslovi, ki ne ustrezajo	Razlaga
student_fmfgmail.com	v naslovu manjka znak '@'
student..fmf@gmail.com	naslov ne sme vsebovati dveh zaporednih pik
student_fmf@.com	v domeni manjka poddomena
student_fmf@gmail	v domeni manjka končnica
@gmail.com	v lokalnem delu mora biti vsaj en znak
student_fmf@@gmail.com	en znak '@' v naslovu je preveč
! "#\$%()./;<>[]@gmail.com	neveljavni znaki v lokalnem delu

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

student_fmf@! "\$%(),./;<>[]].com	neveljavni znaki v domeni
student_fmf@acom	domena mora vsebovati vsaj en znak pika

Pri iskanju pravilnega internetnega poštnega naslova se bo gotovo našel kakšen naslov, ki bo ustrežal regularnemu izrazu, pa ne bi smel. Lahko se zgodi tudi, da se ne ustreza naslovu, čeprav bi moral. Poglejmo si primer internetnih poštnih naslovov, ki ustrezajo regularnim izrazom, pa ne bi smeli:

Naslovi, ki ne bi smeli ustrezati	Razlaga
domenaNeObstaja@domain.moc	email naslov ustreza, čeprav domena ne obstaja
steviloVDomeni@domain.c0m	email ima v domeni namesto črke števko

Poglejmo si še primer, kjer iščemo ustrezen internetni poštni naslov fakultete za matematiko in fiziko. Pogoj za študentski internetni poštni naslov (poleg splošnih pogojev za internetni naslov) je naslov domene, ki je vedno enak. To je "@student.fmf.uni-lj.si". Tudi za lokalni del imamo strožje omejitve, saj so dovoljeni znaki samo male slovenske črke (brez šumnikov), pika in števka na koncu priimka.⁴ Ker imamo za preverjanje internetnega poštnega naslova naše fakultete strožje omejitve nam preostane lažje delo. Tako drugi del internetnega naslova nima nadomestnih znakov, pač pa so vsi znaki običajni.

Kot smo že omenili, so v lokalnem delu edini dovoljeni znaki slovenske črke, neodvisne od velikosti ter pika in števka na koncu priimka. Tako je regularni izraz za preverjanje lokalnega dela bolj preprost. Ker pa je domena vedno enake oblike, se že pri prvem preverjanju ustreznosti regularnega izraza prepričamo, da je domena pravilna. Prej pa smo morali domeno posebej preveriti. V prejšnjem primeru smo preverjali veljavnost vseh dovoljenih znakov v lokalnem delu z regularnim izrazom: `'/^([A-Za-z0-9!#$%&*+V=?^_`{}~\.-]{2,63})$/'`. Sedaj pa posebej preverimo lokalni del, ki vsebuje priimek: `'/^([A-Za-z]{3,30})[0-9]{0,1}$/'`, saj ta lahko vsebuje tudi števko na koncu priimka. Regularni izraz za preverjanje imena, pa se ne razlikuje veliko, tukaj števka ni dovoljen znak: `'/^([A-Za-z]{3,30})$/'`. Ime se nahaja v prvem delu lokalnega dela (do znaka '.'). V kodi ga najdemo na mestu `$lokalniDel[0]`. Iz tega lahko pravilno sklepamo, da se priimek nahaja na mestu: `$lokalniDel[1]`. Število znakov v imenu in priimku smo dovolili med 3 in 30 (saj je število dovoljenih znakov v lokalnem delu največ 63). Izpustili pa smo dodatno preverjanje domene, saj smo to storili že takoj na začetku z regularnim izrazom: `'/^[@]{6,64}@student.fmf.uni-lj.si$/'`.

```
<?php
function preveri_email_naslov ($email) {
    //preverimo, da imamo samo en znak @, dolžina mora biti znotraj dovoljene meje
    $tabela1;
    //preverimo ali domena ustreza vzorcu: student.fmf.uni-lj.si in da ima naslov znak '@'
    //in da je dolžina lokalnega dela pravilna
    if (!preg_match('/^[@]{6,64}@student.fmf.uni-lj.si$/', $email, $tabela1)){
        return false;
    }
    //email razdelimo na manjše nize
    $email_tabela = explode("@", $email);
    //lokalni del razdelimo na dva niza
    $lokalniDel = explode(".", $email_tabela[0]);
    $tabela2;
    //dolžina lokalnega dela mora biti 2 - ime in priimek
    if (sizeof($lokalniDel) != 2){
        return false;
    }
}
```

⁴ Vir: Računalniški center FMF

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
else{
  if ($i == 0){
    if(!preg_match('/^[A-Za-z]{3,30}$/', $lokalniDel[0], $tabela2)){
      return false;
    }
  }
  //za priimkom dovolimo tudi števk
  else if (!preg_match('/^[A-Za-z]{3,30}[0-9]{0,1}$/', $lokalniDel[1], $tabela2)){
    return false;
  }
  else{
    return true;
  }
}
}

/*s spodnjim stavkom "$email= ..." preverimo ali internetni poštni naslov ustreza našemu
regularnemu izrazu, zakomentirali pa smo ga, ker ga pogosto spreminjamo med pisanjem regularnega
izraza, testiramo pravilne in nepravilne vnose
*/
//$email = "jasna2.kobal@student.fmf.uni-lj.si";
if (preveri_email_naslov ($email)) {
  echo $email .' je veljaven email naslov.';
}
else{
  echo $email .' ni veljaven email naslov.';
}
?>
```

Primer ustreznega naslova:

```
jasna.kobal@student.fmf.uni-lj.si je veljaven email naslov.
```

Primer napačnega naslova (v lokalnem delu je številka dovoljena samo na koncu priimka):

```
jasna2.kobal@student.fmf.uni-lj.si ni veljaven email naslov.
```

4.4 Izračun povprečne starosti zaposlenih

V datoteki imamo v formatu CSV imamo shranjene podatke o zaposlenih. Kot smo že omenili so v formatu CSV posamezna polja razdeljena z vejico (lahko tudi drugim ločilom). Zanima nas, kakšna je povprečna starost zaposlenih. Denimo, da ne vemo kateri po vrsti je stolpec s podatki o starosti. Pri obdelavi podatkov si bomo pomagali z regularnim izrazom. Predpostavimo, da je starost edini podatek, ki je številka. Poiskali bomo vse podatke o starosti ter izračunali povprečno starost zaposlenih.

Najprej moramo prebrati datoteko. Uporabimo funkcijo **fgetcsv**, da pridobimo podatke iz datoteke. V datoteki imamo podatke razporejene po stolpcih: ime, priimek, starost ter poklic. S pomočjo funkcije **preg_grep** poiščemo podatke, ki ustrezajo starosti. Starost zaposlenih je običajno med 15 in 70 let. Zapišemo regularni izraz, ki zadosti tem pogojem: **'([1][5-9] | [2-7][0-9])'** Ko dobimo potrebne podatke, jih shranimo v tabelo `$d`. Sprehodimo se čez tabelo, kjer preštejemo število zadetkov in seštejemo vse starosti zaposlenih. Ko "izstopimo" iz zanke, izračunamo povprečje - vsoto delimo s številom zaposlenih. Število zapisov je za ena večje od števila zaposlenih, ker nam funkcija

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

fgetcsv izpisuje tudi naslovno vrstico, v kateri pa ni podatka o starosti. Naslovna vrstica izpiše ime stolpcev, ki jih datoteka v formatu CSV vsebuje. Ker pa zaposlene štejemo šele v zanki, kjer mora stolpec s starostjo vsebovati število, ne pride do napake.

V datoteki zaposleni.csv imamo naslednje podatke:

IME,PRIIMEK,STAROST,POKLIC

Janez,Novak,28,direktor

Marija,Novak,30,tajnica

Ana,Lampe,33,prodajalka

Martin,Jurko,25,natakar

Monika,Petric,40,delavka

Cene,Rutar,35,blagajnik

Petra,Drole,22,programer

Jure,Balog,32,natakar

Igor,Serjun,24,prodajalec

Maja,Kikelj,35,prodajalka

```
<pre>
<?php
    $branje = fopen("zaposleni.csv", "r");
    while (($podatki = fgetcsv($branje, 1000, ",")) !== FALSE){
        $stolpec = preg_grep( '/[0-9]+/', $podatki);
        print_r ($podatki[0]);
        echo " ";
        print_r ($podatki[2]);
        echo "<br/>";
        foreach($stolpec as $kljuc => $starost){
            $vsota = $vsota + $starost;
            $stevilo = $stevilo + 1;
        }
    }
    echo "<br/>Vsota: $vsota";
    $povprecna_starost = $vsota / $stevilo;
    echo "<br/> Povprečna starost zaposlenih: $povprecna_starost";
    fclose($branje);
?>
</pre>
```

Primer izpisa programa, zgornjih podatkov:

```
IME STAROST
Janez 28
Marija 30
Ana 33
Martin 25
Monika 40
Cene 35
Petra 22
Jure 32
Igor 24
Maja 35

Vsota: 304
Povprečna starost zaposlenih: 30.4
```

4.5 Odstranjevanje komentarjev PHP

Denimo, da imamo napisan program v kodi PHP, ki vsebuje enovrstične in večvrstične komentarje. Denimo, da ne dovolimo gnezdenih komentarjev (komentar znotraj komentarja). Enovrstične komentarje v kodi PHP pišemo z '//', večvrstične pa tako, da začetek komentarja napovemo z znakoma '/*', končamo pa z znakoma '*/'.

Sestavimo najprej regularni izraz, kateremu ustrezajo enovrstični komentarji.

Pri sestavljanju regularnega izraza moramo biti pozorni, saj je znak '/' nadomestni znak. Takrat, ko želimo ta znak uporabiti kot običajni znak, moramo pred njega napisati še ubežni znak, to je znak '\\'. Tako znak '/' izgubi svoj posebni pomen. Tako je regularni izraz za enovrstične komentarje videti kot: '\\\\.*/'. Ker je kombinacija poševnice nazaj in poševnice videti kot znak V, moramo opozoriti bralca, da je V\\ niz, sestavljen iz štirih znakov: '\\', ki mu sledi '/' in še enkrat ta kombinacija.

Prav tako moramo tudi pri regularnem izrazu za večvrstične komentarje zapisati ubežni znak pred poševnico naprej. Enako moramo narediti pred znakom zvezdica. Poleg regularnega izraza za večvrstične komentarje: '*.\\?*\\\\/s' imamo tukaj tudi preoblikovalec '/s'. Ta preoblikovalec celoten niz obravnava kot eno vrstico. V tem primeru znak '.' ustreza nevidnemu znaku, prehod v novo vrsto, ki pa ga drugače ne najde.

```
<?php
    $niz = 'besedilo 1 //enovrstični komentar
        besedilo 2 //še en enovrstični komentar
        besedilo 3
        /* prvi
        večvrstični
        komentar
        */
        besedilo 4
        /*
        drugi
        večvrstični
        komentar
        */
        besedilo 5';
    $vzorec = '\\\\*.\\?\\*\\\\/sU';
    $vzorec2 = '\\\\\\.*/';
    $zamenjava = ' ';
    //najprej odstranimo znake za večvrstične komentarje in vsebino komentarjev
    $niz2 = preg_replace($vzorec, $zamenjava, $niz);
    //nato odstranimo še enovrstične komentarje in vsebino komentarjev
    //izpiše se samo niz izven komentarjev
    echo preg_replace($vzorec2, $zamenjava, $niz2);
?>
```

Ko prvič uporabimo funkcijo **preg_replace** smo se znebili večvrstičnih komentarjev. Znebili smo se tako znakov za komentarje '/*' ter '*/' kot tudi same vsebine komentarjev. Preoblikovalec '/s', pa najde ustrezen podniz, ki se začne v eni vrstici in se nadaljuje v več vrsticah.

Ko nam ostanejo komentarji oblike '//' pa še te zamenjamo s praznim nizom.

```
besedilo 1 besedilo 2 besedilo 3 besedilo 4 besedilo 5
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

V primeru smo uporabili tudi preoblikovalec U, ki določa, da so določevalci količine leni. To v našem primeru pomeni, da funkcija ustavi iskanje podniza takoj, ko prvič najde zadnji znak iz regularnega izraza. Če ne bi uporabili preoblikovalca U, bi se iskanje zaključnega dela vrstičnega komentarja bi se končalo pri zadnji pojavitvi niza '*/', s čimer bi odstranili tudi podniz "besedilo 4", ki pa se ne nahaja znotraj komentarjev.

4.6 Odstranjevanje podvojenih presledkov

Denimo, da imamo besedilo, v katerem smo napravili odvečne presledke. Tako sta se med posamezne besede namesto enega prikrdla dva (ali več) presledkov. Teh odvečnih presledkov bi se radi znebili. Potrebno je najti presledke, ki se ponovijo enkrat ali večkrat: `'/( ){2,}/'`. Podvojene presledke zamenjamo s presledkom. Tako dobimo besedilo brez podvojenih presledkov. Pri tem smo uporabili skupino brez vidnih znakov. Kot smo že omenili v razdelku 4. bomo presledke v besedilu označevali z `'_'`, v kodi pa s praznim prostorom.

```
<?php
  $niz = "Tukaj je    veliko    podvojenih    presledkov.";
  $vzorec = '/( ){2,}/';
  $zamenjava = '_';
  echo $nov_niz = preg_replace($vzorec, $zamenjava, $niz);
?>
```

Kadar pa je v besedilu samo en presledek, ga ne odstranimo, ker z regularnim izrazom iščemo presledke, kjer se pojavita zaporedoma dva ali več. Izpis novega besedila brez podvojenih presledkov:

```
Tukaj je veliko podvojenih presledkov.
```

4.7 Preverjanje veljavnosti vnosov v obrazce

Za registracijo uporabnika na našem forumu potrebujemo uporabnikove podatke. Ko uporabnik izpolni podatke v obrazec, se na vsakem vnosnem polju posebej prepričamo, da posamezno polje ni prazno ter da podatek zadostuje posameznim kriterijem. Na primer v polju ime in priimek so edini dovoljeni znaki slovenske črke male in velike ter presledki.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

http://localhost/vnosnaPolja2.php - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Reload Home Search Favorites

Address http://localhost/vnosnaPolja2.php

IME IN PRIIMEK:

UPORABNIŠKO IME:

GESLO:

POŠTNA ŠTEVILKA:

KRAJ POŠTE:

NASLOV:

TELEFONSKA ŠTEVILKA:

E-MAIL NASLOV:

ROJSTNI DATUM

DAN:

MESEC:

LETO:

Potrdi

Slika 4: Preverjanje veljavnosti vnosov v obrazcu

4.7.1 Preverjanje polja ime in priimek

Pri preverjanju imena in priimka ne smemo pozabiti, da ime in priimek vsebuje vsaj en presledek. Predpostavimo, da ima oseba največ dve imeni in dva priimka (oziroma dovolimo štiri besede). Tukaj tudi predpostavljamo, da ima oseba slovensko ime in priimek, torej sta sestavljena iz črk slovenske abecede. Predpostavljamo tudi, da sta vsako ime oz. vsak priimek sestavljena iz najmanj treh črk.

Lahko pa bi bilo preverjanje veljavnosti polja ime in priimek dosti bolj zapleteno. Tukaj nismo upoštevali možnosti, da imamo slovenski državljani v imenih in priimkih tudi tuje črke (na primer č). Preverjanje imena in priimka je zato zelo omejeno. V imenu ne dopuščamo niti znaka '-' niti znaka '''. Prav tako ne dopuščamo inicialk ter nikjer ne preverjamo velikih začetnic.

Poglejmo kako smo se lotili samega preverjanja. S pomočjo funkcije **strtoupper** pretvorimo male črke v velike. S tem smo regularni izraz poenostavili, saj nam ni potrebno preverjati velikih in malih črk, pač pa preverjamo samo velike črke. Izjema so šumniki, ki jih s funkcija **strtoupper** ne zna pretvoriti, v našem primeru deluje samo za angleške črke. S funkcijama **trim** in **preg_replace** se znebimo odvečnih presledkov. Funkcija **trim** odpravi odvečne presledke na začetku in koncu niza. Če pa uporabnik pomotoma vpiše več zaporednih presledkov znotraj niza, to napako odpravimo s funkcijo **preg_replace**. Ta nam zamenja podvojene presledke z enojnim presledkom.

Sestavimo regularni izraz.

Razred za zapis imena, to je slovenskih in angleških črk zapišemo: **[ABCČD-PRSŠTUVŽžčšž]{3,}**. Zraven je dodan še določevalec količine, ki določa, da se mora znak iz razreda ponoviti vsaj trikrat. Drugi del regularnega izraza **([ABCČD-PRSŠTUVŽžčšž]{3,}){1,3}** pa določa priimek. Ta drugi del pomeni, da regularni izraz vsebuje presledek, ki mu sledijo najmanj tri črke. To zaporedje se lahko ponovi enkrat ali trikrat (upoštevali smo tudi primer, da ima oseba dve imeni in/ali dva priimka).

```
<?php
//najprej preverimo, da polje ime ni prazno
if (!$_POST['ime']){
    echo "Prosimo vpišite ime in priimek.";
}
else {
    //če ima vhodni niz male črke jih pretvorimo v velike
    //v spremenljivko $ime shranimo niz s samimi velikimi črkami, ter presledki
```

```
$ime = strtoupper($ime);  
//odstranimo odvečne presledke na začetku in koncu niza  
$ime = trim($ime);  
$vzorec = '/( ){2,}/';  
$zamenjava = ' ';  
//odstranimo vmesne podvojene presledke  
$ime2 = preg_replace($vzorec, $zamenjava, $ime);  
//z RI se prepričamo, da niz vsebuje samo besede, ki so ločene z enim presledkom  
if (preg_match('/^[ABCČD-PRSŠTUVZŽčšž]{3,}([ABCČD-PRSŠTUVZŽčšž]{3,}){1,3}$/',  
$ime2)){  
    echo "Prosimo ponovno vpišite ime in priimek.";  
}  
else{  
    echo "Ime in priimek sta pravilno izpolnjena.";  
}  
}  
?>
```

4.7.2 Preverjanje polja poštne številke

Preverjanje poštne številke je dokaj enostavno, pri pogoju, da dovolimo tudi napačne podatke, na primer številka "9999". Poštna številka je štirimestno število, ki pa na prvem mestu ne sme vsebovati ničle. Uporabniku dovolimo tudi vnos presledkov na začetku, koncu niza, ter tudi med samimi števki. S funkcijo **preg_replace** odstranimo odvečne presledke. Tako nam ostanejo samo števke. S tem zelo poenostavimo regularni izraz, ki zgleda tako: `'/^([1-9][0-4][0-9]{2})$/'`. V viru⁵ imamo seznam poštne številke, tu lahko vidimo, da je omejitev za prvo števko številka med 1 in 9, za drugo števko pa je omejitev številka med 0 in 4. Vendar pa na tem mestu nismo upoštevali poštne številke večjih podjetij in organizacij.

```
<?php  
//najprej preverimo, da polje postna_st ni prazno  
if (!$_POST['postna_st']){  
    echo "Prosimo ponovno vpišite poštno številko.";  
}  
else {  
    //dovolimo vpis 4 števk, prva ima omejitev od 1 do 9, druga od 0 do 4, ostali dve nimata omejitve  
    //odstranimo odvečne presledke  
    $vzorec = '/( ){1,}/';  
    $zamenjava = " ";  
    //odstranimo vse presledke  
    $postna_st2 = preg_replace($vzorec, $zamenjava, $postna_st);  
    if (!preg_match('/^[1-9][0-4][0-9]{2}$/', $postna_st2)){  
        echo "Prosimo ponovno vpišite poštno številko.";  
    }  
    else{  
        echo "Poštna številka je pravilno izpolnjena.";  
    }  
}  
?>
```

⁵Vir: <http://www.posta.si/Namizje.aspx?tabid=76>

4.7.3 Preverjanje polja kraj pošte

Preveriti želimo niz, ki ga je uporabnik vnesel v polje kraj pošte. Veljavni nizi so tisti, ki jih najdemo na seznamu krajev pošte.⁶ Ostali nizi ne ustrezajo krajem pošte, vendar pa spodnji regularni izraz ustreza tudi marsikateremu nepravilnemu podatku sestavljenemu iz črk in presledkov. Za nas bo veljaven skoraj vsak niz, ki bo sestavljen iz črk in presledkov. Uporabniku preprečimo potrditev niza, ki vsebuje angleške črke (q,w,y,x). Prav tako ne dovolimo niza, ki bi bil sestavljen iz petih ali več besed. Uporabniku tudi ne dovolimo vnosa posamezne črke, saj posamezna beseda v kraju pošte vsebuje najmanj dve črki.

Poglejmo kako smo se lotili samega preverjanja. S pomočjo funkcije **strtoupper** pretvorimo male črke v velike. S tem smo regularni izraz poenostavili, saj nam ni potrebno preverjati velikih in malih črk, pač pa preverjamo samo velike črke. Izjema so šumniki, ki jih s funkcijo **strtoupper** ne zna pretvoriti, v našem primeru deluje samo za angleške črke. S funkcijama **trim** in **preg_replace** se znebimo odvečnih presledkov. Funkcija **trim** odpravi odvečne presledke na začetku in koncu niza. Če pa uporabnik pomotoma vpiše več zaporednih presledkov znotraj niza, to napako odpravimo s funkcijo **preg_replace**. Ta nam zamenja podvojene presledke z enojnim presledkom.

Sestavimo regularni izraz. Znakovni razred skupaj z določevalcem količine **[ABCCĎD-PRSŠTUVZZčšž]{3,}** najde besedo, ki ima vsaj 3 črke. Znakovna skupina **([ABCCĎD-PRSŠTUVZZčšž]{3,}){0,3}**, pa nam najde presledek (ali več presledkov), ki mu sledita vsaj 2 črki ali več. Sama skupina pa se lahko v ustreznem regularnem izrazu pojavi nikoli, enkrat, dvakrat ali trikrat. Torej celotni regularni izraz najde besedo ali največ štiri besede, ki so ločene s presledkom.

Ko preverimo, ali niz ustreza regularnemu izrazu **/^[ABCCĎD-PRSŠTUVZZčšž]{3,}([ABCCĎD-PRSŠTUVZZčšž]{3,}){0,3}\$ /** smo preverili, ali niz vsebuje slovenske črke.

```
<?php
//najprej preverimo, da polje posta ni prazno
if (!$_POST['posta']){
    echo "Prosimo ponovno vpišite kraj pošte.";
}
else {
    //če ima vhodni niz male črke jih pretvorimo v velike
    $posta = strtoupper($posta);
    $posta = trim($posta);
    $vzorec = '/( ){2,}/';
    $zamenjava = ' ';
    $posta2 = preg_replace($vzorec, $zamenjava, $posta);
    if (preg_match('/^[ABCCĎD-PRSŠTUVZZčšž]{3,}([ABCCĎD-PRSŠTUVZZčšž]{3,}){0,3}$/',
    $posta2)){
        echo "Kraj pošte je pravilno izpolnjen.";
    }
    else {
        echo "Prosimo ponovno vpišite kraj pošte.";
    }
}
?>
```

4.7.4 Preverjanje polja naslov

Regularni izraz za preverjanje naslova je podoben regularnemu izrazu za preverjanje kraja pošte.

⁶ Vir: <http://www.posta.si/Namizje.aspx?tabid=76>

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

Vendar pa naslov poleg kraja lahko vsebuje tudi hišno številko ali pa tudi ne. Hišna številka pa v nekaterih primerih vsebuje tudi črko.

Poglejmo kako smo se lotili samega preverjanja. S pomočjo funkcije **strtoupper** pretvorimo male črke v velike. S tem smo regularni izraz poenostavili, saj nam ni potrebno preverjati velikih in malih črk, pač pa preverjamo samo velike črke. Izjema so šumniki, ki jih s funkcijo **strtoupper** ne zna pretvoriti, v našem primeru deluje samo za angleške črke. S funkcijama **trim** in **preg_replace** se znebimo odvečnih presledkov. Funkcija **trim** odpravi odvečne presledke na začetku in koncu niza. Če pa uporabnik pomotoma vpiše več zaporednih presledkov znotraj niza, to napako odpravimo s funkcijo **preg_replace**. Ta nam zamenja podvojene presledke z enojnim presledkom.

Tako moramo poleg regularnega izraza, ki smo ga uporabili za preverjanje kraja pošte: **[ABCCD-PRSŠTUVZZčšž]{3,}([ABCCD-PRSŠTUVZZčšž]{3,}){0,3}** pri naslovu dodatno preveriti še, ali naslov vsebuje tudi hišno številko. Hišna številka je sestavljena iz presledka ter ene do treh števk, lahko pa ima tudi črko. Tako prej zapisanemu regularnemu izrazu dodamo še **([0-9]{1,3}[ABCDEFGHJKLMNOPRST]?)?**, kar pomeni eden ali nič presledkov, ki mu sledi ali pa ne sledi eno, dvo ali trimestna številka. Na koncu sledi presledek ter črka slovenske abecede (brez šumnikov). Ta skupina pa se lahko pojavi nikoli ali pa se pojavi enkrat.

```
<?php
//najprej preverimo, da polje naslov ni prazno
if (!$_POST['naslov']){
    echo "Prosimo ponovno vpišite naslov.";
}
else {
    $naslov = strtoupper($naslov);
    $naslov = trim($naslov);
    $vzorec = '/( ) {2,}/';
    $zamenjava = ' ';
    $naslov2 = preg_replace($vzorec, $zamenjava, $naslov);
    if (preg_match('/^[ABCCD-PRSŠTUVZZčšž]{3,}([ABCCD-PRSŠTUVZZčšž]{3,}){0,3}([0-9]{1,3}[ABCDEFGHJKLMNOPRST]?)?$/', $naslov2)){
        echo "Naslov je pravilno izpolnjen.";
    }
    else {
        echo "Prosimo ponovno vpišite naslov.";
    }
}
?>
```

4.7.5 Preverjanje polja telefonska številka

Predpostavimo, da je veljavna telefonska številka, številka stacionarnega ali mobilnega telefona znotraj slovenskega omrežja. Običajno se številka mobilnega telefona začne z 031,040... številka stacionarnega telefona pa z 01,02,03... Določili smo, da je veljavna številka sestavljena iz devetih števk, ki so lahko ločene s presledkom ali več presledki med posameznim sklopom števil. Lahko pa so številke zapisane brez presledkov. Prva številka mora biti nič, druga je lahko med ena in sedem, ostale številke pa nimajo omejitve. Ker bomo najprej odstranili vse presledke, ki jih lahko uporabnik postavlja na poljubnem mestu dovolimo samo številke. Tako je regularni izraz za preverjanje telefonske številke dokaj preprost: **/0[1-7][0-9]{7}/**.

```
<?php
//najprej preverimo, da polje telefonska ni prazno
if (!$_POST['telefonska']){
```

```
echo "Prosimo vpišite telefonsko številko.";
}
else {
    $vzorec = '%( ){1,}/';
    $zamenjava = "";
    //odstranimo vse presledke, ostane nam samo ena številka
    $telefonska2 = preg_replace($vzorec, $zamenjava, $telefonska);
    $dolzina = strlen($telefonska2);
    //izločimo vse številke, ki nimajo devetih števk, ali pa vrstni red ne ustreza pogojem
    if (preg_match('/0[1-7]([0-9]){7}/', $telefonska2)){
        if($dolzina == 9){
            echo "Telefonska številka je pravilno izpolnjena.";
        }
        else{
            echo "Prosimo ponovno vpišite telefonsko številko.";
        }
    }
}
}
if (preveriTelefonsko ($telefonska)) {
    echo "Telefonska številka je pravilno izpolnjena.";
}
else{
    echo "Prosimo ponovno vpišite telefonsko številko.";
}
?>
```

4.7.6 Preverjanje polja rojstni datum

Odločili smo se, da bomo za lažje preverjanje polja rojstni datum polja ločili. Tako je dan rojstva eno polje, drugo polje je mesec rojstva, tretje polje pa leto rojstva. To je lepo vidno na sliki 4. Predpostavimo, da sta veljavni dan ter mesec lahko enomestni ali dvomestni števili. Tako bomo lažje napisali regularni izraz, ter ga razbili na tri dele. Veljavnost vsakega polja bomo preverjali z drugačnim regularnim izrazom.

Za veljavne rojstne podatke smo omejili datum rojstva na uporabnike, ki so rojeni med "01.01.1899" ter "31.12.2004". Predvidevamo, da naši uporabniki niso starejši od 120 let ter mlajši od 5 let. Regularni izraz za preverjanje polja leto, je sledeč: `'/(1889|189[0-9]|19[0-9]{2}|200[0-4])/'`. Z delom regularnega izraza `'1889|189[0-9]'` preverimo ali je uporabnikovo leto rojstva spada v 19. stoletje. Preverjanje leta rojstva, ki spada v 20. stoletje vsebuje omejitev samo za prvi dve številki (morata biti 19), ostali dve sta lahko poljubni. Leto rojstva v 21. stoletju pa dovoljuje samo leta 2000, 2001, 2002, 2003 in 2004.

Mesec rojstva mora biti strogo večji od nič in strogo manjši od trinajst. Regularni izraz `'^(0?[1-9]|1[012])$/'` pomeni da številki nič lahko sledi katerakoli številka med ena in devet (vključno s tema dvema) ali pa številka ena, ki ji lahko sledijo števila 0,1 ali 2. `'^(0?[1-9]|1[012]|0[9]|3[01])$/'` je regularni izraz, ki nam pove, da je dan veljaven, če številki nič sledi katerakoli številka, ki je od nič različna. Lahko pa imamo na prvem mestu tudi številko ena ali dva, ki jima sledi katerakoli številka. Kadar pa je na prvem mestu številka tri, ji mora nujno slediti številka nič ali ena. Ker bi bilo preverjanje, ali ima mesec 31 dni ter ali je leto prestopno v samem regularnem izrazu prezahtevno, smo to preverjanje napisali izven regularnega izraza s pomočjo stavka if. Ker je koda zelo dolga in že vsebuje komentarje se v podrobnosti ne bomo spuščali. Omenila bi samo, da za prestopno leto velja, da je deljivo s 4, ali pa s 100 in hkrati s 400.

```
<?php
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
//preverimo da polje dan ni prazno
//če je polje dan prazno, javimo napako
if (!$_POST['dan'] ){
    $pravilnostDneva = false;
}
// če dan ne ustreza pogojem številka med 1 in 31 opozorimo uporabnika
else if ($_POST['dan'] ){
    if (!preg_match('/^(0?[1-9]|[12][0-9]|3[01])$/', $dan)){
        $pravilnostDneva = false;
    }
    else{
        $pravilnostDneva = true;
    }
}
//preverimo da polje leto ni prazno
if (!$_POST['leto'] ){
    $pravilnostLeta = false;
}
//če leto ne ustreza pogojem številka med 1899 in 2004 javimo napako
else if ($_POST['leto'] ){
    if (!preg_match('/(1889|189[0-9]|19[0-9]{2}|200[0-4])/', $leto)){
        $pravilnostLeta = false;
    }
    else{
        $pravilnostLeta = true;
    }
}
//preverimo da polje mesec ni prazno
//če je polje mesec prazno, javimo napako
if (!$_POST['mesec'] ){
    $pravilnostMeseca = false;
}
//če mesec ne ustreza pogojem številka med 1 in 12 javimo napako
else if ($_POST['mesec'] ){
    if (!preg_match('/^(0?[1-9]|1[012])$/', $mesec)){
        $pravilnostMeseca = false;
    }
    if ($mesec == 2){
        //če je mesec februar in ima več kot 29 dni javimo napako
        if ($dan > 29){
            $pravilnostMeseca = false;
        }
        else if ($dan == 29){
            //če je mesec februar in dan je 29, leto pa je prestopno
            if ($leto % 4 == 0 && ($leto % 100 != 0 || $leto % 400 == 0)){
                $pravilnostMeseca = true;
            }
            //če leto ni prestopno javimo napako
            else{
                $pravilnostMeseca = false;
            }
        }
    }
    else if ($dan < 29){
        $pravilnostMeseca = true;
    }
}
```

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
}
//kadar imamo 31 dni, mesec ima pa samo 30 dni javimo napako
else if (($mesec == 4 || $mesec == 6 || $mesec == 9 || $mesec == 11) && ($dan == 31)){
    $pravilnostMeseca = false;
}
else if (($mesec == 4 || $mesec == 6 || $mesec == 9 || $mesec == 11) && ($dan <= 30)){
    $pravilnostMeseca = true;
}
else if (($mesec == 1 || $mesec == 3 || $mesec == 5 || $mesec == 7 || $mesec == 08 || $mesec
== 10 || $mesec == 12) && ($dan <= 31)){
    $pravilnostMeseca = true;
}
}
if ($pravilnostDneva == false || $pravilnostMeseca == false || $pravilnostLeta == false){
    echo "Prosim ponovno vpišite rojstni datum.";
}
else if ($pravilnostDneva == true && $pravilnostMeseca == true && $pravilnostLeta == true ){
    echo "Rojstni datum je pravilno izpolnjen.";
}
?>
```

4.7.7 Preverjanje polja internetni poštni naslov

Za preverjanje polja internetni poštni naslov uporabimo regularni izraz, ki smo jo napisali že zgoraj (razdelek 5.1). Regularni izraz preverja veljavnost samo najbolj pogostih internetnih poštinih naslovov: `'/^([a-zA-Z0-9]{1,30}[a-zA-Z0-9]?[a-zA-Z0-9]{0,30})|([a-zA-Z0-9]{1,60}))@([a-zA-Z]{2,8})\.([a-zA-Z]{2,8})$/'`.

```
<?php
//najprej preverimo, da polje email ni prazno
if (!$_POST['email']){
    echo "Prosim ponovno vpišite internetni poštni naslov.";
}
else{
    //lokalni del ima lahko eno besedo, ali pa dve besedi, ki ju loči pika, dovoljene so tudi števke
    //domena mora biti nujno sestavljena iz dveh besed, ki ju loči pika, dovoljene so samo črke
    if (!preg_match('/^([a-zA-Z0-9]{1,30}[a-zA-Z0-9]?[a-zA-Z0-9]{0,30})|([a-zA-Z0-9]{1,64}))@([a-zA-Z]{2,8})\.([a-zA-Z]{2,8})$/', $email)){
        echo "Prosim ponovno vpišite internetni poštni naslov.";
    }
    else {
        echo "Internetni poštni naslov je pravilno izpolnjen.";
    }
}
?>
```

4.7.8 Preverjanje polja uporabniško ime

Predpostavimo, da mora biti uporabniško ime sestavljeno iz pet angleških črk, ki jim sledijo tri števke. Izločimo vse vnose z drugačnim vrstnim redom. Črke so lahko poljubne velikosti (velike ali male). Tako regularni izraz zapišemo kot: `'^[a-zA-Z]{5}[0-9]{3}$'`, to je pet angleških črk, ki jim sledijo tri števke.

DIPLOMSKA NALOGA :

FAKULTETA ZA MATEMATIKO IN FIZIKO

```
<?php
//preverimo, da polje upor_ime ni prazno
if (!$_POST['upor_ime']){
    echo "Prosimo vpišite uporabniško ime.";
}
else {
    //dovolimo 5 angleških črk ter 3 števke
    if (!preg_match('/^[a-zA-Z]{5}[0-9]{3}$/', $upor_ime)){
        echo "Prosimo ponovno vpišite uporabniško ime.";
    }
    else{
        echo "Uporabniško ime je pravilno izpolnjeno.";
    }
}
?>
```

4.7.9 Preverjanje polja geslo

Predpostavimo, da je v našem primeru pogoj za pravilno izpolnjeno vnosno polje geslo dolžina gesla osem do deset znakov. Dovoljeni znaki so samo angleške črke ter števke. Vrstni red ni pomemben, lahko jih poljubno kombiniramo. Regularni izraz je preprost: `'^[a-zA-Z\d]{8,10}$'`. Spomnimo se, da je `'\d'` bližnjica za števke.

```
<?php
//preverimo, da polje geslo ni prazno
if (!$_POST['geslo']){
    echo "Prosimo vpišite geslo.";
}
else {
    //dovolimo 8-10 angleških črk ali števk
    if (!preg_match('/^[a-zA-Z\d]{8,10}$/', $geslo)){
        echo "Prosimo ponovno vpišite geslo.";
    }
    else{
        echo "Geslo je dovolj močno.";
    }
}
?>
```

5 Zaključek

V diplomski nalogi smo se dotaknili nekaterih osnov jezika PHP. Večino časa smo se ukvarjali z nizi. Predvsem pa nas je zanimal jezik PHP v povezavi z regularnimi izrazi. NA primerih smo tudi opazili, da PHP nudi v svoji knjižnici PCRE odlično podporo za delo z regularnimi izrazi. Tako smo se v diplomi srečali tudi s pogosto uporabljenima funkcijama `preg_match` in `preg_replace`. Regularni izrazi so pogosto uporabljeni v različnih tekstovnih urejevalnikih, podatkovnih bazah ter internetnih brskalnikih (Texpad, Cobiss, Mozilla...), pa še kje.

Tako smo bralca seznanili z nekaterimi osnovami jezika PHP in osnovnimi pojmi pri delu z regularnimi izrazi. Samo upamo lahko, da smo s tem diplomskim delom vsaj kateremu bralcu zbudili zanimanje za regularne izraze ali jezik PHP.

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

Viri in literatura

Achour, M., Betz, F., Dovgal, A. in drugi, PHP Manual, Pridobljeno 25. 5. 2009 iz <http://si.php.net/manual/en/index.php>

Goyvaerts J., Regular-Expressions.info, Pridobljeno 26.5.2009 iz <http://www.regular-expressions.info>

Bradley A., About.com: PHP/MySQL, Pridobljeno 26.5.2009 iz <http://php.about.com>

Hazel P, pcrematching man page, Pridobljeno 26.5.2009 iz <http://regexkit.sourceforge.net/Documentation/pcre/pcrematching.html#SEC1>

Gammon N., Perl-compatible Regular Expressions (PCRE), Pridobljeno 26.5.2009 iz <http://mushclient.com/pcre/>

Mertz D., Learning to Use Regular Expressions, Pridobljeno 26.5.2009 iz http://gnosis.cx/publish/programming/regular_expressions.html

Kuchling A., Regular Expression HOWTO, Pridobljeno 26.5.2009 iz <http://www.amk.ca/python/howto/regex/>

Zander J., Abrams B., Heuer T., Stagner J., Glas E. in drugi, Pridobljeno 26.5.2009 iz <http://msdn.microsoft.com/en-us/library/3206d374.aspx>

PHP, Pridobljeno 26.5.2009 iz <http://en.wikipedia.org/wiki/PHP>

PHP Tutorial, Pridobljeno 26.5.2009 iz <http://www.w3schools.com/php/default.asp>

Comparing E-mail Address Validating Regular Expressions, Pridobljeno 26.5.2009 iz <http://fightingforalostcause.net/misc/2006/compare-email-regex.php>

The PHP Group, PHP: Test and Code Coverage Analysis, Pridobljeno 2.6.2009 iz http://gcov.php.net/PHP_5_3/lcov_html/pcre/php_pcre.c.gcov.php

Gervalle D., Quantifiers, Pridobljeno 3.6.2009 iz <http://www.softec.st/en/OpenSource/DevelopersCorner/RegularExpressions/Quantifiers.html>