გ. ღვინეფაძე

WEB-დაპროგრამება PHP

ტექნიკური უნივერსიტეტი

საქართველოს ტექნიკური უნივერსიტეტი

გ. ღვინეფაძე

WEB-დაპროგრამება
PHP

დამტკიცებულია სტუ-ს
სარედაქციო-საგამომცემლო
საბჭოს მიერ

თბილისი
2009

უაკ 681.3.06

განხილულია სერვერის მხარეზე დაპროგრამებისათვის განკუთვნილი სპეციალიზებული ენა PHP-ის ბაზისური საშუალებანი.

განკუთვნილია ინფორმატიკის და მართვის სისტემების ფაკულტეტის 22.02, 22.01, 0719.08 სპეციალობათა შემსწავლელი სტუდენტებისა და ამ საკითხებით დაინტერესებული პირებისათვის.

რეცენზენტი: ასოც. პროფ. ე. თურქია

© გამომცემლობა "ტექნიკური უნივერსიტეტი", 2009

ISBN

P H P 4

შესავალი

არცთუ დიდი ხნის წინ Web-კვანძებიდან მომხმარებელს ადგილზე ძირითადად სტატიკური HTML-ფაილები გადმოეცემოდა, რომელთაც იგი ბროუზერის მეშვეობით მხოლოდ კითხულობდა. დღეს კი WEB-სივრცეში უმეტესწილად დინამიკური WEB-ფურცლები ანუ DHTML-ფაილები გადაიგზავნება. მათი აწყობა ხდება WEB-კვანძების სერვერზე განთავსებული WEB-პროგრამების გაშვებით, რასაც მომხმარებელი ახორციელებს საკუთარი კომპიუტერიდან სისტემასთან დიალოგში.

ინტერნეტში პროგრამების შექმნისათვის დღეს ყველაზე პოპულარული WEB-დაპროგრამების ენა გახლავთ PHP4. იგი განსაკუთრებით აადვილებს სერვერებზე განლაგებულ მონაცემთა ბაზებთან ურთიერთობას.

დასაწყისში PHP (personal home page) შეიქმნა, როგორც მაკროსების კრებული, მარტივი სახის პირადი WEB-ფურცლების ფორმირებისთვის. შემდეგ კი იგი იქცა სრულფასოვან დაპროგრამების ენად, რომელზეც დაწერილი კონსტრუქციები ჩაიდგმება სერვერზე მოთავსებულ HTML-ტექსტებში. WEB-ფურცლის გამოძახებისას PHP-ის პრეპროცესორის მიერ ადგილზე ხდება ამ კონსტრუქციების *(პროგრამული ფრაგმენტების)* დამუშავება და HTML-ის არსებულ ფრაგმენტებს შორის ჩაემატება შესაბამისი ინფორმაცია, მაგალითად, ბაზიდან ამოღებული მონაცემები, სერვერზე კონსტრუირებული გრაფიკული გამოსახულებები და სხვ.

PHP-ის ვერსია - PHP4(5) - მისაერთებელი მოდულის სახით კომპილირდება დღეისათვის ყველაზე პოპულარულ WEB-სერვერ-პროგრამა Apache-სთვის. სერვერზე PHP-პროგრამები უმეტესწილად ურთიერთობენ MySQL მონაცემთა ბაზასთან *(რომელიც უფასოდ ვრცელდება).* ასე რომ, Apache, MySQL და PHP ტრიადას დღევანდელ დღეს კონკურენტი არ ჰყავს. თუმცა, ცხადია, PHP-ს მუშაობა შეუძლია სხვა WEB-გარემოში და სხვა ბაზებთანაც.

საერთოდ, PHP-პროგრამებს ინტერპრეტატორი ამუშავებს, მაგრამ შესაძლებელია მათი კომპილირებაც *(თუმცა ეს შესაძლებლობა მხოლოდ კომერციული დამატების სახით არსებობს),* რაც კიდევ უფრო ამაღლებს სისტემის მწარმოებლურობას.

CGI-სთან შედარებით PHP-ს ის მნიშვნელოვანი უპირატესობა გააჩნია, რომ მისი ინტერპრეტატორი ფუნქციონირებს სერვერთან მჭიდრო ინტეგრაციის პირობებში – საჭირო არ გახლავთ თითოეული პროგრამის შესრულებისას სისტემის ხელახლა გაშვება, რაც მნიშვნელოვნად ამაღლებს მუშაობის სისწრაფეს.

PHP-ს რიგი სხვა ღირსებებიც გააჩნია:

- პროდუქტის ღიაობა;
- პროგრამების დაწერის გაადვილება HTML-ტექსტისა და თვით პროგრამული ნაწილის განცალკევების გამო;
- პროგრამების ერთი ოპერაციული სისტემიდან სხვაში გადატანის სიმარტივე;
- პროგრამების შესრულების საკმარისი სისწრაფე.

P H P – ის დაყენება

PHP-ს უფასოდ ჩამოვტვირთავთ http://www.php.net კვანძიდან.

http://www.php.net/manual/ მისამართზე შესაძლებელია პროდუქტის შესახებ უახლესი, ამომწურავი ინფორმაციის მიღება.

შევისწავლოთ, როგორ ხდება PHP-ის დაყენება Apache Web-სერვერთან სამუშაოდ, კომპიუტერის მართვისას Linux ოპერაციული სისტემის მიერ. *(საერთოდ, შესაძლებელია Apache-სთან უკვე მიერთებული PHP-ის ვარიანტის მოძიებაც, მაგრამ როცა ამ პროცესს თვითონ ვახორციელებთ, არჩევანი უფრო ფართო ხდება – ამ შემთხვევაში პარამეტრებს ჩვენ ვანიჭებთ მნიშვნელობებს).*

სისტემაში PHP-ის დაყენების უფლება მხოლოდ ადმინისტრატორს *(სუპერმომხმარებელს)* აქვს. PHP-ის, როგორც Apache-სთვის მოდულის, კომპილირება ორი გზით შეიძლება განხორციელდეს:

- პირველი გულისხმობს Apache-ის ხელახლა კომპილაციას მასთან PHP-ის სტატიკურად მიერთებით *(ლინკირებით);*
- მეორე გზა ითვალისწინებს მხოლოდ PHP-ის კომპილაციას, ოღონდ მანამდე უნდა განხორციელდეს Apache-ის კომპილაცია DSO რეჟიმში *(Dynamic Shared Object - საერთო სარგებლობის დინამიური ობიექტს აღნიშნავს).* სწორედ, ამ რეჟიმით ვისარგებლებთ PHP-ის კომპილირებისას. ეს გზა PHP-ის დაყენების უფრო მარტივი ხერხია და ჩვენც მას განვიხილავთ.

იმის შესამოწმებლად, უზრუნველყოფს თუ არა Apache აღნიშნულ რეჟიმში მოდულების მიერთებას, საჭიროა შესრულებაზე გავუშვათ httpd ფაილი -1 ალმით:

```
/www/bin/httpd -1
```

გამოდის მოდულების სია. თუ მასში ფიგურირებს mod_so.c, დავასკვნით, რომ DSO რეჟიმში მუშაობა შესაძლებელია *(წინააღმდეგ შემთხვევაში მოგვიწევს Apache-ის ხელახლა კომპილირება).*

PHP4-ის ჩამოტვირთვის შედეგად განკარგულებაში გვეძლევა gzip უტილიტით შეკუმშული Php-4.0.4pll.tar.gz დისტრიბუტიული ფაილი.

მისი გაშლა შემდეგი ბრძანებით ხდება:

```
tar -xvzf php-4.0.4pll.tar.gz
```

გადავდივართ PHP-დისტრიბუტივის კატალოგში:

```
cd../php-4.0
```

შესრულებაზე ვუშვებთ აღნიშნულ კატალოგში არსებულ configure პროგრამას პარამეტრებისათვის ყველაზე უფრო მიღებული მნიშვნელობების მინიჭებით:

```
./configure --enable-track-vars \
            --with-gd \
            --with-mysql \
            --with-apxs=/www/bin/apxs
```

ბოლო მნიშვნელობა იმ კატალოგზე მიუთითებს, რომელშიც Apache Web-სერვერია დაყენებული, mysql ბაზისთვის კი იგულისხმება, რომ იგი დუმილით გათვალისწინებულ კატალოგშია დაყენებული. მაგრამ თუკი ეს

ასე არ გახლავთ, საჭიროა ამ განსხვავებული კატალოგის შესახებ ინფორმაციის გათვალისწინებაც:

--with-mysql=/path/to/dir

configure პროგრამის შესრულების შემდეგ ვუშვებთ make უტილიტას. კომპიუტერზე დაყენებული უნდა იყოს C ენის კომპილატორი.

Apache სერვერის კონფიგურირება

Apache-სერვერის კატალოგის conf-ქვეკატალოგში მოვძებნით httpd.conf ფაილს და დავუმატებთ მას შემდეგ სტრიქონებს:

AddType application / x-httpd-php .php

AddType application / x-httpd-php-source .phps

ეს ნიშნავს, რომ PHP-ინტერპრეტატორი .php გაფართოების ფაილებს დაამუშავებს როგორც პროგრამებს, ხოლო .phps გაფართოების მქონეთ – როგორც გაფერადებულ HTML-ტექსტს.

php.ini ფაილის მეშვეობით კი შესაძლებელია PHP-ის პარამეტრების შეცვლა უკვე მისი კომპილირების და დაყენების შემდეგ. UNIX ამ ფაილს ინახავს /usr/local/lib კატალოგში, ხოლო Windows ოპერაციული სისტემა - Windows-ში.

შესაძლებელია ამ ფაილის მიმდინარე კატალოგში განთავსებაც. მაშინ მას ექნება ლოკალური მოქმედების არეალი. ჯერჯერობით კი ჩვენ ვკმაყოფილდებით ამ ფაილის პარამეტრებისთვის მინიჭებული სტანდარტული მნიშვნელობებით.

php.ini ფაილში შეიძლება მნიშვნელობის განსაზღვრა შემდეგი დირექტივისთვისაც: short_open_tag=On (True,Yes), რაც უზრუნველყოფს php-ბლოკების გაფორმებას <? *ბლოკი* ?> სახით. ამ შესაძლებლობის გამორთვა ხორციელდება დირექტივისთვის Off (False,No) მნიშვნელობის მინიჭებით.

თუ Web-ფურცლების ფორმირებისათვის XML-ენასაც ვიყენებთ, მაშინ ამ ე.წ. მოკლე ტეგების გამოყენება ზემოთ აღნიშნული ხერხითვე უნდა ავკრძალოთ და ვისარგებლოთ მხოლოდ სტანდარტული ტეგებით:

```
<?php
?>
```

ჩვენი პირველი პროგრამა PHP-ზე

Notepad-ტექსტში ავკრიბოთ შემდეგი ტექსტი:

```
<?php
print “Hello Web!”;
?>
```

ფაილი დავიმახსოვროთ first.php სახელით.

თუ სერვერზე არ ვმუშაობთ, მოგვიხდება FTP-სერვისით ფაილის მასზე გადაგზავნა *(კოპირება).* შემდეგ კი ამ ფაილს შეიძლება მივმართოთ ჩვენი ბროუზერიდან. ეკრანზე აისახება წარწერა:

```
Hello Web!
```

მაგრამ თუკი სერვერზე PHP დაყენებული არ გახლავთ, ანდა ფაილის გაფართოება არასწორად იქნა განსაზღვრული, მაშინ მოსალოდნელია ეკრანზე გამოვიდეს შემდეგი ტექსტი:

```
<?php
  Hello Web!
?>
```

print() წარმოადგენს ფუნქციას ჩვენ მიერ დაწერილ უმარტივეს პროგრამაში. თუ მას ეკრანზე სტრიქონი გამოჰყავს, მაშინ ეს სტრიქონი ბრჭყალებში უნდა ჩაისვას *(დასაშვებია ერთმაგი ბრჭყალებიც).*

ვიცით, რომ ფუნქციის არგუმენტები ფრჩხილებში უნდა მოთავსდეს. print ფუნქცია კი ამ მხრივ გამონაკლისია - დასაშვებია მის მიერ გამოსაყვანი ინფორმაცია ფრჩხილებში არ მოვაქციოთ.

PHP-ინსტრუქციები წერტილ-მძიმით უნდა დაბოლოვდეს. გამონაკლისს შეიძლება წარმოადგენდეს ბლოკში ბოლო ინსტრუქცია, თუმცა ეს რეკომენდირებული არ გახლავთ *(საეჭვო შემთხვევებში შეიძლება ასეთმა გადაწყვეტილებამ ინტერპრეტატორი შეცდომაში შეიყვანოს).*

ისევე, როგორც JavaScript-ის სცენარები, PHP-ინსტრუქციების ბლოკებიც შეიძლება უშუალოდ HTML-ტექსტში განლაგდეს:

<u>ლისტინგი 1</u>

```
<html>
<head>
<title>გაუმარჯოს PHP-ისა  და  HTML-ის ძმურ  კავშირს! </title>
</head>
</body>
<?php
   print "წინ, პროგრამირებაში ახალი მწვერვალების დასაპყრობად!";
?>
</body>
</html>
```

PHP-ის ინტერპრეტატორი ამ ფურცელზე მხოლოდ PHP-ბლოკებით "ინტერესდება". მათი ერთობლიობა კი ქმნის ერთიან PHP-პროგრამას. შინაარსობრივად ეს იმას ნიშნავს, რომ პირველ ბლოკში გამოცხადებული ცვლადების, ფუნქციების თუ კლასების გამოყენება მომდევნო ბლოკებშიც შეიძლება.

პროგრამებში ხშირად გამოიყენება <u>კომენტარები</u>:

```
// ეს კომენტარია.

# თქვენ წარმოიდგინეთ, ესეც კომენტარია.

/* ეს კი
  რამდენიმე სტრიქონზე
```

გაწელილი კომენტარია.
*/

კომენტარებში ჩაწერილი ინფორმაცია ინტერპრეტატორის მიერ იგნორირდება.

ცვლადები

PHP-ში ცვლადის სახელი $ სიმბოლოთი იწყება! მომდევნო პოზიციებზე კი შეიძლება გამოვიყენოთ ლათინური სიმბოლოები, ციფრები და ხაზგასმის ნიშანი. სხვა სიმბოლოების, მათ შორის ხარვეზის *(ცარიელი არის)* გამოყენება არ დაიშვება. მოვიყვანოთ ცვლადების სწორად დასახელების მაგალითები:

$ a
$ a_b_c
$ b555

ცვლადებს ენიჭებათ მნიშვნელობები. მნიშვნელობები და შესაბამისად ცვლადებიც სხვადასხვა ტიპის შეიძლება იყოს. კერძოდ, ცვლადებში დასაშვებია შევინახოთ: *რიცხვები, სიმბოლოთა სტრიქონები, ობიექტები, მასივები, ლოგიკური მნიშვნელობები...*

როგორც წესი, ცვლადის შექმნა და მისთვის მნიშვნელობის მინიჭება ერთსა და იმავე ინსტრუქციაში ხდება, მაგალითად:

```
$num1 = 8;
$name = “გია”;
```

ინსტრუქციებში მნიშვნელობამინიჭებული ცვლადის გამოყენება, ფაქტობრივად, ამ მნიშვნელობის გამოყენებაა. მაგალითად, გამოსახულება

print $num1;

ეკრანზე გამოიყვანს იმავე ინფორმაციას (8), რასაც - გამოსახულება print 8;

დინამიკური ცვლადები

PHP4(5)-ში სიახლეა - ცვლადის სახელი შეიძლება დავიმახსოვროთ, როგორც სხვა ცვლადის მნიშვნელობა! კერძოდ, გამოსახულება

$user=”bob”;

შეიძლება შეცვლილ იქნეს შემდეგი ორი გამოსახულებით:

```
$holder=”user”;
$$holder=”bob”;
```

ინსტრუქცია print $$holder; ეკრანზე გამოიყვანს bob მნიშვნელობას. განსხვავებულ შედეგს მოგვცემს ინსტრუქცია

print “$$ holder”;

ამ შემთხვევაში ინტერპრეტატორს ეკრანზე გამოჰყავს ბრჭყალებში მოქცეული ცვლადის მნიშვნელობა ანუ მოცემულ შემთხვევაში $user სტრიქონი. ე.ი. ინტერპრეტატორი, ასე ვთქვათ, მეორე ნაბიჯს აღარ დგამს - $user აღიქვას, როგორც ცვლადი, და გამოგვიყვანოს მისი მნიშვნელობა.

საინტერესოა, რომ დინამიკური ცვლადი შეიძლება გამოცხადდეს სტრიქონის *(სტრიქონული მუდმივის)* მეშვეობითაც *(გამოიყენება განსხვავებული სინტაქსი):*

$ {“user”}=”bob”;

ასეთი გადაწყვეტა, როგორც შემდეგ ვაჩვენებთ, აადვილებს დინამიკური ცვლადების გენერირების პროცესს ციკლებისა და კონკატენციის ოპერატორის გამოყენებით.

ცხადია, ფიგურულ ფრჩხილებში ცვლადიც შეიძლება მოთავსდეს:

$ {$holder}

ქვემოთ პროგრამაში მოყვანილია დინამიკური ცვლადის შექმნის სხვადასხვა მაგალითი.

$holder ცვლადში შენახულია “user” სტრიქონი, რომელიც $$holder დინამიკურ ცვლადს აფორმირებს $user ცვლადის სახით. ამ უკანასკნელს ენიჭება “bob” მნიშვნელობა.

შემდეგ მოყვანილია აღნიშნული ცვლადების მნიშვნელობების ეკრანზე გამოყვანის ასევე სხვადასხვა ხერხი:

<u>ლისტინგი 2</u>

```
<html>
<head>
<title> დინამიკური ცვლადის შექმნა და მისდამი მიმართვა</title>
</head>
<body>
<?php
$holder=”user”;
$$holder=”bob”;

print   “$user<br>”;          //გამოჰყავს ”bob”
print   “$$holder”;           //გამოჰყავს ”bob”
print   “<br>”;
print   “$ {$holder}<br>”;    //გამოჰყავს ”bob”
print   “S {‘user’}<br>” ;    //გამოჰყავს ”bob”
?>
</body>
</html>
```

PHP საშუალებას იძლევა, ერთი ცვლადი მეორეს დაეყრდნოს:

```
<?php
    $ Ziritadi_cvladi=42;
    $ Ziritadze_dayrdnobili=&Ziritadi_cvladi;
    print  $Ziritadze_ dayrdnobili;        // გამოდის  42
?>
```

მონაცემთა ტიპები

მონაცემების დამუშავებისას PHP ითვალისწინებს მათ ტიპს. ეს ენა არ მოითხოვს, რომ თავიდანვე გამოცხადდეს მონაცემთა ტიპი. საჭირო ინფორმაციას PHP იღებს ცვლადისადმი მინიჭებული მნიშვნელობის მიხედვით. შევნიშნოთ, რომ ამგვარ მიდგომას ნაკლიც გააჩნია – არ არის გამორიცხული, დიდი მოცულობის პროგრამებში პროგრამისტს მხედველობიდან გამორჩეს, თუ როგორია ამა თუ იმ ცვლადის ტიპი და შედეგად პროგრამამ არასწორად გამოთვალოს შედეგი.

ჩამოვთვალოთ PHP-ში გათვალისწინებული მონაცემთა ტიპები:

ტიპი	მაგალითი	აღწერა
Integer	5	მთელი რიცხვი
Double	3.111	მცურავწერტილიანი რიცხვი
String	“hello”	სიმბოლოების სტრიქონი
Boolean	true (false)	ლოგიკური ტიპი
Object		ობიექტი
Array		მასივი

პროგრამისტს შეუძლია gettype() ფუნქციის მეშვეობით შეამოწმოს მონაცემთა ტიპი:

<u>ლისტინგი 3</u>

```
<html>
<head>
        <title>ცვლადის ტიპის შემოწმება </title>
</head>
<body>
<?php
$testing = 5;
print gettype ($testing);      // integer
print  “<br>”;
$testing = “five”;
print gettype ($testing );      // string
print (“<br>”);
$testing = 5.0;
print gettype ($testing );      //double
print (“<br>”);
$testing = true;
print gettype ($testing );      //Boolean
print  “<br>”;
?>
</body>
</html>
```

ეს პროგრამა გამოიტანს შემდეგ შეტყობინებებს:

integer
string
double
boolean

მონაცემთა ტიპის შეცვლა

PHP-ში საჭიროების შემთხვევაში შესაძლებელია მონაცემთა ერთი ტიპის მეორეთი შეცვლა.

ლისტინგი 4

```
<html>
<head>
        <title> ცვლადის ტიპის შეცვლა </title>
</head>
<body>
<?php
  $undecided = 3.14;
print gettype( $undecided );                    //double
print  " -- $undecided<br>"                     // 3.14
  settype( $undecided, string );
print gettype ( $undecided );                   // string
print  " -- $undecided<br>";                    // 3.14
  settype( $undecided, integer );
print gettype( $undecided );                    // integer
print  " -- $undecided<br>";                    //3
  settype( $undecided, double );
print gettype( $undecided );                    // double
print  " -- $undecided<br>";                    // 3.0
  settype( $undecided, boolean );
print gettype ( $undecided );                   // boolean
print  " -- $undecided<br>";                    // 1
?>
</body>
</html>
```

აღსანიშნავია, რომ მოცემულ ცვლადზე დაყრდნობით შესაძლებელია შევქმნათ ახალი, მაგრამ განსხვავებული ტიპის მქონე ცვლადი. ცხადია, ბაზისური ცვლადის ტიპი ამ შემთხვევაში უცვლელი რჩება *(ბუნებრივია, მნიშვნელობაც).* ახალი, განსხვავებული ტიპის ცვლადის მნიშვნელობა კი განისაზღვრება ბაზისური ცვლადის მნიშვნელობის მიხედვით.

მოვიყვანოთ ასეთი გარდაქმნების მაგალითი:

ლისტინგი 5

```
<html>
<head>
<title> ცვლადის ტიპის გარდაქმნა </title>
```

```
</head>
<body>
<?php
    $undecided = 3.14;
    $holder = ( double ) $undecided;
print gettype( $holder );                    // double
print  " -- $holder<br>";                    // 3.14
    $holder = ( string ) $undecided;
print gettype( $holder );                    // string
print  " -- $holder<br> ";                   // 3.14
    $holder = ( integer ) $undecided;
print gettype( $holder );                    // integer
print  " -- $holder<br>";                    // 3
    $holder = ( double ) $undecided;
print gettype ( $holder );                   // double
print " -- $holder<br>";                     // 3.14
    $holder = ( boolean ) $undecided;
print gettype ($holder );                    // boolean
print  " -- $holder<br>";                    // 1
?>
</body>
</html>
```

გამოსახულებები PHP4-ში სხვა ენების მსგავსად იქმნება. გამოიყენება

+, -, /, * და % *(გაყოფა მოდულის მიხედვით)* ოპერატორები.

სტრიქონის სტრიქონზე გადაბმა კი აქ წერტილის მეშვეობით ხორციელდება:

```
Print  "hello"."world";
```

ამ ინსტრუქციის შედეგად ეკრანზე აისახება

hello world

სტრიქონი.

მნიშვნელობის მინიჭების ჩვეულებრივი ოპერატორი x=x+4; აქაც შეიძლება სხვაგვარად წარმოგვიდგეს:

```
x  += 4;
```

ცხადია, გამოიყენება

-=, /=, *= და %= ოპერატორებიც.

PHP-ში ფუნქციონირებს .= ოპერატორიც. მაგალითად, $x = $x . "test" გამოსახულება შეიძლება შეიცვალოს $x .= "test" გამოსახულებით.

გამოსახულებებში გამოიყენება ჩვენთვის ნაცნობი შედარების ოპერატორები:

==, !=, >, >=, <, <=

აქ სიახლეს წარმოადგენს === ოპერატორი, რომელიც ამოწმებს ოპერანდების არა მარტო მნიშვნელობების, არამედ მათი ტიპების თანხვედრასაც!

რაც შეეხება ლოგიკურ ოპერატორებს, || და && ოპერატორებთან ერთად გამოიყენება იმავე დანიშნულების მქონე or და and ლოგიკური ოპერატორები *(შევნიშნოთ, რომ პირველებს უფრო მაღალი პრიორიტეტი გააჩნიათ).*

"უარყოფის" ლოგიკურ ოპერაციას ახორციელებს ! ოპერატორი, ხოლო "გამომრიცხავი ან" ოპერაციას *(ნიშნავს, რომ მხოლოდ ერთი ოპერანდია ჭეშმარიტი)* – xor ლოგიკური ოპერატორი.

PHP-ში ვხვდებით აგრეთვე ჩვენთვის უკვე ნაცნობ, სპეციფიკური სახის მქონე მინიჭების ოპერატორებს:

```
$x++
$x--
--$
++$
```

ზოგჯერ საჭიროა, პროგრამაში თავიდან ავიცილოთ ცვლადის მნიშვნელობის და ტიპის შეცვლა. მაშინ ცვლადს მხოლოდ ერთხელ, პროგრამის დასაწყისში მიენიჭება მნიშვნელობა *(ამ მნიშვნელობის მიხედვით - ტიპიც)* და მერე ის აღარ იცვლება. ცვლადის ასეთი ნაირსახეობა ფაქტობრივად კონსტანტაა. PHP-ში მას ასეც უწოდებენ. მისი განსაზღვრისას $ სიმბოლოს არ იყენებენ და სახელს დიდი ასოებისაგან ადგენენ:

<u>ლისტინგი 6</u>

```
<html>
<head>
        <title>კონსტანტის შექმნა </title>
</head>
<body>
<?php
define (  "USER", "Gerald" );
print "Welcome ".USER;
?>
</body>
</html>
```

მივაქციოთ ყურადღება - გამობეჭდვისას კონსტანტა სტრიქონს მივუერთეთ კონკატენციის ოპერატორის მეშვეობით.

აღვნიშნოთ, რომ PHP ენა პროგრამისტს აწვდის ე.წ. <u>*წინასწარგანსაზღვრული კონსტანტების*</u> გამოყენების საშუალებასაც.

მაგალითად: _FILE_ კონსტანტა შეიცავს იმ ფაილის სახელს, რომელსაც მოცემულ მომენტში ამუშავებს ინტერპრეტატორი, _LINE_ კი - ფაილის მიმდინარე სტრიქონის ნომერს და სხვ.

ნაკადის მართვა

პროგრამის შესრულების მიმდინარეობა, სიტუაციიდან გამომდინარე, შეიძლება შეიცვალოს. აქაც, პროგრამირების ნაცნობ ენებთან შედარებას თუ მოვახდენთ, განსაკუთრებულ სიახლეებს არ ვხვდებით.

მოვიყვანოთ IF ინსტრუქციის გამოყენების მაგალითი:

<u>ლისტინგი 7</u>

```
<html>
<head>
        <title> if  ინსტრუქცია </title>
</head>
<body>
<?php
$mood = “sad”;
if ($mood == “happy” )
    {
    print  “მე კარგ გუნებაზე ვარ!”;
    }
?>
</body>
</html>
```

რადგან აქ IF-ის მომდევნო ბლოკი ერთადერთ ინსტრუქციას შეიცავს, შეიძლებოდა ფიგურული ფრჩხილები არც გამოგვეყენებინა.

პროგრამა ოდნავ გავართულოთ else ბლოკის დანიშნულების სადემონსტრაციოდ:

<u>ლისტინგი 8</u>

```
<html>
<head>
        <title> if ინსტრუქცია else ბლოკით </title>
</head>
<body>
<?php
$mood = “დარდიანადა ვაარ”;
if ($mood == “happy” )
    {
    print  “მე კარგ გუნებაზე ვარ!”
    }
else
    {
    print   $mood;
    }
?>
</body>
</html>
```

შემდეგი ნაბიჯია elseif კონსტრუქციასთან გაცნობა, რომელიც if-ინსტრუქციაში, else-ბლოკისგან განსხვავებით, ერთმანეთის მიყოლებით რამდენჯერმეც შეიძლება შეგვხვდეს *(ქვემოთ მოყვანილ მაგალითში კი იგი მხოლოდ ერთგან ფიგურირებს):*

ლისტინგი 9

```
<html>
<head>
        <title> else  და  elseif  ბლოკების გამოყენება </title>
</head>
<body>
<?php
$mood = “sad”;
if ($mood == “happy” )
    {
    print  “მე კარგ გუნებაზე ვარ!”;
    }
elseif ( $mood == “sad” )
    {
    print  “არ იდარდო!”;
    }
else
    {
    print  “გაუგებარია ეს  $mood ”;
    }
?>
</body>
</html>
```

პროგრამის შესრულების მიმდევრობა შეიძლება შევცვალოთ ჩვენთვის უკვე ნაცნობი switch-ინსტრუქციის მეშვეობითაც:

ლისტინგი 10

```
<html>
<head>
        <title>switch ინსტრუქცია </title>
</head>
<body>
<?php
    $mood = “sad”;
switch ( $mood )
    {
    case “happy”:
        print  “მე კარგ გუნებაზე ვარ!”;
        break;
    case “sad”:
        print  “არ იდარდო!”;
```

```
        break;
    default:
        print  “გაუგებარია ეს $mood ”;
    }
?>
</body>
</html>
```

დიდი პოპულარობით არ სარგებლობს, მაგრამ უნდა ვიცნობდეთ ? ოპერატორსაც:

(პირობის-შემოწმება)? გამოსახულება–1–დადებითი–პასუხისას:
გამოსახულება–2–უარყოფითი–პასუხისას;

ვაჩვენოთ მისი ფუნქციონირება შემდეგი პროგრამის მაგალითზე:

ლისტინგი 11

```
<html>
<head>
        <title> ? ოპერატორი  </title>
</head>
<body>
<?php
$mood = “sad”;
$text = ($mood == ”happy”)?  “მე კარგ გუნებაზე ვარ!”:  “ხასიათზე ვერ ვარ”;
print  “$text”;
?>
</body>
</html>
```

ციკლები

კიდევ ერთხელ გადავავლოთ თვალი ჩვენთვის ნაცნობ ციკლებს:

ციკლი while

ლისტინგი 12

```
<html>
<head>
        <title>  while ციკლი </title>
</head>
<body>
<?php
    $counter = 1;
while ( $counter <= 12 )
    {
    print  “$counter გავამრავლოთ ორზე, იქნება ”. ($counter*2). “<br>”;
    $counter++;
    }
?>
```

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

ციკლი do… while

ლისტინგი 13

```
<html>
<head>
<title> do… while ციკლი</title>
</head>
<body>
<?php
   $num = 1;
do
   {
   print "გატარების ნომერია: $num<br>\n";
   $num++;
   }
while ( $ > 200  &&  $num < 400 );
?>
</body>
</html>
```

ხაზი უნდა გავუსვათ მოთხოვნას – ციკლის ბოლოს აუცილებლად უნდა დავსვათ ; სიმბოლო.

გავიხსენოთ, რომ do … while ციკლის გამოყენებით ციკლის სხეულში მოცემული ინსტრუქციები ერთხელ მაინც სრულდება, თუნდაც არ კმაყოფილდებოდეს ტესტური პირობა *(ზემოთ მოყვანილ პროგრამაში ზუსტად ასეთი რამ ხდება).*

For ციკლი

რაც ამ ციკლის მეშვეობით ხორციელდება, do … while ციკლითაც მიიღწევა. მაგრამ, რადგანაც for ციკლი უფრო ადვილად აღსაქმელია, ხშირად მას აძლევენ უპირატესობას.

ქვემო მაგალითში 2-ზე მრავლდება პირველი 12 ნატურალური რიცხვი:

ლისტინგი 14

```
<html>
<head>
<title> for ციკლი </title>
</head>
<body>
<?php
for ($counter = 1; $counter <=12; $counter ++ )
   {
   print "$counter-ის ორზე გამრავლებით მივიღებთ ".($counter*2). "<br>";
   }
```

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

ახლა კი ვაჩვენოთ for ციკლში break და continue ინსტრუქციების გამოყენების მაგალითები:

ლისტინგი 15

```
<html>
<head>
        <title> break ინსტრუქციის გამოყენება </title>
</head>
<body>
<?php
for ($counter = -4;  $counter <=10;  $counter++)
    {
    if ($counter == 0)
      break;
    $temp = 4000 / $counter;
    print  “40000-ის  $counter-ზე  გაყოფით მივიღებთ $temp-ს<br>”;
    }
?>
</body>
</html>
```

როცა $counter ცვლადის მნიშვნელობა ნულის ტოლი გახდება, ციკლი შეწყდება.

ლისტინგი 16

```
<html>
<head>
        <title>  continue ინსტრუქციის  გამოყენება </title>
</head>
<body>
<?php
$counter = -4;
for ( ;  $counter <= 10;  $counter++ )
    {
    if ( $counter == 0 )
        continue;
    $temp = 4000/$counter;
    print  “თუ 4000-ს  გავყოფთ  $counter-ზე, მივიღებთ  ”. $temp.”-ს”.“<br>”;
    }
?>
</body>
</html>
```

ერთმანეთში ჩადგმული ციკლები

ციკლის სხეული შეიძლება სხვა ციკლსაც შეიცავდეს. ასეთი კონსტრუქციის გამოყენება განსაკუთრებით ხელსაყრელია ცხრილებთან მუშაობისას.

მოვიყვანოთ ჩადგმული ციკლების მაგალითი, რომელთაც ბროუზერის ეკრანზე გამოჰყავს გამრავლების ცხრილი:

ლისტინგი 17

```
<html>
<head>
        <title> ერთმანეთში ჩადგმული for-ციკლები </title>
</head>
<body>
<?php
print  “<table border=1\n>”;
for ( $y=1;  $y<=12;  $y++ )
    {
    print  “<tr>\n”;
    for ( $x=1;  $x<=12;  $x++ )
        {
          print  “\t<td>”;
          print ($x*$y);
          print  “</td>\n”;
        }
    print  “</tr>\n”;
    }
print  “</table>”;
?>
</body>
      </html>
```

შევნიშნოთ, რომ print ინსტრუქციებით ზემოთ მოყვანილ პროგრამაში ხდება HTML-ენის კონსტრუქციების *(მოცემულ შემთხვევაში ცხრილის)* აგება და შემდგომ ეკრანზე მისი ასახვა.

გარე ციკლი უზრუნველყოფს $y ცვლადის მიერ (1 – 12) დიაპაზონში მნიშვნელობების მიღებას. თითოეულ იტერაციაში ეკრანზე აისახება <TR> ტეგი *(ცხრილის სტრიქონი)* და გაიშვება შიდა ციკლი, რომელიც $x ცვლადს ასევე თანმიმდევრულად ანიჭებს მნიშვნელობებს 1-დან 12-ის ჩათვლით, გამოჰყავს <TD> ტეგი *(ცხრილის უჯრედი)* და მასში ათავსებს $x-ის ნამრავლს $y-ზე. პროგრამის მუშაობის საბოლოო შედეგად მიიღება გამრავლების ცხრილი.

ფუნქციები

ნებისმიერი, მეტ-ნაკლებად "სერიოზული" პროგრამა, როგორც წესი, ფუნქციებს იყენებს.

ფუნქცია შეიძლება ავტომატს შევადაროთ. ფუნქციას მიეწოდება გარკვეული "ნედლეული" - მონაცემები, რომლებსაც იგი გადაამუშავებს და იძლევა შედეგს, რომელიც შეიძლება ეკრანზე გამოვიტანოთ ჩვენთვის დაბრუნებული, მის მიერ გამოთვლილი მნიშვნელობის სახით. ხშირად ფუნქცია ორივე აღნიშნულ ქმედებას თვითონვე ახორციელებს.

განსაკუთრებით სასურველია ფუნქციების გამოყენება მრავალჯერ განმეორებადი მოქმედებების შესასრულებლად.

<u>ფუნქცია</u> განიმარტება, როგორც ინსტრუქციების ბლოკი, რომელიც სრულდება პროგრამის მიერ მისი გამოძახების შემთხვევაში.

ფუნქციები ორი სახისაა: *სისტემაში (PHP-ში) ჩაშენებული* და *მომხმარებლის მიერ შექმნილი.*

უმეტესწილად ფუნქციას მიეწოდება არგუმენტი *(არგუმენტები).* ისინი უნდა განლაგდნენ მრგვალ ფრჩხილებში *(როგორც უკვე ვიცით, გამონაკლისს წარმოადგენს print ფუნქცია).* იმ იშვიათ შემთხვევებშიც კი, როცა რომელიმე ფუნქცია არგუმენტებს არ საჭიროებს, ფრჩხილების დასმა მაინც აუცილებელია. სიაში არგუმენტები ერთმანეთისაგან მძიმით გამოიყოფა.

ფუნქცია გვიბრუნებს მნიშვნელობას. მაგალითად, abs() ფუნქცია უკან აბრუნებს მისთვის გადაცემული რიცხვის აბსოლუტურ *(არაუარყოფით)* მნიშვნელობას. საინტერესოა, რომ მნიშვნელობას აბრუნებს print() ფუნქციაც – ეს გახლავთ true ან false, რათა პროგრამამ შეიტყოს, წარმატებით შესრულდა თუ არა ეს ოპერაცია.

<u>ლისტინგი 18</u>

```
<html>
<head>
        <title> abs() ჩაშენებული ფუნქციის გამოძახება </title>
</head>
<body>
<?php
  $num = -321;
  $newnum = abs($num);

  print $newnum;    // ეკრანზე გამოდის "321"
?>
</body>
</html>
```

ფუნქციას ვქმნით function საკვანძო სიტყვის მეშვეობით, მისი სახელისა და არგუმენტების სიის განსაზღვრით და ფიგურულ ფრჩხილებში ფუნქციის სხეულის აღწერით:

ლისტინგი 19

```
<html>
<head>
        <title>  ფუნქციის  შექმნა </title>
</head>
<body>
<?php
function bighello()
  {
  print  “<h1>HELLO</h1>”; }
bighello();
?>
</body>
</html>
```

ამ კონკრეტულ შემთხვევაში ფუნქციას არგუმენტები არ გადაეცემა, მაგრამ მისი სახელის შემდეგ მრგვალი ფრჩხილების ჩვენება მაინც აუცილებელია.

ახლა კი ვაჩვენოთ არგუმენტის შემცველი ფუნქციის მაგალითი:

ლისტინგი 20

```
<html>
<head>
        <title>  არგუმენტიანი ფუნქციის  შექმნა </title>
</head>
<body>
<?php
function printBR($txt);
  {
  print  (“$txt<br>\n”);
  }
print  (“ეს არის სტრიქონი”);
print  (“ეს მომდევნო სტრიქონია”);
print  (“აი, კიდევ ერთი სტრიქონი”);
?>
</body>
</html>
```

შემდეგი ნაბიჯი იქნება ისეთი ფუნქციის შექმნა, რომელიც მნიშვნელობას გვიბრუნებს:

ლისტინგი 21

```
<html>
<head>
        <title>  მნიშვნელობის  დამბრუნებელი  ფუნქციის  შექმნა </title>
</head>
<body>
```

```
<?php
function addNums($firstnum, $secondnum);
  {
  $result = $firstnum + $secondnum;
  return $result;
  }
print  addNums(3,5);        // დაიბეჭდება  8
?>
</body>
</html>
```

return ოპერაციას შეუძლია შედეგი სხვადასხვა სახით დააბრუნოს:

```
return 4;
return ($a / $b);
return  (function($ argument));
```

დასაშვებია ფუნქციის დინამიკურად გამოძახებაც. მაგალითად, მომხმარებელს შეუძლია კომპიუტერთან დიალოგში გადაწყვიტოს, რომელი ფუნქცია უნდა შესრულდეს; მის მიერ მიწოდებული ინფორმაციის საფუძველზე მოხდება გამოსაძახებელი ფუნქციის სახელის ფორმირება.

<u>ლისტინგი 22</u>

```
<html>
<head>
        <title>  ფუნქციის დინამიკურად გამოძახება </title>
</head>
<body>
<?php
function sayHello( );
        /* აქ პროგრამის გასამარტივებლად ვგულისხმობთ, რომ "sayHello"
        სტრიქონი პროგრამას გარედან მიეწოდება მომხმარებლის მიერ
        დიალოგის რეჟიმში */
  {
  print  ("Hello<br>");
  }
$function_holder =  ("sayHello");
$function_holder =  ( );
?>
</body>
</html>
```

ცვლადების ხილვადობის უბნები

ფუნქციაში შექმნილი ცვლადი მხოლოდ მის არეალში მოქმედებს – იგი <u>ლოკალური</u> მოქმედებისაა. ფუნქციის შიგნით ავტომატურად ვერ მივმართავთ მის გარეთ შექმნილ ცვლადსაც. ასეთი მიდგომები თავიდან გვაცილებს სხვადასხვა ადამიანების მიერ ერთი დიდი პროგრამის შექმნისას მოსალოდნელ გაუგებრობებს. მაგრამ ძალიან ხშირად საჭირო

ხდება ისეთი ცვლადების შექმნაც, რომელთაც პროგრამის ნებისმიერი უბნიდან შეიძლება მივმართოთ. ანუ დგება საკითხი ე.წ. <u>გლობალური</u> მოქმედების ცვლადების შექმნისა. ამ მიზნით, PHP4 იყენებს global ინსტრუქციას:

<u>ლისტინგი 23</u>

```
<html>
<head>
        <title> გლობალური ცვლადების გამოყენება </title>
</head>
<body>
<?php
$life = 42;
function meaningOfLife( );
  {
  global  $life;
  print  “The meaning of  life is $life<br>”;
  }
meaningOfLife( );
?>
</body>
</html>
```

საკითხის ასეთი გადაწყვეტის შედეგად ეკრანზე აისახება შეტყობინება:

The meaning of Life is 42

გამოძახებებს შორის ფუნქციის მდგომარეობის დამახსოვრება

დავუშვათ, გვესაჭიროება ისეთი ფუნქციის შექმნა, რომელსაც ეხსომება, თუ რამდენჯერ მოხდა მისი გამოძახება, ამასთან, გვსურს გამოძახებათა რიცხვი ეკრანზეც ავსახოთ. სწორედ, აქ შეიძლება წარმატებით გამოვიყენოთ global ინსტრუქცია:

<u>ლისტინგი 24</u>

```
<html>
<head>
        <title>ცვლადის მნიშვნელობის შენახვა გამოძახებებს შორის</title>
</head>
<body>
<?php
$num_of_calls = 0;
function andAnotherThing($txt );
  {
  global  $num_of_calls;
  $num_of_calls++;
  print  (“<h1> $num_of_calls. $txt</h1>”;
  }
```

```
andAnotherThing( “ Widgets” );
andAnotherThing( “ Doodads” );
?>
</body>
</html>
```

ვხედავთ, რომ აქაც წინა მაგალითის მსგავს ხერხს ვიყენებთ. ფუნქციის გარეთ შექმნილ ცვლადს რომ მივწვდეთ, ფუნქციაში იგივე ცვლადი global ინსტრუქციის მეშვეობით უნდა გამოვაცხადოთ.

მაგრამ ასეთ მიდგომას ერთი მნიშვნელოვანი ნაკლი ახასიათებს. ფუნქცია, რომელიც global ინსტრუქციას იყენებს, არ შეიძლება დამოუკიდებელ ბლოკად ჩაითვალოს, რადგანაც მას მუშაობა შეუძლია მხოლოდ მოცემული პროგრამის ფარგლებში. საკითხი შეიძლება სხვაგვარადაც გადაწყდეს, მხოლოდ ამ შემთხვევაში უნდა ვისარგებლოთ static ინსტრუქციით.

Static-ით შექმნილი ცვლადი ლოკალურობის თვისებას ინარჩუნებს, ოღონდ ამ ცვლადებისთვის დამახასიათებელი ღირსება ის გახლავთ, რომ ფუნქციისადმი ყოველი მიმართვისას მათ ახსოვთ წინა გამოძახებისას მიღებული მნიშვნელობა:

ლისტინგი 25

```
<html>
<head>
<title> STATIC ინსტრუქცია </title>
</head>
<body>
<?php
 function andAnotherThing ($txt)
 {
    static $num_of_calls = 0;
    $num_of_calls++;
    print "<h1>$num_of_calls. $txt</h1>";
 }
 andAnotherThing ("Widgets");
 andAnotherThing ("Doodads");
?>
</body>
</html>
```

ასეთი ფუნქცია კი უკვე დამოუკიდებელი სახის გახლავთ. ახლა საჭირო აღარაა გლობალური ცვლადების შექმნაზე ზრუნვა, შედეგი კი ისეთივე იქნება, როგორსაც წინა პროგრამა უზრუნველყოფდა.

კვლავ არგუმენტების შესახებ; მათი მნიშვნელობის განსაზღვრა დუმილის პრინციპით

ფუნქციის გამოძახებისას ამა თუ იმ არგუმენტის მნიშვნელობა შესაძლებელია დუმილის წესითაც განისაზღვროს:

<u>ლისტინგი 26</u>

```
<html>
<head>
<title>ფუნქცია არასავალდებულო არგუმენტით</title>
</head>
<body>
<?php
 function fontWrap($txt, $size=3)
    {
print  "<font size=\"$size\"face=\"Helvetica, Arial, Sans-Serif\">$txt</font>";
    }
 fontWrap("A heading<br>",5);
 fontWrap("some body text<br>");
 fontWrap("some more body text <br>");
 fontWrap("yet more body text <br>");
?>
</body>
</html>
```

ფუნქციის იმ გამოძახებებში, რომლებშიც მითითებული არ არის მეორე არგუმენტი, შრიფტის ზომად აირჩევა ფუნქციის განსაზღვრისას ნაჩვენები 3-ის ტოლი მნიშვნელობა. PHP4-ში ასეთი მიდგომას ერთი უხერხული მხარეც ახლავს თან. თუკი ფუნქციის გამოძახებისას რომელიმე არგუმენტი გამოვტოვეთ, მისგან მარჯვნივ მყოფი სხვა არგუმენტებისთვისაც დუმილით გათვალისწინებული მნიშვნელობებით უნდა დავკმაყოფილდეთ.

დაყრდნობის გზით არგუმენტის გადაცემა

როცა ფუნქციას არგუმენტად რომელიმე ცვლადს გადავცემთ, ცხადია, ფუნქციის შესრულება არავითარ ზეგავლენას არ ახდენს ადგილზე ცვლადის მნიშვნელობაზე. მაგრამ შესაძლებელია ფუნქციას ცვლადი გადავცეთ დაყრდნობის ხერხის გამოყენებითაც. ასეთ შემთხვევაში ფუნქციაში არგუმენტზე განხორციელებული ცვლილებები ადგილზე არსებულ ცვლადებზეც აისახება. უკვე ვიცით, რომ ცვლადის *(ამ შემთხვევაში არგუმენტის)* ცვლადზე დასაყრდნობად ვიყენებთ & სიმბოლოს. ამასთან, აღნიშნული ხერხის გამოყენება შესაძლებელია როგორც ფუნქციის გამოძახებისას, ასევე მისი განსაზღვრის დროსაც.

ქვემოთ ნაჩვენებია ეს ორივე ვარიანტი:

ლისტინგი 27

```
<html>
<head>
<title>ფუნქციის გამოძახებისას არგუმენტის გადაცემა დაყრდნობის მეშვეობით</title>
</head>
<body>
<?php
 function addFive($num)
 {
    $num += 5;
 }
 $orignum = 10;
 addFive(&$orignum);
 print $orignum;
?>
</body>
</html>
```

ლისტინგი 28

```
<html>
<head>
<title>დაყრდნობის მეშვეობით არგუმენტის გადაცემა</title>
</head>
<body>
<?php
 function addFive($num)
 {
    $num += 5;
 }
 $orignum = 10;
 addFive($orignum);
 print $orignum;
?>
</body>
</html>
```

შევნიშნოთ, რომ უფრო “პროგრამაგენურად” გამოიყურება მეორე მიდგომა.

მასივები

მასივი ერთი სახელის მქონე მნიშვნელობათა კრებულია, რომელთაგან თითოეულს შეიძლება მივმართოთ ინდექსის ან ტექსტური სტრიქონის მეშვეობით.

დავუშვათ, პროგრამაში საჭიროა, ინახებოდეს რაიმე 100 მნიშვნელობა. შეიძლებოდა თითოეული მათგანისთვის განგვესაზღვრა

ცვლადი *(შესაბამისი მნიშვნელობით),* მაგრამ გაცილებით აადვილებს ამ მნიშვნელობებთან მუშაობას მათი მასივის სახით გაფორმება *(მაგალითად, სორტირებისას, ციკლების გამოყენებისას და ა.შ.).* მასივიდან ელემენტის ამორჩევა უმეტესწილად ნომრის ჩვენებით ხდება *(სხვა შემთხვევები ქვემოთ იქნება განხილული).* PHP-ში მასივის პირველი ელემენტის ნომერია არა "1", არამედ "0". *(გავიხსენოთ, რომ JavaScript-შიც ეს გადაწყვეტილება იყო მიღებული).*

მასივის ელემენტებისთვის მნიშვნელობების მინიჭება ამავე დროს მათ გამოცხადებასაც ნიშნავს. მნიშვნელობების მინიჭება ორი ხერხით ხორციელდება:

I - $users = array ("გია", "მზია", "ლია", "გიგი");

II - $users[] "გია";

$users[] "მზია";

$users[] "ლია";

$users[] "გიგი";

ცხადია, მეორე შემთხვევაში შესაძლებელი იყო ინდექსების ჩვენებაც, მაგრამ მასივის შექმნისას ეს არაფერს არ გვაძლევს, გარდა საქმის გართულებისა.

მასივს მისი შექმნის შემდეგაც შეგვიძლია დავუმატოთ ელემენტები *(ინდექსის ჩვენებით ან მის გარეშეც):*

$users = array ("გია", "მზია", "ლია", "გიგი");

$users[] = "ჯონი";

$users[] = "ჯიმი";

ასოცირებული მასივები

ასოცირებული მასივი ისეთი მასივი გახლავთ, რომლის ელემენტის არჩევა სახელის მეშვეობით შეგვიძლია. დაპროგრამების სხვა ენებში ამგვარ მასივებს **სტრუქტურებს** უწოდებენ.

PHP-ში, სხვა მხრივ, ჩვეულებრივ და ასოცირებულ მასივებს შორის არავითარი განსხვავება არ არსებობს. თუმცა, ცხადია, ასოცირებული მასივების აღნიშნული თავისებურება მათი დამუშავებისას განსხვავებულ მიდგომებს მოითხოვს.

ასოცირებული მასივების შექმნაც ზემოთ მოყვანილი ორივე ხერხით არის შესაძლებელი:

$character = array (saxeli => "გია",

gvari => "ლომიძე",

asaki => 22,

"damatebiti informacia" => "ნიანგებზე მონადირე");

მივაქციოთ ყურადღება – თუ მასივის ელემენტის სახელში შუალედები გვხვდება, საჭიროა ეს სახელი ბრჭყალებში მოვათავსოთ.

ახლა უკვე შეგვიძლია მივმართოთ მასივის ნებისმიერ ელემენტს, მაგალითად, ამგვარად:

```
print $character[asaki];
```

ვაჩვენოთ მეორე გზაც:

```
$character [saxeli] = ”გია”,
$character [gvari] = ”ლომიძე”,
$character [asaki] = 22,
$character [“damatebiti informacia”] = ”ნიანგებზე მონადირე”
```

მრავალგანზომილებიანი მასივები

მასივის ელემენტის როლში შეიძლება გამოვიდეს როგორც რაიმე ტიპის მნიშვნელობა, ასევე ობიექტი და მასივიც კი.

მრავალგანზომილებიანი მასივი ისეთი მასივი გახლავთ, რომლის თითოეული ელემენტი მასივს წარმოადგენს. მისი მეშვეობით ადვილად შეიძლება შევქმნათ მონაცემთა საკმაოდ რთული სტრუქტურები, მაგალითად, ასოცირებული მასივების მასივი:

<u>ლისტინგი 29</u>

```
<html>
<head>
<title>მრავალგანზომილებიანი მასივის შექმნა</title>
</head>
<body>
<?php
 $characters = array (
            array ( name=>"bob",
              occupation=>"superhero",
              age=>30,
              speciality=>"x-ray vision" ),
            array ( name=>" sally",
              occupation=>"superhero",
              age=>24,
              speciality=>"superhuman strength"),
            array ( name=>"Mary",
              occupation=>"arch villain",
              age=>63,
              speciality=>"nanotechnology" )
              );
print $characters[0] [occupation];
// prints "superhero"
?>
</body>
</html>
```

ზოგჯერ ჩვენთვის უცნობია მასივში ელემენტების რიცხვი. ასეთ შემთხვევაში შეგვიძლია ვისარგებლოთ count() ფუნქციით, რომელიც მასივის ელემენტების რიცხვს გვიბრუნებს. მაგალითად, როცა გვსურს ამოვბეჭდოთ უცნობი სიგრძის მასივის ბოლო ელემენტი, ასე შეიძლება ვიმოქმედოთ:

```
$users = array  (“გია”, “ლია”, “მზია”, “გიგი”);
    print  $users[count($users) - 1];
```

გადმოცემული ზომის 1-ით შემცირება გამოწვეულია იმ მიზეზით, რომ მასივის ელემენტების დანომრვა 0-დან იწყება!

მასივის ელემენტების ეკრანზე გამოყვანა შეიძლება განვახორციელოთ ციკლების მეშვეობით. ყველაზე მარტივი ხერხია სპეციფიკური foreach ციკლის გამოყენება, რომლისთვისაც საჭირო არ გახლავთ მითითება, თუ რამდენი ელემენტია მასივში. სამაგიეროდ, აუცილებელია იმ ცვლადის ჩვენება, რომელშიც დროებით შეინახება მასივის თითოეული ელემენტი. მოვიყვანოთ ამ ციკლის გამოყენების მაგალითი:

```
$users = array  (“გია”, “ლია”, “მზია”, “გიგი”);
foreach ($users  as  $val)
 {
print  “$val<br>” ;
 }
```

ასოცირებული მასივის ციკლში ჩათვალიერება კი რამდენადმე განსხვავებულ მიდგომას მოითხოვს:

```
foreach ($array  as  $saxeli => $mniSvneloba)
```

აქ საჭირო ხდება მასივის ელემენტების სახელისა და მნიშვნელობების დროებით შემნახველი ცვლადების განსაზღვრა.

მოვიყვანოთ მაგალითი:

<u>ლისტინგი 30</u>

```
<html>
<head>
<title>ასოცირებული მასივის ჩათვალიერება</title>
</head>
<body>
<?php
 $character = array (
            name=>"bob",
            occupation=>"superhero",
            age=>30,
            "special power"=>"x-ray vision"
                        );
  foreach ( $character as $key=>$val )
  {
   print "$key = $val<br>";
  }
```

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

პროგრამის შესრულების შედეგად ეკრანზე გამოდის შემდეგი ინფორმაცია:

```
name = bob
occupation = superhero
age = 30
special power = x-ray vision
```

მრავალგანზომილებიანი მასივის გამოყენა

ზემოთ განხილული მეთოდების გამოყენებით ახლა ჩვენ უკვე შეგვიძლია ეკრანზე ავსახოთ მრავალგანზომილებიანი მასივებიც:

ლისტინგი 31

```
<html>
<head>
<title>ციკლით ჩათვალიერება</title>
</head>
<body>
<?php
 $characters = array (
            array ( name=>"bob",
              occupation=>"superhero",
              age=>30,
              speciality=>"x-ray vision" ),
            array ( name=>" sally",
              occupation=>"superhero",
              age=>24,
              speciality=>"superhuman strength"),
            array ( name=>"Mary",
              occupation=>"arch villain",
              age=>63,
              speciality=>"nanotechnology" )
              );
 foreach ( $characters as $val )
   {
   foreach ( $val as $key=>$final_val )
       {
       print "$key: $final_val<br>";
       }
     print "<br>";
     }
?>
</body>
</html>
```

ეკრანზე გამოვა ასოცირებული მასივების მასივი:

```
name: bob
        occupation: superhero
         age: 30
         speciality: x-ray vision

         name: sally
         occupation: superhero
         age: 24
         speciality: superhuman strength

        name: Mary
        occupation: arch villain
        age: 63
        speciality: nanotechnology
```

ამ პროგრამაში გარე ციკლი ჩაათვალიერებს ნომრების მიხედვით მოწესრიგებულ $users მასივს და მის თითოეულ ელემენტს მოათავსებს $val ცვლადში. თითოეული ამ ელემენტთაგანი თვითონ წარმოადგენს მასივს, მაგრამ რადგანაც ეს მასივი ასოცირებული გახლავთ, შიდა ციკლით მისი ჩათვალიერებისას გამოიყენება ზემოთ განხილული შემდეგი სინტაქსი - თითოეული ქვეელემენტის სახელი და მნიშვნელობა განთავსდება $key და $final_val ცვლადებში.

მასივების გაერთიანება

მასივების გაერთიანებას ემსახურება array_merge ჩაშენებული ფუნქცია:

```
$first = array ("a", "b", "c");
$second = array ("1", "2", "3");
$third = array_merge ($first, $second);
foreach ($third as $val)
   {
     print "$val<BR>";
     }
```

მასივისთვის ელემენტების დამატება

მასივს ელემენტები array_push() ფუნქციის მეშვეობითაც შეგვიძლია დავუმატოთ *(ცხადია, ეს სხვა ოპერაციაა, ვიდრე მასივების გაერთიანება, რაც თავდაპირველი მასივების უცვლელად დატოვებას გულისხმობს).*

მაგალითად:

```
$first=array ("a", "b", "c");
$total=array_push($first, 1,2,3);
```

საინტერესოა, რომ ამ დროს ხდება არა მარტო $first მასივისთვის სამი ახალი ელემენტის დამატება, არამედ $total ცვლადისათვის

მოდიფიცირებული მასივის სიგრძის სიდიდის მინიჭებაც. ამოვბეჭდოთ ეს მნიშვნელობა და ეკრანზე ავსახოთ მასივის ელემენტები:

```
print "\$first მასივში სულ არის $total ელემენტი <p>");
foreach ($first as $val)
  {
   print "$val<BR>";
   }
```

ზედა მაგალითში გამოყენებულია მასკირების ხერხი – "\" სიმბოლოს დასმა "$" სიმბოლოს წინ ინტერპრეტატორს აცნობებს, რომ დახრილი ხაზის მომდევნო სიტყვა $first მან უნდა აღიქვას არა როგორც ცვლადი, არამედ როგორც "$first" სტრიქონი.

მასივის პირველი ელემენტის ამოგდება

აღნიშნულ მიზანს ემსახურება array_shift() ფუნქცია.

ქვემოთ პროგრამაში ციკლის მეშვეობით ხდება მასივისთვის პირველი ელემენტის მოცილებები მანამ, სანამ მასივი არ დაცარიელდება, რასაც ამოწმებს count() ფუნქცია. ამოგდებული ელემენტის მნიშვნელობა ენიჭება $val ცვლადს. ციკლის თითოეულ ბიჯზე ხდება ამ ცვლადისა და მასივში დარჩენილი ელემენტების რიცხვის ამობეჭდვა.

```
<?php
$an_array = array ( "a",  "b",  "c"  );
 While  ( count ($an_array))
  {
     $val = array_shift ($an_array);
     print  "$val<BR>";
     print  " \ $an_array მასივში არის ". count ($an_array) . "ელემენტი<BR>"
  }
?>
```

მასივიდან ქვემასივის მიღება

ეს პროცესი ხორციელდება array_slice() ფუნქციის მეშვეობით. საწყისი მასივი, ცხადია, არ იცვლება:

```
$first = array ( "a",  "b",  "c", "d",  "e",  "f" );
$second = array_slice ( $first, 2, 3 );
foreach ( $first as $val ) {
                              print  "$val<BR>";
                             }
```

მასივების სორტირება

მარტივი მასივების სორტირებას ახდენს sort() ფუნქცია, ასოცირებულებისას - asort(). მოვიყვანოთ asort() ფუნქციის გამოყენების მაგალითი:

```
$an_array = array ( “x”,  “a”,  “f”, “c”  );
sort ( $an_array );
foreach ($an_array as $val )
      {
        print  “$val<BR>”;
       }
```

შესაძლებელია ასოცირებული მასივის სორტირება ველების სახელების მიხედვითაც, რაშიც გვეხმარება ksort() ფუნქცია:

```
$first = array ( “x”=>5,  “a”=>5, “f”=>5);
ksort ($first);
foreach $first as $key=>$val )
      {
        print  “$key = $val<BR>”;
       }
```

ობიექტები

დაპროგრამებაში შემდგომი, მნიშვნელოვანი ნაბიჯი იქნა წინ გადადგმული, როცა შემოღებულ იქნა ობიექტის ცნება.

ობიექტი არის რაიმე არსის **თვისებების** და ამ თვისებების დამამუშავებული ფუნქციების - **მეთოდების** კრებული.

დაპროგრამების პროცესი მრავალი ნიუანსის გათვალისწინებას მოითხოვს. ობიექტები თავის თავზე იღებენ “შავ” სამუშაოს და თან მომხმარებელს აწვდიან მათთან ინტერფეისის დამყარების მოხერხებულ საშუალებებს.

თვით ობიექტები წარმოგვიდგება, როგორც გარკვეული კლასის კონკრეტული ეგზემპლარები. ამრიგად, კლასი გამოდის სპეციალური შაბლონის როლში, რომლის მიხედვითაც შესაძლებელია შეიქმნეს ობიექტი.

მაშასადამე, ობიექტების შექმნამდე უნდა გამოვაცხადოთ კლასი - განვსაზღვროთ ის თვისებები და მეთოდები, რომლებიც დაახასიათებენ აღნიშნულ კლასს და, შესაბამისად, მის ბაზაზე შექმნილ კლასის ეგზემპლარებს – ობიექტებს.

ობიექტებისთვის კი ხდება თვისებების კონკრეტული მნიშვნელობების განსაზღვრა.

შევნიშნოთ, რომ თვისებებისთვის მნიშვნელობების მინიჭება, დუმილის პრინციპით, კლასის გამოცხადების დროსაც შეიძლება. ამასთან,

დასაშვებია, რომ ობიექტისთვის ყოველი თვისების მნიშვნელობის განსაზღვრა არც მოხდეს.

გამოვაცხადოთ რაიმე მარტივი კლასი და მის საფუძველზე განვსაზღვროთ კლასის ეგზემპლარები - ობიექტები:

```
class first_class
   {
    var $name="harry";
   }
$obj1=new first_class();
$obj2=new first_class();
// პირველი ობიექტის name თვისებას მივანიჭოთ მნიშვნელობა.
// ყურადღება მივაქციოთ თვისების განსაზღვრის წესს!
$obj1 -> name="bob";
// ეკრანზე ავსახოთ ობიექტების თვისების მნიშვნელობები.
Print  "$obj1  ->  name<br>";  // გამოდის ახალი მნიშვნელობა "bob"
Print  "$obj2 -> name<br>";  // გამოდის მნიშვნელობა "harry" (დუმილით)
```

აღვნიშნოთ, რომ ობიექტიდან თვისებაზე გასვლა ხდება –> ოპერატორის მეშვეობით.

ახლა კი განვსაზღვროთ კლასი, რომელიც მეთოდს შეიცავს:

<u>ლისტინგი 32</u>

```
<html>
<head>
<title>მეთოდის შემცველი კლასი</title>
</head>
<body>
<?php
 class first_class
      {
      var  $name;
      function sayHello()
         {
         print "hello";
         }
      }
 $obj1 = new first_class();
 $obj1->sayHello();
 // "hello"
?>
</body>
</html>
```

მეთოდი, ვხედავთ, ჩვეულებრივი ფუნქციაა, ოღონდ ის კლასის შიგნით განისაზღვრება. ობიექტისთვის მისი გამოძახება ხდება –> ოპერატორის მეშვეობით.

ხაზგასასმელია ის გარემოება, რომ მეთოდს შეუძლია მიმართოს კლასში არსებულ ყველა შიდა ცვლადს (თვისებას). ამ შესაძლებლობას უზრუნველყოფს ცვლადისადმი მიმართვისას მისი სახელის წინ $this მაჩვენებლის დასმა:

ლისტინგი 33

```
<html>
<head>
<title>თვისებისადმი მიმართვა მეთოდიდან</title>
</head>
<body>
<?php
 class first_class
      {
      var  $name="harry";
      function sayHello()
         {
         print  "hello! my name is $this->name<BR>";
         }
      }
 $obj1 = new first_class();
 $obj1->sayHello();
 //  print "hello my name is harry"
?>
</body>
</html>
```

რადგან მეთოდი ფუნქცია გახლავთ, მას გამოძახებისას, ცხადია, არგუმენტიც (*არგუმენტები)* შეიძლება გადავცეთ. მაგრამ აქ ხაზი უნდა გავუსვათ ერთ მნიშვნელოვან გარემოებას - თუ კლასში გამოცხადებული მეთოდისა და თვით კლასის სახელები ერთმანეთს ემთხვევა, მაშინ ობიექტის შექმნისას ავტომატურად ხდება ასეთი მეთოდის გამოძახებაც. ამ დროს შესაბამისი არგუმენტების გადაცემით ჩვენ შეგვიძლია განვსაზღვროთ ობიექტის თვისებათა მნიშვნელობები.

ასეთ სპეციალურ ფუნქციებს **კონსტრუქტორები** ეწოდება.

მოვიყვანოთ კონსტრუქტორის მეშვეობით ობიექტის თვისებისათვის დუმილით განსაზღვრული მნიშვნელობის შეცვლის მაგალითი:

ლისტინგი 34

```
<html>
<head>
<title>კონსტრუქტორის შემცველი კლასი</title>
</head>
<body>
```

```
<?php
 class first_class
      {
      var  $name;
      function first_class( $n="anon" )
         {
         $this->name = $n;
         }
       function sayHello()
         {
         print "hello my name is $this->name<BR>";
         }
      }
 $obj1 = new first_class("bob");
 $obj2 = new first_class( "harry");
 $obj2 = new first_class();

 $obj1->sayHello();
 //  print "hello my name is bob"
 $obj2->sayHello();
 //  print "hello my name is harry"
?>
</body>
</html>
```

$obj1 ობიექტის შექმნისას ავტომატურად გამოიძახება first_class სახელის მქონე მეთოდი, მას არგუმენტად გადაეცემა “bob” სტრიქონი, $n ცვლადს დუმილით გათვალისწინებული “anon” მნიშვნელობა შეეცვლება “bob”-ით. ამ ფუნქციის შიგნით არსებული ინსტრუქციით ხდება კლასში გამოცხადებული $this –> name ცვლადისადმი მიმართვა და მისთვის $n1 მნიშვნელობის *(ანუ* “bob”-*ის)* მინიჭება.

ამავე პროგრამაში იქმნება მეორე და მესამე ობიექტებიც. მესამე ობიექტისათვის, რადგანაც მისი შექმნისას არგუმენტის ჩვენება არ ხდება, მას მიენიჭება სახელისთვის დუმილით გათვალისწინებული მნიშვნელობა “anon”.

დაბოლოს, კლასში თითოეული ობიექტისთვის ხდება სხვა მეთოდის გამოძახება, რომლის დანიშნულება არის ობიექტის სახელის შემცველი სტრიქონის ფორმირება და მისი ეკრანზე გამოტანა.

<u>მაგალითი</u>

მწყობრში მოვიყვანოთ ობიექტების შესახებ ჩვენი ცოდნა და შევქმნათ კლასი, რომელიც ეკრანზე გამოგვიყვანს დასათაურებულ სვეტებიან ცხრილს. ამ კლასში გავითვალისწინოთ მეთოდი, რომელიც შექმნილ ობიექტს დაუმატებს მასივის სახით მოცემულ სტრიქონს.

დასასრულ, კიდევ ერთი მეთოდით ავსახოთ ცხრილი ეკრანზე.

<u>კლასის თვისებები</u>

პროგრამის აგებას ვიწყებთ შესაბამისი კლასის ფორმირებით:

```
class Table
   {
   var $table_array = array();  // ცხრილში მონაცემების მასივი
   var $headers = array();      // სვეტების სახელთა მასივი
   var $cols;                   // სვეტების რაოდენობის ამსახველი ცვლადი
   }
```

<u>კონსტრუქტორი</u>

$headers მასივის მნიშვნელობები უნდა განისაზღვროს ობიექტის *(ცხრილის)* შექმნისას. ამ მიზანს ემსახურება კონსტრუქტორი-ფუნქცია. ცხადია, მისი სახელი უნდა ემთხვეოდეს კლასის სახელს, ანუ უნდა იყოს Table. შინაარსობრივი მხარიდან გამომდინარე, არგუმენტის *(რომელიც მასივს წარმოადგენს)* სახელად ავირჩიოთ $headers. შემდეგ, ჩვენთვის უკვე ნაცნობი ხერხით უნდა განისაზღვროს კლასში გამოცხადებული სვეტების სათაურთა მასივის შემცველობა და count() ფუნქციის მეშვეობით დათვლილ იქნეს მასში ელემენტების რაოდენობა.

კონსტრუქტორს ექნება ასეთი სახე:

```
function Table ($headers)
  {
    $this -> headers=$headers;
    $this -> cols=count ($headers);
  }
```

აღვნიშნოთ, რომ ობიექტის თვისებებში განთავსებული ინფორმაცია მისაწვდომია ობიექტის ნებისმიერი მეთოდისთვის.

<u>addRow() მეთოდი</u>

ამ მეთოდის ყოველი გამოძახებისას ობიექტს ემატება მონაცემთა სტრიქონი. ეს სტრიქონი მასივის სახით წარმოგვიდგება, რომლის სიგრძე უნდა ემთხვეოდეს სათაურების სტრიქონის სიგრძეს *(პროგრამა ამოწმებს ამ პირობის შესრულებას):*

```
function addRow ( $row )
  {
  if ( count ( $row ) != $this->cols )
      return false;
   array_push ( $this->table_array, $row);
   return true;
  }
```

array_push ფუნქცია ცხრილს, რომელიც ასევე მასივის სახით წარმოგვიდგება, ახალ სტრიქონს უმატებს. დასამატებელი ელემენტი

(სტრიქონი) თვითონ გახლავთ მასივი, მაგრამ იგი არსებულ მასივს ემატება, როგორც ელემენტი.

ამრიგად, ვიღებთ მრავალგანზომილებიან მასივს - მასივთა მასივს.

AddRowAssocArray() მეთოდი

წინა მეთოდისაგან განსხვავებით, addRowAssocArray() მეთოდს არგუმენტად გადაეცემა არა მოწესრიგებული, არამედ ასოცირებული მასივი. ამასთან, შესაძლებელია მასივის სიგრძე ნაკლებიც იყოს ცხრილის სიგანეზე, ე.ი. მასში ყოველი სვეტის სახელი არც ფიგურირებდეს.

პროგრამა შემდეგნაირად იგება:

- ფუნქცია ზემოთ აღნიშნულ ასოცირებულ მასივს $row-assoc არგუმენტის სახით ღებულობს.
- ყალიბდება $row ცარიელი მასივი, რომელიც თანდათანობით ივსება საჭირო ელემენტებით ასოცირებული მასივიდან.
- $row მასივისათვის ელემენტის დასამატებლად foreach ციკლით ჩათვალიერდება $this–>headers სათაურთა მასივი და მისი ელემენტების მნიშვნელობა თანმიმდევრულად მიენიჭება header ცვლადს.
- isset() ჩაშენებული ფუნქციით ციკლში მოწმდება, არსებობს თუ არა $row-assoc ასოცირებულ მასივში $header სახელის მქონე ელემენტი. თუ ეს ასე არ გახლავთ, ნაკლული $row-assoc მასივი ივსება ამ სახელის მქონე ელემენტით, რომელსაც ცარიელი სტრიქონის მნიშვნელობა ენიჭება. ნებისმიერ შემთხვევაში ამას მოსდევს $row მასივის შესაბამისი ელემენტის ფორმირებაც.
- ციკლის დამთავრების შემდეგ ჩვენს განკარგულებაშია $row მასივი, რომელშიც ფიგურირებს $row-assoc ასოცირებულ მასივში მოცემული ელემენტები. ზოგიერთი ადგილი კი მასში, როგორც ვნახეთ, ცარიელი მნიშვნელობითაც შეიძლება განისაზღვროს.
- პროგრამის ბოლოში array_push ჩაშენებული მეთოდით ხდება $table_array ცვლადში განლაგებული მასივისთვის $row მასივის, როგორც ელემენტის, დამატება.
- return true ინსტრუქციით ხდება შეტყობინება, რომ ფუნქციის მუშაობა ნორმალურად დამთავრდა.

output() მეთოდი

ამ მეთოდით პროგრამას ეკრანზე გამოჰყავს table-array მრავალ-განზომილებიანი მასივიდან ცხრილის სვეტების სათაურები და მასივის მნიშვნელობები.

თავდაპირველად foreach ციკლის მეშვეობით ჩათვალიერდება სვეტების სათაურების მასივი და ყოველი მათგანი ეკრანზე აისახება.

გადავდივართ შემდეგ სტრიქონზე. foreach ცვლადში ჩათვალიერდება table-array მასივის ელემენტებიც, მაგრამ რადგანაც ეს მასივი მრავალგანზომილებიანია *(მოცემულ შემთხვევაში - ორგანზომილებიანი),* საჭირო ხდება მისი თითოეული ელემენტის *(მასივის)* შიდა foreah ციკლში

ჩათვალიერებაც და ცალკეული ელემენტების ეკრანზე გამოყვანა. ზედა დონის თითოეულ ელემენტს *(ანუ მასივის)* ამობეჭვდის შემდეგ პროგრამას კურსორი მომდევნო სტრიქონზე გადაჰყავს.

საბოლოოდ, მთლიანი პროგრამა და მისი მუშაობის შედეგი ასეთი სახით წარმოგვიდგება:

ლისტინგი 35

```
<html>
<head>
<title> Table კლასი</title>
</head>
<body>
<?php
 class Table
      {
     var  $table_array = array ();
     var $headers = array ();
     var $cols;

     function Table ($headers )
        {
        $this->headers = $headers;
        $this->cols = count ( $headers );
        }

     function addRow ($row )
       {
        if ( count ($row ) != $this->cols )
             return false;
             array_push($this->table_array, $row);
             return true;
        }

     function addRowAssocArray ($row_assoc)
        {
            $row = array();
            foreach ($this->headers as  $header )
              {
              if ( ! isset( $row_assoc[$header] ))
                  $row_assoc[$header] = " " ;
              $row[] = $row_assoc[$header];
              }
            array_push($this->table_array, $row);
            return true;
         }
```

```
function output  ()
        {
          print "<pre>";
          foreach ( $this->headers as $header )
            print  "<b>$header</B> ";
            print "\n";
            foreach ( $this->table_array as $y )
              {
                foreach ( $y as $xcell )
                    print "$xcell ";
                    print  "\n";
              }
              print "</pre>";
          }
      }
        $test = new table (array("a","b","c") );
        $test->addRow( array(1,2,3) );
        $test->addRow( array(4,5,6) );
        $test->addRowAssocArray( array ( b=>0, a=>6, c=>3 ));
        $test->output();
?>
</body>
</html>
```

პროგრამის მუშაობის შედეგს ექნება სახე:

a b c

1 2 3

4 5 6

6 0 3

კვლავ კლასების შესახებ

მემკვიდრეობითობა

კლასებს მომხმარებლისთვის მრავალფეროვანი სერვისის გაწევა შეუძლიათ. მაგალითად, ადვილად განსახორციელებელია არსებულის ბაზაზე მოდიფიცირებული კლასების შექმნაც. ამასთან, ახალ კლასებს მემკვიდრეობით გადმოეცემათ მშობელი კლასის წარმომადგენლებისათვის დამახასიათებელი ფუნქციური როლებიც.

მოვიყვანოთ ამგვარი კლასის შექმნის მაგალითი:

<u>ლისტინგი 36</u>

```
<html>
<head>
<title>არსებული კლასის ბაზაზე ახლის შექმნა</title>
</head>
<body>
```

```
<?php
 class first_class
     {
     var  $name = "harry";

     function first_class( $n )
        {
        $this->name = $n;
        }

     function sayHello()
        {
        print "hello my name is $this->name<br>";
        }
     }

  class second_class extends first_class
     {

       }
       $test = new second_class("son of harry");
       $test->sayHello();

 //  print "hello my name is harry"
?>
</body>
</html>
```

პროგრამაში, first_class-ის გარდა, მის ბაზაზე შევქმენით second_class-იც. ამ მიზნით, საკმარისი აღმოჩნდა მისი გამოცხადებისას extends საკვანძო სატყვის გამოყენება.

ვხედავთ, რომ first_class შეიცავს კონსტრუქტორ-ფუნქციასაც. *(გავიხსენოთ, ასეთი ფუნქციის სახელი კლასის სახელს ემთხვევა).*

საინტერესოა, რომ მემკვიდრე კლასის შექმნისას ავტომატურად ხდება მასში არა მარტო მეთოდების გადატანა, არამედ კონსტრუქტორისათვის შესაბამისი სახელის განსაზღვრაც. სწორედ, ასეთი გადაწყვეტილების გამო გახდა შესაძლებელი, კორექტულად ემუშავა მოცემულ პროგრამაში test-sayHello() ინსტრუქციას.

მშობელი კლასის მეთოდების სახეცვლილება

წინა მაგალითში წარმოებული კლასის ობიექტები ზუსტად ისევე იქცეოდნენ, როგორც მშობელი კლასის ობიექტები. რა თქმა უნდა, დასაშვებია, ეს ყოველთვის ასე არც იყოს. მაგალითად, შევქმნათ second-class წარმოებულ კლასისათვის საკუთარი მეთოდი sayHello():

<u>ლისტინგი 37</u>

```
<html>
<head>
```

```
<title>მეთოდის ხელახლა განსაზღვრა</title>
</head>
<body>
<?php
 class first_class
      {
      var  $name = "harry";

      function first_class( $n )
         {
         $this->name = $n;
         }

       function sayHello()
         {
         print "hello my name is $this->name<br>";
         }
       }

  class second_class extends first_class
       {
       function sayHello()
          {
          print "I'm not going to tell you my name<br>";
           }
         }

$test = new second_class("son of harry");
$test->sayHello();

 //  ბეჭდავს  "I'm not going to tell you my name"
?>
</body>
</html>
```

მშობელი კლასის მეთოდის გამოძახება

გამორიცხული არ გახლავთ, ზოგჯერ საჭირო შეიქნეს წარმოებული კლასის ობიექტიდან მშობელი კლასის მეთოდის გამოძახებაც. ობიექტზე ორიენტირებული დაპროგრამება უშვებს ასეთ შესაძლებლობას; იგი ხორციელდება მარტივი ინსტრუქციის საფუძველზე:

<u>ლისტინგი 38</u>

```
<html>
<head>
<title>მშობლიური კლასის მეთოდის გამოძახება</title>
</head>
```

```
<body>
<?php
 class first_class
      {
      var  $name = "harry";

      function first_class( $n )
         {
         $this->name = $n;
         }

       function sayHello()
         {
         print "hello my name is $this->name<br>";
         }
      }

  class second_class extends first_class
      {
                        function sayHello()
        {
            print "I'm not going to tell you my name --";
            first_class::sayHello();
        }
      }

        $test = new second_class("son of harry");
        $test->sayHello();
       / /   გამოჰყავს "I'm not going to tell you my name" --HEllo my name is
             son of harry"
?>
</body>
</html>
```

მემკვიდრეობის გადაცემის მაგალითი

ადრე განხილული Table კლასის ბაზაზე შევქმნათ ახალი, HTMLTable-დ წოდებული კლასი, რომელსაც დამატებითს ფუნქციებს დავაკისრებთ. კერძოდ, მათი მეშვეობით მოხდება ცხრილში გამოსაყვანი მონაცემების დაფორმატება და მათი ჩვეულებრივი, html ცხრილის სახით გამოყვანა. ამ მაგალითში მომხმარებელს შეეძლება, აირჩიოს cellpadding და bgcolor ატრიბუტების მნიშვნელობები.

წინა პროგრამაში უნდა შევიტანოთ შემდეგი ცვლილებები:

ვაცხადებთ table-ს მემკვიდრე HTMLTable კლასს -

```
class HTMLTable extends Table
  {
    var $bgcolor;
    var $cellpadding = 2;     // მნიშვნელობა განისაზღვრება დუმილით
   }
```

კონსტრუქტორის შექმნა

წარმოებულ კლასში შევქმნათ საკუთარი კონსტრუქტორი *(წინააღმდეგ შემთხვევაში ობიექტის შექმნისას მოხდებოდა მშობელი კლასის კონსტრუქტორის გამოძახება):*

```
function  HTMLTable ($headers, $bg="#f f f f f f ")
    {
      Table: :Table ($headers);
      $this -> bgcolor=$bg;
    }
```

ვხედავთ, რომ, მშობელი კლასის კონსტრუქტორის ფუნქციების შესრულების გარდა, წარმოებული კლასის კონსტრუქტორი თავის თავზე იღებს ობიექტის *(ცხრილის)* ახალი თვისებებისთვის არგუმენტის მიღებისა და ამ თვისებებისათვის მნიშვნელობის მინიჭების ფუნქციებსაც.

აღვნიშნოთ, რომ თუ ობიექტის შექმნისას $bg არგუმენტისათვის მნიშვნელობა არ განისაზღვრება, მაშინ მას მიენიჭება დუმილით გათვალისწინებული მნიშვნელობა *(მოცემულ შემთხვევაში #f f f f f f).*

SetCellPadding მეთოდის დამატება

დავამატოთ კლასში მისი საკუთარი მეთოდებიც. მაგალითად, შევქმნათ მეთოდი, რომლის გამოძახების შედეგად შეიცვლება დუმილით გათვალისწინებული მნიშვნელობა cellpadding თვისებისათვის:

```
function  SetCellPadding ($padding)
  {
    $this -> cellpadding = $padding;
  }
```

Output() მეთოდი

დაბოლოს, მოგვიწევს მშობლიურ კლასში არსებული ამ მეთოდის მთლიანად გარდაქმნა, რათა შევძლოთ მონაცემების HTML-ცხრილის სახით გამოყვანა:

```
function Output()
    {
      print "<table cellpadding=\"$this->cellpadding\" border=1>";
   foreach ( $this->headers as $header )
      print "<td  bgcolor=\"$this->bgcolor\"><b> $header </b></td>";
   foreach ( $this->table_array as $row=>$cells )
      {
```

```
            print "<tr>";
             foreach ( $cells as $cell )
               print "<td  bgcolor=\"$this->bgcolor\"> $cell</td>";
            print "/<tr>";
            }
          print "</table>";
        }
```

დამატებით იმისა, რაც ხდებოდა მშობლიურ კლასში, აქ გათვალისწინებულია cellpadding და bgcolor ატრიბუტების ფორმირებაც.

მთლიანი პროგრამა და მისი მუშაობის შედეგი ასეთი სახით წარმოგვიდგება:

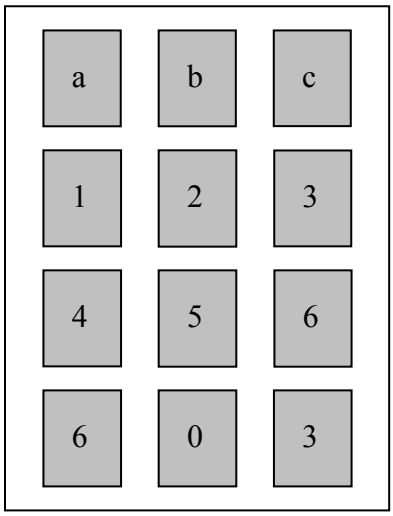
ლისტინგი 39

```
<html>
<head>
<title> Table  და  HTMLTable  კლასები </title>
</head>
<body>
<?php
 class Table
      {
     var $table_array = array();
     var $headers =array ();
        var $cols;
        function Table ($headers )
        {
      $this->headers =$headers;
      $this->cols =count ( $headers );
        }
        function addRow ( $row )
        {
        if ( count ($row) != $this->cols )
           return false;
         array_push($this->table_array, $row);
         return true;
        }
```

```
 function addRowAssocarray( $row_assoc )
 {
    if ( count ($row_assoc) != $this->cols )
       return false;
$row = array();
foreach ( $this->headers as $header )
  {
if ( ! isset( $row_assoc[$header] ))
     $row_assoc[$header] = " ";
   $row[] = $row_assoc[$header];
   }
array_push($this->table_array, $row);
}

function output ()
  {
     print "<pre>";
     foreach ( $this->headers as $header )
     print "<B>$header</B> ";
     print "\n";
     foreach ( $this->table_array as $y )
       {
         foreach ( $y as $xcell )
         print "$xcell ";
         print "\n";
        }
     print "</pre>";
  }
 }

 class HTMLTable extends Table
 {
     var $bgcolor;
     var $cellpadding = "2";
```

```
    function HTMLTable ( $headers, $bg="#ffffff" )
     {
    Table::Table($headers);
    $this->bgcolor=$bg;
    }

    function setCellpadding( $padding )
     {
 $this->cellpadding = $padding;
     }

    function output ()
     {
  print "<table cellpadding= \"$this->cellpadding\" border =1>";
  foreach ( $this->headers as $header )
    print "<td  bgcolor=\"$this->bgcolor\"><b> $header </b></td>";
  foreach ( $this->table_array as $row=>$cells )
      {
    print "<tr>";
    foreach ( $cells as $cell )
      print "<td bgcolor=\"$this-> bgcolor\"> $cell</td>";
     print "</tr>";
     }
  print "</table>";
    }
  }
  $test = new HTMLTable( array( "a", "b", "c"), "#00FF00");
  $test->setCellpadding( 7 );
  $test->addRow( array(1,2,3));
  $test->addRow( array(4,5,6));
  $test->addRowAssocArray ( array ( b=>0, a=>6, c=>3 ));
  $test->output();
  ?>
</body>
</html>
```

პროგრამის მუშაობის შედეგია შემდეგი ცხრილი:



ფორმებთან მუშაობა

დასაწყისში გავიხსენოთ, რომ პროგრამებში გლობალური ცვლადების გამოცხადება ხდება ყველა ფუნქციის გარეთ, ასე ვთქვათ, პროგრამის პირველ დონეზე. თითოეული გლობალური ცვლადი ინახება ჩაშენებულ ასოცირებულ $GLOBALS მასივში.

ვნახოთ, როგორ ხერხდება ამ ცვლადების ჩათვალიერება:

<u>ლისტინგი 40</u>

```
<HTML>
<head>
<title> $GLOBALS მასივის ჩათვალიერება </title>
</head>
<body>
<?php
$user1 = "Bob";
$user2 = "Harry";
$user3 = "Mary";
foreach  ( $GLOBALS as $key=>$value )
{
   print "\$GLOBALS[\"$kay\"] == $value<br>";
}
?>
  </body>
  </html>
```

სისტემურთან ერთად ეკრანზე აისახება ჩვენ მიერ პროგრამაში გამოცხადებული 3 გლობალური ცვლადიც.

მომხმარებლის მიერ შევსებული ფორმის დამუშავების პროგრამა

ჯერ შევქმნათ კოდი Web-ფურცლისათვის, რომელზეც მოხდება ფორმის შევსება:

ლისტინგი 42

```
<HTML>
<head>
<title> მარტივი html-ფორმა  </title>
</head>
<body> <form action="eg9.3.php" method="GET">
<input type="text" name="user">
<br>
<textarea name="address" rows="5" cols="40">
</textarea>
<br>
<input type="submit" value="hit it!">
</form>
 </body>
 </html>
```

ამ მარტივ ფორმაში განლაგებულია “user” სახელის მქონე ტექსტური ველი, “address” ტექსტური უბანი და მონაცემთა გადაცემის ღილაკი “Submit”.

ფორმის შევსების შემდეგ ღილაკზე დაწკაპუნებით გამოიძახება მიმდინარე საქაღალდეში მყოფი eg9.3.php პროგრამა, რომელიც დაამუშავებს ფორმის მეშვეობით გადაცემულ მონაცემებს:

ლისტინგი 43

```
<HTML>
<head>
<title> წინა ლისტინგიდან ფორმის მონაცემების წაკითხვა </title>
</head>
<body>
<?php
print "Welcome <b>$user</b><p>\n\n";
print "your address is:<p>\n\n<b>$address</b>";
?>
 </body>
 </html>
```

ზემოთ მოყვანილ პროგრამას პრინციპული სიახლე ახასიათებს - მისი გამოძახება ხდება არა უშუალოდ, არამედ HTML-ფორმიდან. პროგრამა სრულება სერვერზე. ფორმით გადაცემულ მონაცემებს პროგრამები გლობალური ცვლადების მეშვეობით იღებენ.

მრავალი მნიშვნელობის მქონე ელემენტების დამუშავება

განვიხილოთ ისეთი შემთხვევა, როცა მომხმარებელი პროგრამას გადასცემს ერთ ელემენტთან დაკავშირებულ რამდენიმე მნიშვნელობას, მაგალითად, ჩამოშლად სიაში არჩეულ რამდენიმე პროდუქტის დასახელებას, გვარს და ა.შ.

ეს საკითხი შემდეგნაირად წყდება – ელემენტის სახელის შემდეგ უნდა დასმულ იქნეს ცარიელი კვადრატული ფრჩხილები:

ლისტინგი 44

```
<HTML>
<head>
<title> HTML ფორმა  Select ტეგით  </title>
</head>
<body>
<form action="eg9.5.php" method="POST">
<input type="text" name ="user">
<br>
<textarea name="address" rows="5" cols="40">
</textarea>
<br>
<select name="produqts[]" multiple>
  <option>Sonic Screwdriver
  <option>Tricorder
  <option>ORAC AI
  <option>HAL 2000
</select>
</br>
<input type="submit" value="hit it!">
</form>
</body>
</html>
```

select ტეგში გადასაცემი ელემენტისათვის products სახელი რომ მიგვეინჭებინა, მაშინ პროგრამა დაამუშავებდა სიაში შერჩეულ მხოლოდ ერთ მნიშვნელობას, products[] სახელი კი უზრუნველყოფს ამ პროგრამისათვის მომხმარებლის მიერ არჩეული მნიშვნელობების გადაცემას $products მასივის სახით.

პროგრამას შეიძლება ასეთი სახე ჰქონდეს:

ლისტინგი 45

```
<HTML>
<head>
<title> წინა ლისტინგიდან ფორმის მონაცემების დამუშავება </title>
</head>
<body>
<?php
```

```
print "Welcome <b>$user</b><p>\n\n";
print "Your address is:<p>\n\n<b>$address</b><p>\n\n";
print "Your product choices are:<p>\n\n";
print "<ul>\n\n";
foreach ( $products as $value )
{
print "<li>$value<br>\n";
}
print "</ul>";
?>
</body>
</html>
```

შევნიშნოთ, რომ, select-ის გარდა, ერთი სახის რამდენიმე მნიშვნელობის გადაცემა სხვა ელემენტებსაც შეუძლიათ *(მაგალითად, ერთი სახელის ქვეშ გაერთიანებულ ალმების მიმდევრობას).*

HTML-ტექსტისა და PHP-პროგრამების ერთ ფურცელზე განლაგება

ზოგჯერ, თუ ეს შესაძლებელია, უმჯობესია HTML-ტექსტი და PHP-პროგრამა ერთ Web-ფურცელზე განვალაგოთ. მაგალითად, მაშინ, როცა ფორმის შევსება რამდენჯერმე ხდება. იგულისხმება შემთხვევა, როცა პროგრამა ტექსტისგან განცალკევებულია და არა მასში მრავალ "ნაჭრად" გაფანტული, რაც მის წაკითხვას ართულებს.

შევქმნათ სკოლამდელი ასაკის ბავშვებისათვის პროგრამა, რომელიც «ჩაიფიქრებს» რიცხვს და პასუხობს კითხვაზე, მომხმარებლის მიერ მისთვის გადაცემული რიცხვი მეტია თუ ნაკლებია ამ რიცხვზე.

დავიწყოთ HTML-ფორმის შექმნა. იგი შეიცავს ერთადერთ ტექსტურ ველს. ასეთ შემთხვევაში ფორმის გადასაცემად თანამედროვე ბროუზერებს აღარ სჭირდებათ Submit ღილაკის არსებობა – ამ მიზნით საკმარისია <Return> კლავიშზე ხელის დაჭერა.

შემდეგ, action ელემენტში php პროგრამული ფაილის ნაცვლად მითითებულია $PHP_SELF ცვლადი. მაშასადამე, ფორმის შევსების შემდეგ მიმართვა ხდება საკუთარი თავისადმი:

<u>ლისტინგი 46</u>

```
<HTML>
<head
<title>თავის თავის გამომძახებელი HTML ფორმა </title>
</head>
<body>
<form action=<?php print $PHP_SELF?>" method="POST">
Type your guess here: <input type="text" name="guess">
</form>
</body>
</html>
```

ცხადია, ეკრანზე არაფერი აისახება, ამიტომ შემდეგი ნაბიჯი იქნება იმავე Web-ფურცელზე php-პროგრამის ტექსტის დამატება:

<u>ლისტინგი 47</u>

```
<?php
$num_to_guess = 42;
$message = " ";
if ( ! isset( $guess ) )
{
$message = "Welcome to the guessing machine!";
}
elseif   ( $guess > $num_to_guess )
{
$message = "$guess is too big! Try a smaller number";
}
elseif ( $guess < $num_to_guess )
{
$message = "$guess is too small! Try a larger number";
}
else
{
   $message = "Well done!";
}
?>
<html>
<head>
<title> რიცხვის გამომცნობი პროგრამა </title>
</head>
<body>
<form action=<?php print $PHP_SELF?>" method="POST">
Type your guess here: <input type="text" name="guess">
</form>
</body>
</html>
```

პირველ რიგში, უნდა განისაზღვროს რიცხვი, რომელიც მომხმარებელმა უნდა გამოიცნოს. რეალურ *(არასაცდელ)* პროგრამაში, ცხადია, შემთხვევითი რიცხვის გენერირებას მოვახდენდით. ამჯერად, ამ საქმეს უფრო მარტივად ვწყვეტთ - ცვლადს პირდაპირ ვანიჭებთ მნიშვნელობას.

ამის შემდეგ უნდა გაირკვეს, მომხმარებელმა პირველად მიმართა Web-ფურცელს, თუ ფორმა უკვე შეავსო და იგი განმეორებით გამოიძახა. ამ კითხვაზე პასუხი იმის და მიხედვით გაიცემა, განსაზღვრულია თუ არა $guess გლობალური ცვლადი. გავიხსენოთ, ამ სქემაში გვეხმარება isset() ჩაშენებული ფუნქცია.

შემდეგ კი პირობების შემმოწმებელ ოპერატორებში გარკვევა სირთულეს აღარ წარმოადგენს.

თუ მომხმარებელი პირველად მოხვდა Web-ფურცელზე, მაშინ, ცხადია, $guess ცვლადი განსაზღვრული ვერ იქნება და $message სტრიქონში შეგვაქვს მისალმების გამომხატველი სტრიქონი. წინააღმდეგ შემთხვევაში ხდება შემოწმება, აჭარბებს თუ არა $guess-ის მნიშვნელობა წინასწარ აღებულ რიცხვს და ამისდა მიხედვით, მომხმარებელს მიეწოდება შესაბამისი შეტყობინება $messaqe ცვლადის მეშვეობით.

მდგომარეობის დასაფიქსირებლად დამალული ველების გამოყენება

დავუშვათ, გვსურს, დავიმახსოვროთ, თუ რამდენი ცდა გამოიყენა მომხმარებელმა. ამ საქმეში შეგვიძლია ე.წ. <u>დამალული ველი</u> დავიხმაროთ. იგი ტექსტური ველია, რომელიც ეკრანზე უშუალოდ არ აისახება.

ვაჩვენოთ, თუ როგორ შეიძლება Web-ფურცელზე ასეთი ველის დამატება და PHP-პროგრამით მისი დამუშავება:

<u>ლისტინგი 48</u>

```
<?php
$num_to_guess = 42;
$message = " ";
$num_tires = ( isset( $num_tries ) ) ? ++$num_tries : 0;

if ( ! isset( $guess ) )
{
$message = "Welcome to the guessing machine!";
}
elseif   ( $guess > $num_to_guess )
{
$message = "$guess is too big! Try a smaller number";
}
elseif ( $guess < $num_to_guess )
{
$message = "$guess is too small! Try a larger number";
}
else // must be equivalent
{
   $message = "Well done!";
}
```

```
$guess = (int) $guess;
?>
<html>
<head>
<title> დამალული ველის მეშვეობით მდგომარეობის დამახსოვრება
</title>
</head>
<body>
<h1>
<?php print $message ?>
</h1>
Guess number: <?php print $num_tries?>
<form action=<?php print $PHP_SELF?>" method="POST">
Type your guess here:
<input type="text" name="guess" value="<?php print $guess?>">
<input type="hidden" name="num_tries" value="<?php print $num_tries?>">
</form>
</body>
</html>
```

ვხედავთ, რომ ფორმაში დავამატეთ num_tries სახელის მქონე დამალული ველი *(ველის ტიპზე მიგვითითებს type = ”hidden” ატრიბუტი).* მის value ატრიბუტს ვანიჭებთ მნიშვნელობას:

```
value = ”<?php  print  $num_tries ?>”
```

ასევე ვიქცევით guess ველისთვისაც. მასაც იმავე წესით ვანიჭებთ მნიშვნელობას:

```
value = ”<?php  print  $guess  ?>”
```

მაგრამ რამდენადაც ეს ველი დამალული არაა, ეკრანზე აისახება მისი მნიშვნელობაც, ანუ ის მნიშვნელობა, რომელიც ბოლო ცდაზე შეიტანა მომხმარებელმა ამ ველში. შედეგად მომხმარებელს უადვილდება ახალი არჩევანის გაკეთება ველში არსებული მნიშვნელობის კორექტი-რების მეშვეობით.

პროგრამას დასაწყისში ემატება შემოწმების ოპერატორი, რომელიც, თუ $num_tries გლობალური ცვლადი შექმნილია, 1-ით ზრდის მის მნიშვნელობას, წინააღმდეგ შემთხვევაში ქმნის ცვლადს და ანიჭებს ნულოვან მნიშვნელობას.

კლიენტის "გადართვა" სხვა Web-ფურცელზე

ჩვენ მიერ ზემოთ განხილულ პროგრამას ერთი არსებითი ნაკლი გააჩნია - მას მაშინაც გამოჰყავს ეკრანზე შესავსები ფორმა, როცა კლიენტმა რიცხვი უკვე გამოიცნო.

თუ კი ჩვენ გამოვიყენებთ ე.წ პროგრამის სათაურის გადმომგზავნ header() ფუნქციას, მომხმარებლის მიერ რიცხვის გამოცნობის შემთხვევაში შევძლებთ მას ეკრანზე გამოვუყვანოთ სხვა - მილოცვის შეტყობინება, რისთვისაც საკმარისია წინა პროგრამაში else-ბლოკი შემდეგნაირად შევცვალოთ:

```
else
   {
     header ( "Location: congrats.html" );
     exit;
   }
```

საერთოდ, სერვერიდან კლიენტის მიერ პროგრამის გამოძახების შემთხვევაში მას ავტომატურად გადმოეგზავნება გადმოცემული Web-ფურცლის შესახებ ინფორმაციის შემცველი სათაური. მასში არსებული ინფორმაცია ეკრანზე, თუ სპეციალურ ზომებს არ მივმართეთ, არ აისახება - ამ ინფორმაციის ბროუზერი იყენებს.

აი, ტიპური სათაური, რომელსაც PHP-პროგრამა ბროუზერს უგზავნის:

<u>ლისტინგი 49</u>

```
HEAD /matt/php-book/forms/eg9.1.php HTTP/1.0
HTTP/1.1 200 OK
Date: Sun, 09 Feb 2005 17:23:45 QMT
Server: Apache/1.3.9 (UNIX) PHP 4.0
Connection: close
Content-Type: text/html
```

შესაძლებელია წინა პროგრამა ისე იქნეს მოდიფიცირებული, რომ რიცხვის გამოცნობის შემთხვევაში მომხმარებელი დავაკავშიროთ იმ WEB-ფურცელთან, რომელზეც იქნება შესაბამისი შეტყობინება მილოცვის ტექსტით. ეს ხდება header("Location: congrats.html"); ოპერატორის მეშვეობით:

<u>ლისტინგი 50</u>

```
<?php
$num_to_guess = 42;
$message = " ";
$num_tries = ( isset( $num_tries ) ) ? ++$num_tries : 0;
if ( ! isset( $guess ) )
{
$message = "Welcome to the guessing machine!";
}
```

```
elseif   ( $guess > $num_to_guess )
{
$message = "$guess is too big! Try a smaller number";
}
elseif ( $guess < $num_to_guess )
{
$message = "$guess is too small! Try a larger number";
}
else // must be equivalent
{
   header( "Location: congrats.html" );
   exit;
}
$guess = (int) $guess;
?>
<html>
<head>
<title> header() ფუნქციის მეშვეობით მომხმარებლისათვის საიტის
        მისამართის გაგზავნა
</title>
</head>

<body>
<h1>
<?php print $message ?>
</h1>
  Guess number: <?php print $num_tries?>

<form action=<?php print $PHP_SELF?>" method="POST">
  Type your guess here:
  <input type="text" name="guess" value="<?php print $guess?>">
  <input type="hidden" name="num_tries" value="<?php print
          $num_tries?>">
</form>

</body>
</html>
```

ფაილების სერვერზე გადასაცემად საჭირო ფორმები და პროგრამები

თუ გვსურს HTML-ფორმის დახმარებით სერვერზე რაიმე ფაილი გადავაგზავნოთ *(მაგალითად, GIF ფორმატში ნახატი),* ფორმაში შესაბამის ველებთან ერთად უნდა გავითვალისწინოთ enctype არგუმენტი:

```
enctype = “multipart/form-data”
```

ფორმაში აუცილებელია გავითვალისწინოთ max_file_size სახელის მქონე დამალული ველიც, რომელშიც მითითებული იქნება გადასაცემი ფაილის შესაძლო მაქსიმალური სიგრძე *(იგი არ უნდა აღემატებოდეს php.ini ფაილში ნაჩვენებ ზომას, რომელიც, ჩვეულებრივ, 2 მეგაბაიტის ტოლია).* მას მოსდევს INPUT ელემენტი თვით ფაილის გადაცემისათვის. ფაილს შეგვიძლია დავარქვათ ნებისმიერი სახელი:

<u>ლისტინგი 51</u>

```
<HTML>
<HEAD>
<TITLE> ფაილის გადასაცემი მარტივი ფორმა </title>
</head>
<BODY>
<FORM enctype="multipart/form-data" action="<? print $PHP_SELF ?>"
        metod="POST>"
  <INPUT type="hidden" name="MAX_FILE_SIZE" value="55555">
  <INPUT type="file" name="fupload" ><br>
  <INPUT type="submit"  value="upload!" >
</form>
</body>
</html>
```

ვხედავთ, რომ ფორმა იმავე ფურცელს გამოიძახებს, რომელზეც თვითონ არის ჩაწერილი. ეს საჭიროა იმ php-პროგრამისადმი მიმართვის განსახორციელებლად, რომელიც გადაცემულ ფაილს დაამუშავებს.

სერვერზე გადაცემის შემდეგ ფაილს მიენიჭება უნიკალური სახელი და შეინახება დროებითი ფაილების კატალოგში *(UNIX-ში, ჩვეულებრივ, ეს კატალოგია /tmp).* ფაილისკენ მიმავალი მთელი გზა ჩაიწერება იმ გლობალურ ცვლადში, რომლის სახელი ემთხვევა ფაილის გადამცემი ველის სახელს. მაშასადამე, მოცემულ შემთხვევაში ეს იქნება $fupload.

სერვერზე გადაცემული ფაილის შესახებ PHP სხვა ინფორმაციებსაც ინახავს გარკვეული წესით სახელდებულ შემდეგ გლობალურ ცვლადებში:

ცვლადის სახელი	აღწერა	მაგალითი
$fupload	გზა დროებით ფაილამდე	/tmp/php5Pq3fu
$fupload_name	გადაცემული ფაილის სახელი	test.gif
$fupload_size	ფაილის სიგრძე ბაიტებში	5555
$fupload_type	ფაილის ტიპი MIME სისტემაში	image/gif

ფორმით ერთდროულად რამდენიმე ფაილის გადაცემაც შეიძლება. მათთვის სერვერს დამატებითი ცვლადები აქვს გათვალისწინებული *(ამ საკითხს აქ არ განვიხილავთ).*

ახლა კი დავწეროთ პროგრამა, რომელიც ეკრანზე გამოიყვანს გადაცემული ფაილის შესახებ ინფორმაციას. ამასთან, თუ ფაილის გაფართოება იქნება GIF, მაშინ ეკრანზე ნახატიც აისახება:

ლისტინგი 52

```
<HTML>
<head>
<title> გადაცემული ფაილის დამუშავების პროგრამა </title>
</head>
<?php
  $file_dir= "/home/matt.htdocs/uploads";
  $file_url= "http://www.corrosive.co.uk/matt/uploads";
  if (isset( $fupload ) )
    {
     print "path: $fupload<br>\n";
     print "path: $fupload_name<br>\n";
     print "path: $fupload_size bytes<br>\n";
     print "path: $fupload_type<p>\n\n";
      if ($fupload_type == "image/gif")
        {
         copy ($fupload, "$file_dir/$fupload_name") or die ("Couldn't copy");
         print "<img src=\"$file_url/$fupload_name\"><p>\n\n";
        }
      }
?>
<body>
  <form enctype="multipart/form-data" action="<?php print $PHP_SELF ?>"
          method="POST">
    <input type="hidden" name="MAX_FILE_SIZE" value="51200">
    <input type="file" name="fulpload"><br>
    <input type="submit" value="Send file!">
  </form>
</body>
</html>
```

თავდაპირველად პროგრამა ამოწმებს, განსაზღვრულია თუ არა $fupload ცვლადი. თუ ეს ასეა, ფაილი გადაცემული არის სერვერისათვის, მის შესახებ ინფორმაცია ჩაწერილია გლობალურ ცვლადებში და ეკრანზე გამოდის 4 ცვლადის სახელებისა და მნიშვნელობების ამსახველი ინფორმაცია.

შემდეგ მოწმდება, $fupload_type ცვლადში მოცემული ტიპი თუ ემთხვევა “image/gif”-ს? დადებითი პასუხის შემთხვევაში ხდება დროებითი კატალოგიდან ჩვენს კატალოგში ფაილის კოპირება. Copy() ფუნქცია მოითხოვს ორ არგუმენტს: <u>გზას საწყის ფაილამდე</u> და მის <u>ახალ მდებარეობას</u> *(ფაილის სახელის ჩვენებასთან ერთად).* პირველი განსაზღვრულია $fupload ცვლადში, ხოლო ახალი მდებარეობა ნაჩვენებია პროგრამის დასაწყისშივე, $file_dir ცვლადში.

ფაილებთან მუშაობა

ფაილების ჩართვა PHP-დოკუმენტში

include() დირექტივით შესაძლებელია ნებისმიერ php-დოკუმენტში *(ანუ ფაილში)* სხვა, ასევე ფაილის სახით გაფორმებული php-პროგრამის ჩართვა.

ჩასართავი ფაილი შეიძლება მრავალმა სხვა php-დოკუმენტმაც გამოიძახოს, რაც ფრიად აადვილებს დაპროგრამების პროცესს. კერძოდ, თუ საჭიროა პროგრამულ ფაილში ინფორმაციის კორექტირება, ეს პროცესი განხორციელდება მხოლოდ მასში, ანუ ერთადერთ ადგილზე.

include() ფუნქციას სჭირდება მხოლოდ ფაილის სახელი *(და მისკენ გზის)* ჩვენება. მოვიყვანოთ ამ ფუნქციის გამოყენების მაგალითი:

<u>ლისტინგი 53</u>

```
<HTML>
<head>
<title>include() ფუნქციის გამოყენება </title>
</head>
<body>
 <?php
include("eg10.2.php");
 ?>
</body>
</html>
```

თუ გამოძახებული ფაილი, ვთქვათ, ასეთი სტრიქონია: I have been included! - იგი დამუშავდება, როგორც უბრალო HTML -ტექსტი და ასევე აისახება ეკრანზე. მაგრამ თუკი გვსურს, რომ გამოძახებული ფაილი აღქმული იქნეს, როგორც PHP-პროგრამა, მაშინ იგი შესაბამის ტეგებში უნდა მოვათავსოთ:

<u>ლისტინგი 54</u>

```
<?php
print  "I have been included!!<BR>";
print  "but now I can add up... 4+4 = ".(4+4);
 ?>
```

ჩასართავ ფაილებს PHP-ში, ფუნქციების დარად, return ოპერატორის მეშვეობით შეუძლიათ პროგრამის შესრულების შეწყვეტა და მნიშვნელობების დაბრუნება. მოვიყვანოთ მაგალითები:

<u>ლისტინგი 55</u>

```
<HTML>
<head>
<title> include() ფუნქცია აბრუნებს მნიშვნელობას </title>
</head>
<body>
```

```
<?php
  $addResult = include("eg10.6.php");
  print "The include file returned $addResult";
?>
</body>
</html>
```

<u>ლისტინგი 56</u>

```
<?php
$retval = ( 4 + 4 );
return $retval;
 ?>
```

{ ეს ტექსტი ეკრანზე, ცხადია ვერ აისახება! }

აღვნიშნოთ, რომ include() ინსტრუქციის გამოყენება შესაძლებელია პირობით ოპერატორებსა და ციკლებშიც. მოვიყვანოთ მაგალითები:

```
$test = false;
if ($test)
  {
    include ("a_file.txt");
  }
```

<u>ლისტინგი 57</u>

```
<html>
<head>
<title> include() ფუნქციის ციკლში გამოყენება </title>
</head>
<body>
<?php
for ($x=1; $x<=3; $x++)
   {
   $incfile="incfile$x".".txt";
   print "ვრთავთ $incfile  ფაილს <br>";
   include ("$incfile");
   print "<p>";
   }
?>
</body>
</hmtl>
```

ამ პროგრამის შესრულებისას Web-ფურცლის ტექსტში ჩაემატება 3 ფაილი:

incfile1.txt, incfile2.txt და incfile3.txt

შევნიშნოთ, რომ სანამ ფაილთან მუშაობას დავიწყებდეთ, შესაძლებელია რიგი ფუნქციებით მის შესახებ ინფორმაციის მიღება-შემოწმება.

მაგალითად, ფუნქცია file_exists (“*ფაილის_სრული_მისამართი* ან მხოლოდ *ფაილის_დასახელება*”) ამოწმებს ფაილის არსებობას. ამ კითხვაზე დადებითი პასუხი ნიშნავს ფუნქციის მიერ true-ს დაბრუნებას, წინააღმდეგ შემთხვევაში – false-ს:

```
if (file_exists("test.txt"))
    print "test.txt ფაილი მოიძებნა";
```

არსებობს ფაილის სხვა თვისებების *(სტატუსის, სიგრძის, შექმნის თარიღის და ა.შ.)* შესახებ ინფორმაციის მომწოდებელი ფუნქციებიც.

დაბოლოს, შესაძლებელია PHP-პროგრამების მეშვეობით ფაილების კატალოგების შექმნა, კორექტირება და განადგურება *(ამჯერად ამ საკითხებს არ განვიხილავთ).*

DBM-ფუნქციებთან მუშაობა

PHP-პროგრამებს შეუძლიათ იმუშაონ ისეთ მძლავრ მონაცემთა ბაზებთან, როგორიცაა, მაგალითად, ORACLE, SQL, MySQL და სხვ. თუმცა მონაცემების შენახვა ბაზის ფალების სახით და მათთან მუშაობა მაშინაც კი არის შესაძლებელი, როცა ზემოთ ჩამოთვლილი ბაზები ჩვენს განკარგულებაში არ იმყოფება. საქმე ისაა, რომ PHP-ს ე.წ. DBM-ფუნქციების მეშვეობით შეუძლია მონაცემთა ბაზის ფუნქციების ემულირება. ცხადია, ასეთი გზით შექმნილ ელემენტარულ ბაზებს ზემოთ მოყვანილი ბაზების ყველა შესაძლებლობა არ გააჩნიათ, მაგრამ რიგ პრაქტიკულ შემთხვევებში მათ წარმატებით შეუძლიათ, გადაწყვიტონ ჩვენ მიერ დასახული ამოცანები.

DBM-მონაცემთა ბაზის გახსნა-დახურვა

ამ მიზანს ემსახურება dbmopen() ფუნქცია. მას ესაჭიროება ორი არგუმენტის გადაცემა:

<u>DBM-ფაილებამდე გზის</u> და <u>ალმების სტრიქონის.</u>

განსახილველი ფუნქციით სარგებლობისას ვქმნით ცვლადს - გაღებული მონაცემთა ბაზის იდენტიფიკატორს, რომლითაც შემდგომ სარგებლობენ სხვა DBM-ფუნქციებიც.

შევნიშნოთ, რომ ამ ფუნქციის გამომყენებულ პროგრამებს უნდა ჰქონდეს იმ კატალოგთან შეღწევაზე ნებართვა, რომელშიც განთავსებულია მონაცემთა ბაზა.

ქვემოთ ჩამოთვლილი ალმები განსაზღვრავენ მონაცემთა ბაზასთან მუშაობის რეჟიმებს:

- r – *მონაცემთა ბაზა იღება მხოლოდ წაკითხვისათვის;*
- w – *მონაცემთა ბაზა იღება წაკითხვისა და ჩაწერისთვის;*
- c – *დასაშვებია მონაცემთა ბაზის შექმნა (ანდა მისი წაკითხვა და კორექტირება, თუ იგი უკვე არსებობს);*
- n – *იქმნება ახალი მონაცემთა ბაზა (ნადგურდება ამ სახელის მქონე ბაზის შემცველობა, თუკი იგი უკვე არსებობს).*

ქვემოთ მოყვანილ მაგალითში იღება მონაცემთა ბაზა, მაგრამ თუკი ასეთი სახელის მქონე ბაზა არ არსებობს, მაშინ იქმნება ახალი:

$dbh = dbmopen(“./data/products”, “c”) or die/(“ვერ მოხერხდა DBM-ის გაღება”);

ბაზებთან მუშაობას დასამთავრებლად ფუნქციაში მიეთითება გაღებული ბაზის იდენტიფიკატორი. მოცემულ შემთხვევაში მას ექნება სახე:

dbmclose($dbh);

ბაზებში მონაცემების დამატება

ამ მიზანს ემსახურება dbminsert() ფუნქცია. მოვიყვანოთ მისი გამოყენების მაგალითი:

<u>ლისტინგი 58</u>

```
<HTML>
<head>
<title> DBM_მონაცემთა ბაზაში მონაცემთა დამატება </title>
</head>
<body>
Adding produts now...
 <?php
$dbh = dbmopen( "./data/products", "c") or die( "couldn't open DBM" );

dbminsert($dbh, "snic crewdriver", "23.20" );
dbminsert($dbh, "Tricorder", "55.50" );
dbminsert($dbh, "ORAC AI", "2200.500" );
dbminsert($dbh, "HAl 2000", "4500.50" );

dbmclose($dbh);
 ?>
</body>
</html>
```

ხაზი გავუსვათ შემდეგ გარემოებას. მონაცემთა ბაზაში ყოველი სახის ინფორმაცია, მათ შორის ციფრულიც, სტრიქონის სახით შეიტანება. ამ მოთხოვნის გამო ზედა მაგალითში ფასის ამსახველი მონაცემიც ბრჭყალებში ჩავსვით.

თუ ამის შემდეგ შეცდომით იგივე სახის მქონე ველის დამატებას შევეცდებით, ამას არავითარი შედეგი არ მოჰყვება:

სისტემა დააბრუნებს არა ოპერაციის წარმატებულად ჩატარების გამომხატველ “0” კოდს *(ან შეცდომის შესახებ მაუწყებელ “-1”-ს),* არამედ – “1”-ს.

ბაზის მონაცემთა კორექტირება

ბაზაში მონაცემთა კორექტირება ხორციელდება dbmreplece() ფუნქციის გამოყენებით. ზედა პროგრამაში ჩამატების ფუნქცია სწორედ ამ ფუნქციით შევცვალოთ. ცხადია, ვცვლით ველების მნიშვნელობებსაც:

ლისტინგი 59

```
<HTML>
<head>
<title> DBM_მონაცემთა ბაზაში მონაცემთა დამატება ან შეცვლა
</title>
</head>
<body>
Adding produts now...
 <?php
$dbh = dbmopen( "./data/products", "c") or die( "couldn't open DBM" );

 dbmreplace($dbh, "snic screwdriver", "25.20" );
 dbmreplace($dbh, "Tricorder", "56.50" );
 dbmreplace($dbh, "ORAC AI", "2209.50" );
 dbmreplace($dbh, "HAl 2000", "4535.50" );

 dbmclose($dbh);
 ?>
</body>
</html>
```

ბაზიდან მონაცემების წაკითხვა

ბაზიდან მონაცემები dbmfetch() ფუნქციის მეშვეობით წაიკითხება. მას არგუმენტად უნდა გადაეცეს გაღებული მონაცემთა ბაზის იდენტიფიკატორი და წასაკითხი ელემენტის სახელი. ფუნქცია მონაცემებს სტრიქონის სახით დაგვიბრუნებს.

მაგალითად, Tricorder ელემენტის ფასის გასაგებად ვწერთ შემდეგ ოპერატორს:

```
$price = dbmfetch ($dbh, “Tricorder”);
```

თუ ამ სახელის მქონე ელემენტი ბაზაში არ არსებობს, ფუნქცია დააბრუნებს ცარიელ სტრიქონს.

მაგრამ როგორ მოვიქცეთ, თუკი ბაზაში შენახული ელემენტების სახელები ჩვენთვის უცნობია, ანდა ზუსტად არ გვახსოვს?

ვთქვათ, გვაინტერესებს ბაზაში შეტანილი ყოველი პროდუქტის ფასი. ასეთ შემთხვევებში დაგვეხმარება dbmfirstkey() ფუნქცია, რომელიც გვიბრუნებს ბაზაში პირველი ელემენტის სახელს. შესაძლებელია, ეს ელემენტი პირველად შეტანილი არც გახლდეთ *(ასეთია DBM ბაზის შიდა ორგანიზების თავისებურება).* თუმცა ეს საქმეს მაინცდამაინც არ გვირთულებს - dbmnexthey() ფუნქციის მეშვეობით ჩვენ შეგვიძლია ბაზის სხვა ელემენტებსაც *(აქ მათ გასაღებებს უწოდებენ)* მივადგეთ. ელემენტის

მნიშვნელობის წაკითხვა კი, როგორც ვიცით, ხორციელდება dbmfetch() ფუნქციით.

სამივე ფუნქციისათვის საჭირო გახლავთ მონაცემთა ბაზის იდენტიფიკატორის მითითება.

მივიღოთ ინფორმაცია ბაზაში არსებული ყოველი საქონლის შესახებ:

<u>ლისტინგი 60</u>

```
<HTML>
<head>
<title> DBM_მონაცემთა ბაზიდან მონაცემთა წაკითხვა </title>
</head>
<body>
Here at the Impossible Gadget Shop
we're offering the following exciting
products:
<p>
<table border=1 cellpadding ="5">
  <tr>
   <td aling="center"> <b>product</b></td>
   <td aling="center"> <b>price</b></td>
  </tr>
 <?php
$dbh = dbmopen( "./data/products", "c") or die( "couldn't  open DBM" );
    $skey = dbmfirstkey($dbh);
    while ($skey !="")
    {
        $value=dbmfetch($dbh,$key);
        print "<tr><td align =\"left\">$skey </td>";
        print "<td align = \"right\">\$$value </td></tr>";
        $key =dbmnextkey($dbh,$key);
}
 dbmclose($dbh);
 ?>
 </table>
</body></html>
```

ბაზაში ელემენტის არსებობის შემოწმება

მონაცემების კორექტირების წინ უმჯობესია, შევამოწმოთ ბაზაში მისი არსებობა. ამ მიზნით ვიყენებთ dbmexists() ფუნქციას:

```
if (dbmexists ($dbh, “Tricorder”))
    print dbmfetch ($dbh, “Tricorder”))
```

მონაცემთა ბაზიდან ელემენტის ამოგდება

ხორციელდება მარტივად dbmdelete() ფუნქციის მეშვეობით. მაგალითად:

```
dbmdelete($dbh, “Tricorder”)
```

ბაზაში უფრო რთული სტრუქტურების შენახვა

რადგანაც ბაზაში ინფორმაცია მხოლოდ სტრიქონის სახით ინახება, პრინციპში შესაძლებელია *(მაგრამ საკმაოდ რთულდება)* მასში მასივებისა და ობიექტების შენახვა. ამ მიზნით იყენებენ სპეციალურ ფუნქციებს. თუმცა როცა საქმე ასეთი სტრუქტურების გამოყენებამდე მიდის, მაშინ უმჯობესია ვისარგებლოთ უფრო მძლავრი ბაზების შესაძლებლობებით.

მონაცემთა ბაზებთან კავშირი MySQL-ის მაგალითზე

პირველი, რაც უნდა გავაკეთოთ, ეს გახლავთ სერვერთან დაკავშირება. PHP-ში ამ მიზანს ემსახურება mysql-connect() ფუნქცია. მას გააჩნია სამი არგუმენტი:

1. *კომპიუტერის სახელი,*
2. *მომხმარებლის სახელი,*
3. *პაროლი.*

თუ სერვერთან დაკავშირება მოხერხდა, ფუნქცია გვიბრუნებს <u>მიერთების იდენტიფიკატორს</u>. მას ცვლადის მნიშვნელობად იმახსოვრებენ და ბაზასთან შემდგომი მუშაობისთვის იყენებენ.

ვაჩვენოთ აღნიშნული ფუნქციის გამოყენებით ბაზასთან შეერთების მაგალითი:

```
$link = mysql_connect (“locallhost”, “root”, “nickel”);
if ( ! $link )
    die  (“couldn’t connect to MySQL”);
```

აღვნიშნოთ, რომ mysql_connect() ფუნქციაში კომპიუტერის სახელად “locallhost”-ის გამოყენებით კავშირს ვამყარებთ ჩვენსავე კომპიუტერზე განთავსებულ SQL ბაზასთან.

Apache Web-სერვერთან მიერთება mysql-pconnect() ფუნქციის მეშვეობითაც არის შესაძლებელი. ასეთ შემთხვევაში სერვერთან შეერთება არ წყდება პროგრამის მუშაობის დამთავრებისთანავე ანდა mysql_close() ფუნქციის გამოძახებით.

მონაცემთა ბაზის არჩევა

სერვერთან შეერთების შემდეგ ვირჩევთ მონაცემთა ბაზას mysql_select_db() ფუნქციის მეშვეობით. ამ ფუნქციას პირველ არგუმენტად გადავცემთ ბაზის სახელს, მეორე – არააუცილებულ არგუმენტში მიეთითება სერვერთან მიერთების იდენტიფიკატორი. თუ ეს იდენტიფიკატორი არ ვუჩვენეთ, დუმილით აირჩევა ბოლო მიერთების იდენტიფიკატორი. მაგალითად:

```
$database = ”sample”;
mysql_select_db($sample)  or  die  (“couldn’t  open  $sample”);
```

შეცდომების დამუშავება

აქამდე ფუნქციის შესრულებისას შეცდომის შემთხვევაში die ფუნქციის მეშვეობით ვწყვეტდით პროგრამის მუშაობას. მაგრამ შესაძლებელია შეცდომის შესახებ დაწვრილებითი ინფორმაციის მიღებაც, რაც მის აღმოსაფხვრელად გამოგვადგება. კერძოდ, mysql_errno() ფუნქციით განისაზღვრება შეცდომის ნომერი, ხოლო mysql_error()-ით – სტრიქონი მისი აღწერით.

ვაჩვენოთ mysql_error()-ფუნქციის გამოყენების მაგალითი:

<u>ლისტინგი 61</u>

```
<HTML>
<head>
<title>მონაცემთა ბაზის გაღება, მიერთება და ამორჩევა </title>
</head>
<body>
<?php
$user = "harry";
$pass = "elbomonkey";
$db = "sample";
$link = mysql_connect( "localhost", $user, $pass );
if ( ! $link )
   die( "Coudn't connect to MySQL");
print “Successfully connected to server<P>”;
mysql_select_db( $db) or die (“Could/t open $db: “.mysql_error());
print “Successfully selected database \”$db\”<P>”;
mysql_close( $link);
?>
</body>
</html>
```

თუ ამ პროგრამაში $db ცვლადს, არარსებული ბაზის სახელს, *(ვთქვათ, “sample2”-ს)* მივანიჭებთ, die() ფუნქცია გამოგვიყვანს შემდეგ შეტყობინებას:

Couldn’t open sample2: Access denied for user: ‘harry@localhost’ to database ‘sample2’

ცხრილში მონაცემების ჩამატება

დავუშვათ, ვქმნით Web-კვანძს, რომელშიც მომხმარებლებს შეუძლიათ თავიანთვის დომენური სახელების ყიდვა.

Sample მონაცემთა ბაზაში შევქმნათ ცხრილი domains სახელითა და 4 ველით:

```
create  table domains ( id INT NOT NULL AUTO_INCREMENT,
                        PRIMARY EY ( id ),
                        domain VARCHAR (20),
```

```
sex CHAR (1),
mail VARCHAR (20) );
```

ამ ცხრილში კონკრეტული მონაცემების ჩასამატებლად საჭიროა ბაზას მივმართოთ SQL-მოთხოვნით. ამისათვის PHP-ში გათვალისწინებულია mysql-query() ფუნქცია, რომელსაც გადაეცემა <u>მოთხოვნის სტრიქონი</u> და არააუცილებელი მიერთების იდენტიფიკატორი:

<u>ლისტინგი 62</u>

```
<html>
<head>
<title> ცხრილში ჩანაწერის დამატება
 </title>
</head>
<body>
<?php
$user = "harry";
$pass = "elbomonkey";
$db = "sample";
$link = mysql_connect( "localhost", $user, $pass );
if ( ! $link )
            die( "Coudn't connect to MySQL");
mysql_select_db( $db, $link)
     or  die ("Couldn't open $db: ".mysql_error() );
$query=”INSERT INTO domains ( domain, sex, mail )
          values( ‘123xyz.com’, ‘F’, sharp@adomain.com’ )”;
mysql_query( $query, $link)
     or  die ("Couldn't add data to \”domains\”  table: “.mysql_error() );
mysql_close( $link);
?>
</body>
</html>
```

მივაქციოთ ყურადღება: ბაზაში სტრიქონის ჩამატებისას id-ველისათვის მნიშვნელობა არ გვიჩვენებია. გავიხსენოთ, გასაღებური ველის მნიშვნელობა სისტემის მიერ ავტომატურად განისაზღვრება.

ამ პროგრამას რამდენჯერაც შევასრულებთ, ცხრილს იმდენჯერვე დაემატება ერთი და იგივე მონაცემების შემცველი ახალი ჩანაწერი *(განსხვავება მხოლოდ id ველის მნიშვნელობაში იქნება).*

ქვემოთ მოგვყავს პროგრამა, რომელიც ცხრილში ამატებს მომხმარებლის მიერ ფორმაში შეტანილ მონაცემებს:

<u>ლისტინგი 63</u>

```
<HTML>
<HEAD>
<TITLE> ბაზაში მომხმარებლის მიერ მოწოდებული ინფორმაციის
          დამატება
 </title>
```

```
</head>
<BODY>
<?php
if ( isset( $domain ) && isset( $sex ) &&  isset( $domain ) )
 {
    // check user input here!
    $dberror = "";
    $ret = add_to_database( $domain, $sex, $mail, $dberror);
    if ( ! $ret )
          print "Error: $dberror<BR>";
    else
          print “Thank you very much”;
  }
else  {
          write_form();
         }

function add_to_database( $domain, $sex, $mail, &$dberror)
{
  $user = "harry";
  $pass = "elbomonkey";
  $db = "sample";
  $link = mysql_pconnect( "localhost", $user, $pass );

if ( ! $link )
  {
    $dberror = "Couldn't connect to MySQL server";
    return false;
  }
if (! mysql_seleqt_db( $db, $kink ) )
  {
    $dberror =mysql_error();
    return faise;
   }

$query = "INSERT INTO domains ( domain, sex, mail )
values( '$domain','$sex','$mail' )";

if ( ! mysql_query( $query, $link ) )
  {
    $dberror = mysql_error();
    return false;
  }
return true;
}

function write_form()
{
```

```
    global$PHP_SELF;
    print"<form action=\'$PHP _SELF\" method=\"POST\">\n";
    print"<input type=\"text\" name=\"domain\"> ";
    print"The domain you would like<p>\n";
    print"<input TYPE=\"text\" name\"mail\"> ";
    print"Your mail address<p>\n" ;
    print"<select name=\"sex\">\n";
    print"\t<option value=\"F\" Female\n";
    print"\t<option value=\"M\"> Male\n";
    print"\select>\n";
    print"<input type=\"submit\" value=\"submit! \">\n</form>\n";
}

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

თავდაპირველად პროგრამა ამოწმებს $domain, $mail და $sex ცვლადების არსებობას *(ისინი ფორმიდან გადმოიცემა).* დადებითი პასუხის შემთხვევაში გამოიძახება add_to_database() ფუნქცია. *(მანამდე კი განისაზღვრება ჯერჯერობით ცარიელი მნიშვნელობის მქონე $dberror ცვლადი).*

თუ აღმოჩნდა, რომ ფორმა ჯერ არ გადმოცემულა *(მივაქციოთ ყურადღება - ფორმა გადმოეცემა ამავე პროგრამას)* ან მასში თუნდაც ერთი ცვლადის მნიშვნელობა არ იქნება განსაზღვრული, გამოიძახება write-form() ფუნქცია.

add_to_database() ფუნქციას 4 არგუმენტი გააჩნია - მომხმარებლის მიერ გადმოცემულ 3 მნიშვნელობას ემატება $dberror ცვლადთან დაკავშირებული $dberror სტრიქონიც. რომელიმე ოპერაციის უშედეგოდ დასრულებისას ხდება შეცდომის შესახებ ინფორმაციის ჩაწერა ჩვენ მიერ გამოცხადებულ $dberror გარე ცვლადშიც *(და არა მარტო მის ასლში).*

როგორც ვხედავთ, add_to_database() ფუნქციაში ხორციელდება მცდელობები:

- *სერვერთან დაკავშირების,*
- *ბაზის შერჩევის,*
- *SQL-მოთხოვნის გაცემის.*

თუ რომელიმე მათგანი წარუმატებლად დასრულდა, შეცდომის შესახებ ინფორმაცია გადაეცემა $dberror ცვლადს, ხოლო add_to_database() ფუნქცია PHP-პროგრამას დაუბრუნებს false მნიშვნელობას.

საპირისპირო შემთხვევაში add_to_database() ფუნქცია პროგრამას უბრუნებს true მნიშვნელობას. ნებისმიერი შემთხვევისთვის პროგრამას გამოჰყავს შესაბამისი შეტყობინება *(მოწმდება $ret ცვლადი).*

ავტომატურად კორექტირებადი ველის მნიშვნელობის გაგება

ამ მიზნით დასაშვებია SQL-მოთხოვნის გამოყენება, მაგრამ თუ გვსურს ინფორმაცია ჩანაწერის შექმნისთანავე მივიღოთ, უნდა ვისარგებლოთ mysql_insert_id() ფუნქციით, რომელიც გვიბრუნებს ბოლო ჯერზე ცხრილისთვის დამატებული ჩანაწერის გასაღებური ველის მნიშვნელობას.

მაგალითად, როცა გვსურს მომხმარებელს შევატყობინოთ ნომერი, რომელიც მის შეკვეთას მიენიჭა, ვიყენებთ ასეთ პროგრამულ ფრაგმენტს:

```
$query="INSERT INTO domains ( domain, sex, mail ) values ('$domain',
'$sex', '$mail' )";

mysql_query( $query, $link );

$id=mysql_insert_id ();

print "Thank you. Your tranzaction number is $id. Please quote it in any queries.";
```

ბაზიდან მონაცემების წაკითხვა

ამჯერად ვიყენებთ Select ტიპის SQL-მოთხოვნას. მისი წარმატებით შესრულებისას mysql_query() ფუნქცია აბრუნებს <u>მოთხოვნის შედეგის იდენტიფიკატორს</u>, რომელიც შეიძლება გადავცეთ მიღებული შედეგის დამამუშავებულ ფუნქციებს.

SQL-მოთხოვნით მოძებნილი ჩანაწერების რიცხვის განსაზღვრა

ამ ინფორმაციის მიღება შესაძლებელია mysql_num_rows() ფუნქციით. იგი იყენებს ერთადერთ არგუმენტს – მანამდე შესრულებული მოთხოვნის იდენტიფიკატორს. მაგალითად:

<u>**ლისტინგი 64**</u>

```
<HTML>
<head>
<title>სტრიქონების რიცხვის განსაზღვრა</title>
</head>
<body>
<?php

$user = "harry";
$pass = "elbomonkey";
$db = "sample";
$link = mysql_connect( "localhost", $user, $pass );
if ( ! $link )
  die ( "Could't connect to MySQL");
   mysql_select_db($db, $link)
        or die ("Couldn't open $db: ".mysql_error() );
   $result=sql_query("SELECT *FROM  domains");
   $num_rows=mysql_num_rows( $result);
```

```
  print "There are currently $num_rows rows in the table<P>";
 mysql_close ($link);
?>
</body>
</html>
```

მოთხოვნის შედეგების ჩათვალიერება

ეს პროცესი ციკლში შეიძლება განვახორციელოთ. მოძებნილი ჩანაწერებისათვის PHP ქმნის ე.წ <u>შინაგან მაჩვენებელს</u>. როცა მიმდინარე ჩანაწერთან ვმუშაობთ, ეს მაჩვენებელი მიგვითითებს მომდევნო ჩანაწერის პოზიციაზე. ამასთან, mysqe-fetch() ფუნქციით შესაძლებელია თითოეული ჩანაწერისათვის მივიღოთ მისი ველებისგან შემდგარი მასივი. ცხადია, ამ ფუნქციასაც უნდა გადაეცეს მოთხოვნის იდენტიფიკატორი. ჩანაწერების ჩათვალიერების ბოლოს ფუნქცია გვიბრუნებს false-ს:

<u>ლისტინგი 65</u>

```
<HTML>
<head>
<title>ცხრილის ყველა ჩანაწერის გამოყვანა</title>
</head>
<body>
<?php

$user = "harry";
$pass = "elbomonkey";
$db = "sample";
$link = mysql_connect( "localhost", $user, $pass );
if ( ! $link )
  die ( "Could't connect to MySQL");
   mysql_select_db($db, $link)
        or die ("Couldn't open $db: ".mysql_error() );
   $result=sql_query("SELECT *FROM  domains");
   $num_rows=mysql_num_rows( $result);
  print "There are currently $num_rows rows in the table<P>";
  print "<table border=1>\n";
  while ( $a_row =mysql_fetch ($result))
     {
     print "<tr>\n";
     foreach ($a_row as $field)
         print "\t<td>$field</td>\n";
         print"</tr>\n";
     }
print "</table>\n";
mysql_close ($link);
?>
</body>
</html>
```

WHILE-ციკლში mysql_fetch() ფუნქციის მიერ დაბრუნებულ მნიშვნელობას ვანიჭებთ $a_row ცვლადს.

WHILE-ციკლში გვაქვს კიდევ ერთი ციკლი foreach. მისი მეშვეობით ხდება მასივის ჩათვალიერება და მისი თითოეული ელემენტის გამოყვანა ცხრილის უჯრედში.

შევნიშნოთ, რომ ჩანაწერის ველებს სახელითაც შეიძლება მივმართოთ. ამასთან, შეიძლება გამოვიყენოთ შემდეგი ორი გზიდან ერთ-ერთი:

- *პირველი გულისხმობს WHILE-ის სათაურში mysql_fetch() ფუნქციის ნაცვლად mysql_fetch_array-ის გამოყენებას (იგი ასოცირებულ მასივს გვიბრუნებს;*
- *მეორე – სტრიქონის აღქმას ობიექტად და მისი თვისებების mysql_fetch_object() ფუნქციით წაკითხვას:*

```
print “<TABLE BORDER=1>\n”;
while ( $a_row = mysql_fech_array ($result) )
        {
          print “<TR>\n”;
          print “<TD>$a_row [mail]</TD><TD>$a_row [domain]</TD>\n”;
          print “</TR>”\n”;
        }
print “</TABLE>\n”;

print “<table border=1>\n”;
while ( $a_row=mysql_fetch_object ($result) )
        {
          print “<tr>\n”;
          print “<td> $a_row -> mail</td><td>$a_row -> domain </td>\n”;
          print “</tr>\n”;
        }
 print “</table>\n”;
```

მონაცემების შეცვლა

mysql-query() ფუნქციას ვიყენებთ ცხრილში მონაცემების შესაცვლელადაც.

მოთხოვნაში ფიგურირებს UPDATE ინსტრუქცია, რომელსაც შეიძლება, ვთქვათ, ასეთი სახე ჰქონდეს:

UPDATE *ცხრილის-სახელი* SET *ველის-სახელი* =*ახალი-მნიშვნელობა*
Where *პირობის შესრულება;*

UPDATE-ინსტრუქციის წარმატებით შესრულება ჯერ კიდევ არ ნიშნავს მონაცემების ფაქტობრივად შეცვლას, რისი შემოწმებაც შეიძლება მოხდეს mysql_affected_rows() ფუნქციით *(ეს ფუნქციაც მხოლოდ მიერთების იდენტიფიკატორს საჭიროებს არგუმენტად და აქაც, თუ ამ*

იდენტიფიკატორს არ მივუთითებთ, გამოყენებული იქნება უკანასკნელი მიერთების იდენტიფიკატორი).

შევნიშნოთ, რომ ამ ფუნქციის გამოყენება შეიძლება მონაცემებში ნებისმიერი ცვლილებების შეტანისას.

ქვემოთ ნაჩვენებია პროგრამა, რომელიც მონაცემთა ბაზის ადმინისტრატორს საშუალებას აძლევს, ცვლილებები შეიტანოს domains ცხრილის domain ველში:

<u>ლისტინგი</u>

```
<HTML>
<head>
<title>mysql-ფუნქციის გამოყენება ცხრილში მონაცემების
      შესაცვლელად
</title>
</head>
<body>
<?php

$user = "harry";
$pass = "elbomonkey";
$db = "sample";
$link = mysql_connect( "localhost", $user, $pass );
if ( ! $link )
  die ( "Could't connect to MySQL");
   mysql_select_db($db, $link)
       or die ("Couldn't open $db: ".mysql_error() );
   if ( isset ($domain) && isset ($id))
       {
       $query="UPDATE domains SET domain = '$domain' where id=$id";
      $result = mysql_query ($query);
      if  (! $result)
         die ("Couldn't update: ".mysql_error());
     print "<h1>Table updated ". mysql_affected_rows().
         "row(s) hanged</h1><p>";
   ?>

  <form action = "<? print $PHP_SELF ?>" method = 'POST'>
   <select name = "id">
   <?
   $result=mysql_query("SELECT domain, id FROM domains");
   while ($a_row = mysql_fetch_object ($result))
       {
       print "<OPTION VALUE = \"$a_row->id\"";
       if ( isset ($id) && $id == $a_row->id)
           print "SELECTED";
       print" > $a_row->domain\n";
       }
```

```
  mysql_close ($link);
?>
</select>
<input type = "text" name = "domain">
</form>
</body>
</html>
```

სერვერთან შეერთებისა და მონაცემთა ბაზის არჩევის შემდეგ ვამოწმებთ $domain და $id ცვლადების არსებობას. დადებითი პასუხის შემთხვევაში ვაყალიბებთ SQL-მოთხოვნას. მასში domain ველის მნიშვნელობას ვცვლით იმ ჩანაწერებისთვის, რომლებისთვისაც id ველის მნიშვნელობა ემთხვევა *(ფორმაში განსაზღვრულ)* $id ცვლადის მნიშვნელობას.

აღვნიშნოთ, რომ შეცდომის შესახებ შეტყობინებას ვერ მივიღებთ, თუ $id-ისთვის შევირჩევთ არარსებულ მნიშვნელობას, ან როცა domain ველის მნიშვნელობა დაემთხვევა ჩვენ მიერ შეთავაზებულს. უბრალოდ, ასეთ შემთხვევებში mysql_affected_rows() ფუნქცია დაგვიბრუნებს “0”-ს. საპირისპირო შემთხვევებში კი *(მაგალითად, ზემოთ მოყვანილი პროგრამისათვის)* დაგვიბრუნდება მნიშვნელობა “1”, რომელიც ეკრანზეც აისახება.

პროგრამას ეკრანზე გამოჰყავს ფორმა, რომლის დახმარებითაც შეიძლება ცხრილში ცვლილებები შევიტანოთ. Domain და id ველების შესახებ ინფორმაციას გვაწვდის mysql_query() ფუნქცია, რომელიც HTML-ტექსტში სიის ფორმირებისათვის გამოიყენება. სიაში ვირჩევთ მონაცემებს, მათი შემდგომი კორექტირების მიზნით. თუ ფორმა უკვე გადავეცით და ამორჩეული id მნიშვნელობა ემთხვევა მიმდინარე ველის მნიშვნელობას, Option ელემენტში შეგვყავს Selected სტრიქონი, რათა ახალი მნიშვნელობა გამოირჩეს ფორმაში.

გარემოს შესახებ ინფორმაციის შემცველი ცვლადები

ზოგიერთ ასეთ ცვლადს, რომელთაც PHP ან სერვერი ქმნის, უკვე გავეცანით.

სერვერზე განთავსებულ PHP-პროგრამიდან ჩვენ შეგვეძლება ამ ცვლადების მეშვეობით მივიღოთ ინფორმაცია, მაგალითად, საკუთარი საიტის მნახველების შესახებ. მაგრამ რადგანაც ყოველთვის ვერ ხერხდება სისტემაში ან სერვერზე არსებულ რესურსებთან შეღწევა, ჯობს, წინასწარ შევამოწმოთ მცდელობათა შედეგები. მოვიყვანოთ ამის მაგალითი:

<u>ლისტინგი 67</u>

```
<HTML>
<head>
```

```
<title>ბროუზერის ფანჯარაში ცვლადების მნიშვნელობების
        გამოყვანა
</title>
</head>
<body>
<?php
$envs = array ( "HTTP_REFERER", "HTTP_USER_AGENT",
"REMOTE_ADDR", "REMOTE_HOST",
                "QUERY_STRING", "PATH_INFO");
foreach ($envs as $env)
   {
   if  ( isset ($$env))
       print "$env: ${$env}<br>";
   }
?>
</body>
</html>
```

ვხედავთ, რომ, სტრიქონების ასეთივე სახელის მქონე ცვლადებად გარდაქმნის მიზნით, პროგრამაში გამოყენებულია დინამიკური ცვლადები.

ეკრანზე აისახება ზემოთ მოყვანილი პროგრამის მუშაობის შედეგები. მიღებული მონაცემები გენერირდება ამ პროგრამის გამოძახების შედეგად სხვა WEB-ფურცლიდან შემდეგ კავშირზე დაწკაპუნებით:

```
<A HREF=’eg13.1.php/my_path_info?query_key=query_value’> go </a>
```

ჩანს, რომ პროგრამის გამოსაძახებლად კავშირი იყენებს ფარდობით გზას. დამატებითი ინფორმაცია გზის შესახებ ჩაიწერება დოკუმენტის my_path_info სახელის შემდეგ და გადაეცემა $PATH_INFO ცვლადს.

? კითხვის ნიშნის შემდეგ განთავსებულია ე.წ. <u>მოთხოვნის სტრიქონი</u> query_key=query_value, რომელიც აისახება შესაბამის $QUERY-STRING ცვლადში.

მოთხოვნის სტრიქონი შედგება &-სიმბოლოებით გაყოფილ შემდეგი წყვილებისგან:

სახელი/მნიშვნელობა

ამასთან, ორაზროვნობის თავიდან ასაცილებლად ეს წყვილები ექვემდებარება ე.წ. <u>URL-კოდირებას</u> – “საეჭვო” სიმბოლოები იცვლება მათი 16-ობითი ეკვივალენტებით.

PHP უზრუნველყოფს სტრიქონში მითითებული თითოეული ცვლადის გლობალურ ცვლადად აღქმის შესაძლებლობას. მაგალითად, ჩვენი შემთხვევისთვის ეს გლობალური ცვლადი იქნება $query_value, ხოლო მისი დეკოდირებული მნიშვნელობა კი – “query_value”.

კლიენტის სერვერთან HTTP ოქმით შეერთების ზოგიერთი საკითხი

ინტერნეტში ჰიპერტექსტი, ჩვეულებრივ, http ოქმით გადაიგზავნება. ამასთან, კლიენტი და სერვერი ერთმანეთს ინფორმაციას უცვლიან საკუთარი თავის შესახებაც. ბევრი რამ ამ ინფორმაციიდან გარემოს ამსახველ ცვლადებშიც შეიძლება მოვიპოვოთ.

<u>მოთხოვნა</u>

კლიენტის მიერ სერვერიდან მონაცემების მოთხოვნა სამ კომპონენტს შეიძლება მოიცავდეს:

- *მოთხოვნის სტრიქონი* // აუცილებელი კომპონენტია
- *სათაურების სტრიქონი* // ზოგჯერ არ იყენებენ
- *მოთხოვნის სახელი* // ზოგჯერ არ იყენებენ

მოთხოვნის სტრიქონში მითითებულია:

- მეთოდი (ჩვეულებრივ, აირჩევა GET, HEAD ან POST);
- მოთხოვნილი დოკუმენტის მისამართი;
- გამოყენებული HTTP-ის ვერსია (HTT /1.0 ან HTTP/1.1).

მოთხოვნის სტრიქონის ტიპური მაგალითია:

GET / mydoc.html HTTP.1.0

მოყვანილ მაგალითში კლიენტი სერვერს გადასცემს GET მოთხოვნას, ანუ იგი მოითხოვს მთელი დოკუმენტის გადმოგზავნას, თვითონ კი, ფაქტობრივად, მონაცემებს არ აგზავნის *(დასაშვებია მხოლოდ მცირე მოცულობის მონაცემები "მივაბათ" მოთხოვნის სტრიქონის სახით URL-მისამართს).*

HEAD მეთოდს მაშინ ვიყენებთ, როცა მხოლოდ დოკუმენტის შესახებ გვსურს ინფორმაციის მიღება.

POST მეთოდი გამოიყენება კლიენტიდან სერვერზე უფრო "სოლიდური" ინფორმაციის გადაგზავნის საჭიროებისას *(მაგალითად, HTML-ფორმების).*

როგორც ზემოთ აღვნიშნეთ, ზოგჯერ იფარგლებიან მოთხოვნის მხოლოდ პირველი კომპონენტით. ოღონდ ასეთ შემთხვევაში სერვერს მოთხოვნის დასრულების შესახებ უნდა ვამცნოთ ამ მოთხოვნის ბოლოს ცარიელი სტრიქონის გადაგზავნით.

კლიენტების უმეტესობა სერვერს უგზავნის *სახელი / მნიშვნელობა* წყვილებისაგან შემდგარ სათაურების განყოფილებასაც. ზოგიერთი ამ წყვილებიდან მისაწვდომი ხდება გარემოს ცვლადების მეშვეობით. წყვილები *(სათაურები)* შედგება ერთმანეთისგან ორწერტილით გამოყოფილი გასაღების დასახელებისა და მნიშვნელობისაგან.

ცხრილში მოყვანილია ინფორმაცია ზოგიერთი გასაღების შესახებ:

სახელი	აღწერა
Accept	ინფორმაციის მატარებელთა ტიპები, რომლებთანაც შეუძლია კლიენტს მუშაობა.
Accept-Encoding	მონაცემთა შეკუმშვის ტიპები, რომლებ-თანაც შეუძლია კლიენტს მუშაობა.
Accept-Charset	სიმბოლოთა ცხრილები, რომელთაც კლიენტი უპირატესობას ანიჭებს.
Accept-Language	კლიენტისათვის სასურველი ენა (მაგალი-თად, “en” - ინგლისური)
Host	კომპიუტერის ქსელური სახელი, რომელსაც მომხმარებელის მოთხოვნა ეგზავნება.
Referer	დოკუმენტი, რომლიდანაც მოხდა მოთხოვნის გაცემა.
User-Agent	ინფორმაციის პროგრამა-კლიენტის ტიპისა და ვერსიის შესახებ.

GET და HEAD მეთოდებში სერვერს ცარიელი სტრიქონი ეგზავნება სათაურების უბნის, ხოლო POST მეთოდში – მოთხოვნის სხეულის შემდეგ.

რაც შეეხება მოთხოვნის სხეულს, ის ძალზე ჰგავს მოთხოვნის სტრიქონს - მასშიც ფიგურირებენ URL-კოდირებული წყვილები:

სახელი/მნიშვნელობა

ქვემოთ მოყვანილია Netscape-ბროუზერის მიერ სერვერისადმი გადაგზავნილი მოთხოვნის ტიპური ვარიანტი:

<u>ლისტინგი 68</u>

GET / HTTP / 1.0

Referer: http:/ zink.demon.co.uk :8080/ matt/ php-book/ network/ test2.php

Connection: Keep-Alive

User-Agent: Mozilla 4.6 (XLL; T; Linux 2.2.6-15ampac ppc)

Host : www.corrosive.co.uk

Accept: image/ gif,image/ x-xbitmap, image/ jpg,image/ pjpeg, image/png, */*

Accept-Encoding: gzip

Accept-Language: en

Accept-Charset: iso-8859-1, *, utf-8

<u>პასუხი</u>

მოთხოვნის მიღების შემდეგ სერვერი კლიენტს უგზავნის პასუხს, რომელიც, ჩვეულებრივ, ასევე სამი ნაწილისგან შედგება:

- *სტატუსის სტრიქონი;*
- *სათაურების უბანი;*
- *მოთხოვნის სხეული.*

ზოგიერთი სათაური შეიძლება ისეთივე იყოს, როგორც წინა შემთხვევაში *(პირველ ყოვლისა, ეს ეხება ინფორმაციას მესამე პუნქტის შესახებ).*

სტატუსის სტრიქონი შედგება სერვერის მიერ გამოყენებული HTTP ოქმის ვერსიის, პასუხის კოდისა და ამ კოდის აღმწერი ინფორმაციისგან.

მოვიყვანოთ ზოგიერთი კოდი თავისი აღწერილობით:

კოდი	**ტექსტი**	**აღწერა**
200	OK	მოთხოვნა წარმატებით შესრულდა. მოთხოვნილი მონაცემები გადმოიცა.
301	Moved Permanetly (მონაცემები გადაადგილებულია)	განლაგების ამსახველ სათაურში გამოდის ახალი მისამართი.
302	Moved Temporarily (მონაცემები დროებით ადგილშეცვლილია)	განლაგების ამსახველ სათაურში გამოდის ახალი მისამართი.
404	Not Found (მონაცემები ვერ მოიძებნა)	ამ მისამართზე მონაცემები ვერ მოიძებნა.
500	Internal Server Error (სერვერის შიდა შეცდომა)	პრობლემებია სერვერზე ან CGI-პროგრამაში.

აი, წარმატებულად შესრულებული მოთხოვნის ამსახველი პასუხი: HTTP/1.1 200 OK

ახლა კი მოვიყვანოთ სერვერიდან გადმოგზავნილი პასუხების ყველაზე ტიპური სათაურები:

სახელი	აღწერა
Date	მიმდინარე თარიღი
Server	სერვერის სახელი და ვერსია
Content-Type	სხეულის შემცველობის MIME-ტიპი
Content-Length	სხეულის ზომა (ბაიტებში)
Location	ალტერნატიული დოკუმენტის სრული მისამართი

ლიტერატურა:

1. Освой самостоятельно PHP4 за 24 часа. Мэт Зандстра, «Вильямс», 2001.
2. А. Гончаров. Самоучитель HTML, «Питер» , 2001.
3. А. Матросов, А. Сергеев, М. Чаунин. HTML4, «Вильямс» , 2003.
4. Учебный курс «Компьютерные сети», Microsoft Press. Санкт-Петербург, 1999.
5. Освой самостоятельно JavaScript за 24 часа. Майкл Монкур, «Вильямс», 2002.
6. Э. Кингсли-Хью, К. Кингсли-Хью, JavaScript 1.5, Учебний курс, «Питер» , 2001.

სარჩევი

- შესავალი ---------- 3
- PHP–ის დაყენება ---------- 4
- Apache სერვერის კონფიგურირება ---------- 5
- ჩვენი პირველი პროგრამა PHP-ზე ---------- 5
- ცვლადები ---------- 7
- ნაკადის მართვა ---------- 13
- ციკლები ---------- 15
- ფუნქციები ---------- 19
- მასივები ---------- 25
- ობიექტები ---------- 32
- კვლავ კლასების შესახებ---------- 39
- ფორმებთან მუშაობა ---------- 47
- ფაილებთან მუშაობა ---------- 58
- DBM-ფუნქციებთან მუშაობა ---------- 60
- მონაცემთა ბაზებთან კავშირი MySQL-ის მაგალითზე ---- 64
- გარემოს შესახებ ინფორმაციის შემცველი ცვლადები ---- 73
- კლიენტის სერვერთან HTTP ოქმით შეერთების ზოგიერთი საკითხი ---------- 75
- ლიტერატურა ---------- 78