**ლექცია 10**. **ფუნქციები, მასივები**

PHP -ის სიძლიერე მრავალწილად განპირობებულია მისი ფუნქციებით. PHP -ში არის 1000 მეტი ჩაშენებული ფუნქცია. მაგრამ PHP -ში არსებობს საშუალება მომხმარებელმა თავისი საკუთარი ფუნქციებიც შექმნას.

ფუნქცია, ეს არის ოპერატორთა ბლოკი, რომელიც შეიძლება არაერთხელ იქნას გამოყენებული პროგრამაში. ამასთან პროგრამის შესრულება თავისთავად, პროგრამის, კოდის ჩატვირთვისას არ ხდება. მისი შესრულება შესაძლებელია ფუნქციის გამოძახების პირობებში.

მომხმარებელს შეუძლია განსაზღვროს - გამოაცხადოს თავისი ფუნქცია სამოსამსახურე სიტყვის ” **function**” -ის საშუალებით.

სინტაქსისი:

**function** funqciis\_saxeli**()** **{**
*შესასრულებელი კოდი*;
**}**

ფუნქციის სახელები უნდა იწყებოდნენ ლათინური ანბანის ასოთი ან ქვედა ტირეთი. ფუნქციის სახელები შედგება ასოებისგან, ციფრებისა და ქვედა ტირესგან. სასურველია ფუნქციის სახელი ასახავდეს იმას, თუ რისი შესრულება იგეგმება ამ ფუნქციის საშუალებით. ყველა ფუნქცისას უნდა ჰქონდეს სხვადასხვა სახელი. ფუნქციის სახელი არაა რეგისტრ-მგრძნობიარე.

გამხსნელი ფიგურული ფრჩხილი მიუთითებს იმას, რომ იწყება ფუნქციის კოდი, დამხურავი ფიგურული ფრჩხილი მიუთითებს იმას, რომ დასრულდა ფუნქციის კოდი. კონკრეტული ფუნქცი­ის გამოძახება, ანუ იმ კოდის შესრულება, რომელიც მოთავსებულია გამხსნელ და დამხურავ ფრჩხილებს შორის შესაძლებელია, თუ ჩავწერთ ამ კონკრეტული ფუნქციის სახელს, მაგალითად: funqciis\_saxeli();

მოვიყვანოთ ფუნქციის გამოყენებით ცხრილის შექმნის კოდის მაგალითი:

ლისტინგი 1 ფუნქცია

!DOCTYPE html>

<html>

<body>

<table border=1>

<?php

**function tskhrili() {**

 **echo "\n<tr><td>სტრიქონი </td>**

**<td> სვეტი 1</td><td> სვეტი 2</td></tr>";**

**};**

for($a=1;$a<=5;$a++){

**tskhrili()**;

};

?>

</body>

</html>

ლისტინგ 1-ის შესრულების შემდეგ მივიღებთ:

|  |  |  |
| --- | --- | --- |
| სტრიქონი | სვეტი 1 | სვეტი 2 |
| სტრიქონი | სვეტი 1 | სვეტი 2 |
| სტრიქონი | სვეტი 1 | სვეტი 2 |
| სტრიქონი | სვეტი 1 | სვეტი 2 |
| სტრიქონი | სვეტი 1 | სვეტი 2 |

შედეგი 1. ცხრილის შექმნა ფუნქციის გამოძახებით

ჩვენს მიერ მოყვანილ მაგალითში, ლისტინგ 1-ის სახით, ფუნქცია **tskhrili()**-ის ყოველი გამოძა­ხებისას შესრულდება ერთი და იგივე ოპერატორები, ერთი და იგივე გარემოში - არცერთი პარა­მეტრი არ იცვლება. ეს ერთის მხრივ თავად ფუნქციის კონკრეტული კოდისგანაა განპირობებული, მაგრამ არანაკლებ მნიშვნელოვანის ისიც, რომ ფუნქციის გამოძახებისას, ჩვენ მას ის მნიშვნელობა არ მივაწოდეთ, რომელიც შეიძლება გამოყენებული ყოფილიყო კოდის შესრულებისას. არადა ფუნქციისთვის ყოველი გამოძახებისას სხვადასხვა მნიშვნელობების მიწოდების რეალიზაცია შესაძლებელია ე.წ. პარამეტრების საშუალებით.

ფუნქციის პარამეტრები ჩაიწერება მრგვალ ფრჩხილებში, რომელიც ფუნქციის გამოცხადებისას ფუნქციის სახელის შემდეგ იწერება. ამდენად პარამეტრების გათვალისწინებით ფუნქციის სინტაქსისია:

**function** funqciis\_saxeli**([**cvladi][=constant][,……]**)** **{**
*შესასრულებელი კოდი*;
**}**

ჩანაწერი [,……] აღნიშნავს cvladi[=constant] -ის რამდენჯერმე განმეორების საშუალებას, ამასთან ცვლადების სახელები, რა თქმა უნდა უნდა განსხვავდენოდეს. კვადრატული ფრჩხილები, როგორც გვახსოვს, არააუცილებელ ჩანაწერზე მიუთითებს.

მოვახდინოთ ლისტინგ 1-ის მოდიფიცირება, გადავცეთ ფუნქციას პარამეტრი, მაგალითად, სტრიქონის ნომრის მნიშვნელობა და შევიტანოთ იგი სტრიქონის ფორმირებისას:

ლისტინგი 2 ფუნქციისთვის პარამეტრის გადაცემა:

<!DOCTYPE html>

<html>

<body>

<table border=1>

<?php

function tskhrili(**$b**) {

 echo "\n<tr><td>სტრიქონი $b</td><td> სვეტი 1</td>

<td> სვეტი 2</td></tr>";

};

for($a=1;$a<=5;$a++){

tskhrili(**$a**);

};

?>

</body>

</html>

ლისტინგ 2-ის შესრულების შემდეგ მივიღებთ:

|  |  |  |
| --- | --- | --- |
| სტრიქონი 1 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 2 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 3 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 4 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 5 | სვეტი 1 | სვეტი 2 |

შედეგი 2. ცხრილის შექმნა ფუნქციის გამოძახებით და მისთვის არგუმენტის გადაცემით

კოდში ფუნქციის გამოძახებისას, მგვალ ფრჩხილებში ჩაწერილი არგუმენტის ეს მნიშვნელო­ბები, შესაბამისად გადაეცემათ ფუნქციის გამოცხადებისას მითითებულ ცვლადებს და ხდება ფიგურულ ფრჩხილებში მოთავსებული კოდის შესრულება.

პარამეტრების რაოდენობა შეზღუდული არაა. მაგალითად ჩვენ შეგვიძლია გადავცეთ ფუნქციას ორი არგუმენტი. ვთქვათ საჭიროა სტრიქონები გამოვიტანოთ სხვადასხვა ფერებში. ამდენად მე­ორე არგუმენტი უნდა მიუთითებდეს ფერის მნიშვნელობას. სხვადასხვა ფერები ჩავწეროთ მასივში და იქიდან ავიღოთ კონკრეტული ფერი:

ლისტინგი 3. ფუნქცია ორი არგუმენტით

<!DOCTYPE html>

<html>

<body>

<table border=1>

<?php

function tskhrili($b,$d) {

 echo "\n<tr bgcolor=$d ><td>სტრიქონი $b</td>

<td> სვეტი 1</td><td> სვეტი 2</td></tr>";

};

$feri=array("yellow","LemonChiffon","GreenYellow","LightCyan","Aqua");

for($a=1;$a<=5;$a++){

$col=$feri[$a-1];

tskhrili($a,$col);

};

?>

</body>

</html>

ლისტინგ 3-ის შესრულების შემდეგ მივიღებთ:

|  |  |  |
| --- | --- | --- |
| სტრიქონი 1 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 2 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 3 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 4 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 5 | სვეტი 1 | სვეტი 2 |

შედეგი 3. ცხრილის შექმნა ფუნქციის გამოძახებით და მისთვის ორი არგუმენტის გადაცემით

შევნიშნოთ, რომ თუმც ჩანაწერი $sakheli არ შეიძლება იყოს ფუნქციის სახელი, მაგრამ ფუნქციის ასეთმა გამოძახება - $sakheli () , მაინც კორექტულად - სწორედ იმუშაოს, რადგანაც PHP, აიღებს $sakheli ცვალდის, მნიშვნელობას და ეძებს იმ ფუნქციას, რომლის სახელი ამ მნიშვნელობას ემ­თხვევა. ფუნქციის ასეთ ტიპს უწოდებენ ფუნქცია-ცვლადს (დინამიურ ფუნქციას). განვიხილოთ მაგალითი რომელშიც გამოვიყენებთ ფუნქცია-ცვლადს :

ლისტინგი 4. ფუნქცია-ცვლადი

<!DOCTYPE html>

<html>

 <body>

<table border=1>

<?php

function tskhrili($b,$d) {

 echo "\n<tr bgcolor=$d ><td>სტრიქონი $b</td>

<td> სვეტი 1</td><td> სვეტი 2</td></tr>";

};

$feri=array("yellow","LemonChiffon","GreenYellow","LightCyan","Aqua");

for($a=1;$a<=5;$a++){

$col=$feri[$a-1];

tskhrili($a,$col);

} ?>

</table>

<br>

<table border=4>

<?php

$sakheli="tskhrili";

for($a=1;$a<=5;$a++){

$col=$feri[$a-1];

**$sakheli($a,$col)**;

}; ?>

</table>

</body>

</html>

ლისტინგ 4-ის შესრულების შემდეგ მივიღებთ:

|  |  |  |
| --- | --- | --- |
| სტრიქონი 1 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 2 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 3 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 4 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 5 | სვეტი 1 | სვეტი 2 |

|  |  |  |
| --- | --- | --- |
| სტრიქონი 1 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 2 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 3 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 4 | სვეტი 1 | სვეტი 2 |
| სტრიქონი 5 | სვეტი 1 | სვეტი 2 |

შედეგი 4. ცხრილის მეორედ გამოტანა ფუნქციის არასტანდარტული - ფუნქცია-ცვლადის გამოძახებით

სად შეიძლება იქნეს ჩაწერილი ფუნქციის გამოძახება? ამ კითხვაზე პასუხი სხვადასხვაა იმის და მიხედვით PHP-ის თუ, რომელ ვერსიაზეა საუბარი. ადრინდელ ვერსიებში, ფუნქციის გამოძახება უნდა მომხდარიყო კოდის იმ ნაწილში, რომელიც ფუნქციის გამოცხადება მოსდევს. მაგრამ PHP4-ის შემდეგ ეს შეზღუდვა მოსხნილია, და PHP-ს ბოლო ვერსიებში ფუნქციის გამოძახება შეიძლება წინ უსწროდეს ფუნქციის აღწერას. მაგრამ, აქ გვაქვს ერთი მოთხოვნა, ფუნქცია არ უნდა იქნეს გა­მოცხადებული ბლოკში, ანუ კოდის იმ ნაწილში, რომელიც შემოსაზღვრულია ფიგურული ფრჩხი­ლებით: ამ შემთხვევაში ფუნქციის გამოძახება უნდა იყოს ჩაწერილი ფუნქციის გამოცხადების შემ­დეგ, თანაც აღარ აქვს მნიშვნელობა, იმას თუ სად წერია ფუნქციის გამოძახება - ბლოკის შემდეგ, თუ ბლოკშივე. ამ პირობის დარღვევა იწვევს შეცდომის გამოტანას, სადაც მითითებული იქნება არგამოცხადებული ფუნქციისადმი მიმართვა.

ლისტინგი 5. ფუნქციის გამოცხადების ადგილი

1.<!DOCTYPE html>

2.<html>

3. <body>

4.<?php

 5.// ლოგიკური ცვლადის გამოცხადება

6. $flag = TRUE;

 7. if($flag)

 8. {

 9. function get\_sum()

10. {

11. $sum = 10 + 5;

 **12. return $sum**;

13. $sum1 = 10 + 10;

14. **return $sum1;**

 15. }

16. echo 'ჯამი='.get\_sum();

 17. }

 18. echo '<br>ჯამი1='.get\_sum();

 19.?>

 20. </body>

 21 .</html>

ლისტინგ 5-ის შესრულების შემდეგ მივიღებთ: ჯამი=15 ჯამი1=15

ლისტინგ 5-ში ფუნქციის ორივე გამოძახება იმუშავებს, რადგანაც ისინი ჩაწერილნი არიან ბლოკში მოთავსებული ფუნქციის შემდეგ. ფუნქციის გამოძახება რომ ყოფილიყო ჩაწერილი უფრო ადრე, ამ მაგალითში, მეცხრე სტრიქონის წინ, მივიღებდით შეცდომას -და დაიწერებოდა:

 **Fatal error**: Call to undefined function get\_sum() in **X:\home\localhost\www\laboratoriuli\_10\lab\_10\_4\_fun\_gamots\_adgili\index.php** on line **8**

ლისტინგ 5-ში გამოყენებულია PHP-ს კიდევ ერთი კონსტრუქცია - სამოსამსახურე სიტყვა **return ,** რომელიც აჩერებს ფუნქციის შესრულებას და მართვას გადასცემს იმ ოპერატორს, რომე­ლიც ჩაწერილია ფუნქციის გამოძახების შემდეგ. ამრიგად არ შესრულდება მე-13-ე და მე-14-ე სტრიქონებში ჩაწერილი ოპერატორები და მართვა გადაეცემა მე-17-ე სტრიქონში ბლოკის დამ­თავრების დამხურავ ფრჩხილს. თუ კი **return**-ის შემდეგ ჩაწერილია რაიმე ცვლადი, ობიექტი, თუ სხვა, მაშინგარდა იმისა, რომ შეჩერდება ფუნქციის შესრულება, გამოტანილი-დაბრუნებული იქნება ამ ცვლადის თუ სხვა ობიექტის მნიშვნელობა, რაც ნაჩვენებია ლისტინგ 5-ში, მე-16-ე სტრი­ქონში ჩაწერილ ოპერატორში - echo 'ჯამი='.get\_sum(); აქ დაიბეჭდება ის მნიშვნელობა რაც ფუნქცი­ის შესრულებისას მიიღო ცვლადმა **$sum**. და აქ გასათვალისწინებელია ცვლადების მოქ­მედების არეალი. ჩვემ რომ ჩაგვეწერა echo 'ჯამი='.$sum ოპერატორი, გამოიბეჭდებოდა მხოლოდ **ჯამი=** რადგანაც $sum ცვლადი ლოკალური ცვლადია და მისი მნიშვნელობა ფუნქციის გარეთ იკარგება, მაგრამ რადგანაც ლისტინგ 5-ში წერია გამოტანა ფუნქციის დაბრუნების მნიშვნელობა, მაშინ ეს დასაშვებია. ფუნქციის გარეთ echo 'ჯამი='.$sum; ოპერატორი კორექტულად შესრულ­დებოდა, თუ კი $sum ცვლადს ფუნქვიის შიგნით გამოვაცხადებდით, როგორც გლობალურს.

ფუნქციათა გამოყენება შეიძლება **რეკურსიულად**, ანუ ფუნქციას შეუძლია თავის თავის გამოძა­ხება. მართალია რეკურსიაზე უფრო მეტი დრო და მეხსიერების რესურსი იხარჯება, და ზოგ შემთხვევაში იგი ჯობია იტერაციებით შეიცვალოს, მაგრამ რეკურსიული გამოძახება გვაძლებს კოდი ჩაწერის უფრო მარტივ და ადვილად წასაკითხ კოდს. განვიხილოთ რეკუსიის გამოყენების მაგალითი არაუარყოფითი მთელი რიცხვის ფაქტორიალის გამოთვლისას. გავიხსენოთ, რომ არაუარყოფითი მთელი რიცხვის n ფაქტორიალი ჩაიწერება შემდეგნაირად n! და გამოითვლება: n!=1\*2\*3\*…..\*(n-1)\*n; შეთანხმებისამებრ 0!=1.

ლისტინგი 6 ფაქტორიალი რეკურსიით

<!DOCTYPE html>

<html>

<body>

<?php

 function factorial($n=10)

{

if ($n==0) return 1;

else

**return $fact = $n \* factorial($n-1);**

}

echo '3!='.factorial(3).'<br>'; // 3-ის ფაქტორიალი

echo '50!='.factorial(50).'<br>'; // 50-ის ფაქტორიალი

echo '10!='.factorial().'<br>'; // 10-ის ფაქტორიალი

 ?>

 </body>

 </html>

 ლისტინგი 6-ის შესრულების შედეგად მივიღებთ:

3!=6
50!=3.0414093201713E+64
10!=3628800

ფუნქციის რეკურსიული გამოძახება ხდება ლისტინგი 6-ის მე-9-ე სტრიქონში. აღსანიშნავია, რომ მე-11-ე და მე-12-ე სტრიქონებში ფუნქციის გამოძახებისას ჩვენ ფუნქციას მივაწოდეთ პარამეტრე­ბის მნიშვნელობები და მათთვის ფუნქციამ დაგვითვალა ფაქტორიალები, მაგრამ მე-13-ე სტრი­ქონში პარამეტრის გადაცემა არ ხდება და ამ შემთხვევაში ჩუმათობის პრინციპით, იგი დაამუშა­ვებს პარამეტრის იმ მნიშვნელობას, რომელიც მას მითითებული აქვს გამოცხადებისას. ჩვენს შემ­თხვევაში - 10. თუ კი ფუნქციის გამოცხადებისას ზოგიერთ პარამეტრს მრგვალ ფრჩხილებში მინი­ჭებული აქვს მნიშვნელობა, მათ ეწოდება **არააუცილებელი პარა**მ**ეტრი** - ან **პარამეტრები ჩუმათო­ბით**, იმ თვალსაზრისით, რომ ამ ცვლადს, თუ გამოძახებისას მნიშვნელობა მითითებული არ აქვს, ფუნქცია დაამუშავებს არააუცილებელი პარამეტრის ფუნქციაში მითითებულ მნიშვნელობას. თუ ფუნქციაში არის არააუცილებელი და აუცილებელი პარამეტრები, ჯერ უნდა ჩაიწეროს აუცილებელი, ხოლო შემდეგ არააუცილებელი პარამეტრები.

პარამეტრების გადაცემს აქამდე ჩვენ ვანხორციელებდით მნიშვნელობით, ანუ იმ ცვლადის მნიშ­ვნელობა, რომელიც მითითებულია ფუნქციის გამოძახებაში, გადაეცემა ფუნქციის აღწერაში მი­თითებულ ცვლადებს. არსებობს ინფორმაციის გადაცემის სხვა გზაც, ანუ ფუნქციის კოდს მნიშ­ვნელობას კი არ გადავცემთ, არამედ მივუთითებთ მხოლოდ ცვლადის სახელს. ამისთვის ფუნ­ქციის აღწერაში პარამეტრის სახელის წინ ჩაიწერება სიმბოლო ”&” - ამპერსანდი. როცა შესრულ­დება ფუნქციის კოდი, ამპერსანდის საშუალებით მითითებული პარამეტრი, რომელიც ფაქტი­ურად გლობალური ცვლადია, შეიძლება შეიცვალოს (თუკი ამას კოდი ითვალისწინებს). როგორც გვახსოვს, ფუნქციის შიგნით გვაქვს ლოკალური ცვლადები და ისინი ფუნქციის დასრულების შემ­დეგ, ფუნქციის გარეთ აღარ მოქმედებენ. პარამეტრის მითითებით გადაცემისას კი ფაქტიურად ხდება გლობალური ცვლადის მნიშვნელობის შეცვლა ფუნქციის მიერ, ისე, რომ ცვლადების გლო­ბალურად გამოცხადებაც არ დაგვჭირდა.

მოვიყვანოთ პარამეტრის მითითებით გადაცემის მაგალითი, ვთქვათ ნებისმიერ რეგისტრში აიკ­რიფა **Francis Scott Key** **Fitzgerald** -ის სახელი და საჭიროა , რომ ის გავასწოროთ - ყველა სახელის პირველი ასო დაიწეროს მაღალ რეგისტრში, დანარჩენები კი დაბალში. ვთქვათ თავიდან ჩაიწერა: FRAncis scott KEY FITZGERALD, შევქმნათ კოდი, რომელიც ამ ამოცანას გადაწყვეტს:

ლისტინგი 7 პარამეტრის მითითებით გადაცემა

<!DOCTYPE html>

<html>

<body>

<?php

$f1 = "FRAncis";

$f2 = "scott";

$f3 = "KEY";

$f4 = "FITZGERALD";

echo $f1." ".$f2." ".$f3." ".$f4."<br>";

fix\_names($f1,$f2,$f3,$f4);

echo $f1." ".$f2." ".$f3." ".$f4;

function fix\_names (&$n1,&$n2,&$n3,&$n4)

{

$n1 = ucfirst(strtolower($n1));

$n2 = ucfirst(strtolower($n2));

$n3 = ucfirst(strtolower($n3));

$n4 = ucfirst(strtolower($n4));

}

?>

 </body>

 </html>

ლისტინგი 7-ის შესრულების შემდეგ მივიღებთ:

FRAncis scott KEY FITZGERALD
Francis Scott Key Fitzgerald

ლისტინგში გამოყენებულია PHP-ის ჩაშენებული ფუნქციები: **strtolower($str)** - რომელიც $str ტექ­სტურ ცვლადს ჩაწერს დაბალ რეგისტრში და **ucfirst($str**) - რომელიც $str ტექსტური ცვლადის პირ­ველ პოზიციაში ჩაწერს სიმბოლოს მაღალ რეგისტრში. როგორც ვხედავთ გლობალური ცვლადები ფუნქციის გამოძახების შემდეგ შეიცვალა - რადგანაც პარამეტრები მითითებით გადაეცა.

 PHP-ში შესაძლებელია ერთი ფუნქციის შიგნით გამოცხადებულ იქნას სხვა ფუნქცია, რომელსაც **ჩადგმულ ფუნქციას** ვუწოდოთ. ჩვეულებრივი ფუნქციისგან განსხვავებით ჩადგმულ ფუნქციას მხოლოდ მაშინ თუ გამოიყენებ-გამოიძახებ, როცა მიმართავ იმ ფუნქციას, რომელშიც ის ჩადგმუ­ლია.

 PHP-ში 5.3 ვერსიიდან შესაძლებელია ანონიმური ფუნქციების გამოყენება. სახელწოდება წარმო­იშვა იმის გამო, რომ ამ ფუნქციებს სახელი არ აქვს, სხვანაირად მათ ლამბდა-ფუნქციებსაც უწოდე­ბენ. ისინი განისაზღვრებიან და მიენიჭებათ ცვლადებს როგორც ჩვეულებრივი მნიშვნელობა, ამი­ტომაც მათი გამოყენების შემდეგ არ უნდა დაგვავიწყდეს წერტილმძიმის დასმა. ცვლადი, რომელ­საც მიენიჭა ანონიმური ფუნქცია, ინარჩუნებს თავის სახელს, მაგრამ მუშაობს როგორც ფუნქცია. ანონიმური ფუნქციის გამოყენება შეიძლება მას მერე, რაც ის განსაზღვრული იქნება. ანონიმური ფუნციის შექმნისა და გამოყენების მაგალითი;

ლისტინგი 8 ანონიმური ფუნქცია.

<!DOCTYPE html>

<html>

<body>

<?php

 $anonim\_func = function($str) {

 echo "საბურავი, ზეთი, აალების სანთლები - $str"."<br>";

 }; // წერტილმძიმე

 $anonim\_func(' ავტონაწილებია!');

 $anonim\_func(' საქონელია!');

?>

 </body>

 </html>

ლისტინგი 8-ის შესრულების შემდეგ ვღებულობთ:

**საბურავი, ზეთი, აალების სანთლები - ავტონაწილებია!
საბურავი, ზეთი, აალების სანთლები - საქონელია!**

თუ კი ფუნქციის გამოძახებისას ხდება იმაზე ნაკლები არგუმენტების გადაცემა, ვიდრე ეს ფუნქციის გამოცხადებისაა მითითებული, გამოდის გაფრთხილება, რომ არგუმენტების რაოდენობა საკმარისი არაა.

**func\_num\_args** --  ეს ფუნქცია რომელიც მიმდინარე მომხმარებლის ფუნქციაში გამოიყენება, აბრუნებს მომხმარებლის ფუნქციისთვის გადაცემული არგუმენტების რაოდენობას. სინტაქსისი

int **func\_num\_args** ( )

მომხმარებლის ფუნქციის გარეთ func\_num\_args -- ფუნქცის გამოიყენება იძლევა გაფრთხილებას.

**func\_get\_arg** – აბრუნებს ელემეტს არგუმენტების სიიდან.

სინტაქსისი mixed **func\_get\_arg** ( int arg\_num )

აბრუნებს რიგით *arg\_num* არგუმენტს მომხმარებლის ფუნქციისთვის გადაცემული არგუმენტების სიიდა. ნუმერაცია ნულიდან იწყება. მომხმარებლის ფუნქციის გარეთ. **func\_get\_arg()** ფუნქცის გამოიყენება იძლევა გაფრთხილებას.თუ კი *arg\_num* მეტია გადაცემული არგუმენტების რაოდენობაზე, მაშინ მოხდება გაფრთხილების გენერირება და **func\_get\_arg()** აბრუნებს **FALSE -ს**.

**func\_get\_args** --  აბრუნებს არგუმენტების ელემეტს მასივს. სინტაქსისი

array **func\_get\_args** ( void )

აბრუნებს მასივს, რომლის ყოველი ელემენტი არის შესაბამისი არგუმენტი გადაცემული მომხმარებლის ფუნქციისთვის. **func\_get\_args()**ფუნქცის გამოიყენება იძლევა გაფრთხილებას.

**მასივები**

რიგ შემთხვევაში სასარგებლოა სხვადასხვა იდენტიფიკატორების და სხვადასხვა ტიპის ცვლადები გაერთიანებული იქნენ ერთი სახელის ქვეშ - ასეთ ცვლადებს მასივები ეწოდება და ისინი განსხვა­ვდებიან სკალარული ცვლადებისgან (integer, float, string). როცა ვსაუბრობთ იმაზე, რომ მასივში შეიძლება გაერთიანებული იყვნენ სხვადასხვა ტიპის ცვლადები, არ გამოირიცხება, რომ ეს ცვლა­დები თავად იყვნენ მასივები. ესე იგი, საქმე გვაქვს მრავალგანზონილებიან მასივთან. მაგრამ ჩვენ მივყვეთ თანამიმდევრობით და დავიწყოთ ერთგანზომილებიანი მასივიდან. უპირველეს ყოვლისა გავეცნოთ მასივის სტრუქტურას.

მასივი წარმოადგენს მეხსიერებაში გამოყოფილ სახელობით არეს, რომელშიც მოთავსებულია მნიშვნელობების ერთობლიობა. ასე მაგალითად კომპანია ”ავტონაწილები ბუბასგან” შემოთავა­ზებული საქონლის ერთობლიობა შეიძლება მასივის სახით წარმოვადგინოთ, ანუ მოვათავსოთ ისინი მეხსიერების არეში და ამ არეს სახელი მივანიჭოდ, მაგალითად $products:

|  |  |  |
| --- | --- | --- |
| საბურავები | ზეთი | აალების სანთლები |

ნახატი 1. კომპანია ”ავტონაწილები ბუბასგან” შემოთავაზებული საქონლელი წარმოვადგენილი მასივის სახით

$products მასივში მოთავსებულია სამი ტექსტური ცვლადის მნიშვნელობები: ”საბურავები”, ”ზე­თი”, ”აალების სანთლები”. მასივში მოთავსებულ მნიშვნელობებს **ელემენტები** ეწოდება. ყოველ ელემენტს გააჩნია მასთან დაკავშირებული **ინდექსი** ან **გასაღები,** რომლებიც გამოიყენებიან მასი­ვის ამ მნიშვნელობის წვდომისათვის. ტერმინი ”ინდექსი” გამოიყენება იმ შემთხვევაში, როცა მისი მნიშვნელობა რიცხვია - არაუარყოფითი მთელი, ასეთ მასივს **ინდექსირებული მასივი** ეწოდება. ტერმინი ”გასაღები” გამოიყენება იმ შემთხვევაში, როცა მისი მნიშვნელობა სტრიქონულია - ტექ­სტური, ასეთ მასივს **ასოცირებული მასივი** ეწოდება.

**ინდექსირებული მასივის ინიციალიზაცია**

მასივის ინიციალიზაციის ერთ-ერთი მეთოდია array() კონსტრუქციის გამოყენება. მრგვალ ფრჩხი­ლებში ჩაიწერება მასივის ელემენტების მნიშვნელობა, რომელიც მიენიჭება იმ მასივს, რომლის სა­ხელი იქნება მინიჭების ოპერატორის - ”=” მარცხნივ, ხოლო array(), იქნება ამ ოპერატორისგან მარ­ჯვნივ ჩაწერილი. განვიხლო] თუ როგორ სახეს მიიღებს $products მასივის ინიციალიზაცია:

$products = array('საბურავები', 'ზეთი', 'აალების სანთლები' );

ამ შემთხვევაში ჩვენ გვაქვს სამ ელემენტიანი ინდექსირებული მასივი და მისი ელემენტის წვდო­მისთვის უნდა გამოვიყენოთ მასივის სახელი, კვადრატული ფრჩხილები, რომელშიც ჩაწერილი იქნება ინდექსი - ანუ მთელი არაუარყოფითი რიცხვი. რადგანაც გვაქვს სამი ელემენტი, მაშინ ინ­დექსენის სიმრავლეა {0,1,2} - ანუ ინექსირება PHP -ში, ისევე, როგორც Javascript-ში იწყება ნული­დან. ასე მაგალუთად $products მასივის ელემენტის - ”აალების სანთლები”-ს წვდომისათვის ჩავ­წერთ:

$products[2]

აღსანიშნავია, რომ ახალი მასივის ინიციალიზაცია შესაძლებელია, თუ კი მასივის ყოველი ელემენ­ტის განსაზღვრას ცალ-ცალკე მოვახდენთ, ასე მაგალითად:

ლისტინგი 9 . მასივების ცხადი სახით ინიციალიზაცია

$products[0]= ”საბურავები”;

$products[1]= ”ზეთი”;

$products[2]= ”აალების სანთლები”;

ლისტინგი 9-ის პირველი ოპერატორის შესრულებისას შეიქმნება ერთელემენტიანი მასივი მნიშ­ვნელობით - ”საბურავები”; მეორე ოპერატორის შესრულებისას, ერთელემენტიანი მასივი ”გაიზ­რდება” - გაფართოვდება ორგელემენტიანამდე, რადგანაც მას ემატება მეორე ელემენტი - ”ზეთი”; ანალოგირად მოხდემა მასივის გაზრდა მესამე ოპერაატორის შესრულებისას სამელემენტიან მასი­ვამდე. ამრიგად PHP -ში შესაძლებელია **მასივის დინამიური გაფართოება**.

**for** ციკლის გამოყენება ინდექსირებული მასივისთვის ეფექტურია, რადგანაც შესაძლებელია მასივის ელემენტის ინდექსი მივუთოთ ციკლის მთვლელის საშუალებით. განვიხილოთ მასივის ელემენტების გამოტანის შესაძლებლობები ციკლით და მის გარეშე:

ლისტინგი 10 მასივის ელემენტების გამოტანა

<!DOCTYPE html>

<html>

<body>

<?php

$products = array('საბურავები', 'ზეთი', 'აალების სანთლები' );

echo 'მასივის ელემენტების ცალ-ცალკე გამოტანა<br>';

echo "$products[0] $products[1] $products[2]<br>";

echo "<br>მასივის ელემენტების ციკლით გამოტანა<br>";

for($i=0;$i<3;$i++){

echo $products[$i]." ";

}

?>

</body>

</html>

ლისტინგი 10 -ის გაშვების შედეგად მივიღებთ

მასივის ელემენტების ცალ-ცალკე გამოტანა
საბურავები ზეთი აალების სანთლები

მასივის ელემენტების ციკლით გამოტანა
საბურავები ზეთი აალების სანთლები

შევნიშნოთ, რომ, ლისტინგი 10-ში, for -ციკლის ოპერატორში გვიწერია ინდექსის ცვლილების ზე­და ზღვარი, რომელიც ჩვემს მაგალითში 3-ის ტოლია, ანუ ელემენტების რაოდენობის მაჩვენებე­ლი. მაგრამ თუ კი დინამიური მასივების შემთხვევაში, რაღაც მომენტში ჩვენ არ ვიცით მასივის ელემენტების რაოდენობა, მაგრამ საჭიროა ე.წ. ციკლის დატრიალება მასივის ელემენტების რა­ოდენობის ტოლჯერ. ამისთვის გამოიყენება მასივის ელემენტის დამთვლელ ფუნქცია **count()** და ჩვენს მაგალითში 3-ის ნაცვალად ჩავსვათ ეს ფუნქცია, ხოლო ფრჩხილებში მივუთითო, იმ მასივის დასახელება, რომლის ელემენტების რაოდენობასაც ვითვლით: for($i=0;$i< count($products) ;$i++).

მასივისთვის, როგორც გვახსოვს ციკლის სპეციალური ოპერატორია foreach, რომელიც ჩვენი მაგალითში ასე ჩაიწერება:

ლისტინგი 11. მასივის ელემენტების გამოტანა foreach ციკლით

echo "<br><br>მასივის ელემენტების foreach ციკლით გამოტანა<br>";

foreach($products as $mimdinare){

 echo $mimdinare." ";

}

ლისტინგი 11-ის ლისტინგი 10-ისთვის დამატებით და შემდეგ გაშვებით მივიღებთ:

მასივის ელემენტების ცალ-ცალკე გამოტანა
საბურავები ზეთი აალების სანთლები

მასივის ელემენტების for ციკლით გამოტანა
საბურავები ზეთი აალების სანთლები

მასივის ელემენტების foreach ციკლით გამოტანა
საბურავები ზეთი აალების სანთლები

foreach-ი მუშაობს შემდეგნაირად: $products მასივის ყოველი ელემენტი, ყოველი იტერაციისას მიენიჭება $mimdinare ცვლადს, შემდეგ კი ციკლის ტანში ხდება მისი მნიშვნელობის გამოტანა. ზოგჯერ მასივს შეიძლება ჰქონდეს ინდექსების მნიშვნელობები, რომლებიც არავითარ კანონზო­მიერებას არ ექვემდებარებიან (მაგალითად არ წარმოადგენენ, არც არითმეტიკულ, არც გეომეტრი­ულ პროგრესისა), ამიტომ ასეთი მასივების ელემენტების შემოვლა for ციკლით პრობლემურია; ამიტომ ხშირად იყენებენ foreach-ციკლის ოპერატორს, რომელიც დაუვლის მასივის ნებისმიერ ელემენტს, რა რთული და არაკანონზომიერი ინდექსაციაც არ უნდა ჰქონდეს მას.

**ასოცირებული მასივის ინიციალიზაცია**

PHP-ში არის საშუალება მასივების ელემენტები დავუკავშიროთ (მოვახდინოთ ასოციაცია) ნები­სმიერ გასაღებს. ასეთი ასოცირებული მასივის ინიციალიზაცია შეიძლება მოვახდინოთ ინდექ­სირებული მასივის მსგავსად, მაგრამ არსებობს მცირედი განსხვავებები. ვთქვათ ჩვენ ვქმნით ავტონაწილების მაღაზიის საქონელის ფასების მასივს. ამ მასივის სახელი იყოს $fasi, მის გასაღებე­ბად ავიღოთ საქონლის დასახელება, ხოლო ელემენტებად ერთეული საქონლის ფასის მნიშვნელობა :

$faci=array ('საბურავები'=>100, 'ზეთი'=>10, 'აალების სანთლები'=>6);

ლისტინგი 12. ასოცირებული მასივის ინიციალიზაცია

<!DOCTYPE html>

<html>

<body>

<?php

$faci=array ('საბურავები'=>100, 'ზეთი'=>10, 'აალების სანთლები'=>6);

var\_dump($faci);

?>

</body>

</html>

ლისტინგი 12-ის გაშვებისას ხდება ასოცირებული მასივის ინიციალიზაცია, ხილო მისი არსებობი­სა და ელემენტებისა და გასაღების ჩვენებისთვის გამოვიყენეთ ფუნქცია var\_dump(i);

array(3) { ["საბურავები"]=> int(100) ["ზეთი"]=> int(10) ["აალების სანთლები"]=> int(6) }

ასოცირებული მასივის ყოველი ელემენტის წვდომისათვის გამოიყენება მასივის იდენტიფიკატო­რი და მის შემდეგ გამხსნელი და დამხურავი კვადრატული ფრჩხილები, რომელშიც ჩასმულია გა­საღების მნიშვნელობა - ანუ ჩვენს მაგალითში მასივი $faci-ის მეორე ელემენტის მნიშვნელობის წვდომისათვის ჩავწერთ $faci['ზეთი'].

ასოცირებული მასივის ინიციალიზაცია შესაძლებელია ასევე სხვა გზითაც, ცალკეული ელემენ­ტების მინიჭებით. მაგალითად:

 $faci=array ('საბურავები'=>100);

$faci['ზეთი']=10;

$faci['აალების სანთლები']=6;

აქ, ჯერ იქმნება მასივი ერთი ელემენტით, ხოლო შემდეგ ამ მასივს ემატება ახალი ელემენტები. რადგან პირველ ოპერატორში არის მასივზე მითითება - array - ს სახით, ამ შემთხვევაშიც, ისევე, როგორც ზემოთ მოყვანილ კოდში, ხდება მასივის ცხადად გამოცხადება. მასივის არაცხადი გამოც­ხადებისთვის აღარ ვუთითებთ - array -ს, არამედ ცალ-ცალკე მასივის ელემენტებს ვანიჭებთ მნიშ­ვნელობებს და ასეთნაირად წარმოებს მასივის ინიციალიზაცია:

$faci['საბურავები']=100;

$faci['ზეთი']=10;

$faci['აალების სანთლები']=6;

ასოცირებული მასივისთვის, ციკლის სპეციალური ოპერატორი foreach-ისთვის ხდება მცირედი შეცვლა მისი გამოყენებისას, რათა გამოვიტანოთ, არა მარტო მასივის ელემენტის მნიშვნელობა, არამედ, ასევე შესაბამისი გასაღების მნიშვნელობაც:

ლისტინგი 13 . ასოცირებული მასივისთვის ციკლის ოპერატორი foreach-ის მოდიფიცირება

<!DOCTYPE html>

<html>

<body>

<?php

$faci=array ('საბურავები'=>100, 'ზეთი'=>10, 'აალების სანთლები'=>6);

foreach ($faci as $gasaRebi=>$mniSvneloba) {

echo $gasaRebi." ".$mniSvneloba."<br>";

}

?>

</body>

</html>

ლისტინგი 14-ის გაშვებისას ხდება ასოცირებული მასივის ინიციალიზაცია და შემდეგ ციკლის foreach ოპერატორით გასაღებისა და მნიშვნელობის გამოტანა (ლაბორატორიული 11):

საბურავები 100
ზეთი 10
აალების სანთლები 6

ჩვენ შეგვეძლო არ შეგვეცვალა არაფერი და foreach ოპერატორი ასოცირებული მასივისთვის დაგ­ვეწერა ზუსტად ისევე, როგორც ინდექსირებული მასივისთვის, მაგრამ ამ შემთხვევაში ვკარგავთ გასაღებების მნიშვნელობების გამოტანის საშუალებას.

ლისტინგი 15 ასოცირებული მასივისთვის ციკლის ოპერატორი foreach-ის მოდიფიცირების გარეშე

<!DOCTYPE html>

<html>

<body>

<?php

$faci=array ('საბურავები'=>100, 'ზეთი'=>10, 'აალების სანთლები'=>6);

foreach ($faci as $mimdinare) {

echo $mimdinare." ";

}

?>

</body>

</html>

ლისტინგი 15-ის გაშვებისას ხდება ასოცირებული მასივის ინიციალიზაცია და შემდეგ ციკლის foreach ოპერატორით მხოლოდ მასივის ელემენტის მნიშვნელობის გამოტანა (ლაბორატორიული 12):

100 10 6

**მასივის ელემენტების მიმთითებლის ოპერატორები**

როდესაც ჩვენ ვახდენთ მასივის ელემენტების შემოვლას, ვთქვათ ციკლის საშუალებით, ან უშუ­ალოდ - მასივის ელემენტის წვდომით, ჩაირთვება მიმთითებლის მექანიზმი. ყოველ მასივს გააჩ­ნია შინაგანი მიმთითებელი (pointer), იგივე კურსორი, რომელშიც აღინიშნება - მასივის თუ რო­მელი ელემენტი განიხილება მიმდინარე მომენტში - ანუ, იმყოფება ფოკუსში.

მიმთითებლის მიმდინარე მნიშვნელობის გასაგებად გამოიყენება ფუნქცია **current()**, სადაც მრგვალ ფრჩხილებში ჩაიწერება მასივის სახელი. ფუნქცია current() აბრუნებს მიმთითებლის მიმდინარე მნიშვნელობას.

მიმთითებლის შემდეგ ელემენტზე გადაყვანას ახდენს ფუნქცია **next()** და აბრუნებს მის მნიშვნელობას ;

მიმთითებლის წინა ელემენტზე გადაყვანას ახდენს ფუნქცია **prev()** და აბრუნებს მის მნიშვნელობას;

მიმთითებლის ბოლო ელემენტზე გადაყვანას ახდენს ფუნქცია **end()** და აბრუნებს მის მნიშვნელობას;

მიმთითებლის პირველ ელემენტზე გადაყვანას ახდენს ფუნქცია **reset()** და აბრუნებს მის მნიშვნელობას;

მასივის მიმდინარე ელემენტის გასაღების დაბრუნებას ახდენს ფუნქცია **key()**.

 ვნახოთ ლისტინგ 15-ში, ფუნქციების **current(),** **next(), prev(), end(), reset() -**ისდამატებითრა შედეგს მივიღებთ (ლაბორატორიული 13):

ლისტინგი 16. ფუნქციების **current(),** **next(), prev(), end(), reset(), key()** გამოყენება

<!DOCTYPE html>

<html>

<body>

<?php

echo '1.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა -'.current($faci).'; გასაღების -'.key($faci).'<br>';

echo 'ეს შედეგი დადგა იმის გამო, რომ მასივი ჯერ განსაზღვრული არაა'.'<br><br>';

$faci=array ('საბურავები'=>100, 'ზეთი'=>10, 'აალების სანთლები'=>6);

$mode = current($faci);// $mode = '100'

echo '2.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა -'.current($faci).'; გასაღების -'.key($faci).'<br>';

echo 'ეს შედეგი დადგა იმის გამო, რომ მასივის განსაზღვრისას მიმთითებელი პირველ ელემენტზეა'.'<br><br>';

$mode = next($faci); // $mode = '10';

echo '3.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - '.$mode.'; გასაღების -'.key($faci).'<br>';

echo '4.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - '.current($faci).'; გასაღების -'.key($faci).'<br>';

echo 'ეს შედეგი დადგა იმის გამო, რომ მასივის მიმთითებელი next() ფუნქციით გადავიდა მასივის შემდეგ ელემენტზე'.'<br><br>';

$mode = prev($faci); // $mode = '100';

echo '5.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - '.$mode.'; გასაღების -'.key($faci).'<br>';

echo '6.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - '.current($faci).'; გასაღების -'.key($faci).'<br>';

echo 'ეს შედეგი დადგა იმის გამო, რომ მასივის მიმთითებელი prev() ფუნქციით გადავიდა მასივის წინა ელემენტზე'.'<br><br>';

$mode = end($faci); // $mode = '6';

echo '7.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - '.$mode.'; გასაღების -'.key($faci).'<br>';

echo '8.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - '.current($faci).'; გასაღების -'.key($faci).'<br>';

echo 'ეს შედეგი დადგა იმის გამო, რომ მასივის მიმთითებელი end() ფუნქციით გადავიდა მასივის ბოლო ელემენტზე'.'<br><br>';

$mode = reset($faci); // $mode = '100';

echo '9.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - '.$mode.'; გასაღების -'.key($faci).'<br>';

echo '10.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - '.current($faci).'; გასაღების -'.key($faci).'<br>';

echo 'ეს შედეგი დადგა იმის გამო, რომ მასივის მიმთითებელი reset() ფუნქციით გადავიდა მასივის პირველ ელემენტზე'.'<br><br>';

?>

</body>

</html>

ლისტინგ 16-ის გაშვებით მივიღებთ:

**Warning**: current() expects parameter 1 to be array, null given in **X:\home\localhost\www\laboratoriuli\_10\lab\_10\_14\_current\index.php** on line **5**

**Warning**: key() expects parameter 1 to be array, null given in **X:\home\localhost\www\laboratoriuli\_10\lab\_10\_14\_current\index.php** on line **5**
1.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა -; გასაღების -
ეს შედეგი დადგა იმის გამო, რომ მასივი ჯერ განსაზღვრული არაა

2.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა -100; გასაღების -საბურავები
ეს შედეგი დადგა იმის გამო, რომ მასივის განსაზღვრისას მიმთითებელი პირველ ელემენტზეა

3.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - 10; გასაღების -ზეთი
4.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - 10; გასაღების -ზეთი
ეს შედეგი დადგა იმის გამო, რომ მასივის მიმთითებელი next() ფუნქციით გადავიდა მასივის შემდეგ ელემენტზე

5.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - 100; გასაღების -საბურავები
6.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - 100; გასაღების -საბურავები
ეს შედეგი დადგა იმის გამო, რომ მასივის მიმთითებელი prev() ფუნქციით გადავიდა მასივის წინა ელემენტზე

7.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - 6; გასაღების -აალების სანთლები
8.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - 6; გასაღების -აალების სანთლები
ეს შედეგი დადგა იმის გამო, რომ მასივის მიმთითებელი end() ფუნქციით გადავიდა მასივის ბოლო ელემენტზე

9.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - 100; გასაღების -საბურავები
10.მასივის მიმთითებლის მიმდინარე მნიშვნელობაა - 100; გასაღების -საბურავები
ეს შედეგი დადგა იმის გამო, რომ მასივის მიმთითებელი reset() ფუნქციით გადავიდა მასივის პირველ ელემენტზე

ლისტინგ 16-ის გაშვების შედეგი

 **each() და list() ფუნქციების გამოყენება**

მასივებისთვის each() და list() ფუნქციების გამოყენებას ზოგჯერ თვლიან მოძველებულად, რადგა­ნაც მათ ნაცვლად ძირითადად foreach ოპერატორი გამოიყენება. მაგრამ PHP-ს 5.5.0 ვერსიიდან არის **E\_DEPRECATED** и **E\_USER\_DEPRECATED** - ორი დონის შეცდომების მაჩვენებელი. **E\_DEPRECATED** -ის შეცდომების დონე გამოიყენება იმისთვის, რომ აღინიშნოს ფუნქციისა და შესაძლებლობების მოძველება. **E\_USER\_DEPRECATED** -ის შეცდომების დონე გამოიყენება იმისთვის, რომ აღინიშნოს მოძველებული ფუნქცია სამომხმარებლო კოდში, ისევე, როგორც ეს ხდება შემდეგ დონეებზე - **E\_USER\_ERROR** და **E\_USER\_WARNING**. ამიტომაც თუ კი ამ ტიპის შეტყობინენებები არ გამოჩნდება, ამ ფუნქციების გამოყენება თამამად შეიძლება.

ფუნქცია each()-ს აქვს სახე array **each** ( array &$array ) , რაც ნიშნავს, იმას, რომ ფუნქციის პატამეტ­რია მასივი, ხოლო შედეგად ვღებულობთ ისევ მასივს. each() ფუნქცია პარამეტრში მოთავსებული მასივის მიმდინარე გასაღებ/მნიშვნელობას გადასცემს ახალ, მის მიერ ფორმირებად მასივში, რომელსაც ოთხი ელემენტი აქვს. ამ ახალ მასივს აქვს შემდეგი გასაღებები [0],[1].[key],[value]. ავღნიშნოთ ახალი მასივი შემდეგნაირად $akhali\_masivi, მაშინ $akhali\_masivi[0] და $akhali\_masivi[key ] ელემენტები ერთმანეთის ტოლია და იღებენ პარამეტრში მოთავსებული მასივის მიმდინარე ელემენტის გასაღებს, ხოლო $akhali\_masivi[1] და $akhali\_masivi[value ] ელე­მენტები ასევე ერთმანეთის ტოლია და იღებენ პარამეტრში მოთავსებული მასივის მიმდინარე ელემენტის მნიშვნელობას. ამის შემდეგ პარამეტრში მოთავსებული მასივის მიმთითებელი შემდეგ ელემენტზე გადადის. თუ კი მიმთითებელი გადის ინდექსის არედან each() აბრუნებს მნიშვნელობას false . მაგალითად გამოვიყენოთ ფუნქცია each() $faci მასივისთვის

ლისტინგი 17. ფუნქცია each()

<!DOCTYPE html>

<html>

<body>

<?php

$faci=array ('საბურავები'=>100, 'ზეთი'=>10, 'აალების სანთლები'=>6);

$akhali\_masivi=each($faci);

print\_r($akhali\_masivi);

echo'<br>';

$akhali\_masivi=each($faci);

print\_r($akhali\_masivi);

echo'<br>';

$akhali\_masivi=each($faci);

print\_r($akhali\_masivi);

echo'<br>';

$akhali\_masivi=each($faci);

print\_r($akhali\_masivi);

echo'<br>';

?>

</body>

</html>

ლისტინგი 17-ის გაშვებისა ვღებულობთ

Array ( [1] => 100 [value] => 100 [0] => საბურავები [key] => საბურავები )
Array ( [1] => 10 [value] => 10 [0] => ზეთი [key] => ზეთი )
Array ( [1] => 6 [value] => 6 [0] => აალების სანთლები [key] => აალების სანთლები )

ლისტინგ 16-ის ბოლო $akhali\_masivi=each($faci); ოპერატორი არ მუშაობს, რადგანაც მის წინ, მე-12-ე სრტიქონში მომუშავე each() ოპერატორმა მადაადგილა რა მიმთითებელი, იგი გადაიყვანა ინრექსების არის გარეთ.

ლისტინგ 17-ში მოცემული კოდი შეიძლება ციკლით შევცვალოთ, მაგალითად:

ლისტინგი 18 each() ფუნქცია while ციკლში

<!DOCTYPE html>

<html>

<body>

<?php

$faci=array ('საბურავები'=>100, 'ზეთი'=>10, 'აალების სანთლები'=>6);

echo 'each() ფუნქცია while ციკლში<br>';

while ($akhali\_masivi=each($faci)){

print\_r($akhali\_masivi);

echo'<br>';

}

?>

</body>

</html>

ლისტინგი 18-ის გაშვების შემდეგ მივიღებთ:

each() ფუნქცია while ციკლში
Array ( [1] => 100 [value] => 100 [0] => საბურავები [key] => საბურავები )
Array ( [1] => 10 [value] => 10 [0] => ზეთი [key] => ზეთი )
Array ( [1] => 6 [value] => 6 [0] => აალების სანთლები [key] => აალების სანთლები )

list() ფუნქცია array() -ის საპირისპიროა, იმ გაგებით, რომ, თუ კი array() -ის საშუალებით იქმნება მასივი, ფუნქცია list()-ით შესაძლებელია მასივიდან ამოვიღოთ მონაცემენი და მივანიჭოთ ცვლადების იმ სიას, რომელიც ამ ფუნქციის პარამერების სახითაა მოცემული. სინტაქსისია:

list(var1[,var2...])

სადაც var1 -აუცილებელი პარამეტრია, var2 - დამატებითი.

ამიტომაც ჩვენს მაგალითში $faci მასივიდან შეიძლება each() ფუნქციით მიმდინარე ელემენტისგან შევქმნათ ახალი მასივი, რომლის ელემენტებიც უკვე list() ფუნქციის საშუალებით მივანიჭოთ ორ ცვლადს და მერე გამოვიტანოთ ისინი ბეჭდვაზე, თანაც ეს ყველაფერი მოვათავსოთ ციკლში. მაგალითად,

ლისტინგი 19 list() ფუნქცია

<!DOCTYPE html>

<html>

<body>

<?php

$faci=array ('საბურავები'=>100, 'ზეთი'=>10, 'აალების სანთლები'=>6);

echo 'list() და each() ფუნქციები while ციკლში<br>';

while (list($product, $price)=each($faci)){

echo "$product - $price<br>";

}

?>

</body>

</html>

ლისტინგი 19 -ის გაშვებით მივიღებთ -

list() და each() ფუნქციები while ციკლში
საბურავები - 100
ზეთი - 10
აალების სანთლები - 6