

თეიმურაზ სტურუა  
ბესიკ ტაბატაძე  
თეა თოდუა

# ვებდაპროგრამების გაქნოლოგია

php



საქართველოს ტექნიკური უნივერსიტეტი

თეიმურაზ სტურუა  
ბესიკ ტაბატაძე  
თეა თოდუა

# ვებდაპროგრამების ტექნოლოგია - PHP



დამტკიცებულია სახელმძღვანელოდ  
საქართველოს ტექნიკური უნივერსიტეტის  
სარედაქციო-საგამომცემლო საბჭოს  
მიერ. 27.12.2016, ოქმი №3

თბილისი  
2017

## უკ 681.3

სახელმძღვანელოში განხილულია PHP ენის შესაძლებლობები, PHP5 ვერსიაში შემოღებული სიახლეების გათვალისწინებით. გადმოცემულია და დეტალურად ახსნილი ყველა ის საკითხი და ამოცანა, რომელსაც ვებპროგრამისტი შეიძლება გადააწყდეს ვებსაიტის აგების პროცესში.

განკუთვნილია ინფორმატიკისა და მართვის სისტემების ფაკულტეტის პროფესიული სწავლების, ბაკალავრიატის და მაგისტრატურის სტუდენტებისთვის, გამოადგება ყველას ვინც დაინტერესებულია PHP ენის გამოყენებით ვებგვერდების შექმნით.

რეცენზენტები: საქართველოს ტექნიკური უნივერსიტეტის  
პროფესორი გელა ღვინეფაძე,

საქართველოს ტექნიკური უნივერსიტეტის  
პროფესორი რომან სამხარაძე.

© საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, 2017

**ISBN 978-9941-20-743-3**

<http://www.gtu.ge/publishinghouse/>

ყველა უფლება დაცულია. ამ წიგნის არც ერთი ნაწილის (იქნება ეს ტექსტი, ფოტო, ილუსტრაცია თუ სხვა) გამოყენება არანაირი ფორმით და საშუალებით (იქნება ეს ელექტრონული თუ მექანიკური), არ შეიძლება გამომცემლის წერილობითი ნებართვის გარეშე.

სავატორო უფლებების დარღვევა ისჯება კანონით.



## რა არის PHP?

PHP არის ღია საწყისი კოდით საერთო დანიშნულების სცენარის ფართოდ გამოყენებული ენა. მარტივად რომ ვთქვათ, დაპროგრამების ენა, რომელიც სპეციალურად web-სერვერზე გამოსაყენებელი web-დანართების (სცენარების) დასაწერად შეიქმნა.

აბრევიატურა PHP თავდაპირველად იკითხებოდა როგორც “Personal Homepage Programming” (საშინაო web-გვერდების დაპროგრამება), ხოლო ამჟამად, იკითხება, როგორც „Hypertext Preprocessor (ჰიპერტექსტის პრეპროცესორი)“. ენის სინტაქსის საწყისი დაპროგრამების C, Java და Perl ენებია. PHP შესასწავლად საკმაოდ მარტივია. PHP-ის უპირატესობა ის არის, რომ დინამიკურად გენერირებადი web-გვერდის სწრაფად შექმნის საშუალებას იძლევა.

Perl და C ენებთან შედარებით PHP ენის, მნიშვნელოვან უპირატესობას PHP-ის ბრძანებებით ჩანერგილი HTML დოკუმენტების შექმნა წარმოადგენს.

კლიენტის მხარეს შესრულებადი სხვა ნებისმიერი კოდი-საგან, მაგალითად, JavaScript-საგან, PHP მნიშვნელოვნად იმით განსხვავდება, რომ PHP-სკრიპტები სერვერის მხარეს სრულდება. მომხმარებელს თავისი სერვერის ისეთი კონფიგურაციის შერჩევა შეუძლია, რომ HTML-ფაილები PHP პროცესორით დამუშავდეს. ისე, რომ კლიენტის ვერც კი მიხვდეს ჩვეულებრივი HTML-ფაილის თუ სკრიპტის შესრულების შედეგს იღებს.

PHP საშუალებას იძლევა საკმაოდ მცირე დროში შეიქმნას web-დანართი, მივიღოთ მარტივად მოდიფიცირებადი და მომავლით მხარდაჭერილი პროდუქტი.

PHP ასათვისებლად მარტივია და პროფესიონალი პროგრამისტის ყველა მოთხოვნასაც აკმაყოფილებს.

## PHP-ის შესაძლებლობანი

PHP-ს ძალიან ბევრი შესაძლებლობა გააჩნია. ყველაზე მთავარი ის არის, რომ PHP-ის სერვერის მხარეს მომუშავე სკრიპტების გამოყენებაზეა ფოკუსირებული. ამგვარად, PHP-ს შეუძლია შეასრულოს ყველაფერი, რასაც CGI-ის ნებისმიერი სხვა პროგრამა ასრულებს. მაგალითად, შეუძლია მონაცემთა ფორმების დამუშავება, დინამიკური გვერდების გენერირება, გადააგზავნოს და მიიღოს cookies. მაგრამ, გარდა ამისა PHP-ს კიდევ მრავალი სხვა ამოცანის შესრულებაც შეუძლია.

PHP-ის გამოყენების სამი ძირითადი სფერო არსებობს:

- სერვერის მხარეს შესასრულებელი სკრიპტების შექმნა. PHP ამ თვალსაზრისით განსაკუთრებით ფართოდ გამოიყენება. იმისათვის, რომ PHP-სკრიპტების შესრულების შედეგების ნახვა შევძლოთ, საჭიროა მომუშავე web-სერვერი და კომპიუტერზე დაყენებული PHP.
- ბრძანების სტრიქონში შესასრულებლად სკრიპტების შექმნა. შეგიძლიათ შექმნათ PHP-სკრიპტები, რომელთა გაშვება web-სერვერისა და ბრაუზერისაგან დამოუკიდებლად იქნება შესაძლებელი.
- კლიენტის მხარეს შესრულებადი GUI დანართის შექმნა. მართალია, PHP მსგავსი დანართების შესაქმნელად საუკეთესო ენა არ არის, მაგრამ, თუ კარგად ვიცით იგი, მაშინ ეს შესაძლებელი იქნება.

PHP-ის უმეტესობა ისეთი ოპერაციული სისტემებისათვისაა მისაწვდომი, როგორცაა Linux, Unix-ის მრავალი მოდიფიკაცია (HP-UX, Solaris და OpenBSD), Microsoft Windows, Mac OS X, RISC OS

და მრავალი სხვ. გარდა ამისა, PHP-ში მრავალი თანამედროვე web-სერვერის მხარდაჭერაა ჩართული, ისეთი, როგორცაა Apache, Microsoft Internet Information Server, Personal Web Server, სერვერები Netscape და iPlanet, Oreilly Website Pro, Caudium, Xitami, OmniHTTPd და მრავალი სხვ. ამგვარად, თუ აირჩევთ PHP-ს, თქვენ თავისუფლად შეგეძლებათ აირჩიოთ თქვენთვის სასურველი ოპერაციული სისტემა და web-სერვერი. გარდა ამისა, თქვენ შეგეძლებათ პროცედურულ და ობიექტ-ორიენტირებულ დაპროგრამებას შორის აირჩიოთ ერთ-ერთი ან მათი შეთავსება.

PHP-ს შეუძლია არამარტო HTML-ისა და PHP-ს შესაძლებლობების გაცემა, არამედ გამოსახულების, PDF ფაილებისა და Flash (libswf და Ming გამოყენებით) რგოლების ფორმირებაც. PHP ასევე, ნებისმიერი ტექსტური მონაცემის, მაგალითას, XHTML-ისა და სხვა XML-ფაილების გამოტანის, ავტომატური გენერირებისა და შენახვის საშუალებასაც იძლევა და ამგვარად, კემ დინამიკური მეხსიერების ორგანიზებას და სერვერის მხარეს მის განთავსებას ახდენს.

PHP-ის ერთ-ერთ მნიშვნელოვან უპირატესობას მონაცემთა ბაზის ფართო წრის მხარდაჭერა წარმოადგენს. მონაცემთა ბანკის გამოყენების სკრიპტის შექმნა ძალიან მარტივია. ამჟამად, PHP შემდეგ მონაცემთა ბაზებს უჭერს მხარს:

Adabas D	Ingres	Oracle (OCI7 и OCI8)
dBase	InterBase	Ovrimos
Empress	FrontBase	PostgreSQL
FilePro (მხოლოდ წასაკითხად)	mSQL	Solid
Hyperwave	Direct MS-SQL	Sybase
IBM DB2	MySQL	Velocis

Informix	ODBC	Unix dbm
----------	------	----------

PHP აგრეთვე, სხვა სერვისებთან „ერთიერთობას“ LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (Windows-ის პლატფორმა) და სხვა მრავალი პროტოკოლის გამოყენებით უჭერს მხარს.

## PHP-ის უპირატესობა

PHP ენის ყველაზე მნიშვნელოვან თვისებას მისი პრაქტიკულობა წარმოადგენს. PHP პროგრამისტს უნდა აძლევდეს დასმული ამოცანის სწრაფად და ეფექტურად გადაწყვეტის საშუალებას. PHP-ის პრაქტიკულობა განპირობებულია მისი ხუთი მნიშვნელოვანი თვისებით:

- ტრადიციულობით;
- სიმარტივით;
- ეფექტურობით;
- უსაფრთხოებით;
- მოქნილობით.

არსებობს კიდევ ერთი „მახასიათებელი“, რომელიც PHP-ს კიდევ უფრო მიმზიდველს ხდის: იგი უფასოდ ვრცელდება! ამასთან, საწყისი ღია კოდით (Open Source).

## ტრადიციულობა

PHP ენა სხვადასხვა სფეროში მომუშავე მრავალ პროგრამისტს ნაცნობად მოეჩვენება, რადგან ენის მრავალი კონსტრუქცია C, Perl ენებიდანაა აღებული.

PHP კოდები ძალიან ჰგავს იმ კოდებს, რომლებიც C ან Pascal-ზე დაწერილ ტიპობრივ პროგრამებში გვხვდება. ეს საკმაოდ ამცირებს მისი შესწავლის დროს.



მართალია, PHP საკმაოდ ახალგაზრდა ენაა, მაგრამ მან web-პროგრამისტებს შორის ისეთი პოპულარობა მოიპოვა, რომ მოცემულ მომენტში იგი web-დანართების (სკრიპტების) შესაქმნელი თითქმის ყველაზე პოპულარული ენაა.

## სიმარტივე

PHP სცენარი შეიძლება შედგებოდეს როგორც 10000, ისე ერთი სტრიქონისაგან, ყველაფერი ამოცანის სპეციფიკაზეა დამოკიდებული. მომხმარებელს არ მოუხდება ბიბლიოთეკის ჩატვირთვა, კომპილაციის სპეციალური პარამეტრების, ან სხვა მსგავსი პარამეტრების მითითება. PHP-ს მექანიზმი უბრალოდ კოდის შესრულებას პირველი ეკრანული მიმდევრობიდან (<?) იწყებს და იმ მომენტამდე აგრძელებს, ვიდრე არ შეხვდება მისი წყვილი ეკრანული მიმდევრობა (?>). თუ კოდს სწორი სინტაქსი აქვს, იგი ზუსტად ისე შესრულდება, როგორც პროგრამისტმა მიუთითა.

PHP არის ენა, რომელიც შეიძლება ჩაშენებულ იქნეს უშუალოდ გვერდის html-კოდში, რომელიც, თავის მხრივ, PHP-ინტერპრეტატორის მიერ კორექტულად დამუშავდება. მომხმარებელს შეუძლია PHP CGI-სცენარების დასაწერად გამოიყენოს და ტექსტის გამოტანის მრავალ არასასიამოვნო ოპერატორს დააღწიოს თავი. PHP შეიძლება აგრეთვე, HTML-დოკუმენტების ფორმირებისათვის გამოვიყენოთ, რითაც თავს დავაღწევთ გარე სცენარების მრავალ გამოძახებას.

PHP-ის მრავალრიცხოვანი ფუნქცია C ან Pascal-ზე მომხმარებლის მრავალსტრიქონიანი ფუნქციის ჩაწერისაგან გაგვანთავისუფლებს.

## ეფექტურობა

მრავალმომხმარებლიანი არისათვის, რომელთა რიცხვსაც web-პროგრამირება მიეკუთვნება, განსაკუთრებით მნიშვნელოვანი თვისებაა მისი ეფექტურობა.

PHP-ის ძალიან მნიშვნელოვან უპირატესობას მისი „ამძრავი ძალა“ (ბირთვი) წარმოადგენს. PHP-ს „ამძრავი ძალა“ არც კომპილატორი და არც ინტერპრეტატორი არ არის. იგი ინტერპრეტირებულ ტრანსლატორს წარმოადგენს. PHP-ს „ამძრავი ძალის“ ასეთი მოწყობილობა სცენარის განსაკუთრებით სწრაფად დამუშავების საშუალებას იძლევა.

ზოგიერთი შეფასების მიხედვით, PHP-სცენარების უმეტესობა (განსაკუთრებით არც ისე დიდი ზომის) უფრო სწრაფად მუშავდება, ვიდრე Perl-ზე დაწერილი ანალოგიური პროგრამები. მაგრამ, კომპილირებული შესრულებადი ფაილები გაცილებით უფრო სწრაფად - ათჯერ, ზოგჯერ კი ასჯერ უფრო სწრაფად სრულდება. ასე რომ, PHP-ს მწარმოებლობა ძალზე სერიოზული web-დანართების შესაქმნელად სრულიად საკმარისია.

## უსაფრთხოება

PHP-ის შემქმნელებისა და ადმინისტრატორების განკარგულებაში იძლევა უსაფრთხოების მოქნილ და ეფექტურ საშუალებას, რომელიც პირობითად ორ კატეგორიად იყოფა: სისტემური დონის და დანართის დონის საშუალებანი.

1. სისტემური დონის უსაფრთხოების საშუალებანი.

PHP-ში რეალიზებულია ადმინისტრატორის მიერ მართული უსაფრთხოების მექანიზმები; PHP-ის სწორი მომართვის შემთხვევაში ეს მოქმედების მაქსიმალურ თავისუფლებასა და უსაფრთხოებას უზრუნველყოფს. PHP მუშაობს ე. წ. უსაფრთხო რეჟიმში (safe mode), რომელიც ზოგიერთი მნიშვნელოვანი

მაჩვენებლით PHP-ის გამოყენების შესაძლებლობებს ზღუდავს. მაგალითად, შეიძლება შეიზღუდოს შესრულების დრო და გამოყენებული მეხსიერების მაქსიმალური მნიშვნელობა.

## 2. დანართის დონის უსაფრთხოების საშუალებანი.

PHP-ს ფუნქციების სტანდარტულ ნაკრებში დაშიფვრის რამდენიმე საიმედო მექანიზმი შედის. აგრეთვე, PHP თავსებადია დამოუკიდებელი ფირმების მრავალ დანართთან, რაც ელექტრონული კომერციის (e-commerce) დამცავ ტექნოლოგიებთან ინტეგრირების საშუალებას იძლევა. PHP-სცენარის ტექსტის ბრაუზერში დათვალიერება შეუძლებელია, ვინაიდან სცენარის კომპილაცია მომხმარებელთან მის გაგზავნამდე ხდება, რაც მის კიდევ ერთ უპირატესობას წარმოადგენს.

## მოქნილობა

ვინაიდან PHP ჩაშენებული (embedded) ენაა, იგი დამმუშავებლის მოთხოვნების მიმართ განსაკუთრებული მოქნილობით გამოირჩევა. ჩვეულებრივ, PHP-ის გამოყენება რეკომენდებულია HTML-თან ერთად, თუმცა იგი JavaScript, WML, XML და სხვა ენებთანაც ისეთივე წარმატებით ინტეგრირდება.

PHP ასევე არ არის ბრაუზერზე დამოკიდებული, ვინაიდან სცენარის კლიენტისათვის გაგზავნის წინ ხდება სერვერის მხარეს PHP-ის სრულად კომპილირება. ძირითადად PHP-სცენარის ბრაუზერით გადაცემა ნებისმიერი მოწყობილობით ხდება, იქნება ეს მობილური ტელეფონი, ელექტრონული ბლოკნოტი, პეიჯერი თუ პორტატიული კომპიუტერი.

ვინაიდან PHP რომელიმე კონკრეტულ web-სერვერზე ორიენტირებულ კოდს არ შეიცავს, ამიტომ მომხმარებელი რომელიმე კონკრეტული სერვერის გამოყენებით არ არის შეზღუდული. PHP ნებისმიერ ქვემოთ ჩამოთვლილ სერვერზე მუშაობს - Apache,

Microsoft IIS, Netscape Enterprise Server, Stronghold და Zeus. ვინაიდან ყველა ეს სერვერი სახვადასხვა პლატფორმაზე მუშაობს, ამიტომ PHP პლატფორმისაგან დამოუკიდებელ ენას წარმოადგენს და შემდეგ პლატფორმებთან მუშაობს - Unix, Solaris, FreeBSD და Windows NT/2000/XP/2003/7/8.

## PHP-ის განვითარების ისტორია

PHP-მ ბოლო წლების განმავლობაში განვითარების დიდი გზა გაიარა და web-დაპროგრამების ერთ-ერთი ყველაზე პოპულარული ენა გახდა.

PHP-ის საფუძველს წარმოადგენს ძველი პროდუქტი, რომელსაც PHP/FI (Personal Home Page / Forms Interpreter - პერსონალური მთავარი გვერდი / ფორმების ინტერპრეტატორი) ერქვა. PHP/FI 1995 წელს რასმუს ლერდორფმა შექმნა, რომელიც მისი რეზიუმეს მონახულების სტატისტიკის წარმოებისათვის Perl-სკრიპტების ნაკრებს წარმოადგენს. Web-ის განვითარება ის-ის იყო იწყებოდა და ამ ამოცანის გადაწყვეტის არავითარი სპეციალური საშუალებები არ არსებობდა, რამაც ავტორის მიმართ მრავალი შეკითხვა წარმოშვა. ლერდორფმა თავისი ინსტრუმენტების უფასოდ დარიგება დაიწყო. მათ „Personal Homepages Tools“ (PHP) – „პერსონალური მთავარი გვერდის ინსტრუმენტებს“ უწოდებდნენ. მალე მეტი ფუნქციურობა გახდა საჭირო და მან C ენაზე უფრო სრულყოფილი ვერსია შექმნა, რომელიც ბაზებთან მუშაობდა და მარტივი web-დანართების დამუშავება შეეძლო.

1997 წელს PHP/FI 2.0 გამოვიდა. მალე, იგი PHP 3.0-ის ალფა-ვერსიამ შეცვალა. PHP 3.0 ამჟამად არსებული PHP-ის მსგავსი პირველი ვერსია იყო. 1997 წელს ენდი გუტმანსმა (Andi Gutmans) და ზივ სურასკიმ (Zeev Suraski) ხელახლა გადაწერეს კოდები. მათ PHP/FI 2.0 თავიანთი საქმისათვის უსარგებლოდ ჩათვალეს და

რასმუს ლერდორფთან ერთად თავი PHP 3.0 PHP/FI-ის ოფიციალურ მემკვიდრედ გამოაცხადეს, ხოლო PHP/FI-ზე მუშაობა პრაქტიკულად შეწყდა.

დაპროგრამების აბსოლუტურად ახალმა ენამ ახალი სახელი მიიღო. ენას უბრალოდ „PHP“ ეწოდა, რაც ასე იკითხება „Hypertext Preprocessor“ – „ჰიპერტექსტის პრეპროცესორი“.

PHP 3.0 ცხრათვიანი ტესტირების შემდეგ ოფიციალურად 1998 წლის ივნისში იქნა გამოშვებული.

1998 წლის ზამთარში, პრაქტიკულად PHP 3.0-ის ოფიციალური გამოსვლისთანავე, ენდი გუტმანსმა და ზივ სურასკიმ PHP ბირთვის გადამუშავება დაიწყეს. შესაძლებლობების გაფართოებამ PHP 3.0-ს მონაცემთა ბაზების ნაკრებთან მუშაობისა და მრავალი სხვადასხვა API და პროტოკოლის მხარდაჭერის საშუალება მისცა. მაგრამ მას მოდულების მხარდაჭერა არ ჰქონდა და დანართებიც არაეფექტურად მუშაობდნენ.

ახალი „ამძრავი ძალა“ ანუ ბირთვი, რომელსაც „Zend Engine“ უწოდეს, ამ პრობლემას წარმატებით ართმევს თავს და პირველად 1999 წლის შუაში იქნა წარმოდგენილი. PHP 4.0, რომელიც ამ „ამძრავ ძალას“ დაეფუძნა და რომელმაც ხმარებაში შემოიტანა დამატებითი ფუნქციების ნაკრები, ოფიციალურად 2000 წლის მაისში გამოვიდა. ზემოთ ჩამოთვლილი სიახლეების გარდა PHP 4.0-ში კიდევ მრავალი სიახლე იყო, როგორცაა სესიების მხარდაჭერა, ინფორმაციის გამოტანის ბუფერიზაცია, მომხმარებლის მიერ შეტანილი ინფორმაციის გადამუშავების უფრო მეტი უსაფრთხოება და რამდენიმე ახალი ენობრივი კონსტრუქცია.

PHP-ის მეხუთე ვერსია 2004 წლის 13 ივლისს იქნა გამოშვებული. ცვლილება მისი ბირთვის Zend (Zend Engine 2) განახლებას მოიცავს, რამაც ინტერპრეტატორის ეფექტურობა მნიშვნელოვნად გაზარდა. შემოტანილ იქნა XML - მონიშვნის ენის

მხარდაჭერა. სრულად გადამუშავდა ობიექტორიენტირებული დაპროგრამების ფუნქციები, რომელიც Java-ში გამოყენებულ მოდელს ემთხვევა. კერძოდ, შემოღებულია დესტრუქტორი, ღია, დახურული და დაცული წევრები და მეთოდები, საბოლოო წევრები და მეთოდები, ინტერფეისები, ობიექტთა კლონირება და მრავალი სხვა.

PHP-ის მეექვსე ვერსიაზე მუშაობა 2006 წლის ოქტომბერში დაიწყო. მისი ერთ-ერთი ძირითადი სიახლე Unicode-ის მხარდაჭერა უნდა ყოფილიყო. მაგრამ 2010 წლის მარტში PHP 6-ის დამუშავება უპერსპექტივოდ იქნა მიჩნეული Unicode-ის მხარდაჭერის სირთულის გამო და მასზე მუშაობა შეწყდა. PHP 6-ის საწყისმა კოდმა 5.4-ის ვერსიის დამუშავებაში გადაინაცვლა.

## PHP-ის ბირთვი

დაპროგრამების ენა ორი სახის არსებობს: ინტერპრეტირებადი და კომპილირებადი. როგორია PHP ენა? ამ კითხვაზე პასუხის გასაცემად საჭიროა ჯერ ტერმინებში გავერკვეთ.

პროგრამას, რომელიც დაპროგრამების ერთ ენაზე დაწერილ კოდებს სხვა ენაზე გადათარგმნის ტრანსლატორი ეწოდება. კომპილატორი - ეს იგივე ტრანსლატორია, ოღონდ მაღალი დონის ენაზე დაწერილ კოდებს მანქანურ ენაზე გადაიყვანს. კომპილაციის პროცესის შედეგად ორობით კოდებში ჩაწერილი შესრულებადი ფაილი მიიღება, რომელიც შესასრულებლად შეიძლება კომპილატორის გარეშე გაეშვას.

ინტერპრეტატორი არ თარგმნის კოდს, იგი ასრულებს მას. ინტერპრეტატორი პროგრამის კოდს ანალიზებს და თითოეულ სტრიქონს ასრულებს. ასეთი კოდის შესრულების დროს ინტერპრეტატორით სარგებლობა აუცილებელია.

ინტერპრეტატორი შესრულების სისწრაფით კომპილატორს საგრძნობლად ჩამორჩება, ვინაიდან ორობითი კოდი ბევრად უფრო სწრაფად სრულდება. სამაგიეროდ, ინტერპრეტატორი საშუალებას იძლევა პროგრამა მისი შესრულების პროცესში სრულად ვაკონტროლოთ.

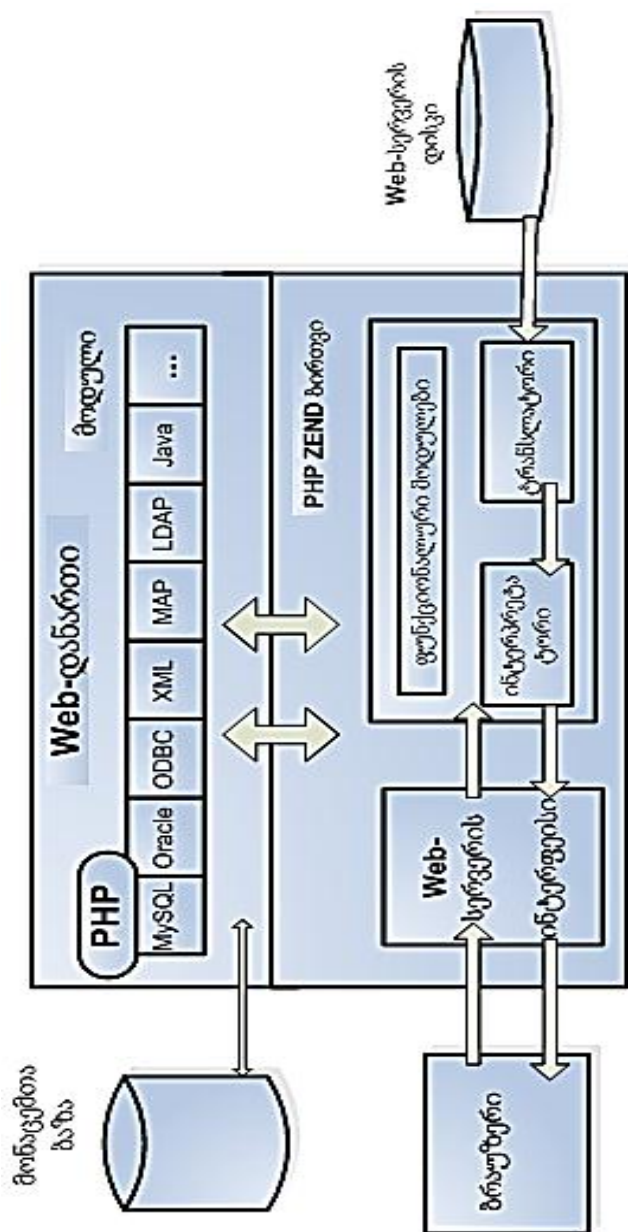
რაც შეეხება PHP-ს, ის არც კომპილატორია და არც ინტერპრეტატორი. იგი კომპილატორსა და ინტერპრეტატორს შორის რაღაც საშუალოს წარმოადგენს. ვნახოთ, როგორ ამუშავებს PHP კოდს.

განვიხილოთ სურათი 1:

ვხედავთ, რომ PHP ორი, თითქმის დამოუკიდებელი ბლოკისაგან შედგება - ტრანსლატორისა და ინტერპრეტატორისაგან. ეს მისი სწრაფქმედების გასაზრდელად გახდა საჭირო.

PHP-ის დასაწყისში სცენარი მიეწოდება. იგი თარგმნის (ტრანსლირებას უკეთებს) მას, ამოწმებს სინტაქსს და სპეციალურ ბაიტ-კოდში (შიგა წარმოდგენა) გადაჰყავს. შემდეგ PHP ბაიტ-კოდს (და არა თვით პროგრამის კოდს) ასრულებს, ამასთან ის შესრულებად ფაილს არ ქმნის.

ბაიტ-კოდი უფრო კომპაქტურია, ვიდრე ჩვეულებრივი პროგრამის კოდი, ამიტომ მისი შესრულება (ინტერპრეტირება) უფრო მარტივია. სინტაქსური განხილვა მხოლოდ ერთხელ ხორციელდება ტრანსლაციის ეტაპზე, ხოლო შემდეგ უკვე „ნახევარფაბრიკატი“- ბაიტ-კოდი სრულდება, რომელიც ამ მიზნისათვის უფრო ხელსაყრელია. ამიტომ, PHP უფრო ინტერპრეტატორია, ვიდრე კომპილატორი. ასეთი ორმაგი სამუშაო შემდეგი მიზნით ჩატარდა:





დავუშვათ, გვაქვს ციკლი 1-დან 10-მდე, რომელიც 100 ოპერატორს შეიცავს. ასეთი ციკლი ათჯერ დატრიალდება. ინტერპრეტატორმა თითოეული ნაბიჯი ყოველი გავლის დროს უნდა შეამოწმოს, ე. ი. სულ  $10 \cdot 100 = 1000$ -ჯერ. თუ ამ ციკლს ერთხელ ბაიტ-კოდში გადავიყვანთ, მაშინ ინტერპრეტატორს მისი შემოწმება მხოლოდ 10-ჯერ დასჭირდება. ეს კი ნიშნავს, რომ ეს სცენარი 10-ჯერ სწრაფად შესრულდება.

აქედან გამომდინარეობს, რომ PHP ინტერპრეტირებულ ტრანსლატორს წარმოადგენს.

PHP 3 ვერსია მთლიანად „ინტერპრეტატორი“ იყო, შემდგომი ვერსიები კი ინტერპრეტირებული ტრანსლატორებია.

ინტერპრეტატორის გამოყენებას (ე. ი. PHP-საც) შემდეგი უპირატესობა აქვს:

- გამოყოფილი მეხსიერების გათავისუფლება აუცილებელი არ არის, ასევე არ არის საჭირო სამუშაოს დამთავრების შემდეგ ფაილების დახურვა - ყოველივე ამას ინტერპრეტატორი გააკეთებს;
- არ არის საჭირო ფიქრი ცვლადების ტიპზე, ასევე არ არის საჭირო ცვლადის წინასწარ გამოცხადება მისი პირველი გამოყენების წინ;
- პროგრამების გამართვა და შეცდომების აღმოჩენა მნიშვნელოვნად გამარტივებულია - ინტერპრეტატორი ამ პროცესს მთლიანად აკონტროლებს;
- web-დანართების კონტექსტში ინტერპრეტატორს კიდევ ერთი მნიშვნელოვანი უპირატესობა აქვს - არ არსებობს სერვერის „დაკიდება“ პროგრამის არასწორი მუშაობის გამო.

PHP-ის ერთადერთ ნაკლს მისი შედარებით დაბალი სწრაფქმედება წარმოადგენს, რომელიც მხოლოდ დიდი და რთული

ციკლების, დიდი რაოდენობის სტრიქონების დამუშავების დროსაა შესამჩნევი. ეს ნაკლი მძლავრი პროცესორების არსებობის შემთხვევაში სულ უფრო და უფრო ნაკლებად შესამჩნევი ხდება.

# Web-სერვერის ჩატვირთვა და ინსტალაცია

PHP-სკრიპტების მუშაობა რომ შევამოწმოთ და გავმართოთ, Web-სერვერის კომპიუტერზე დაყენება და ინსტალაციაა საჭირო. Web-სერვერის რამდენიმე კონფიგურაციის პარამეტრები არსებობს, რომლებიც განსაზღვრავს, როგორ იმუშავებს მასში PHP-პროგრამა. უმეტეს შემთხვევაში Web-სერვერის ავტომატური ინსტალაციის რეჟიმი გამოიყენება, რომლის დეტალური ილუსტრაცია ქვემოთაა მოყვანილი.

იმისათვის, რომ Web-სერვერმა იმუშაოს, კომპიუტერი, რომელზედაც ის არის დაყენებული, აუცილებელი არ არის ინტერნეტში ან ლოკალურ ქსელში იყოს ჩართული. Web-სერვერთან, რომელიც მოცემულ კომპიუტერშია დაყენებული, წვდომა ამავე კომპიუტერში არსებული Web-ბრაუზერის საშუალებით შეიძლება განხორციელდეს, იმ შემთხვევაშიც კი, როცა ამ კომპიუტერს არც მოდემი და არც ქსელური პლატა არ გააჩნია. თუმცა მართალია, თვით პროგრამული უზრუნველყოფის ჩატვირთვისა და მისი დაყენებისათვის ინტერნეტი აუცილებელია.

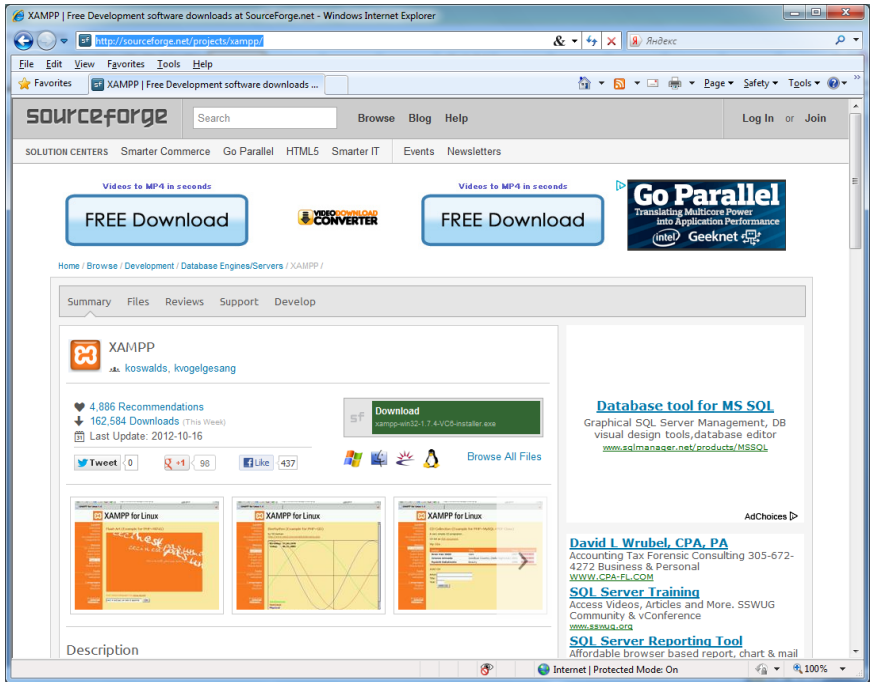
PHP-ს მუშაობისათვის განკუთვნილი Web-სერვერია Apache, მაგრამ უფრო მოხერხებული სერვერია XAMPP 1.7.4 Win32, XAMPP Windows 1.8.1.

Web-სერვერის გადმოწერა შემდეგი საიტებიდანაა შესაძლებელი:

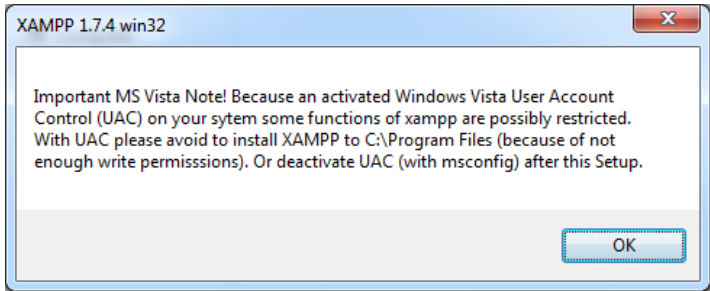
<http://sourceforge.net/projects/xampp/>

<http://www.apachefriends.org/en/xampp-windows.html>

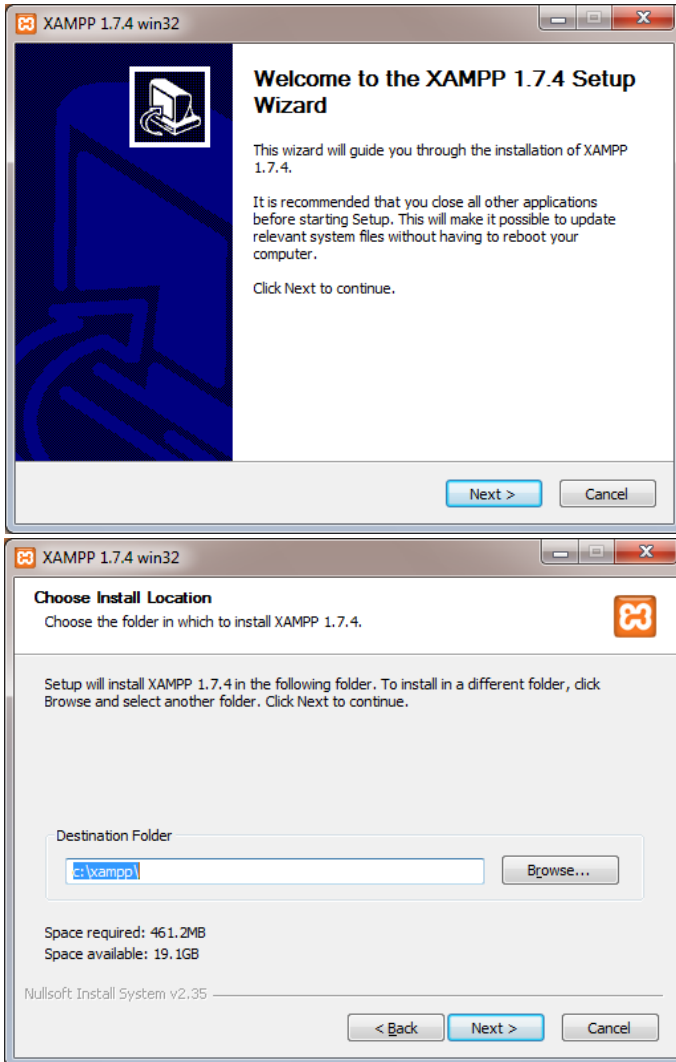
პირველ შემთხვევაში ეკრანზე გაიხსნება ფანჯარა, საიდანაც XAMPP 1.7.4 Win32 სერვერის გადმოწერაა შესაძლებელი.



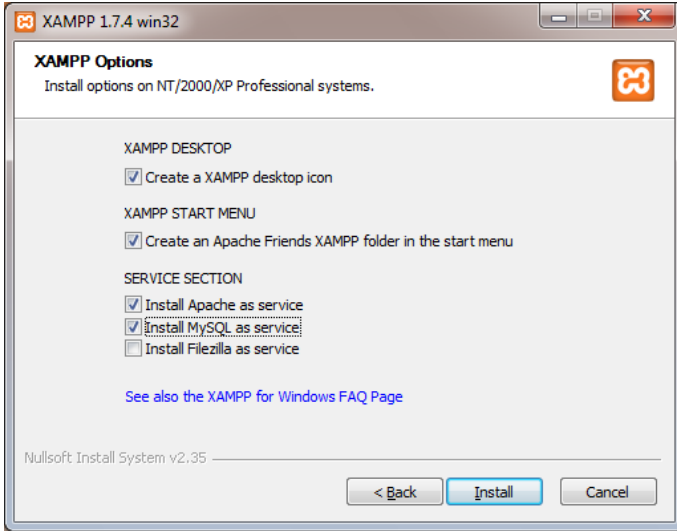
Download დილაგზე თითის დაჭერით ახალი ფანჯარა გაიხსნება და რამდენიმე წამში Web-სერვერის გადმოწერა დაიწყება. გადმოწერის შემდეგ უნდა მოვახდინოთ მისი ინსტალაცია, რისთვისაც - xampp-win32-1.7.4-VC6-installer ფაილი გამოიყენება. ამ ფაილის ნიშნაკზე ორჯერ დაწკაპუნებით გაიხსნება შემდეგი დიალოგური ფანჯარა:



და OK ღილაკზე თითის დაჭერით საინსტალაციო დიალოგი დაიწყება:

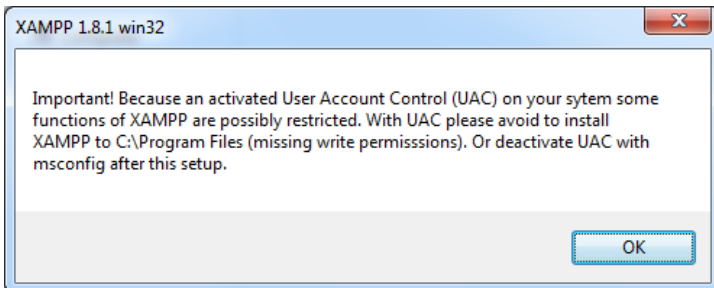


ინსტალაციის შემდეგ ეტაპზე გადასვლა Next ღილაკით ხორციელდება:

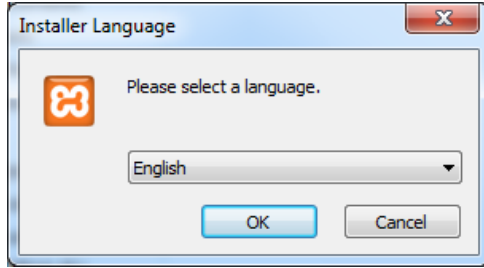


აქ კი SERVICE SECTION ველში სასურველი სერვისები უნდა მოვნიშნოთ და Install ღილაკის საშუალებით ინსტალაციის ბოლო ეტაპი ჩავრთოთ.

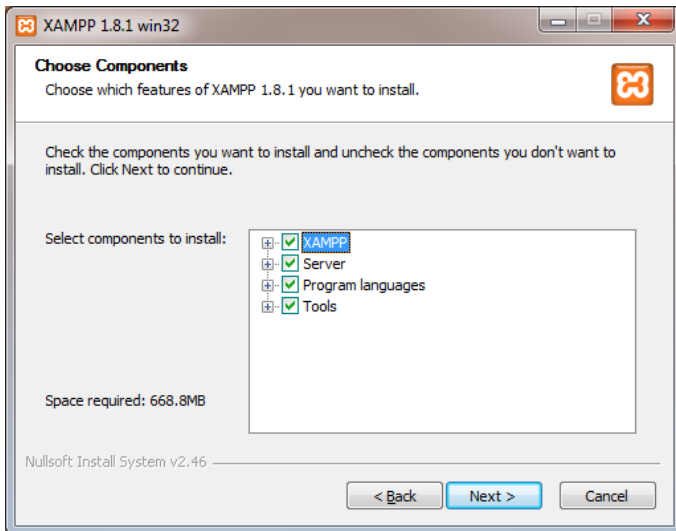
თითქმის ანალოგიურად მიმდინარეობს XAMPP Windows 1.8.1 სერვერის ინსტალაციაც, ოღონდ აქ უნდა გავხსნათ შესაბამისი ფაილი - xampp-win32-1.8.1-VC9-installer, სადაც პირველი დიალოგური ფანჯრის შემდეგ

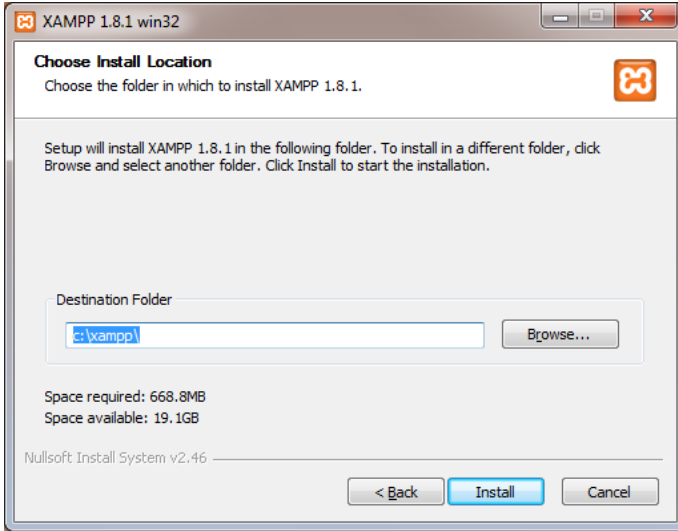


ინსტალაციის ენა უნდა განისაზღვროს:

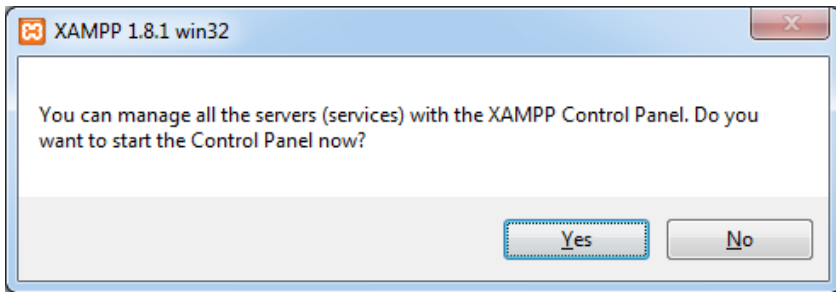


საინსტალაციო ენის შერჩევის შემდეგ დაიწყება სერვერის ინსტალაცია, რომლის დროსაც პირველი დიალოგური ფანჯრის შემდეგ განსხვავებული ფანჯრები გამოდის:



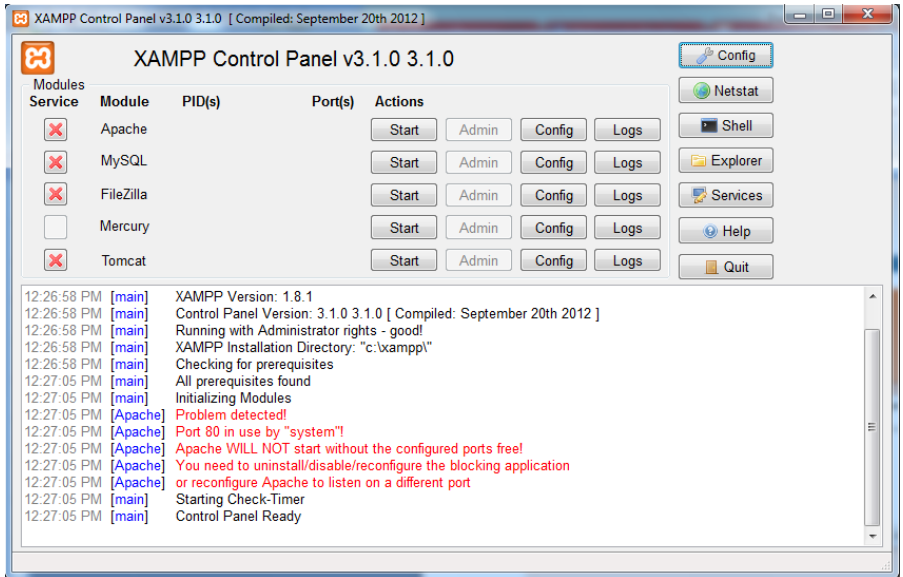


ინსტალაციის დასრულების შემდეგ ეკრანზე გამოჩნდება დიალოგური ფანჯარა, რომლის მიხედვითაც შესაძლებელია სერვერის Control Panel-ის დაყენებაც:



თუ ამ დროს დავაჭერთ No ღილაკს, მაშინ ინსტალაციის რეჟიმი დასრულდება, ხოლო თუ თითს Yes ღილაკს დავაჭერთ, ამ შემთხვევაში გამოდის სერვერის Control Panel-ის დაყენების შემდეგი დიალოგური ფანჯარა:





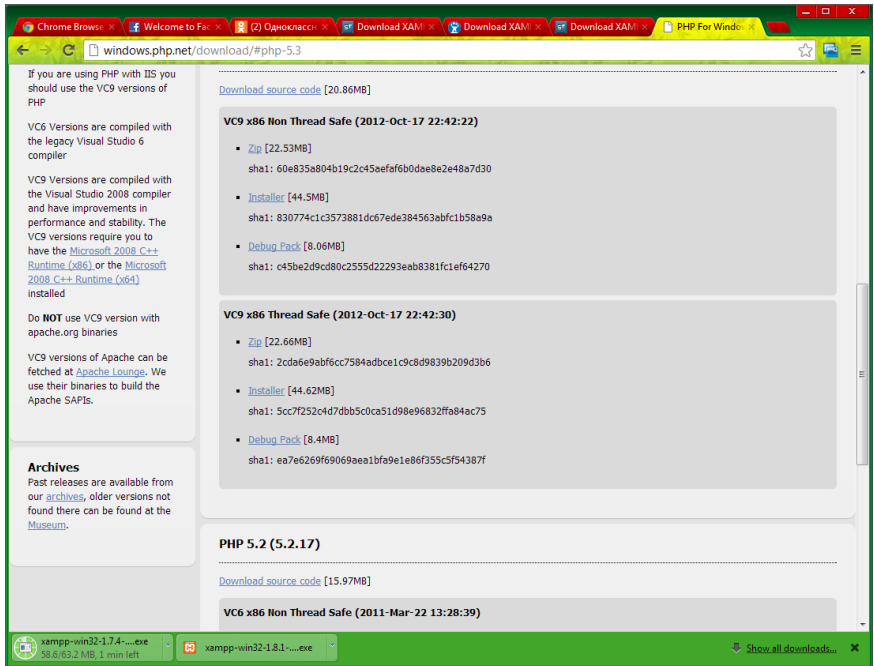
შესაბამისი რეჟიმების არჩევის შემდეგ სერვერის ინსტალაცია დამთავრდება.

## PHP 5-ის ჩატვირთვა და ინსტალაცია

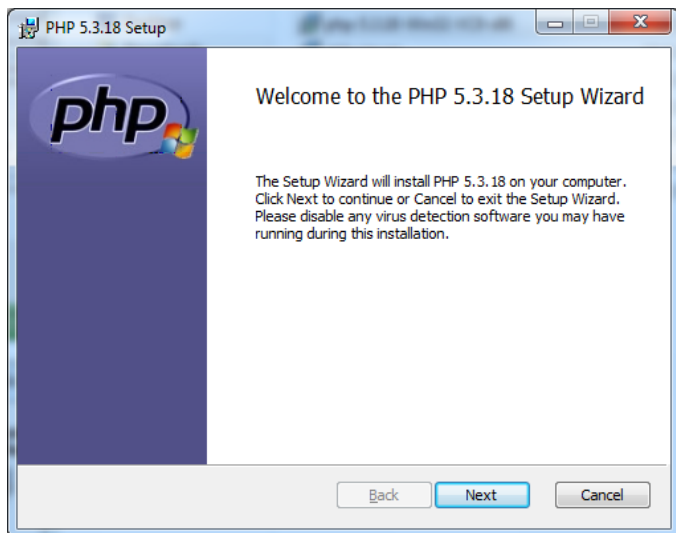
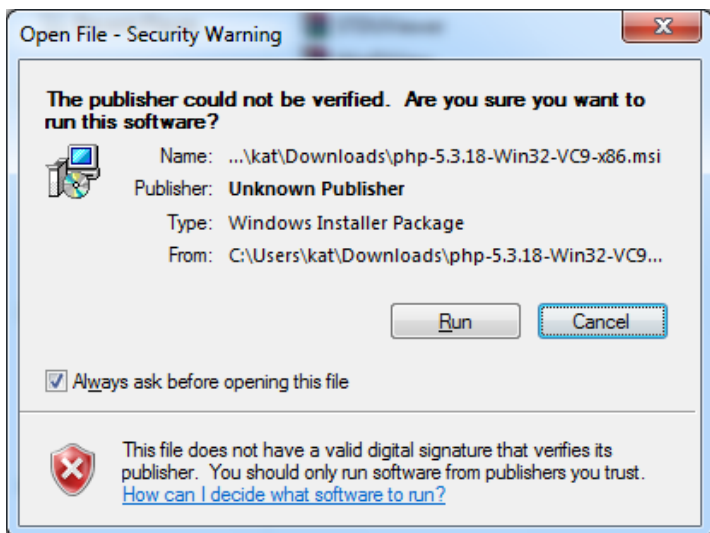
Web-სერვერის დაყენებისა და გაშვების შემდეგ PHP 5-ის დაყენებაც შეიძლება. PHP 5-ის ყველაზე ბოლო ვერსიის მიღება შესაძლებელია [www.php.net](http://www.php.net) Web-საიტის „Downloads“ გვერდზე გადასვლით, რომელიც ქვემოთ სურათზეა მოცემული (მისი სახე დროის მიხედვით შეიძლება შეიცვალოს). მასში ავარჩიოთ ჩვენთვის მისაღები ვერსია და გადმოვწეროთ კომპიუტერში.

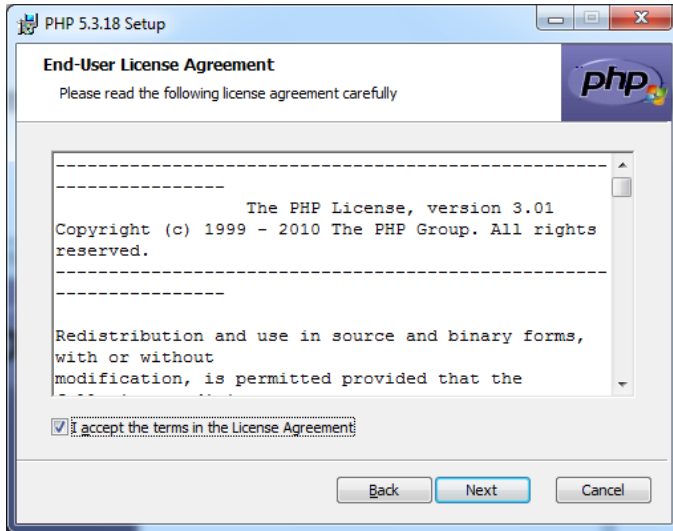
თუ გადმოვწერთ დაარქივებულ (zip-კოდი) ფაილს ანუ Zip რეჟიმს ავირჩევთ, მაშინ ახალი ფოლდერი (საქადალდე) უნდა შევქმნათ, სადაც ამ ფაილს შევინახავთ და დავაარქივებთ. ამის შემდეგ საინსტალაციო პროგრამას - `php-5.3.17-nts-Win32-VC9-x86` გავუშვებთ შესასრულებლად.

თუ Installer რეჟიმს ავირჩევთ, მაშინ კომპიუტერში საინსტალაციო პროგრამა გადმოიწერება და შესაძლებელი იქნება მისი პირდაპირ შესასრულებლად გაშვება.

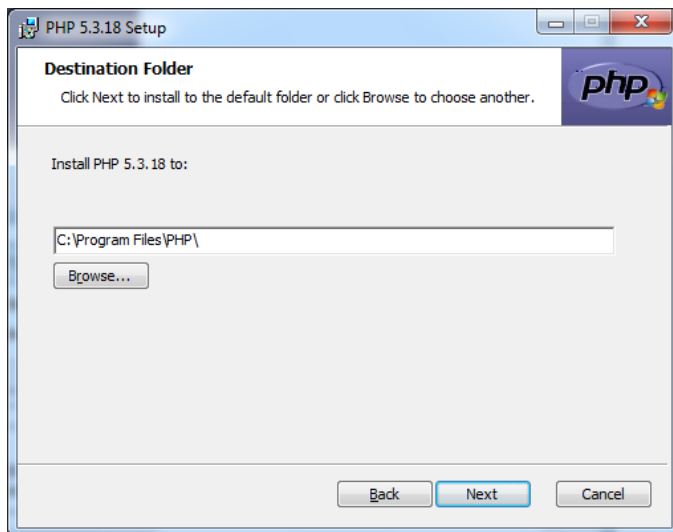


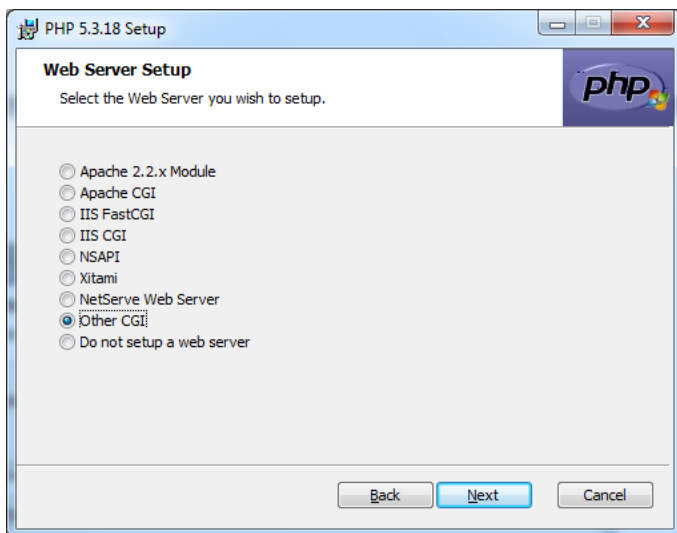
პროგრამის გაშვების შემდეგ ეკრანზე გამოვა Open File – Security Warning დიალოგური ფანჯარა, სადაც Run ღილაკზე თითის დაჭერით პროგრამის ინსტალაცია დაიწყება, ხოლო ყოველ შემდეგ ნაბიჯზე გადასასვლელად Next ღილაკი უნდა გამოვიყენოთ:





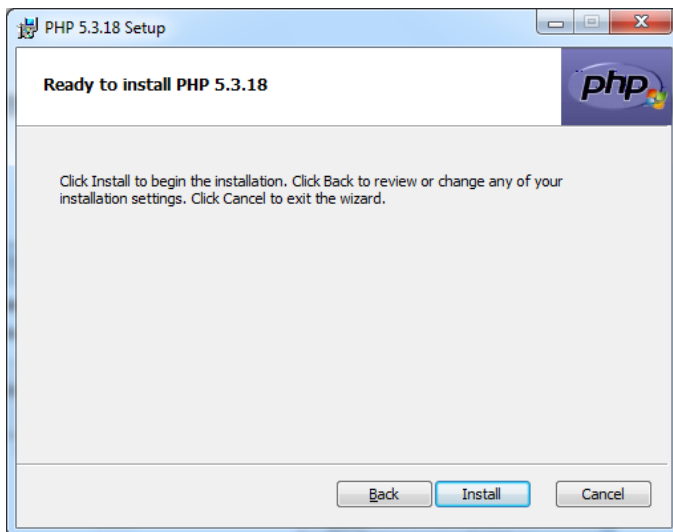
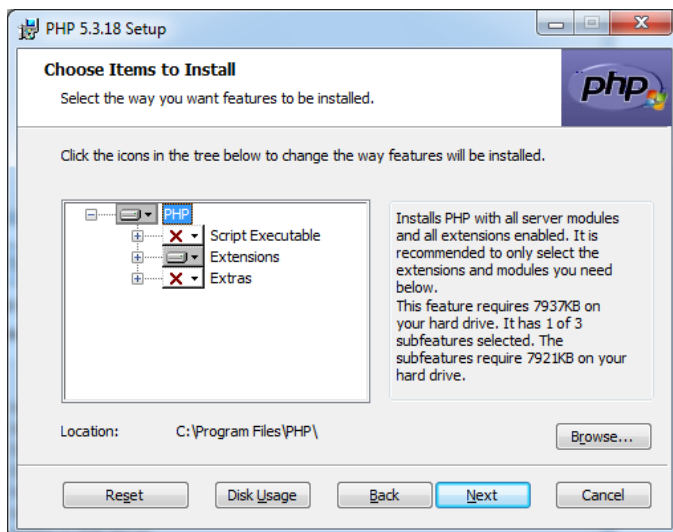
ინსტალაციის შემდეგ ეტაპზე გადასასვლელად I accept the terms in the License Agreement ველი ალმით უნდა მოინიშნოს.



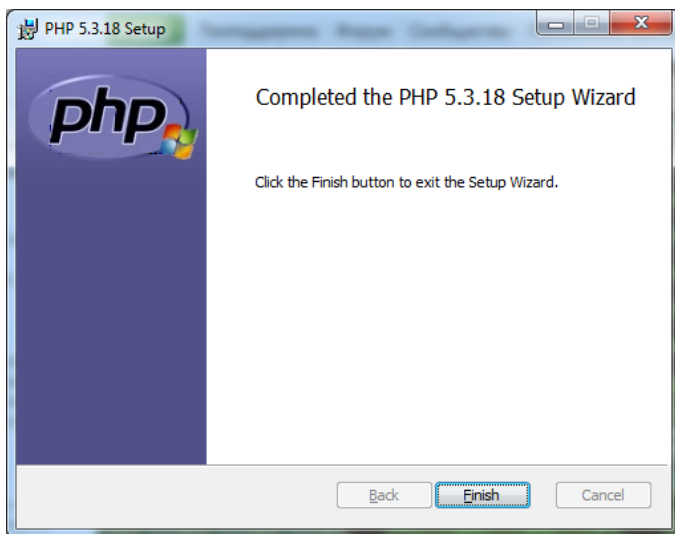


PHP 5-ს ინსტალაციის დროს აუცილებელია Web-სერვერზე გადაწყდეს, სახელდობრ, როგორ იმუშავებს PHP: ან როგორც CGI-პროგრამა, ან როგორც ცალკეული სტატიკური ან დინამიკური მოდული. CGI (Common Gateway Interface – „შლუზის (კარიბჭე)“ გარე პროგრამის საერთო ინტერფეისი) - ინტერფეისის სტანდარტი Web-სერვერთან კავშირისათვის. პროგრამას, რომელიც ასეთი ინტერფეისით Web-სერვერთან ერთად მუშაობს, ხშირად „შლუზებს“ უწოდებენ, თუმცა ბევრი მიზანშეწონილად თვლის „სკრიპტი“ (სცენარი) ან „CGI-პროგრამა“ ვუწოდოთ. CGI ისეთი ინტერპრეტატორებისათვის, როგორც PHP 5-ია, ძალზე მნიშვნელოვანი საშუალებაა. სცენარების უსაფრთხოების რისკებთან დაკავშირებით, უმეტეს შემთხვევაში PHP 5-ის კომპილაცია, რეკომენდებულია განხორციელეს, როგორც სტატიკური ან დინამიკური მოდული, თუმცა ყველაზე ხშირად PHP Web-სერვერთან ერთად მუშაობს ისე, რომ .php გაფართოების მქონე ფაილებით წარმოდგენილი Web-გვერდი, ბრაუზერისათვის გადაგზავნის წინ PHP-ინტერპრეტატორის მიერ დამუშავდეს.

ამ თემის შესახებ დიდი მოცულობის დოკუმენტაცია შეიძლება PHP-ის საიტზე ([www.php.net](http://www.php.net)) მოვიძიოთ.



Install დილაკის საშუალებით ჩავრთოთ ინსტალაციის ბოლო ეტაპი.



ამით დასრულდება პროგრამის ინსტალაცია.



## პირველი პროგრამა PHP-ზე

PHP-ზე პროგრამის დასაწერად ისევე, როგორც HTML-ზე საჭიროა ნებისმიერი ტექსტური რედაქტორი (თუმცა არსებობს სპეციალური რედაქტორები, რომლებიც PHP-სინტაქსს აკონტროლებენ, მაგალითად, PHPEdit). რა თქმა უნდა, იგულისხმება, რომ Web-სერვერი და PHP ინტერპრეტატორი დაყენებულია.

ზოგადად PHP-სკრიპტი HTML-ის ელემენტების გარეშე ძალზე მარტივად გამოიყურება - იგი იწყება და მთავრდება სპეციალური სიმბოლოებით (ტეგებით).

```
<?php  
echo "Hello, World!";  
?>
```

PHP-სკრიპტში ფუნქციებისა და ოპერატორების ჩაწერის დროს რეგისტრს (ასომთავრული თუ პატარა ასოები) არა აქვს მნიშვნელობა: <?PHP და <?php ერთნაირად აღიქმება. ყოველი PHP-ოპერატორი სიმბოლო „წერტილ-მძიმით“ - „;“ უნდა დამთავრდეს. ეს სიმბოლო ინტერპრეტატორს მიუთითებს, რომ ოპერატორი დამთავრდა და მისი შესრულება შეიძლება. PHP-სკრიპტის დაწერის დროს სასურველია ყოველი ოპერატორის ჩაწერა ახალი სტრიქონიდან დავიწყოთ.

პროგრამის გაშვებამდე საჭიროა იგი ჯერ სერვერზე დავაყენოთ. ამისათვის, ჩაწერილი PHP-სკრიპტი start.php სახელით შევინახოთ. შემდეგ მისი კოპირება ჩვენი სერვერის Computer/Local Disk (C:)/Program Files/Apache Group/Apache2/htdocs ან Computer/Local Disk (C:)/xampp/htdocs საქაღალდეში (ფოლდერში). Windows-ში ამ საქაღალდის განთავსება დაყენებულ web-სერვერზე და მის მომართვაზე დამოკიდებული. ამის შემდეგ,

თქვენი ბრაუზერის სამისამართო სტრიქონში აკრიფეთ მისამართი `http://localhost/start.php` და თუ ყველაფერი სწორადაა დაყენებული და მომართული, დაინახავთ ტექსტს Hello, World!



გავარჩიოთ ჩვენი მარტივი სკრიპტის კოდი:

PHP-ს კოდი მოთავსებულია სპეციალურ ტეგებს `<? და ?>` შორის. სკრიპტის დაწყება აღინიშნება გამხსნელი ტეგით `<?>`, ხოლო დამთავრება - დამთავრების ტეგით `?>`. გამხსნელი ტეგის შემდეგ პირველი ოპერატორი `echo` მოდის, რომელსაც ეკრანზე გამოაქვს ინფორმაცია. ოპერატორი `echo` ყველაზე მთავარი ოპერატორია, ვინაიდან მას ინფორმაცია ბრაუზერში გამოჰყავს, მისი შესრულების შედეგად ჩვენ ეკრანზე ვხედავთ Hello, World!-ს

ახლა ჩვენი სკრიპტი ცოტათი გავართულოთ, რისთვისაც მასში `html`-ტეგების გამოტანა ჩავამატოთ:

```
<?php
echo "<html><body>";
echo "<h1>Hello, World!</h1>";
echo "</body></html>";
?>
```

ჩვენი მოდიფიცირებული სკრიპტი, ამჯერად, დიდი ასოებით გამოიტანს ტექსტს Hello, World!

# Hello, World!

ახლა დავეწეროთ სკრიპტი ისე, რომ მან შეძლოს ტექსტის ქართულად გამოტანა. ამისათვის, საჭიროა სკრიპტის აკრების შემდეგ მისი შენახვის დროს Notepad ტექსტურ რედაქტორს მივცეთ Save ან Save As ბრძანება ან გამოვიყენოთ <Ctrl>+<S> კლავიშთა კომბინაცია. გახსნილ დიალოგური ფანჯრის File name ველში ჩავწეროთ მოცემული კოდის სახელი, ხოლო Encoding ჩამოშლად სიაში ავირჩიოთ UTF-8. მაგალითი:

```
<?php
echo "<html><head>";
echo "<title>ჩემი პირველი PHP სკრიპტი</title>";
echo "</head>";
echo "<body>";
echo "<h1>სალამი! ეს PHP სკრიპტია!</h1>";
echo "</body>";
echo "</html>";
?>
```

მოყვანილ მაგალითში ბრაუზერს ჯერ გავუგზავნეთ სათაური <head><....></head> და აქვე მივუთითეთ ბრაუზერის სათაური <title>ჩემი პირველი PHP სკრიპტი </title>. ხოლო შემდეგ გამოვიტანეთ ტექსტი ქართულ ენაზე: <body><h1>სალამი! ეს PHP სკრიპტია! </h1></body>.

# სალამი! ეს PHP სკრიპტია!

როგორც ზემოთ განხილულ მაგალითში იყო, აქაც მთელი ინფორმაცია html-კოდების ჩათვლით, echo ოპერატორის საშუალებით გამოგვეყვდა. ზოგადად, PHP-სკრიპტები შეიძლება გამოტანილი იყოს არა echo ოპერატორის საშუალებით, არამედ PHP-კოდები შეიძლება უშუალოდ იყოს ჩასმული html-დანართში. განვიხილოთ მაგალითი:

```
<html>
  <head>
    <title>ჩემი პირველი PHP სკრიპტი</title>
  </head>
  <body>
    <?php
      echo "სალამი! ეს PHP სკრიპტია!";
    ?>
  </body>
</html>
```

ახლა, ჩვენ შეგვიძლია მოცემული PHP-სკრიპტი .php ფაილის სახით სერვერზე შევინახოთ, მაგალითად, mag3.php სახელით და შედეგები შევამოწმოთ.

Html-კოდი PHP ინტერპრეტატორის მიერ დამუშავდება, ამიტომ ამ სკრიპტის შესრულების დროს შეცდომა არ წარმოიშობა.

სალამი! ეს PHP სკრიპტია!

თუ იგივე სკრიპტში გამოსატანი ტექსტი გვინდა გამოვიტანოთ სათაურის შრიფტის სახით, მაშინ მას უნდა ჩავუმატოთ შესაბამისი ტეგები, მაგალითად:

```
<html>
  <head>
    <title>ჩემი პირველი PHP სკრიპტი</title>
  </head>
  <body>
    <h1><?php
      echo "სალამი! ეს PHP სკრიპტია!";
    ?></h1>
  </body>
</html>
```

სალამი! ეს PHP სკრიპტია!

გამოსატანი ტექსტის დაფორმატებისათვის (სტრიქონები, გვერდებად დაყოფა, პოზიციონირება) შეიძლება HTML- და PHP-

კოდების კომბინირება. ქვემოთ მოგვყავს მაგალითი, სადაც HTML-ტეგებს შორის გამოტანილი სტრიქონების დაფორმატება PHP-კოდებით ხდება.

```
<html>
  <head>
    <title>HTML-PHP</title>
  </head>
  <body>
    <b>
      <?php
        echo "მე ვსწავლობ PHP-ს";
      ?>
    </b>
    <p><b> ეს ჩვეულებრივი HTML-ტეგებია!</b>
    <br>
    <?php
      echo "ეს კვლავ PHP-კოდის მუშაობის შედეგია!";
    ?>
  </body>
</html>
```

**მე ვსწავლობ PHP-ს**

**ეს ჩვეულებრივი HTML-ტეგებია!**  
**ეს კვლავ PHP-კოდის მუშაობის შედეგია!**

ამ მაგალითიდან ნათლად ჩანს, რომ სკრიპტის ერთ ფაილში პროგრამისტმა შეიძლება რამდენიმე HTML- და PHP-ფრაგმენტი ისეთი მიმდევრობით გააერთიანოს, როგორც კონკრეტული ამოცანის გადასაწყვეტადაა საჭირო.

აგრეთვე შესაძლებელია echo ოპერატორის საშუალებით გამოტანილი სტრიქონებში HTML-ტეგები იყოს გამოყენებული. PHP-ისთან მუშაობის დროს შეიძლება ორი სტრატეგიის გამოყენება:

1. პირველი სტრატეგიით PHP არის HTML-ის დანამატი და, შესაბამისად, PHP-სკრიპტი HTML-დოკუმენტში ჩადგმულ ფრაგმენტს წარმოადგენს;
2. მეორე სტრატეგიით კი HTML არის PHP-ის დანამატი და HTML-ტეგი PHP-სკრიპტის მიერ გამოტანილი შეტყობინების დაფორმატების საშუალებას წარმოადგენს.

ერთი შეხედვით ეს ორი მიდგომა თითქოს ერთმანეთს გამორიცხავს, მაგრამ პრაქტიკულად მათი გამოყენება ერთსა და იმავე სკრიპტში წარმატებითაა შესაძლებელი.

ქვემოთ მოყვანილია მეორე მიდგომის მაგალითი:

```
<?php  
echo "მე ვსწავლობ PHP-ს";  
echo "<h2>სათაურის მაგალითი</h2>";  
echo "<hr>";  
echo "ახალი PHP-კოდი";  
?>
```

მოცემული სკრიპტის მუშაობის შედეგი იქნება

მე ვსწავლობ PHP-ს

## სათაურის მაგალითი

ახალი PHP-კოდი

როდესაც PHP ამუშავებს ფაილს, იგი უბრალოდ გადასცემს მას ტექსტს, ვიდრე არ შეხვდება ერთ-ერთი სპეციალური ტეგი, რომელიც მას ატყობინებს, რომ აუცილებელია დაიწყოს ტექსტის, როგორც PHP კოდის ინტერპრეტაცია. შემდეგ იგი მთლიანად ასრულებს ნაპოვნ კოდს დახურვის ტეგამდე, რომელიც ინტერპრეტატორს ისევ ატყობინებს, რომ შემდეგ მას ისევ მოსდევს უბრალო ტექსტი. ეს მექანიზმი საშუალებას გვაძლევს PHP-კოდი HTML-ში ჩავნერგოთ - PHP ტეგებს გარეთ ყველაფერი უცვლელი რჩება, მაშინ როდესაც მის შიგნით - მისი ინტერპრეტირება ხდება, როგორც PHP კოდისა.

არსებობს ტეგების ოთხი ნაკრები, რომლებიც PHP-კოდის აღსანიშნავად შეიძლება იყოს გამოყენებული. მათ შორის ყოველთვის მისაწვდომია მხოლოდ ორი (`<?php. . .?>` და `<script language="php">. . .</script>`): სხვები `php.ini` ფაილის კონფიგურაციაში შეიძლება ჩართული იყოს ან გამორთული. თუ გვსურს PHP-კოდი XML-ში ან XHTML-ში ჩავსვათ, რათა XML-ს შეესაბამებოდეს, საჭიროა `<?php. . .?>` ფორმის გამოყენება.

ქვემოთ მოყვანილია PHP-ს მიერ მხარდაჭერილი ტეგები:

1. `<?php echo("თუ თქვენ გინდათ იმუშაოთ XHTML ან XML დოკუმენტებთან, გააკეთეთ ასე\n"); ?>`



2. `<? echo ("ეს SGML დამუშავების უმარტივესი ინსტრუქციაა \n"); ?>`

`<?= გამოსახულება ?>` ეს

სინონიმია გამოსახულებისათვის "`<? echo გამოსახულება ?>`"

3. `<script language="php">`

`echo ("ზოგიერთ რედაქტორს (მაგალითად, FrontPage)`

`გადამუშავების ინსტრუქცია არ უყვარს");`

`</script>`

4. `<% echo ("თქვენ შეგიძლიათ ამორჩევით გამოიყენოთ ტეგები ASP სტილში"); %>`

`<%= $variable; # ეს სინონიმია "<% echo . . ."-სათვის %>`

პირველი ვარიანტი, `<?php. . ?>`, ყველაზე მისაღებია, ვინაიდან იგი PHP კოდებში საშუალებას იძლევა XML წესების შესაბამისად გამოვიყენოთ, მაგალითად, XHTML-ში.

მეორე ვარიანტის გამოყენება ყოველთვის არ არის შესაძლებელი. მოკლე ტეგის გამოყენება შეიძლება მაშინ, როდესაც ის ჩართულია. ეს შეიძლება გაკეთდეს `short_tags()` (მხოლოდ PHP 3-ში) ფუნქციის გამოყენებით PHP ფაილის კონფიგურაციაში `short_open_tag` პარამეტრის ჩართვით, ან PHP-ს კომპილაცია `configure`-სათვის უნდა განხორციელდეს - `-enable_short_tags` პარამეტრით. და თუ მაინც `php.ini-dist`-ში ჩუმათობის პრინციპით იგი ჩართულია, მაშინაც მოკლე ტეგის გამოყენება რეკომენდებული არ არის.

მეოთხე ვარიანტის გამოყენება შეიძლება, თუ ASP სტილის ტეგები იყო ჩართული, რისთვისაც კონფიგურაციული `asp_tags` იყო გამოყენებული.

*შენიშვნა. 1. ASP სტილის ტეგების მხარდაჭერა პირველად PHP 3.0.4 ვერსიაში დაემატა;*

*2. გასავრცელებელ ან ჩვენი კონტროლის ქვეშ არ არსებულ PHP-სერვერებზე განსათავსებელი დანართებისა და ბიბლიოთეკების დამუშავების დროს მოკლე ტეგების გამოყენებას უნდა მოვერიდოთ, რადგან მოკლე ტეგები შეიძლება სხვა სერვერებზე არ იყოს მხარდაჭერილი.*

PHP-კოდის ბლოკში დახურვის ტეგის შემდეგ აუცილებლად მოდის სტრიქონზე გადასვლის ბრძანება. ამასთან, დახურვის ტეგი ავტომატურად გულისხმობს წერტილ-მძიმეს. თქვენ არ გჭირდებათ ბლოკში კოდის ბოლო სტრიქონი წერტილ-მძიმით დაამთავროთ. PHP-ბლოკში ფაილის ბოლოს დახურვის ტეგი აუცილებელი არ არის.

PHP შემდეგი სტრუქტურის გამოყენების საშუალებას იძლევა:

```
<?php
if ($expression) {
    ?>
    <strong>ეს ჭეშმარიტია.</strong>
    <?php
} else {
    ?>
    <strong>ეს მცდარია.</strong>
    <?php
}
?>
```

ეს კოდი მუშაობს ისე, როგორც მოსალოდნელი იყო, იმიტომ, რომ, როდესაც PHP დახურვის ტეგს - ?> ხვდება, მას უბრალოდ

გამოჰყავს ყველაფერი რაც მომდევნო საწყის ტეგამდე შეხვდება. მოყვანილი მაგალითი ტექსტის დიდი ბლოკების გამოტანის დროს უფრო ეფექტურია, ვიდრე ტექსტის echo(), print() ან მსგავსი სხვა ოპერატორით გაგზავნის შემთხვევაში.

## კომენტარი PHP სკრიპტებში

პრაქტიკულად თითქმის შეუძლებელია პროგრამის ჩაწერა კომენტარის გარეშე.

PHP მხარს უჭერს კომენტარებს 'C', 'C++' და Unix გარსის სტილში.

PHP-ში კომენტარი შეიძლება იყოს სამი ტიპის:

```
<?php
    echo "ეს ტესტია;"; // ეს არის ერთსტრიქონიანი კომენტარი C++
სტილში
    /* არის მრავალსტრიქონიანი კომენტარი
    კომენტარის კიდევ ერთი სტრიქონი */
    echo " ეს კიდევ ერთი ტესტია;";
    echo " ბოლო ტესტი."; # ეს არის კომენტარი Unix გარსის
სტილში
?>
```

ეს ტესტია; ეს კიდევ ერთი ტესტია; ბოლო ტესტი.

ერთსტრიქონიანი კომენტარი სტრიქონის ან PHP-კოდის მომდევნო ბლოკის ბოლომდე გრძელდება.

```
<h1>ეს <?php # echo "მარტივი";?> მაგალითია.</h1>
<p>ზემოთ სათაურში გამოვა 'ეს მაგალითია'.
```

## ეს მაგალითია.

ზემოთ სათაურში გამოვა 'ეს მაგალითია'.

ყურადღებით უნდა იყოთ, რომ არ წარმოიშვას ერთმანეთში ჩადგმული 'C' ტიპის კომენტარები. მათი გამოყენება შეიძლება დიდი ბლოკების კომენტირების დროს.

```
<?php
/*
    echo "ეს ტესტია"; /* ეს კომენტარი პრობლემას გამოიწვევს */
*/
?>
```

ზემოთ მოყვანილია ერთმანეთში ჩადგმული კომენტარების მაგალითი, რომელმაც შეცდომა გამოიწვია.

```
Parse error: syntax error, unexpected '*' in
C:\xampp\htdocs\Mag9.php on line 4
```

თუ მეორე კომენტარში /\*...\*/ სიმბოლოების მაგივრად გამოყენებული იქნებოდა // სიმბოლოები, მაშინ შეცდომა არ წარმოიშობოდა.

## PHP-info

PHP-ში კონფიგურაციის შესახებ ინფორმაციის მისაღებად სპეციალური საშუალებაა გათვალისწინებული. ეს არის ფუნქცია:

`phpinfo ()`

ეს ფუნქცია შესრულების შემდეგ ბრაუზერში სპეციალურ ცხრილს გამოიტანს, სადაც PHP-ის ინსტალაციის შედეგად მინიჭებული კონფიგურაციის მაჩვენებლები იქნება მოცემული. კერძოდ, ინფორმაცია მიერთებული გაფართოების, სერვერის, გამოყენებული მონაცემთა ბაზის და სხვათა შესახებ. ამ ინფორმაციის მისაღებად შემდეგი სკრიპტის შესრულებაა საკმარისი:

```
<?php  
phpinfo ();  
?>
```

ბრაუზერი შემდეგი სახის ცხრილს გამოიტანს:

phpinfo() - Windows Internet Explorer

http://localhost/mag10.php

Почук@Mail.Ru

File Edit View Favorites Tools Help

phpinfo()

## PHP Version 5.3.8

<b>System</b>	Windows NT IRAKLI2-PC 6.1 build 7601 (Windows 7 Ultimate Edition Service Pack 1) i586
<b>Build Date</b>	Aug 23 2011 11:47:20
<b>Compiler</b>	MSVC9 (Visual C++ 2008)
<b>Architecture</b>	x86
<b>Configure Command</b>	cs script /nologo configure.js "--enable-snapshot-build"--disable-isapi"--enable-debug-pack"--disable-isapi"--without-mssql"--without-pdo-mssql"--without-pi3web"--with-pdo-oci=D:\php-sdk\oracle\instantclient10\sdk\shared"--with-oci8=D:\php-sdk\oracle\instantclient10\sdk\shared"--with-oci8-11g=D:\php-sdk\oracle\instantclient11\sdk\shared"--enable-object-out-dir=.\obj"--enable-com-dotnet"--with-mcrypt-static"--disable-static-analyze"
<b>Server API</b>	Apache 2.0 Handler
<b>Virtual Directory Support</b>	enabled
<b>Configuration File (php.ini) Path</b>	C:\Windows
<b>Loaded Configuration File</b>	C:\xampp\php\php.ini
<b>Scan this dir for additional .ini files</b>	(none)
<b>Additional .ini files parsed</b>	(none)
<b>PHP API</b>	20090626
<b>PHP Extension</b>	20090626
<b>Zend Extension</b>	220090626
<b>Zend Extension Build</b>	API220090626,TS,VC9
<b>PHP Extension Build</b>	API20090626,TS,VC9
<b>Debug Build</b>	no
<b>Thread Safety</b>	enabled
<b>Zend Memory Manager</b>	enabled
<b>Zend Multibyte Support</b>	disabled
<b>IPv6 Support</b>	enabled
<b>Registered PHP</b>	php file.php data http:// #.zip compress zip compress bin2.php

Done

Local intranet | Protected Mode: Off

100%

## ცვლადები PHP-ში

თითქმის დაპროგრამების ყველა ენაში არსებობს ცვლადის ცნება.

PHP-ზე დაპროგრამების დროს შეიძლება არ ვიძუნწოთ და რამდენიც გვსურს იმდენი ახალი ცვლადი შემოვიტანოთ. მეხსიერების ეკონომიის პრინციპს, რომელიც რამდენიმე წლის წინ ძალზე აქტუალური იყო, დღეს არ ვითვალისწინებთ, მაგრამ დიდი მოცულობის ცვლადების შენახვის დროს უმჯობესია გამოუყენებელი ცვლადები წაშალოთ, რისთვისაც Unset ოპერატორი უნდა გამოვიყენოთ.

საერთოდ, ცვლადი - ეს არის ოპერატიული მეხსიერების არე, რომელზეც წვდომა სახელით ხორციელდება. ყველა მონაცემი, რომელთანაც პროგრამა მუშაობს, ცვლადების სახით ინახება (გამონაკლისია - მუდმივა, რომელიც შეიძლება მხოლოდ რიცხვს ან სტრიქონს შეიცავდეს). მაჩვენებლის ცნება, როგორც C-შია, PHP-ში არ არსებობს - მინიჭების შემთხვევაში, როგორი რთული სტრუქტურაც არ უნდა ჰქონდეს ცვლადს, მისი ერთი ერთზე კოპირება ხდება. მიუხედავად ამისა, PHP-ში, მე-4 ვერსიიდან დაწყებული, მყარი და სიმბოლური ბმულის ცნება არსებობს, რომელსაც ქვემოთ განვიხილავთ.

ცვლადის სახელი ყოველგვარი შეზღუდვის გარეშე შეიძლება შეირჩეს. სახელი შეიძლება შეიცავდეს ასოებს, რიცხვებს და ქვედა ხაზს. ცვლადის სახელში არ შეიძლება ინტერვალის (ჰარი) გამოყენება. PHP-ში ყველა ცვლადის სახელი დოლარის - \$ ნიშნით უნდა დაიწყოს, რომელსაც მოსდევს ასო ან ქვედა ხაზი - ეს ინტერპრეტატორისათვის უფრო მარტივია, რათა ისინი გაარჩიოს. ცვლადების სახელები აგრეთვე, ასოების რეგისტრის მიმართაც მგრძნობიარეა: მაგალითად, \$var, \$Var და \$VAR სრულიად სხვადასხვა ცვლადები იქნება.

ცვლადისათვის მნიშვნელობის მინიჭება სპეციალური ოპერატორის (მინიჭების ოპერატორი) საშუალებით ხორციელდება, რომელიც ტოლობის (=) ნიშნით აღინიშნება. ამ ნიშნის მარცხნივ ცვლადის სახელი მიეთითება, ხოლო მარჯვნივ - მისანიჭებელი მნიშვნელობა. მინიჭების ოპერატორი, როგორც ყველა ოპერატორი PHP-ში წერტილ-მძიმით მთავრდება. თუ მისანიჭებელი მნიშვნელობა ტექსტურ სტრიქონს წარმოადგენს, მაშინ იგი ბრჭყალებში (") ან აპოსტროფებში (') უნდა მოვათავსოთ. თუ ცვლადს რიცხვით მნიშვნელობას ვანიჭებთ, მაშინ მას არავითარი შეზღუდვა არ ესაჭიროება, ხოლო თუ რიცხვის როგორც ტექსტური ინფორმაციის დამუშავება გვჭირდება, მაშინ ისიც ბრჭყალებში ან აპოსტროფებში უნდა მოვათავსოთ. ამასთან, უნდა გვახსოვდეს, რომ PHP მონაცემთა ტიპებს ძალიან მარტივად მიმართავს და აუცილებლობის შემთხვევაში ტექსტურ სტრიქონს შესაბამისი ფორმატის რიცხვად გარდაქმნის.

ცვლადის მნიშვნელობის ბრაუზერის ეკრანზე გამოტანა ჩვენთვის უკვე ნაცნობი echo ოპერატორის საშუალებით ხდება.

თუ ცვლადს მიმდევრობით ორ სხვადასხვა მნიშვნელობას მივანიჭებთ, მაშინ მეორე მინიჭება პირველს მოსპობს და ცვლადის მეორე მნიშვნელობა შეინახება. სხვა სიტყვებით რომ ვთქვათ, ცვლადში შეტანილი ახალი მნიშვნელობა მის ძველ მნიშვნელობას შეცვლის.

```
<?php  
$var = "Anna";  
$Var = "Mari";  
echo "$var, $Var"; // გამოიტანს "Anna, Mari"  
$4city = 'Tbilisi'; // არასწორია; იწყება ციფრით  
$_4city = 'Tbilisi'; // სწორია; იწყება ქვედა ხაზით  
$code = 995; //სწორია; მიანიჭებს რიცხვით მნიშვნელობას
```



```
$newyear = "2015"; //სწორია; მიანიჭებს ტექსტურ მნიშვნელობას
echo "<br>";
echo "$_4city";
echo "$code";
echo "<br>";
echo "$_4city $code";
echo "<br>";
echo "$newyear";
?>
```

ამ პროგრამის შესრულების შედეგს შემდეგი სახე ექნება:

```
Anna, Mari
Tbilisi995
Tbilisi 995
2015
```

PHP-ში ცვლადების აღწერა, მათი ტიპის მითითება, არ არის საჭირო. ამას ინტერპრეტატორი თვითონ აკეთებს. მაგრამ, ზოგჯერ შეიძლება ისიც შეცდეს (მაგალითად, თუ ტექსტურ სტრიქონში სინამდვილეში ათობითი რიცხვია მოცემული), ამიტომ ხანდახან ტიპის მითითების აუცილებლობა წარმოიშობა.

## მონაცემთა ტიპები PHP-ში

ზოგჯერ, პირდაპირ პროგრამის შესრულების პროცესში საჭირო ხდება გავიგოთ ცვლადის ტიპი (მაგალითად, ფუნქციის პარამეტრებში გადაცემული ცვლადის).

ახლა გავიგოთ რა ტიპის მონაცემები (ცვლადები) შეიძლება შეგვხვდეს PHP-ში.

PHP მხარს უჭერს რვა მარტივი ტიპის მონაცემს (ცვლადს):  
ოთხი სკალარული ტიპის:

- boolean (ორობითი მონაცემები);
- integer (მთელი რიცხვები);
- float (მცოცავმძიმძიანი რიცხვები ანუ 'double');
- string (სტრიქონი).

ორი შერეული ტიპის:

- array (მასივები);
- object (ობიექტები).

ორი სპეციალური ტიპის:

- resource (რესურსები);
- NULL (ცარიელი ტიპი).

არსებობს აგრეთვე, რამდენიმე ფსევდოტიპი:

- mixed (შერეული ტიპის);
- number (რიცხვითი);
- callback (უკუგამომახების).

განვიხილოთ ზემოთ ჩამოთვლილი PHP-ის მონაცემთა ტიპები.

## **ტიპი Boolean (ორობითი მონაცემები)**

ეს არის უმარტივესი ტიპი. იგი გამოსახულების ჭეშმარიტებას გამოხატავს და შეიძლება იყოს ან TRUE ან FALSE. ბულის ტიპის გამოსახულება პირველად PHP 4-ში იყო შემოტანილი.

იმისათვის, რომ განსაზღვრული იყოს ბულის ტიპი, გამოიყენება საგასაღებო სიტყვა TRUE ან FALSE. არც ერთი არ არის რეგისტრზე დამოკიდებული.

```
<?php
$x = True; // $x-ს მიენიჭოს მნიშვნელობა TRUE
?>
```

ჩვეულებრივ, რაიმე ოპერატორს ვიყენებთ, რომელიც ლოგიკურ გამოსახულებას დააბრუნებს, ხოლო შემდეგ მმართველ კონსტრუქციას მიანიჭებს მას.

```
<?php
// == ეს ოპერატორია, რომელიც ეკვივალენტობას ამოწმებს
// და აბრუნებს ბულის მნიშვნელობას
if ($action == "აჩვენე_ვერსია") {
    echo "ვერსია 1.23";
}
// ეს არ არის აუცილებელი...
if ($show_separators == TRUE) {
    echo "<hr>\n";
}
// ...იმიტომ, რომ შეიძლება იგი მარტივად ჩაწერო
if ($show_separators) {
    echo "<hr>\n";
}
?>
```

მნიშვნელობის ბულის ტიპად გარდაქმნის მიზნით ტიპის მოცემა (bool) ან (boolean) გამოიყენება. მაგრამ უმეტეს შემთხვევაში ტიპის მოცემის გამოყენება არ არის აუცილებელი, ვინაიდან, თუ ოპერატორს, ფუნქციას ან მმართველ კონსტრუქციას ბულის არგუმენტი ესაჭიროება, მნიშვნელობის გარდაქმნა ავტომატურად მოხდება.

ლოგიკურ ტიპად გარდაქმნის დროს, შემდეგი მნიშვნელობები განიხილება როგორც FALSE:

- თვით ბულის FALSE;
- მთელი 0 (ნული);
- მცოცავმძიმის რიცხვი 0.0 (ნული);
- ცარიელი სტრიქონი და სტრიქონი "0";
- ნულელებმენტის მასივი;
- ნულცვლადიანწვერების ობიექტი;
- NULL სპეციალური ტიპი.

ყველა დანარჩენი მნიშვნელობა განიხილება, როგორც TRUE (ნებისმიერი რესურსის ჩათვლით). -1 ითვლება TRUE, როგორც ნებისმიერი არანულოვანი (უარყოფითი ან დადებითი) რიცხვი.

```
<?php
var_dump((bool) "");           // bool(false)
var_dump((bool) 1);           // bool(true)
var_dump((bool) -2);          // bool(true)
var_dump((bool) "foo");       // bool(true)
var_dump((bool) 2.3e5);        // bool(true)
var_dump((bool) array(12));    // bool(true)
var_dump((bool) array());      // bool(false)
var_dump((bool) "false");     // bool(true)
?>
```

### ტიპი integer (მთელი რიცხვი)

მთელი, ეს არის რიცხვი სიმრავლიდან  $Z = \{\dots, -2, -1, 0, 1, 2, \dots\}$ , ჩვეულებრივ, ის არის 32 ბიტი სიგრძის (-2 147 483 648-დან 2 147 483 647-მდე) რიცხვი.

მთელი რიცხვები შეიძლება მოცემული იყოს ათვლის ათობით, თექვსმეტობით ან რვაობით სისტემაში და შეიძლება ჰქონდეს ნიშანი (- ან +).

თუ ვიყენებთ ათვლის რვაობით სისტემას, მაშინ რიცხვი უნდა იწყებოდეს 0-ით (ნული), ხოლო თუ ვიყენებთ ათვლის თექვსმეტობით სისტემას, მაშინ რიცხვი უნდა იწყებოდეს 0x-ით.

```
<?php
$a = 1234; // ათობითი რიცხვი
$a = -123; // უარყოფითი რიცხვი
$a = 0123; // რვაობითი რიცხვი (ეს ეკვივალენტურია 83 ათობით
სისტემაში)
$a = 0x1A; // თექვსმეტობითი რიცხვი (ეს
ეკვივალენტურია 26 ათობით სისტემაში)
?>
```

### ტიპი float (მცოცავმძიმანი რიცხვი)

**Double** - ძალიან დიდი სიზუსტის ნამდვილი რიცხვი (ამ რიცხვების გამოყენება შეიძლება მათემატიკური გამოთვლების უმეტესობაში).

მცოცავმძიმანი რიცხვები (იგივე ორმაგი სიზუსტის რიცხვები ან ნამდვილი რიცხვები) შეიძლება ნებისმიერი შემდეგი სინტაქსით იყოს განსაზღვრული:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

## ტიპი string (სტრიქონი)

სტრიქონი PHP-ში არის სიმბოლოების ნებისმიერი სიგრძის ნაკრები. C ენისაგან განსხვავებით სტრიქონი ნულოვან სიმბოლოებს შეიძლება შეიცავდეს, რაც პროგრამაზე არავითარ გავლენას არ ახდენს. სხვა სიტყვებით რომ ვთქვათ, სტრიქონი შეიძლება ბინარული მონაცემების შესანახად გამოვიყენოთ. სტრიქონის სიგრძე მხოლოდ თავისუფალი ოპერატიული მექანიზმებითაა შეზღუდული.

სტრიქონის დამუშავება სტანდარტული ფუნქციების საშუალებით ძალზე მარტივად შეიძლება განხორციელდეს. შესაძლებელია აგრეთვე უშუალოდ მივმართოთ მის ნებისმიერ სიმბოლოს.

ქვემოთ მოცემულია სტრიქონული ცვლადის მაგალითი:

```
<? php
$a = "ეს უბრალოდ სტრიქონულ ცვლადში ჩაწერილი ტექსტია";
echo $a; //გამოიტანს 'ეს უბრალოდ სტრიქონულ ცვლადში
ჩაწერილი ტექსტია'
?>
```

## აპოსტროფებით სტრიქონის განსაზღვრა

სტრიქონის განსაზღვრის უმარტივესი ფორმა არის მისი მოვათავსება აპოსტროფებში (სიმბოლო ' ).

თუ სტრიქონის შიგნით საჭიროა აპოსტროფების გამოყენება, მაშინ, როგორც სხვა მრავალ ენაში, აუცილებელია ამ ნიშნის წინ საპირისპიროდ დახრილი ხაზი (\) გამოვიყენოთ ანუ მოვახდინოთ მისი ეკრანირება. თუ ეს დახრილი ხაზი უნდა მოდიოდეს აპოსტროფის წინ ან სტრიქონის ბოლოს, მაშინ აუცილებელია მისი დუბლირება. უნდა მივაქციოთ ყურადღება იმას, რომ, თუ

რომელიმე სხვა სიმბოლოს ეკრანირებას მოვინდომებთ, მაშინ საპირისპიროდ დახრილი ხაზის გამოტანაც მოხდება.

აპოსტროფებში მოთავსებულ სტრიქონში ცვლადებისა და სპეციალური სიმბოლოების ეკრანირების მიმდევრობის დამუშავება არ მოხდება. ქვემოთ მოყვანილია აპოსტროფების გამოყენების მაგალითები:

```
<?php
echo 'ეს მარტივი სტრიქონია';

echo 'ასევე შეგიძლიათ სტრიქონში ჩასვათ
ახალი სტრიქონის სიმბოლო,
ვინაიდან ეს ნორმალურია';

// გამოიტანს: ერთხელ ზურამ თქვა: "I'll be back"
echo 'ერთხელ ზურამ თქვა: "I\'ll be back"';

// გამოიტანს: თქვენ წაშალეთ C:\*.*?
echo 'თქვენ წაშალეთ C:\\*.*?';

// გამოიტანს: თქვენ წაშალეთ C:\*.*?
echo 'თქვენ წაშალეთ C:/*.*?';

// გამოიტანს: ეს არ ჩასვამს: \n ახალ სტრიქონს
echo 'ეს არ ჩასვამს: \n ახალ სტრიქონს';

// გამოიტანს: ცვლადის $expand ასევე $either მნიშვნელობა არ
გამოდის
echo 'ცვლადის $expand ასევე $either მნიშვნელობა არ გამოდის ';
?>
```

**ბრჭყალებით სტრიქონის განსაზღვრა:**

თუ სტრიქონი ბრჭყალებშია (") მოთავსებული, მაშინ PHP სპეციალური სიმბოლოების უფრო მეტი რაოდენობის მმართველ ბრძანებას განასხვავებს:

ქვემოთ მოყვანილია მმართველი ბრძანებების ცხრილი:

სპეციალურ სიმბოლოთა მიმდევრობა	მნიშვნელობა
\n	ახალი სტრიქონი
\r	კურსორის დაბრუნება სტრიქონის დასაწყისში
\t	ჰორიზონტალური ტაბულაცია
\\	საწინააღმდეგოდ დახრილი ხაზი
\\$	დოლარის ნიშანი
\"	ორმაგი ბრჭყალი
\[0-7]{1,3}	რეგულარული გამოსახვის შესაბამისი სიმბოლოების მიმდევრობა, სიმბოლო ათვლის რვაობით სისტემაში
\x[0-9A-F a-f]{1,2}	რეგულარული გამოსახვის შესაბამისი სიმბოლოების მიმდევრობა, სიმბოლო ათვლის თექვსმეტობით სისტემაში

*შენიშვნა: კიდევ ერთხელ შეგახსენებთ, რომ თუ მოვინდომებთ რომელიმე სხვა სიმბოლოს მნემონიზაციას, მაშინ საპირისპიროდ დახრილი ხაზიც დაიბეჭდება!*

ბრჭყალებში მოთავსებული სტრიქონის ყველაზე მნიშვნელოვან თვისებას მონაცემთა დამუშავება წარმოადგენს.



## Herdoc-სინტაქსით სტრიქონის განსაზღვრა:

სტრიქონის განსაზღვრის სხვა საშუალება არის herdoc-სინტაქსის ("`<<<`") გამოყენება. `<<<` ნიშანის შემდეგ უნდა მივუთითოთ რაიმე იდენტიფიკატორი, შემდეგ ახალ ხაზზე მოდის სტრიქონის მნიშვნელობა, ხოლო შემდეგ ახალ ხაზზე იგივე იდენტიფიკატორი, რომელიც ჩანართს ხურავს. დამხურავი იდენტიფიკატორი სტრიქონის პირველივე სვეტიდან უნდა იწყებოდეს.

დამხურავი იდენტიფიკატორი დასახელების იმავე წესებს უნდა შეესაბამებოდეს, რომელსაც შეესაბამება PHP-ში არსებული ყველა სხვა ჭდე: უნდა შეიცავდეს მხოლოდ ასოებს, ციფრებს და ქვედა ხაზს და უნდა იწყებოდეს ასოთი ან ქვედა ხაზით.

მნიშვნელოვანია აღინიშნოს, რომ სტრიქონი დამხურავი იდენტიფიკატორით სხვა სიმბოლოებს, გარდა წერტილ-მძიმისა (;), არ უნდა შეიცავდეს. ეს ნიშნავს, რომ იდენტიფიკატორი არ უნდა შევიტანოთ არც აბზაცის, არც ტაბულაციის ღილაკის, არც რაიმე ინტერვალების გამოყენებით, არც წერტილ-მძიმის წინ და არც შემდეგ. ასევე უნდა აღინიშნოს, რომ დამხურავი იდენტიფიკატორის წინ პირველი სიმბოლო უნდა იყოს მოცემულ ოპერაციულ სისტემაში გამოყენებული ახალი სტრიქონის სიმბოლო. მაგალითად, Windows-ში ეს არის `\r`.

თუ ეს წესი დარღვეულია და დამხურავი იდენტიფიკატორი არ არის „სუფთა“, იგულისხმება, რომ დამხურავი იდენტიფიკატორი არ არის და PHP განაგრძობს მის ძებნას, რაც გამოიწვევს შეცდომას.

მასში ისევე შეიძლება ზემოთ ჩამოთვლილი მმართველი ბრძანებების გამოყენება, როგორც ბრჭყალებში მოთავსებულ სტრიქონში.

ქვემოთ მოყვანილია heredoc-სტრიქონის განსაზღვრის მაგალითები:

```
<?php
$str = <<<EOD
სტრიქონის მაგალითი,
რომელიც რამდენიმე სტრიქონს მოიცავს,
heredoc-სინტაქსის გამოყენებით.
EOD;

/* უფრო რთული მაგალითი ცვლადებით. */
class foo
{
    var $foo;
    var $bar;

    function foo()
    {
        $this->foo = 'Foo';
        $this->bar = array('Bar1', 'Bar2', 'Bar3');
    }
}

$foo = new foo();
$name = 'ჩემი სახელი';

echo <<<EOT
მე მქვია "$name". მე ვბეჭდავ $foo->foo-ს.
ახლა მე ვბეჭდავ {$foo->bar[1]}-ს.
ამან უნდა გამოიტანოს ასომთავრულით 'A': \x41
```

```
EOT;  
?>
```

## მონაცემთა გაერთიანება (კონკატენაცია)

დაპროგრამების სხვადასხვა ენაში სტრიქონების გასაერთიანებლად (კონკატენაციისათვის) სხვადასხვა ოპერატორი გამოიყენება. მაგალითად, Pascal-ში გამოიყენება ოპერატორი "+". PHP-ში "+" ოპერატორის გამოყენება არაკორექტული იქნება, რადგან თუ სტრიქონი რიცხვებს შეიცავს, მაშინ სტრიქონების გაერთიანების მაგივრად ორი რიცხვის შეკრება შესრულდება.

PHP-ში კონკატენაციის (გაერთიანება - ზოგჯერ მას შერწყმის ოპერაციასაც უწოდებენ) შესასრულებლად ორი ოპერატორი გამოიყენება.

პირველი - კონკატენაციის ოპერატორი წერტილით აღინიშნება ('.'), რომელიც შედეგად მარცხენა და მარჯვენა არგუმენტის გაერთიანებას აბრუნებს.

მეორე - მინიჭების ოპერატორი კონკატენაციით, რომელიც მარცხენა არგუმენტს მიუერთებს მარჯვენა არგუმენტს.

მაგალითად,

```
<?php  
$name = "ჩემი სახელია ";  
$name .= "ნიკოლოზი!";  
echo "$name"; // გამოიტანს "ჩემი სახელია ნიკოლოზი!"  
?>
```

ჩემი სახელია ნიკოლოზი!

გაერთიანების ოპერაციის მიმდევრობათა საშუალებით, არსებულ ცვლადს დამატებით რამდენიმე ფრაგმენტი მივუერთოთ. უნდა გვახსოვდეს, რომ ყოველი მიერთებული ფრაგმენტი ინტერვალით დავიწყოთ ან დავამთავროთ, რათა გაერთიანების შემთხვევაში სიტყვები არ გადაეხას ერთმანეთს. მაგალითად,

```
<?php
$name = "ჩემი სახელია ";
echo "$name"; // გამოიტანს "ჩემი სახელია"
echo "<br>";
$name .= "ანა! ";
echo "$name"; // გამოიტანს "ჩემი სახელია ანა!"
echo "<br>"
$name .= "მე 15 წლის ვარ. ";
echo "$name"; // გამოიტანს "ჩემი სახელია ანა! მე 15 წლის ვარ."
?>
```

ჩემი სახელია  
ჩემი სახელია ანა!  
ჩემი სახელია ანა! მე 15 წლის ვარ.

ზემოთ განხილული შესაძლებლობის გარდა, გაერთიანების ოპერაცია შეიძლება ორი სტრიქონის გასაერთიანებლად

გამოვიყენოთ ისე, რომ შედეგი სხვა ცვლადს მივანიჭოთ. მაგალითად,

```
<?php
$name1 = "ჩემი სახელია ";
$name2 = $name1 . "ნიკოლოზი!";
echo "$name2"; // გამოიტანს "ჩემი სახელია ნიკოლოზი!"
?>
```

შესაძლებელია არა მხოლოდ ცვლადისა და კონსტანტის გაერთიანება, არამედ რამდენიმე კონსტანტისა და რამდენიმე ცვლადის გაერთიანება. ამასთან, ერთი ცვლადი გაერთიანებაში შეიძლება რამდენიმეჯერ მონაწილეობდეს. მაგალითად,

```
<?php
$name1 = "ჩემი სახელია ";
echo "$name1"; // გამოიტანს "ჩემი სახელია"
echo "<br>";
$name2 = $name1 . "ანა! ";
echo "$name2"; // გამოიტანს "ჩემი სახელია ანა!"
echo "<br>";
$name3 = $name2 . "მე 15 წლის ვარ. ";
echo "$name3"; // გამოიტანს "ჩემი სახელია ანა! მე 15 წლის ვარ."
$name4 = "<hr> ".$name3 . " კიდევ ერთხელ. " . $name3;
echo "$name4";
?>
```

ჩემი სახელია  
ჩემი სახელია ანა!  
ჩემი სახელია ანა! მე 15 წლის ვარ.

ჩემი სახელია ანა! მე 15 წლის ვარ. კიდევ ერთხელ. ჩემი სახელია ანა! მე 15 წლის ვარ.

## ცვლადის არსებობის შემოწმება

ზოგჯერ აუცილებელია შემოწმდეს არსებობს თუ არა რომელიმე ცვლადი, ანუ მოხდა თუ არა მისი ინიციალიზაცია (მოხდა თუ არა ამ ცვლადისათვის რაიმე მნიშვნელობის მინიჭება). ამ ამოცანის გადასაწყვეტად PHP-ში სპეციალური ფუნქცია `isset ()` გამოიყენება. ისეთი ფუნქცია, როგორცაა

```
isset ($var1)
```

შედეგად აბრუნებს მნიშვნელობას `true` თუ `$var1` ცვლადი ინიციალიზებულია (არსებობს). წინააღმდეგ შემთხვევაში შედეგად აბრუნებს `false`. ასეთივე მნიშვნელობას `false` დააბრუნებს იგი თუ `$var1` ცვლადი სპეციალური ფუნქციის `unset ()` მეშვეობით არის განადგურებული. `$var1` ცვლადისათვის მისი წაშლა შეიძლება `unset` ოპერატორის საშუალებით განხორციელდეს:

```
unset ($var1)
```

შედეგად ეს ცვლადი წყვეტს არსებობას და ინტერპრეტატორი ივიწყებს მას. ცვლადის განადგურება და შიგთავსის გასუფთავება ერთმანეთში არ უნდა ავურიოთ. გასუფთავება - ცვლადში მონაცემების შეტანის კერძო შემთხვევაა (მაგალითად, როდესაც ტექსტურ ცვლადში ცარიელი სტრიქონი " " შეაქვთ). გასუფთავებულ მონაცემს ინტერპრეტატორის მეხსიერებაში თავისი ადგილი კვლავ უჭირავს, ხოლო წაშლის შემთხვევაში ცვლადის მიერ დაკავებული მეხსიერება თავისუფლდება.

იმისათვის, რომ შემოწმდეს მონაცემი შეიცავს თუ არა ცარიელ მნიშვნელობას, უნდა გამოვიყენოთ ფუნქცია `empty ()`. ამ ფუნქციის

`empty ($var1)`

შესრულების შედეგი ნებისმიერ შემთხვევაში იქნება `false`, გარდა სიტუაციის, როდესაც `$var1` ცვლადის მნიშვნელობა `0`-ის ტოლია ან ცვლადი ცარიელი სტრიქონია " ".

### **ტიპი array (მასივი)**

PHP-ში მასივი არის მონაცემთა მოწესრიგებული ნაკრები, რომელშიც მნიშვნელობასა და ინდექსს შორის შესაბამისობაა დამყარებული და რომელიც ერთნაირი ტიპის ელემენტების სიას წარმოადგენს.

ინდექსი მასივში ელემენტის ცალსახა იდენტიფიცირებისათვის გამოიყენება. ერთ მასივში არ შეიძლება იყოს ორი ელემენტი ერთი და იმავე ინდექსით.

PHP ნებისმიერი სირთულის მასივის შექმნის საშუალებას იძლევა.

ელემენტთა იდენტიფიკაციის საშუალების მიხედვით ორი ტიპის მასივს განასხვავებენ.

1. მარტივი მასივი - ეს ისეთი მასივია, რომელშიც ელემენტი მიმდევრობაში ინდექსით განისაზღვრება;
2. ასოციური მასივი - ასეთ მასივს ასოცირებული ბუნება აქვს და ელემენტზე მიმართვისათვის გამოიყენება ე. წ. გასაღები, რომელიც მის მნიშვნელობასთან ლოგიკურადაა დაკავშირებული.

PHP-ის მნიშვნელოვანი თავისებურება ის არის, რომ PHP, სხვა ენებისაგან განსხვავებით, უშუალოდ პროგრამის ტანში

(სკრიპტებში) ნებისმიერი სირთულის მასივის შექმნის საშუალებას იძლევა.

მასივი შეიძლება იყოს როგორც ერთგანზომილებიანი, ისე მრავალგანზომილებიანი.

## მარტივი მასივები და სიები PHP-ში

მარტივი ინდექსირებული მასივის ელემენტებზე მიმართვის დროს გამოიყენება მთელრიცხვიანი ინდექსი, რომელიც ელემენტის პოზიციას განსაზღვრავს.

### ერთგანზომილებიანი მარტივი მასივი:

ერთგანზომილებიანი მარტივი მასივის ელემენტის ზოგადი სინტაქსი ასეთია:

```
$სახელი[ინდექსი];
```

მასივი, რომლის ინდექსი ნულთ დაწყებული რიცხვია - არის სია.

მარტივი მასივის (სიის) ელემენტებზე წვდომა შემდეგნაირად ხდება:

```
<?php
// მასივის ინიციალიზაციის მარტივი საშუალება
$names[0]="ფორთოხალი";
$names[1]="ბანანი";
$names[2]="მსხალი";
$names[3]="ვაშლი";
// აქ: names - მასივის სახელია, ხოლო 0, 1, 2, 3 - მასივის ინდექსები
// მასივის ელემენტები გამოგვაქვს ბრაუზერში:
echo $names[0]; // names მასივის ელემენტის გამოტანა ინდექსით 0
echo "<br>";
echo $names[3]; // names მასივის ელემენტის გამოტანა ინდექსით 3
```



```
// გამოიტანს:  
// ფორთოხალი  
// ვაშლი  
?>
```

ტექნიკური თვალსაზრისით მარტივ მასივსა და სიას შორის განსხვავება არ არის.

მარტივი მასივი შეიძლება შეიქმნას ისე, რომ მასივის ახალი ელემენტის ინდექსი არ იყოს მითითებული, ამას თქვენ მაგვირად PHP გააკეთებს. მაგალითად:

```
<?php  
// მასივის ინიციალიზაციის მარტივი საშუალება  
$names[]="ფორთოხალი";  
$names[]="ბანანი";  
$names[]="მსხალი";  
$names[]="ვაშლი";  
// აქ: names - მასივის სახელია, ხოლო 0, 1, 2, 3 - მასივის ინდექსები  
// მასივის ელემენტები გამოგვაქვს ბრაუზერში:  
echo $names[0]; // names მასივის ელემენტის გამოტანა ინდექსით 0  
echo "<br>";  
echo $names[3]; // names მასივის ელემენტის გამოტანა ინდექსით 3  
// გამოიტანს:  
// ფორთოხალი  
// ვაშლი  
?>
```

განხილულ მაგალითში names მასივის ელემენტის დამატება მარტივად შეიძლება, ანუ შეგიძლიათ მასივის ელემენტის ინდექსი არ მიუთითოთ:

```
$names[]="ატამი";
```

მარტივი მასივის (სიის) ახალი ელემენტი მასივის ბოლოს დაემატება. მასივის ელემენტის ყოველი შემდგომი დამატებისას ინდექსის მნიშვნელობა ერთით გაიზრდება.

### მრავალგანზომილებიანი მარტივი მასივი:

მრავალგანზომილებიანი მარტივი მასივის ელემენტის ზოგადი სახეა:

\$სახელი[ინდექსი1][ინდექსი2]...[ინდექსიN];

მრავალგანზომილებიანი მარტივი მასივის მაგალითი:

```
<?php
// მრავალგანზომილებიანი მარტივი მასივი:
$arr[0][0]="ბოსტნეული";
$arr[0][1]="ხილი";
$arr[1][0]="გარგალი";
$arr[1][1]="ფორთოხალი";
$arr[1][2]="ბანანი";
$arr[2][0]="კიტრი";
$arr[2][1]="პომიდორი";
$arr[2][2]="გოგრა";
// მასივის ელემენტების გამოტანა:
echo "<h3>".$arr[0][0]."</h3>";
for ($q=0; $q<=2; $q++) {
echo $arr[2][$q]."<br>";
}
echo "<h3>".$arr[0][1]."</h3>";
for ($w=0; $w<=2; $w++) {
echo $arr[1][$w]."<br>";
```

```
}  
?>
```

ამ პროგრამის შესრულების შედეგი შემდეგნაირად გამოიყურება:

### **ბოსტნეული:**

კიტრი  
პომიდორი  
გოგრა

### **ხილი:**

გარგალი  
ფორთოხალი  
ბანანი

## **ასოციური მასივები PHP-ში**

PHP-ში მასივის ინდექსი შეიძლება იყოს არა მარტო რიცხვი, არამედ სტრიქონიც. ამასთან, სტრიქონს არავითარი შეზღუდვა არ ედება: ის შეიძლება ჰარებს და სპეციალურ სიმბოლოებს შეიცავდეს და სიგრძეც ნებისმიერი ჰქონდეს.

მასივს, რომლის ინდექსი სტრიქონია ასოციური მასივი ეწოდება. ასოციური მასივის ინდექსს გასაღები ეწოდება.

ასოციური მასივის გამოყენება მაშინ არის მოსახერხებელი, როდესაც მასივის ელემენტის სიტყვასთან დაკავშირება უფრო მარტივია, ვიდრე რიცხვთან.

## ერთგანზომილებიანი ასოციური მასივი:

ერთგანზომილებიანი ასოციური მასივი შეიცავს მხოლოდ ერთ გასაღებს (ელემენტს), რომელიც ასოციური მასივის კონკრეტულ ინდექსს შეესაბამება. მოვიყვანოთ მაგალითი:

```
<?php
// ასოციური მასივი
$names["ბერიძე"]="ლაშა";
$names["მიქელაძე"]="ნიკა";
$names["გელოვანი"]="კახა";
// მოცემულ მაგალითში: გვარი - ასოციური მასივის გასაღებია,
// ხოლო სახელები - მასივის ელემენტები
?>
```

ერთგანზომილებიანი ასოციური მასივის ელემენტზე მიმართვა ისევე ხორციელდება, როგორც ჩვეულებრივი მასივის ელემენტზე და მას მიმართვა გასაღებით ეწოდება:

```
echo $names["ბერიძე"];
```

## მრავალგანზომილებიანი ასოციური მასივი:

მრავალგანზომილებიანი ასოციური მასივი შეიძლება შეიცავდეს რამდენიმე გასაღებს (ელემენტს), რომელთაგან თითოეული ასოციური მასივის კონკრეტულ ინდექსს შეესაბამება. მოვიყვანოთ მრავალგანზომილებიანი ასოციური მასივის მაგალითი:

```
<?php
// მრავალგანზომილებიანი ასოციური მასივი
$A["ბერიძე"] = array("name"=>"ბერიძე
ლაშა", "age"=>"25", "email"=>"beridze@gmail.com");
```

```
$A["მიქელაძე"] = array("name"=>"მიქელაძე  
ნიკა", "age"=>"34", "email"=>"miqeladze@yahoo.com");  
$A["გელოვანი"] = array("name"=>"გელოვანი  
კახა", "age"=>"47", "email"=>"gelovani@gmail.com");  
?>
```

მრავალგანზომილებიანი ასოციური მასივები Pascal-ის ენაზე ჩანაწერს ან C ენის სტრუქტურას ჰგავს. მრავალგანზომილებიანი ასოციური მასივის ელემენტზე მიმართვა შემდეგნაირად ხდება:

```
echo $A["ბერიძე"]["name"]; // გამოიტანს ბერიძე ლაშა  
echo $A["მიქელაძე"]["email"]; // გამოიტანს  
miqeladze@yahoo.com
```

როგორც უკვე შენიშნეთ მრავალგანზომილებიანი ასოციური მასივის შესაქმნელად გამოვიყენეთ სპეციალური ფუნქცია array, რომელსაც დაწვრილებით მასივებზე მოქმედებების შესწავლის დროს განვიხილავთ.

მრავალგანზომილებიანი ასოციური მასივის შექმნა კლასიკური მეთოდებითაცაა შესაძლებელი, მაგრამ ეს ნაკლებად მოსახერხებელია:

```
<?php  
// მრავალგანზომილებიანი ასოციური მასივი  
$A["ბერიძე"] ["name"]="ბერიძე ლაშა";  
$A["ბერიძე"]["age"]="25";  
$A["ბერიძე"]["email"]="beridze@gmail.com";  
  
$A["მიქელაძე"]["name"]="მიქელაძე ნიკა ";  
$A["მიქელაძე"]["age"]="34";  
$A["მიქელაძე"]["email"]="miqeladze@yahoo.com";
```

```

$A["გელოვანი"]["name"]="გელოვანი კახა";
$A["გელოვანი"]["age"]="47";
$A["გელოვანი"]["email"]="gelovani@gmail.com";

// მრავალგანზომილებიანი ასოციური მასივის ელემენტზე
მიმართვა
echo $A["ბერიძე"]["name"]; // გამოიტანს ბერიძე ლაშა
echo $A["გელოვანი"]["age"]."<br>"; // გამოიტანს 47
echo $A["მიქელაძე"]["email"]; // გამოიტანს miqeladze@yahoo.com?>

```

## ტიპი object (ობიექტი)

ობიექტი ობიექტ-ორიენტირებული დაპროგრამების ერთ-ერთ საბაზო ცნებას წარმოადგენს. ობიექტის გარე სტრუქტურა ხეს ჰგავს, გარდა იმისა, რომ ცალკეულ ელემენტებზე და ფუნქციებზე მიმართვისათვის, კვადრატული ფრჩხილების ნაცვლად, -> ოპერატორი გამოიყენება.

ობიექტის ინიციალიზაციისათვის გამოიყენება new გამოსახულება, რომელიც ობიექტის ეგზემპლარს ცვლადად გარდაქმნის.

```

<?php
class foo
{
    function do_foo()
    {
        echo "Doing foo.";
    }
}
$bar = new foo;

```

```
$bar->do_foo();  
?>
```

ობიექტებს დაწვრილებით შევისწავლით PHP-ს კლასებისა და ობიექტების განხილვის დროს.

## ტიპი resource (რესურსები)

რესურსი - ეს სპეციალური ცვლადია, რომელიც გარე რესურსზე მიმართვას (ბმულს) შეიცავს. რესურსი სპეციალური ფუნქციის საშუალებით იქმნება და გამოიყენება.

ვინაიდან ტიპი resource ფაილის გასახსნელად სპეციალურ, მონაცემთა ბაზებთან შეერთების, გამოსახულების არეს და სხვა მსგავს მაჩვენებელს შეიცავს, ამიტომ რაიმე მნიშვნელობის გარდაქმნა რესურსად არ შეგვიძლია.

Zend PHP 4 ბირთვში შემოტანილი ბმულების დათვლის სისტემა ავტომატურად განსაზღვრავს, რომ რესურსის მიერ მიმართვა უკვე აღარ ხორციელდება. როცა ეს ხდება მეხსიერებას ყველა გამოყენებული რესურსისაგან „ნაგვის“ შემკრები ათავისუფლებს. ამიტომ, ნაკლებ სავარაუდოა, რომ მეხსიერების ხელით გასუფთავება რომელიმე free\_result ფუნქციის გამოყენებით დაგვჭირდეს.

## ტიპი NULL (ცარიელი ტიპი)

სპეციალური მნიშვნელობა NULL გვაჩვენებს, რომ ამ ცვლადს არა აქვს მნიშვნელობა. NULL - ეს არის NULL (ცარიელი ტიპი) ტიპის ერთადერთი შესაძლო მნიშვნელობა. ცვლადი NULL-ად ითვლება თუ:

- მას მიენიჭა კონსტანტა NULL;
- მას ჯერ კიდევ არა აქვს მინიჭებული რაიმე მნიშვნელობა;
- მისი წაშლა unset()-ის მეშვეობით მოხდა.

```
<?php
$var = NULL;
?>
```

`is_null` - განსაზღვრავს არის თუ არა ცვლადი `NULL` ტიპის. მისი სინტაქსია:

```
bool is_null (mixed var)
```

თუ `var` არის `Null` ტიპის შედეგად აბრუნებს `TRUE`-ს, წინააღმდეგ შემთხვევაში `FALSE`-ს.

### ფსევდოტიპი `mixed` (შერეული ტიპი)

`mixed` მიუთითებს, რომ პარამეტრი შეიძლება მრავალნაირი ტიპის იყოს.

`gettype ()`, მაგალითად, `PHP`-ს ყველა ტიპს იღებს, მაშინ როდესაც `str_replace ()` მხოლოდ სტრიქონისა და მასივის ტიპს იღებს.

### ფსევდოტიპი `number` (რიცხვი)

`number` იმაზე მიუთითებს, რომ პარამეტრი შეიძლება იყოს ან `integer`, ან `float`.

### ფსევდოტიპი `callback` (უკუ გამოძახების)

ზოგიერთი ფუნქცია, როგორცაა `call_user_func()` ან `usort()` პარამეტრის სახით მომხმარებლის მიერ განსაზღვრულ `callback`-ფუნქციას ღებულობს. `callback`-ფუნქცია არა მარტო მარტივი ფუნქცია, არამედ ობიექტთა მეთოდებიც შეიძლება იყოს, კლასთა სტატიკური მეთოდების ჩათვლით.

`PHP`-ფუნქციის გადაცემა, როგორც მისი სახელის, სტრიქონის სახით ხდება. თქვენ შეიძლება გადასცეთ ნებისმიერი



სტანდარტული ან მომხმარებლის მიერ განსაზღვრული ფუნქცია, გარდა array(), echo(), empty(), eval(), exit(), isset(), list(), print() და unset() ფუნქციებისა.

ქვემოთ მოყვანილია callback ფუნქციის მაგალითი:

```
<?php

// callback-ის მარტივი მაგალითი
function my_callback_function() {
    echo 'hello world!';
}
call_user_func('my_callback_function');

// callback-მეთოდის მაგალითი
class MyClass {
    function myCallbackMethod() {
        echo 'Hello World!';
    }
}

// სტატიკური კლასის მეთოდის გამოძახება ობიექტის შექმნის
გარეშე
call_user_func(array('MyClass', 'myCallbackMethod'));

// ობიექტის მეთოდის გამოძახება
$obj = new MyClass();
call_user_func(array(&$obj, 'myCallbackMethod'));
?>
```

მონაცემთა ტიპები PHP-ში ძალზე ზედაპირულად განვიხილეთ. თუ გსურთ უფრო დაწვრილებით გაცნობა, მაშინ გადადით ბმულზე <http://www.php.su/learnphp/datatypes/>.

## კონსტანტა PHP-ში

კონსტანტა ეწოდება სიდიდეს, რომელიც პროგრამის შესრულების პროცესში არ იცვლება.

ცვლადისაგან განსხვავებით, პროგრამის მუშაობის პროცესში კონსტანტის იმ მნიშვნელობის შეცვლა, რომელიც მას მიენიჭა მისი გამოცხადების დროს, არ შეიძლება. კონსტანტა შეიძლება გამოყენებულ იქნეს რაიმე მნიშვნელობის შესანახად, რომელიც პროგრამის მუშაობის პროცესში არ უნდა შეიცვალოს. კონსტანტა მხოლოდ სკალარულ მონაცემებს (ლოგიკური, მთელი, მცოცავმძიმძიანი და სტრიქონული ტიპის) შეიძლება შეიცავდეს.

PHP-ში კონსტანტა `define()` ფუნქციის საშუალებით განისაზღვრება. ამ ფუნქციას შემდეგი ფორმატი აქვს:

`define ($name, $value, $case_sensitive)`, სადაც:

`$name` - კონსტანტის სახელი;

`$value` - კონსტანტის მნიშვნელობა;

`$case_sensitive` - ლოგიკური ტიპის არააუცილებელი პარამეტრი, რომელიც მიუთითებს გათვალისწინებული უნდა იყოს ასოების რეგისტრი (`true`) თუ არა (`false`).

ქვემოთ მოყვანილია PHP-ში კონსტანტის განსაზღვრისა და გამოყენების მაგალითი:

```
<?php
define("pi",3.14,true);
echo pi;
// გამოიტანს 3.14
?>
```

თუ \$case\_sensitive პარამეტრი უდრის true-ს, მაშინ ინტერპრეტატორი კონსტანტასთან მუშაობის დროს სიმბოლოთა რეგისტრს გაითვალისწინებს. ყურადღება უნდა მიექცეს იმას, რომ კონსტანტა გამოიყენება \$ ნიშნის გარეშე.

კონსტანტასა და ცვლადს შორის შემდეგი განსხვავებაა:

- კონსტანტას არ აქვს წინსართი დოლარის ნიშნის სახით (\$);
- კონსტანტა შეიძლება განისაზღვროს მხოლოდ define() ფუნქციის გამოყენებით და არა მნიშვნელობის მინიჭების გზით;
- კონსტანტა შეიძლება განისაზღვროს და გამოყენებულ იქნეს ნებისმიერ ადგილზე;
- პირველი გამოცხადების შემდეგ არ შეიძლება მათი ხელახლა გამოცხადება ან ანულირება;
- კონსტანტა შეიძლება მხოლოდ სკალარული სიდიდე იყოს.

### კონსტანტის არსებობის შემოწმება

კონსტანტის არსებობის შესამოწმებლად defined() ფუნქცია გამოიყენება. მოცემული ფუნქცია შედეგად აბრუნებს true-ს, თუ კონსტანტა გამოცხადებულია. მაგალითად:

```
<?php
// კონსტანტა pi-ის გამოცხადება
define("pi",3.14,true);
if (defined("pi")==true) echo "კონსტანტა pi გამოცხადებულია!";
// სკრიპტი გამოიტანს 'კონსტანტა pi გამოცხადებულია!'
?>
```

## PHP-ის სტანდარტული კონსტანტები

PHP-ში შემდეგი წინასწარ განსაზღვრული კონსტანტები არსებობს:

PHP თითოეული შესასრულებელი სკრიპტისათვის დიდი რაოდენობის წინასწარ განსაზღვრული კონსტანტის სიას იძლევა. ბევრი ამ კონსტანტიდან ცალკეული მოდულით განისაზღვრება და მხოლოდ იმ შემთხვევაში გამოჩნდება, როდესაც ეს მოდულები დინამიკური ჩატვირთვის ან სტატიკური აწყობის დროს ხელმისაწვდომი გახდება.

არსებობს ხუთი წინასწარ განსაზღვრული კონსტანტა, რომელთა მნიშვნელობა კონტექსტიდან გამომდინარე იცვლება. მაგალითად, `__LINE__` კონსტანტა სკრიპტში იმ სტრიქონზეა დამოკიდებული, რომელშიც ის არის მითითებული. სპეციალური კონსტანტები რეგისტრზე არ არიან დამოკიდებულნი. ქვემოთ მოყვანილია მათი სია.

სახელი	აღწერა
<code>__LINE__</code>	ფაილში მიმდინარე სტრიქონი
<code>__FILE__</code>	მიმდინარე ფაილის სრული გზა და სახელი
<code>__FUNCTION__</code>	ფუნქციის სახელი. (დამატებულია PHP 4.3.0.)
<code>__CLASS__</code>	კლასის სახელი. (დამატებულია PHP 4.3.0.)
<code>__METHOD__</code>	კლასის მეთოდის სახელი. (დამატებულია PHP 5.0.0)

## გამოსახულება PHP-ში

გამოსახულება PHP-ს ერთ-ერთი ძირითადი ელემენტია. თითქმის ყველაფერი რაც PHP-ში იწერება გამოსახულებას წარმოადგენს. PHP-ში გამოსახულება გულისხმობს ყველაფერს, რასაც მნიშვნელობა აქვს მინიჭებული.

გამოსახულების ძირითადი ფორმებია კონსტანტა და ცვლადი. მაგალითად, თუ დავწერეთ "\$a = 100", ეს ნიშნავს, რომ მნიშვნელობა '100' მივანიჭეთ \$a ცვლადს:

```
$a = 100;
```

მოყვანილ მაგალითში \$a არის ცვლადი, (=) მინიჭების ოპერატორი, ხოლო 100 - გამოსახულება. მისი მნიშვნელობაა 100.

გამოსახულება ცვლადიც შეიძლება იყოს, თუ მას რაიმე მნიშვნელობა აქვს მინიჭებული:

```
$x = 7;
```

```
$y = $x;
```

განხილული მაგალითის პირველ სტრიქონში გამოსახულება არის კონსტანტა 7, ხოლო მეორე სტრიქონში - ცვლადი \$x, ვინაიდან მას ადრე მინიჭებული ჰქონდა მნიშვნელობა 7. (\$y = \$x)-ც გამოსახულებაა.

გამოსახულების ოდნავ რთულ მაგალითს წარმოადგენს ფუნქცია. მაგალითად, განვიხილოთ შემდეგი ფუნქცია:

```
<?php
function                                funct                                ()
{
    return 5;
}
?>
```

ფუნქციის კონცეფციიდან გამომდინარე  $\$x = \text{funct}()$  ჩანაწერი  $\$x = 5$  ჩანაწერის იდენტურია. ფუნქცია არის გამოსახულება, რომლის მნიშვნელობასაც წარმოადგენს ის მნიშვნელობა, რომელსაც ფუნქცია აბრუნებს. ვინაიდან  $\text{funct}()$  აბრუნებს 5-ს, ამიტომ გამოსახულება ' $\text{funct}()$ ' წარმოადგენს 5-ს. როგორც წესი, ფუნქცია აბრუნებს არა სტატიკურ მნიშვნელობას, არამედ გამოთვლით მნიშვნელობას.

PHP მხარს უჭერს სამი ტიპის სკალარულ მნიშვნელობას: მთელი რიცხვითი, მცოცავმიმდინანი და სტრიქონული მნიშვნელობა (სკალარული არის მნიშვნელობა რომლის უფრო მცირე ნაწილებად დაყოფა, მასივისაგან განსხვავებით, შეუძლებელია). PHP აგრეთვე, მხარს უჭერს ორი კომბინირებული (არა სკალარული) ტიპის მნიშვნელობას: მასივებს და ობიექტებს. თითოეული ამ მნიშვნელობათაგანი შეიძლება მიენიჭოს ცვლადს ან დაბრუნდეს ფუნქციის სახით.

PHP არის გამოსახულებაზე ორიენტირებული ენა, რომელიც ყველაფერს განიხილავს, როგორც გამოსახულებას.

დავუბრუნდეთ ჩვენს მაგალითს: ' $\$x = 7$ '. ადვილი შესამჩნევია, რომ აქ ორი მნიშვნელობაა - მთელი რიცხვითი კონსტანტის მნიშვნელობა '7' და  $\$x$  ცვლადის მნიშვნელობა, რომელიც ასევე იღებს მნიშვნელობა 7-ს. მაგრამ სინამდვილეში აქ გვაქვს კიდევ ერთი მნიშვნელობა - თვითონ მინიჭების მნიშვნელობა. თვითონ მინიჭება მისანიჭებელი მნიშვნელობით გამოითვლება, მოცემულ შემთხვევაში - 7-ით. პრაქტიკულად, ეს ნიშნავს, რომ ' $\$x = 7$ ', იმისდა მიუხედავად თუ რას აკეთებს იგი, არის გამოსახულება მნიშვნელობით 7. ამგვარად, ჩანაწერი ' $\$y = (\$x = 7)$ ' იგივეა რაც ჩანაწერი ' $\$x = 7; \$y = 7;$ ' (წერტილ-მიმღე გამოსახულების დასასრულს აღნიშნავს). ვინაიდან მინიჭების ოპერაციის ანალიზი

მარჯვნიდან მარცხნივ მიმდინარეობს, ამიტომ ეს გამოსახულება შეიძლება ასეც ჩაიწეროს '\$y = \$x = 7'.

PHP-ში გამოსახულებები ძირითადად დაკავშირებულია არითმეტიკულ ოპერაციებთან, რომლებსაც ქვემოთ გავეცნობით.



# PHP-ის ოპერატორები

ოპერატორი ეწოდება კონსტრუქციას, რომელიც ერთი ან რამდენიმე მნიშვნელობისაგან შედგება და რომლის გამოთვლა შეიძლება, როგორც ახალი მნიშვნელობისა (ამგვარად, მთელი კონსტრუქცია შეიძლება განვიხილოთ, როგორც გამოსახულება). აქედან, გამომდინარეობს, რომ ფუნქცია ან ნებისმიერი სხვა კონსტრუქცია, რომელიც მნიშვნელობას აბრუნებს (მაგალითად, `print()`), სხვა ენობრივი კონსტრუქციებისაგან განსხვავებით (მაგალითად, `echo()`), რომლებიც არაფერს არ აბრუნებენ, ოპერატორს წარმოადგენს.

## არითმეტიკული ოპერატორები

ალბათ, ყველას ახსოვს სკოლის არითმეტიკის საფუძვლები. ქვემოთ მოყვანილი ოპერატორებიც ასევე მუშაობს.

მაგალითი	დასახელება	შედეგი
$-\$a$	უარყოფა	ნიშნის შეცვლა $\$a$ .
$\$a + \$b$	შეკრება	$\$a$ -ს და $\$b$ -ს ჯამი.
$\$a - \$b$	გამოკლება	$\$a$ -ს და $\$b$ -ს სხვაობა.
$\$a * \$b$	გამრავლება	$\$a$ -ს და $\$b$ -ს ნამრავლი.
$\$a / \$b$	გაყოფა	$\$a$ -ს $\$b$ -ზე გაყოფის შედეგი.
$\$a \% \$b$	მოდულით გაყოფა	$\$a$ -ს $\$b$ -ზე განაყოფის მთელი რიცხვითი ნაშთი.

გაყოფის (" $/$ ") ოპერაციის შედეგი ყოველთვის არის ნამდვილი ტიპის რიცხვი, თუნდაც ორივე მნიშვნელობა მთელი რიცხვი იყოს (ან სტრიქონი, რომელიც მთელ რიცხვად გარდაიქმნება).

ასევე შესაძლებელია ფრჩხილების გამოყენება. მათემატიკური ოპერაციების შესრულების პრიორიტეტი და ფრჩხილების

გამოყენებით ამ პრიორიტეტების შეცვლის წესი, ჩვეულებრივ, მათემატიკურ წესებს შეესაბამება.

### ინკრემენტისა და დეკრემენტის ოპერატორები

PHP, C ენის ანალოგიურად, მხარს უჭერს პრეფიქსულ და პოსტფიქსულ ინკრემენტისა და დეკრემენტის ოპერატორებს.

მაგალითი	დასახელება	მოქმედება
++\$a	პრეფიქსული ინკრემენტი	\$a-ს მნიშვნელობას ერთით ზრდის და აბრუნებს \$a-ს მნიშვნელობას.
\$a++	პოსტფიქსული ინკრემენტი	აბრუნებს \$a-ს მნიშვნელობას, ხოლო შემდეგ \$a-ს მნიშვნელობას ერთით ზრდის.
--\$a	პრეფიქსული დეკრემენტი	\$a-ს მნიშვნელობას ერთით ამცირებს და აბრუნებს \$a-ს მნიშვნელობას.
\$a--	პოსტფიქსული დეკრემენტი	აბრუნებს \$a-ს მნიშვნელობას, ხოლო შემდეგ \$a-ს მნიშვნელობას ერთით ამცირებს.

როგორც C ენაში პოსტფიქსული ინკრემენტისა და დეკრემენტის ოპერატორები, აქაც ცვლადის მნიშვნელობას ზრდიან ან ამცირებენ, ხოლო გამოსახულებაში \$a ცვლადის ცვლილებამდე მნიშვნელობას აბრუნებენ. მაგალითად:

```
$a=10;
$b=$a++;
echo "a=$a, b=$b"; // გამოიტანს a=11, b=10
```

როგორც მაგალითიდან ჩანს, ჯერ \$b ცვლადს მიენიჭა \$a ცვლადის მნიშვნელობა და მხოლოდ ამის შემდგომ მოხდა მისი

ინკრემენტირება. ასეთ ოპერაციებს პოსტფიქსული ოპერაციები ეწოდება.

ასევე არსებობს ინკრემენტისა და დეკრემენტის ოპერატორები, რომლებიც ცვლადის სახელამდე სრულდება. შესაბამისად ისინი მნიშვნელობებს ცვლილების შემდეგ აბრუნებენ. მაგალითად:

```
$a=10;  
$b--$a;  
echo "a=$a, b=$b"; // გამოიტანს a=9, b=9
```

პრაქტიკაში ინკრემენტისა და დეკრემენტის ოპერაციები ძალიან ხშირად გვხვდება. მაგალითად, ისინი პრაქტიკულად ნებისმიერ for ციკლის ოპერატორში გამოიყენება.

```
<?php  
echo "<h3>პოსტფიქსული ინკრემენტი</h3>";  
$a = 5;  
echo "a++ უნდა იყოს 5: " . $a++ . "<br>";  
echo "a უნდა იყოს 6: " . $a . "<br>";  
  
echo "<h3>პრეფიქსული ინკრემენტი</h3>";  
$a = 5;  
echo "++a უნდა იყოს 6: " . ++$a . "<br>";  
echo "a უნდა იყოს 6: " . $a . "<br>";  
  
echo "<h3>პოსტფიქსული დეკრემენტი </h3>";  
$a = 5;  
echo "a-- უნდა იყოს 5: " . $a-- . "<br>";  
echo "a უნდა იყოს 4: " . $a . "<br>";  
  
echo "<h3>პრეფიქსული დეკრემენტი </h3>";  
$a = 5;
```

```
echo "--a უნდა იყოს 4: " . --$a . "<br>";  
echo "a უნდა იყოს 4: " . $a . "<br>";  
?>
```

### პოსტივსული ინკრემენტი

```
a++ უნდა იყოს 5: 5  
a უნდა იყოს 6: 6
```

### პრეფიქსული ინკრემენტი

```
++a უნდა იყოს 6: 6  
a უნდა იყოს 6: 6
```

### პოსტივსული დეკრემენტი

```
a-- უნდა იყოს 5: 5  
a უნდა იყოს 4: 4
```

### პრეფიქსული დეკრემენტი

```
--a უნდა იყოს 4: 4  
a უნდა იყოს 4: 4
```

ქვემოთ მოყვანილია სიმბოლურ ცვლადებთან ოპერაციის მაგალითი:

```
<?php  
$i = 'W';  
for($n=0; $n<6; $n++)  
    echo ++$i . "<br>";  
?>
```



ბულის ტიპის მონაცემები ინკრემენტირებასა და დეკრემენტირებას არ ექვემდებარება.

### მინიჭების ოპერატორები

მინიჭების საბაზო ოპერატორი აღინიშნება (=)-ით. ერთი შეხედვით შეიძლება მოგვეჩვენოს, რომ ეს არის ოპერატორი „ტოლია“. სინამდვილეში ეს ასე არ არის. სინამდვილეში, მინიჭების ოპერატორი გვიჩვენებს, რომ მარცხენა ოპერანდი მარჯვენა გამოსახულების მნიშვნელობას იღებს.

მინიჭების ოპერატორის შესრულების შედეგს თვით მინიჭებული მნიშვნელობა წარმოადგენს. ამგვარად,  $a = 3$  შესრულების შედეგი 3-ის ტოლი იქნება. ეს შემდეგი ტიპის კონსტრუქციის გამოყენების საშუალებას იძლევა:

```
<?php  
$a = ($b = 4) + 5; // შედეგი: $a-ს მიენიჭება  
მნიშვნელობა 9, ხოლო $b-ს მიენიჭება 4.  
?>
```

საბაზო მინიჭების ოპერატორის გარდა, ყველა ბინარული არითმეტიკული და სტრიქონული ოპერაციისათვის არსებობს

„კომბინირებული ოპერატორი“, რომელიც საშუალებას გვაძლევს გამოსახულებაში გამოვიყენოთ ზოგიერთი მნიშვნელობა, ხოლო შემდეგ იგი მივიღოთ, როგორც ამ გამოსახულების შედეგი. მაგალითად:

```
<?php
$a = 3;
$a += 5; // $a-ს მნიშვნელობა იქნება 8, ეს ჩანაწერი
ანალოგიურია: $a = $a + 5;
?>
```

ყურადღება მიაქციეთ იმას, რომ მინიჭება ორიგინალური ცვლადის ახალში კოპირებას ახდენს (მინიჭება მნიშვნელობით), ამგვარად, ერთ-ერთი ცვლადის შემდგომი ცვლილება სხვა ცვლადზე არაფრით არ აისახება. PHP ასევე, მხარს უჭერს მინიჭებას ბმულით, რომლის დროსაც `$var = &$othervar;` სინტაქსს იყენებს. „მინიჭება ბმულით“ ნიშნავს, რომ ორივე ცვლადი ერთი და იმავე მონაცემზე მიუთითებს და არავითარი კოპირება არ ხდება.

## ბმულები PHP-ში

მართალია PHP-ში მაჩვენებლის ცნება არ არის, მაგრამ სხვა ცვლადზე ბმულის შექმნის შესაძლებლობა მაინც არის. არსებობს ორი სახის ბმული: მყარი და სიმბოლური (დინამიკური ცვლადები). მყარი ბმული PHP-ის მე-4 ვერსიაში გაჩნდა და მას ხშირად ჩვეულებრივ ბმულს უწოდებენ.

ბმულები PHP-ში - ეს არის იმის საშუალება, რომ ერთი ცვლადის მნიშვნელობას სხვადასხვა სახელით მივმართოთ. ის C ენის მაჩვენებელს არ ჰგავს და სიმბოლოების ცხრილის ფსევდონიმსაც არ წარმოადგენს. PHP-ში ცვლადის სახელი და მასი მნიშვნელობა სხვადასხვა რამეა, ამიტომ ერთ მნიშვნელობას შეიძლება სხვადასხვა სახელი ჰქონდეს. ბმულები PHP-ში - ეს Unix-ის ფაილურ სისტემაში მყარი ბმულების (hardlinks) ანალოგია.

## მყარი ბმულები PHP-ში

მყარი ბმული უბრალო ცვლადს წარმოადგენს, რომელიც სხვა ცვლადის სინონიმია. მრავალდონიანი ბმული (როგორც ეს Perl-შია შესაძლებელი) აქ მხარდაჭერილი არ არის. ასე, რომ მყარი ბმული სინონიმზე უფრო რთულ მაქანიზმად არ უნდა აღვიქვათ.

იმისათვის, რომ მყარი ბმული შეიქმნას, უნდა გამოვიყენო ოპერატორი & (ამპერსანტი). მაგალითად,

```
$a=10;  
$b = &$a; // ახლა $b - იგივეა რაც $a  
$b=0; // სინამდვილეში $a=0  
echo "b=$b, a=$a"; // გამოიტანს: "b=0, a=0"
```

მიმართვა შესაძლებელია არა მარტო ცვლადებზე, არამედ მასივის ელემენტებზეც (ამით მყარი ბმულები მკვეთრად განსხვავდება სიმბოლური ბმულებისაგან). მაგალითად,

```
$A=array('a' => 'aaa', 'b' => 'bbb');  
$b=&$A['b']; // ახლა $b - იგივეა, რაც მასივის ელემენტი ინდექსით  
'b'  
$b=0; // სინამდვილეში იგივეა, რაც $A['b']=0;  
echo $A['b']; // გამოიტანს 0
```

თუმცა, მასივის ელემენტი, რომლისთვისაც ბმულის შექმნა იგეგმება, შეიძლება საერთოდაც არ არსებობდეს, როგორც ეს შემდეგ მაგალითშია მოცემული:

```
$A=array('a' => 'aaa', 'b' => 'bbb');  
$b=&$A['c']; // ახლა $b - იგივეა, რაც მასივის ელემენტი ინდექსით  
'c'  
echo "ელემენტი ინდექსით 'c': (".$A['c'].")";
```

განხილული სკრიპტის შესრულების შედეგად, მართალია \$b ბმულს რაიმე მნიშვნელობა არ მიენიჭა, მაგრამ \$A მასივში შეიქმნა ახალი ელემენტი c ინდექსით და მნიშვნელობით - ცარიელი სტრიქონი (ეს შეგვიძლია echo ოპერატორით განვსაზღვროთ). მაშასადამე, მყარ ბმულს არ შეუძლია არარსებულ ობიექტს მიმართოს, მაგრამ თუ ასეთ ქმედებას აქვს ადგილი, მაშინ ობიექტი იქმნება.

თუ პროგრამიდან მყარი ბმულის შექმნის სტრიქონს ამოვიღებთ, მაშინ წარმოიშობა შეტყობინება, რომ \$A მასივში ელემენტი ინდექსით c განსაზღვრული არ არის.



მომხმარებლის ფუნქციებისათვის პარამეტრების მიწოდებისა და მათგან მნიშვნელობის დაბრუნების დროს მყარი ბმულების გამოყენება ძალზე მოსახერხებელია.

## დინამიკური ცვლადები

დინამიკური ცვლადი PHP-ში რომელიმე ცვლადში სხვა ცვლადის სახელის დამახსოვრების შესაძლებლობას გულისხმობს. პროგრამისტებში ეს მაჩვენებლის ასოციაციას იწვევს. ამ მექანიზმის აღსანიშნავად დოკუმენტაციაში სიმბოლური ბმული გამოიყენება. იმისათვის, რომ რომელიმე ასეთი ცვლადის მნიშვნელობა გამოვიტანოთ საჭიროა გამოვიყენოთ ორმაგი ნიშანი \$\$.

მაგალითად,

```
<?php
$name1 = "name2";
$name2 = "ჩემი სახელია ნიკოლოზი!";
echo $name1; // გამოიტანს "name2"
echo "<br>";
echo $$name1; // გამოიტანს "ჩემი სახელია ნიკოლოზი!"
?>
```

```
name2
ჩემი სახელია ნიკოლოზი!
```

## მყარი ბმულები და მომხმარებლის ფუნქციები

### ბმულებით მნიშვნელობის გადაცემა.

მომხმარებლის ფუნქციაში ცვლადები შეიძლება ბმულებით გადასცეთ, თუ გსურთ ფუნქციას თავისი არგუმენტების მოდიფიკაციის ნება დართოთ. ასეთ შემთხვევაში მომხმარებლის ფუნქციას შეუძლია თავისი არგუმენტი შეცვალოს. ამის სინტაქსი ასეთია:

```
<?php
function foo(&$var)
{
    $var++;
}
$a=5;
foo($a);
// აქ $a ტოლია 6-ის
?>
```

უნდა შევნიშნოთ, რომ ფუნქციის გამოძახებაში ბმულის ნიშანი არა გვაქვს - იგი მხოლოდ ფუნქციის განსაზღვრაშია გამოყენებული. ეს არგუმენტის ბმულით გადასაცემად სავსებით საკმარისია. მაგალითად,

```
<?php
function funct(&$string)
{
    $string .= 'ეს სტრიქონი ფუნქციის შიგნითაა.';
}
$str = 'ეს სტრიქონი ფუნქციის გარეთაა, ';
funct($str);
```

```
echo $str; // გამოიტანს 'ეს სტრიქონი ფუნქციის გარეთაა, ეს
           სტრიქონი ფუნქციის შიგნითაა.'
```

```
?>
```

ბმულით შეიძლება გადავცეთ:

- ცვლადები, მაგალითად `foo($a)`;
- ოპერატორი `new`, მაგალითად `foo(new foobar())`;
- ბმული, რომელსაც დაბრუნებს ფუნქცია, მაგალითად:

```
<?php
function &bar()
{
    $a = 5;
    return $a;
}
foo(bar());
?>
```

არც ერთი სხვა გამოსახულების გადაცემა ბმულით არ შეიძლება, ვინაიდან შედეგი განსაზღვრული არ იქნება. მაგალითად, ბმულის შემდეგი გადაცემა არასწორია:

```
<?php
function bar() // ოპერაცია & არა გვაქვს
{
    $a = 5;
    return $a;
}
foo(bar());
```

```
foo($a = 5);    // არგუმენტად გვაქვს გამოსახულება და არა
               ცვლადი
foo(5);        // არგუმენტად გვაქვს კონსტანტა და არა ცვლადი
?>
```

### ბმულით მნიშვნელობის დაბრუნება.

განვიხილოთ მომხმარებლის ფუნქციის კიდეც ერთი მაგალითი - ბმულის დაბრუნება. ბმულის დაბრუნება იმ შემთხვევაში გამოიყენება, როდესაც ფუნქციის გამოყენება იმ ცვლადის ამოსარჩევად გვსურს, რომელთანაც მოცემული ბმული უნდა იყოს დაკავშირებული. ამ დროს შემდეგი სინტაქსი გამოიყენება:

```
<?php
function &find_var($param)
{
    /* ... კოდი... */
    return $found_var;
}
$foo =& find_var($bar);
$foo->x = 2;
?>
```

ამ მაგალითში ცვლადს find\_var ფუნქციის მიერ დაბრუნებული ობიექტის და არა მისი ასლის თვისება ექნება, როგორც ეს ბმულის გამოყენების გარეშე იქნებოდა.

განვიხილოთ მომხმარებლის ფუნქციის მიერ ბმულით დაბრუნებული მნიშვნელობის კიდეც ერთი მაგალითი:

```

<?php
$a = 100;
/* შემდეგ მოდის ფუნქცია, რომელიც ბმულს აბრუნებს */
function &s () {
    global $a;
    // ბმულს ვაბრუნებთ ცვლადზე $a
    return $a;
}
// $b ცვლადს ვანიჭებთ ბმულს
$b = &s();
$b = 0;
echo $a; // გამოიტანს 0
?>

```

## ბმულის წაშლა

ბმულის წაშლის შემთხვევაში, უბრალოდ, წყდება კავშირი ცვლადის სახელსა და მის მნიშვნელობას შორის. ეს არ ნიშნავს, რომ ცვლადის შინაარსი დაინგრევა. მაგალითად,

```

<?php
$a = 1;
$b =& $a;
unset($a);
?>

```

ეს კოდი წაშლის მხოლოდ \$a ცვლადს და არ წაშლის \$b-ს.

მყარი ბმული არ არის აბსოლუტურად ზუსტი სინონიმი იმ ობიექტისა, რომელსაც იგი მიმართავს. საქმე ისაა, რომ მყარი ბმულისათვის შესრულებული unset() ოპერატორი, არ წაშლის იმ

ობიექტს, რომელსაც ის მიმართავს, იგი კავშირს გაწყვეტს ბმულსა და ობიექტს შორის.

ამგვარად, მყარი ბმული და ცვლადი (ობიექტი), რომელსაც ის მიმართავს სრულიად თანაბარუფლებიანია, მაგრამ ერთის ცვლილება მეორის ცვლილებას იწვევს. ოპერატორი unset() ბმულსა და ობიექტს შორის კავშირს წყვეტს, მაგრამ ობიექტი მხოლოდ მაშინ იშლება, როდესაც მას აღარავინ მიმართავს.

## ბიტური ოპერატორები

მოცემული ოპერატორები განკუთვნილია მთელი ცვლადებისათვის ბიტური ჯგუფების მისანიჭებლად ან მოსახსნელად. რადგანაც ნებისმიერი რიცხვი ეს არის ბიტების მიმდევრობა. მთელი რიცხვები PHP-ში 32-ბიტია. ერთი რიცხვის წარმოსადგენად გამოიყენება 32 ბიტი:

- 0000 0000 0000 0000 0000 0000 0000 0000 - ეს ნულია;
- 0000 0000 0000 0000 0000 0000 0000 0001 - ეს 1;
- 0000 0000 0000 0000 0000 0000 0000 0010 - ეს 2;
- 0000 0000 0000 0000 0000 0000 0000 0011 - ეს 3;
- 0000 0000 0000 0000 0000 0000 0000 0100 - ეს 4;
- 0000 0000 0000 0000 0000 0000 0000 0101 - ეს 5;
- ...
- 0000 0000 0000 0000 0000 0000 0000 1111 - ეს 15;
- ...

ბიტური ოპერატორებია:

მაგალითი	დასახელება	შედეგი
\$a & \$b	ბიტური 'და'	მხოლოდ იმ ბიტების მინიჭება მოხდება, რომელიც მინიჭებულია \$a-შიც და \$b-შიც.

$\$a \mid \$b$	ბიტური 'ან'	იმ ბიტების მინიჭება მოხდება, რომელიც მინიჭებულია ან $\$a$ -ში ან $\$b$ -ში.
$\$a \wedge \$b$	'ან'-ის უარყოფა	იმ ბიტების მინიჭება მოხდება, რომელიც მინიჭებულია ან მხოლოდ $\$a$ -ში ან მხოლოდ $\$b$ -ში.
$\sim \$a$	უარყოფა	იმ ბიტების მინიჭება მოხდება, რომელიც $\$a$ -ში მინიჭებული არს არის და პირიქით.
$\$a \ll \$b$	მარცხნივ წანაცვლება	$\$a$ ცვლადის ყველა ბიტი წანაცვლებს $\$b$ პოზიციით მარცხნივ (ყოველი პოზიცია 2-ზე გამრავლებას გულისხმობს)
$\$a \gg \$b$	მარჯვნივ წანაცვლება	$\$a$ ცვლადის ყველა ბიტი წანაცვლებს $\$b$ პოზიციით მარჯვნივ (ყოველი პოზიცია 2-ზე გაყოფას გულისხმობს)

## შედარების ოპერატორები

შედარების ოპერატორები, როგორც სახელიდან ჩანს, საშუალებას გვაძლევს ერთმანეთს ორი მნიშვნელობა შევადაროთ.

PHP-ში მხოლოდ სკალარული მონაცემების შედარებაა შესაძლებელი. PHP-ში მასივებისა და ობიექტების შედარება არ შეიძლება.

შედარების ოპერატორები:

მაგალითი	დასახელება	შედეგი
$\$a == \$b$	ტოლია	TRUE თუ $\$a$ ტოლია $\$b$ -სი.

$\$a === \$b$	იგივეურად ტოლია	<b>TRUE</b> თუ $\$a$ ტოლია $\$b$ -სი და იგივე ტიპისაა
$\$a !== \$b$	არ უდრის	<b>TRUE</b> თუ $\$a$ არ უდრის $\$b$ -ს.
$\$a <> \$b$	არ უდრის	<b>TRUE</b> თუ $\$a$ არ უდრის $\$b$ -ს.
$\$a !== \$b$	იგივეურად არ არის ტოლი	<b>TRUE</b> თუ $\$a$ არ უდრის $\$b$ -ს ან თუ ისინი სხვადასხვა ტიპისაა.
$\$a < \$b$	ნაკლებია	<b>TRUE</b> თუ $\$a$ მკაცრად ნაკლებია $\$b$ -ზე.
$\$a > \$b$	მეტია	<b>TRUE</b> თუ $\$a$ მკაცრად მეტია $\$b$ -ზე.
$\$a <= \$b$	ნაკლებია ან ტოლია	<b>TRUE</b> თუ $\$a$ ნაკლებია ან ტოლი $\$b$ -ზე.
$\$a >= \$b$	მეტია ან ტოლია	<b>TRUE</b> თუ $\$a$ მეტია ან ტოლი $\$b$ -ზე.

## ლოგიკური ოპერატორები

ქვემოთ მოყვანილია PHP-ს ლოგიკური ოპერატორების ცხრილი:

მაგალითი	დასახელება	შედეგი
$\$a$ and $\$b$	ლოგიკური 'და'	<b>TRUE</b> თუ $\$a$ -ც და $\$b$ -ც არის <b>TRUE</b> .
$\$a$ or $\$b$	ლოგიკური 'ან'	<b>TRUE</b> თუ ან $\$a$ , ან $\$b$ არის <b>TRUE</b> .
$\$a$ xor $\$b$	'ან'-ის უარყოფა	<b>TRUE</b> თუ $\$a$ , ან $\$b$ არის <b>TRUE</b> , მაგრამ ორივე ერთად არა.
! $\$a$	უარყოფა	<b>TRUE</b> თუ $\$a$ არ არის <b>TRUE</b> .
$\$a$ && $\$b$	ლოგიკური 'და'	<b>TRUE</b> თუ $\$a$ -ც და $\$b$ -ც <b>TRUE</b> .



\$a    \$b	ლოგიკური 'ან'	TRUE თუ ან \$a, ან \$b არის TRUE.
------------	---------------	-----------------------------------

(++) ინკრემენტის და (--) დეკრემენტის ოპერატორები ლოგიკურ ცვლადებთან არ მუშაობენ.

## PHP-ის ოპერატორების პრიორიტეტები

უფრო მაღალი პრიორიტეტის მქონე ოპერატორები პირველ რიგში სრულდება:

პრიორიტეტი	ოპერატორი	შესრულების მიმდევრობა
13	(პოსტფიქსი)++ (პოსტფიქსი)--	მარცხნიდან მარჯვნივ
12	++(პრეფიქსი) (პრეფიქსი)--	მარჯვნიდან მარცხნივ
11	* / %	მარცხნიდან მარჯვნივ
10	+ -	მარცხნიდან მარჯვნივ
9	<< >>	მარცხნიდან მარჯვნივ
8	< <= > >=	მარცხნიდან მარჯვნივ
7	== !=	მარცხნიდან მარჯვნივ
6	&	მარცხნიდან მარჯვნივ
5	^	მარცხნიდან მარჯვნივ
4		მარცხნიდან მარჯვნივ

3	&&	მარცხნიდან მარჯვნივ
2		მარცხნიდან მარჯვნივ
1	= += -= *= /= %= >>= <<== &= ^=  =	მარჯვნიდან მარცხნივ

ნებისმიერ შემთხვევაში, თუ გეჟებხათ, ან გეშინიათ რომ შეცდეთ, უმჯობესია გამოიყენოთ ფრჩხილები.

## სტრიქონული ოპერატორები

PHP-ში სტრიქონთან სამუშაოდ ორი ოპერატორი გვაქვს. ერთია - კონკატენაციის (გაერთიანების) ოპერატორი ('.'), რომელიც ტოლობის მარჯვენა ნაწილში მდგომ მარცხენა და მარჯვენა არგუმენტებს აერთიანებს. მეორე - მინიჭების ოპერატორი კონკატენაციით (გაერთიანებით), რომელიც ტოლობის მარცხენა ნაწილში მდგომ არგუმენტს მარჯვენა ნაწილში მდგომ არგუმენტთან აერთიანებს. მოვიყვანოთ ამის კონკრეტული მაგალითი:

```
<?php
$a = "Hello ";
$b = $a . "World!"; // $b შეიცავს სტრიქონს "Hello World!"

$a = "Hello ";
$a .= "World!"; // $a შეიცავს სტრიქონს "Hello World!"
?>
```

ამ ოპერატორების შესახებ საუბარი ზემოთაც გვქონდა.

## PHP-ს მასივებთან სამუშაო ოპერატორები

მოვიყვანოთ PHP-ს მასივებთან სამუშაო ოპერატორების სია:

მაგალითი	დასახელება	შედეგი
<code>\$a + \$b</code>	გაერთიანება	<code>\$a</code> მასივისა და <code>\$b</code> მასივის გაერთიანება.
<code>\$a == \$b</code>	ტოლია	<b>TRUE</b> იმ შემთხვევაში, თუ <code>\$a</code> და <code>\$b</code> მასივი ერთსა და იმავე ელემენტებს შეიცავს.
<code>\$a === \$b</code>	იგივეურად ტოლია	<b>TRUE</b> იმ შემთხვევაში, თუ <code>\$a</code> და <code>\$b</code> მასივი ერთსა და იმავე ელემენტებს ერთი და იგივე მიმდევრობით შეიცავს.
<code>\$a != \$b</code>	არ უდრის	<b>TRUE</b> თუ <code>\$a</code> მასივი არ უდრის <code>\$b</code> მასივს.
<code>\$a &lt;&gt; \$b</code>	არ უდრის	<b>TRUE</b> თუ <code>\$a</code> მასივი არ უდრის <code>\$b</code> მასივს.
<code>\$a !== \$b</code>	იგივეურად არ უდრის	<b>TRUE</b> თუ <code>\$a</code> მასივი იგივეურად არ უდრის <code>\$b</code> მასივს.

+ ოპერატორის მარცხენა მხარეს მდგომ მასივს მიუერთებს მის მარჯვნივ მდგომ მასივს ისე, რომ დუბლირებული გასაღების მქონე ელემენტებს არ გადააწერს.

```
<?php
$a = array("a" => "ვაშლი", "b" => "ბანანი");
$b = array("a" => "ატამი", "b" => "მარწყვი", "c" => "ბალი");
$c = $a + $b; // $a და $b-ს გაერთიანება
echo "\$a და \$b-ს გაერთიანება: <br>";
for ($i="a"; $i<="c"; $i++) {
echo $c["$i"].<br>";
}
echo "<br>";
$c = $b + $a; // $b და $a-ს გაერთიანება
echo "\$b და \$a-ს გაერთიანება: <br>";
for ($i="a"; $i<="c"; $i++) {
echo $c["$i"].<br>";
}
}
```

```
?>
```

ამ სკრიპტის შესრულების შედეგს შემდეგი სახე ექნება:

```
$a და $b-ს გაერთიანება:  
ვაშლი  
ბანანი  
ბალი  
  
$b და $a-ს გაერთიანება:  
ატამი  
მარწყვი  
ბალი
```

მასივების შედარების დროს მათი ელემენტები იდენტურად ითვლება, თუ გასაღები და შესაბამისი მნიშვნელობა ერთმანეთს ემთხვევა.

```
<?php  
$a = array("ვაშლი", "ბანანი");  
$b = array(1 => "ბანანი", "0" => "ვაშლი");  
  
var_dump($a == $b); // bool(true)  
var_dump($a === $b); // bool(false)  
?>
```

## სტრიქონის დამუშავება

თუ სტრიქონი ბრჭყალებით ან heredoc-ის საშუალებითაა განსაზღვრული, მაშინ მასში მოთავსებული ცვლადები დამუშავდება.

არსებობს ორი სახის სინტაქსი: მარტივი და რთული. მარტივი სინტაქსი შედარებით ადვილად მოსახერხებელია. იგი ცვლადების, მასივის (array) მნიშვნელობის ან ობიექტის (object) თვისების დამუშავების საშუალებას იძლევა.

რთული სინტაქსი PHP 4-ში იყო შემოტანილი და მისი ამოცნობა შესაძლებელია ფიგურულ ფრჩხილებში მოთავსებული გამოსახულებით.

## მარტივი სინტაქსი

თუ ინტერპრეტატორი დოლარის ნიშანს (\$) ხვდება, მაშინ მას შეუძლია იმდენი სიმბოლო მოიცვას, რამდენიცაა საჭირო ცვლადის სწორი სახელის ფორმირებისათვის. თუ გსურთ სახელის დაბოლოება ზუსტად განსაზღვრით, მაშინ ცვლადის სახელი ფიგურულ ფრჩხილებში მოათავსეთ.

```
<?php
$beer = 'Heineken';
echo "$beer's taste is great"; // მუშაობს, "" ცვლადის სახელისათვის
არასწორი სიმბოლოა
echo "He drank some $beers"; // არ მუშაობს, 's' ცვლადის
სახელისათვის სწორი
სიმბოლო არ არის
echo "He drank some ${beer}s"; // მუშაობს
echo "He drank some {$beer}s"; // მუშაობს
?>
```

ზუსტად, ასევე შეიძლება დამუშავდეს მასივის (array) ელემენტები ან ობიექტის (object) თვისებები. მასივის ინდექსებში დახურვის კვადრატული ფრჩხილი ([]) ინდექსის განსაზღვრის დამთავრებას აღნიშნავს. ობიექტის თვისებებისათვის იგივე წესები გამოიყენება, რაც ჩვეულებრივი ცვლადებისათვის.

```
<?php
// ეს არის სტრიქონში მასივების გამოყენების სპეციფიკური
მაგალითები // სტრიქონს გარეთ მასივის სტრიქონული გასაღები
ყოველთვის ბრჭყალებში მოათავსეთ // და სტრიქონის გარეთ არ
გამოიყენოთ {ფრჩხილები}.

// აქ ყველა შესაძლო შეცდომა ვაჩვენოთ
error_reporting(E_ALL);
$fruits = array('მარწყვი' => 'წითელი', 'ბანანი' => 'ყვითელი');

// მუშაობს, მაგრამ სტრიქონები ბრჭყალების გარეშე სხვანაირად
მუშაობს
echo "ბანანი $fruits[ბანანი]ა";
echo "<br>";

// მუშაობს
echo "ბანანი {$fruits['ბანანი']}ა.";
echo "<br>";

// მუშაობს, მაგრამ PHP, როგორც ქვემოთაა აღწერილი, ჯერ ეძებს
კონსტანტას ბანანი.
echo "ბანანი {$fruits[ბანანი]}ა. ";
echo "<br>";

// მუშაობს
echo "ბანანი " . $fruits['ბანანი'] . "ა.";
```

```
echo "<br>";  
?>
```

ბანანი ყვითელია  
ბანანი ყვითელია.

Notice: Use of undefined constant ბანანი - assumed 'ბანანი' in C:\xampp\htdocs\mag19.php on line 18  
ბანანი ყვითელია.

ბანანი ყვითელია.

უფრო რთული ამოცანისათვის შეიძლება რთული სინტაქსი გამოიყენოთ.

## რთული (ფიგურული) სინტაქსი

მოცემულ სინტაქსს რთული იმიტომ კი არ ეწოდება, რომ რთულია გასაგებად, არამედ იმიტომ, რომ რთული გამოსახულების გამოყენების საშუალებას იძლევა.

ფაქტობრივად, ნებისმიერი მნიშვნელობა შეგიძლიათ გამოიყენოთ, რომელიც სტრიქონის სახელისათვის არის დასაშვები. თქვენ უბრალოდ გამოსახულებას იმავე წესით ჩაწერთ, როგორც სტრიქონის გარეთ, ხოლო შემდეგ მას ფიგურულ ფრჩხილებს ({ და }) შორის მოათავსებთ. ვინაიდან არ შეგიძლიათ '{'-ის ეკრანირება, ამიტომ ამ სინტაქსს მაშინ აღიქვამს, როდესაც „\$“ სიმბოლო უშუალოდ „{,“ სიმბოლოს მოსდევს. (გამოიყენეთ „{\\$“ ან „\{\$“ რათა „{\\$“-ის ასახვა მოხდეს). ქვემოთ მოყვანილია მაგალითები:

```
<?php  
// ვაჩვენოთ ყველა შესაძლო შეცდომა  
error_reporting(E_ALL);
```

```

$great = 'fantastic';

// არ მუშაობს, გამოიტანს: This is { fantastic}
echo "This is { $great}";

// მუშაობს, გამოიტანს: This is fantastic
echo "This is {$great}";
echo "This is ${great}";

// მუშაობს
echo "ეს კვადრატია, რომლის სიგანეა {$square-
>width}00 სანტიმეტრი.";

// მუშაობს
echo "ეს მუშაობს: {$arr[4][3]}";

// ეს არ არის სწორი, თუმცა იმუშავებს. ვინაიდან PHP ჯერ ეძებს
კონსტანტას foo,

// ეს გამოიწვევს E_NOTICE დონის შეცდომას (განუსაზღვრელი
კონსტანტა)
echo "ეს არ არის სწორი: {$arr[foo][3]}";

// მუშაობს. მრავალგანზომილებიანი მასივის შემთხვევაში,
სტრიქონის
// შიგნით ყოველთვის გამოიყენეთ ფიგურული ფრჩხილები
echo "ეს მუშაობს: {$arr['foo'][3]}";

// მუშაობს.
echo "ეს მუშაობს: " . $arr['foo'][3];

```



```
echo "შეგიძლიათ ასეც ჩაწეროთ {$obj->values[3]->name}";
```

```
echo "ეს არის ცვლადის მნიშვნელობა, რომლის  
სახელია $name: {${$name}}";  
?>
```

## სტრიქონში სიმბოლოზე წვდომა და მისი შეცვლა

სტრიქონში სიმბოლოების გამოყენება და მოდიფიცირება შეიძლება, თუ სტრიქონის დასაწყისიდან განისაზღვრება მისი ადგილმდებარეობა. ათვლა ნულიდან იწყება და მითითება სტრიქონის სახელის შემდეგ ფიგურულ ფრჩხილებში ხდება. მოვიყვანოთ მაგალითები:

```
<?php  
// პირველი სიმბოლოს მიღება  
$str = 'This is test.';  
$first = $str{0};  
echo "$first";  
echo "<br>";  
  
// მეოთხე სიმბოლოს მიღება  
$fourth = $str{3};  
echo "$fourth";  
echo "<br>";  
  
// სტრიქონის ბოლო სიმბოლოს მიღება  
$str = 'ეს ისევ ტესტია.';  
$last = $str{strlen($str)-1};  
echo "$last";  
echo "<br>";
```

```
// სტრიქონის ბოლო ელემენტის შეცვლა
$str = 'This is test.';
$str{strlen($str)-1} = '!';
echo "$str";
?>
```

```
T
s
-
This is test;
```

## PHP ენის კონსტრუქციები

PHP-ის ნებისმიერი სცენარი რამდენიმე კონსტრუქციითაა ფორმირებული. კონსტრუქცია შეიძლება იყოს ოპერატორები, ფუნქციები, ციკლები, პირობითი კონსტრუქციები, ასევე კონსტრუქციები, რომლებიც არაფერს არ აკეთებენ (ცარიელი კონსტრუქციები). ჩვეულებრივ, კონსტრუქცია წერტილ-მძიმით მთავრდება. ამის გარდა, კონსტრუქციები შეიძლება დაჯგუფებული იყოს, რაც ფიგურულ ფრჩხილებში {...} მოთავსებულ კონსტრუქციის ჯგუფს ქმნის. კონსტრუქციის ჯგუფიც ცალკე კონსტრუქციას წარმოადგენს. PHP ენის კონსტრუქცია C ენის კონსტრუქციას ჰგავს.

მოკლედ განვიხილოთ PHP ენის ძირითადი კონსტრუქციები. ესენია:

- პირობითი ოპერატორები:

if

else

elseif

- ციკლები:

while

do-while

for

foreach

break

continue

- ამორჩევის კონსტრუქცია:

switch

- მნიშვნელობის დაბრუნების კონსტრუქცია:

return

- ფაილის PHP კოდის შედგენილობაში ჩასმა:

require ()  
include ()  
require\_once ()  
include\_once ()

## პირობითი ოპერატორები

პირობითი ოპერატორი დაპროგრამების ყველა ენაში ყველაზე მეტად გავრცელებულ კონსტრუქციას წარმოადგენს. ახლა განვიხილოთ PHP ენის ძირითადი პირობითი ოპერატორები.

### კონსტრუქცია if

If კონსტრუქციის სინტაქსი C ენის If კონსტრუქციის ანალოგიურია:

```
<?php  
if (ლოგიკური გამოსახულება) ოპერატორი;  
?>
```

PHP-ს გამოსახულების თანახმად, if კონსტრუქცია ლოგიკურ გამოსახულებას შეიცავს. თუ ლოგიკური გამოსახულება ჭეშმარიტია (true), მაშინ if კონსტრუქციის შემდეგ მდგომი ოპერატორი შესრულდება, ხოლო თუ ლოგიკური გამოსახულება მცდარია (false), მაშინ if კონსტრუქციის შემდეგ მდგომი ოპერატორი არ შესრულდება. მოვიყვანოთ მაგალითი:

```
<?php  
if ($a > $b) echo "a-ს მნიშვნელობა b-ს მნიშვნელობაზე მეტია";  
?>
```

შემდეგ მაგალითში, თუ \$a ცვლადი არ უდრის ნულს, მაშინ გამოტანილი იქნება შემდეგი სტრიქონი „a-ს მნიშვნელობა ჭეშმარიტია (true)“:

```
<?php
if ($a) echo "a-ს მნიშვნელობა ჭეშმარიტია (true) ";
?>
```

შემდეგ მაგალითში, თუ \$a ცვლადი უდრის ნულს, მაშინ გამოტანილი იქნება შემდეგი სტრიქონი "a-ს მნიშვნელობა მცდარია (false):

```
<?php
if (!$a) echo "a-ს მნიშვნელობა მცდარია (false) ";
?>
```

ხშირად საჭირო ხდება ოპერატორების ბლოკის გამოყენება, რომელიც გარკვეული კრიტერიუმის შემდეგ უნდა შესრულდეს. ამ დროს ეს ოპერატორები აუცილებლად ფიგურულ ფრჩხილებში {...} უნდა მოთავსდეს, მაგალითად:

```
<?php
if ($a > $b) {
    echo "a მეტია b-ზე";
    $b = $a;
}
?>
```

თუ მოცემულ მაგალითში  $a > b$ , გამოიტანს შეტყობინებას "a მეტია b-ზე" და შემდეგ \$b ცვლადს \$a ცვლადის მნიშვნელობას

მიანიჭებს. შევნიშნოთ, რომ მოცემული ოპერატორები if კონსტრუქციის ტანში შესრულდება.

## კონსტრუქცია else

ზოგჯერ წარმოიშობა სიტუაცია, როცა ოპერატორის შესრულება არა მარტო if კონსტრუქციის პირობის შესრულების დროსაა საჭირო, არამედ მაშინაც, როდესაც if კონსტრუქციის პირობა არ სრულდება. მოცემულ შემთხვევაში else ინსტრუქციის გარეშე არაფერი გამოგვივა. მთლიანობაში ასეთ კონსტრუქციას if-else კონსტრუქციას უწოდებენ.

if-else კონსტრუქციის სინტაქსი შემდეგია:

```
if (ლოგიკური გამოსახულება)
ინსტრუქცია-1;
else
ინსტრუქცია-2;
```

if-else კონსტრუქცია შემდეგნაირად მოქმედებს: თუ ლოგიკური გამოსახულება ჭეშმარიტია, მაშინ სრულდება ინსტრუქცია-1, თუ მცდარია ინსტრუქცია-2. როგორც ყველა ენაში, აქაც შეიძლება else კონსტრუქციის გამოტოვება.

თუ ინსტრუქცია-1 ან ინსტრუქცია-2 რამდენიმე ბრძანებისაგან შედგება, მაშინ ისინი, როგორც ყოველთვის ფიგურულ ფრხილებში უნდა იყოს მოთავსებული. მაგალითად:

```
<?php
if ($a > $b) {
    echo "a მეტია b-ზე";
} else {
    echo "a არ არის მეტი b-ზე";
```

```
}  
?>
```

if-else კონსტრუქციას კიდეც ერთი ალტერნატიული სინტაქსი აქვს, რომელსაც ქვემოთ განვიხილავთ.

### კონსტრუქცია elseif

elseif არის if და else კონსტრუქციების კომბინაცია. ეს კონსტრუქცია if-else პირობითი კონსტრუქციის შესაძლებლობებს აფართოებს. მოვიყვანოთ კონსტრუქციის სინტაქსი:

```
if (ლოგიკური გამოსახულება-1):  
  ოპერატორი-1;  
elseif (ლოგიკური გამოსახულება-2):  
  ოპერატორი-2;  
else  
  ოპერატორი-3;  
endif
```

როგორც წესი, შეიძლება elseif და else ბლოკების გამოტოვება. ქვემოთ მოყვანილია elseif კონსტრუქციის გამოყენების მაგალითი:

```
<?php  
if ($a > $b) {  
  echo "a მეტია b-ზე";  
} elseif ($a == $b) {  
  echo "a უდრის b";  
} else {  
  echo "a ნაკლებია b-ზე";
```

```
}  
?>
```

ზოგადად, კონსტრუქცია არც ისე მოსახერხებელია, რის გამოც მას იშვიათად იყენებენ.

## ციკლები PHP-ში

გამოყენების სიხშირის მიხედვით ციკლი მეორე ადგილზეა პირობითი კონსტრუქციის შემდეგ.

ციკლი საშუალებას იძლევა სხვადასხვა ოპერატორი განსაზღვრული (ზოგჯერ კი განუსაზღვრელი - როდესაც ციკლის მუშაობა რაიმე პირობაზეა დამოკიდებული) რაოდენობით გავიმეოროთ. მოცემულ ოპერატორებს ციკლის ტანი ეწოდება. ციკლის გავლას იტერაცია ეწოდება.

PHP-ში სამი სახის ციკლი გამოიყენება:

- ციკლი წინაპირობით (while);
- ციკლი შემდგომი პირობით (do-while);
- ციკლი მთვლელით (for);
- მასივის გადავსების სპეციალური ციკლი (foreach).

ციკლების მუშაობის დროს break და continue ოპერატორების გამოყენებაა შესაძლებელი. მათ შორის პირველი მთლიანი ციკლის, ხოლო მეორე - მხოლოდ მიმდინარე იტერაციის შესრულებას წყვეტს.

განვიხილოთ PHP-ს ციკლები:

### ციკლი წინაპირობით while

ციკლი წინაპირობით while შემდეგი პრინციპით მუშაობს:

1. ლოგიკური გამოსახულების მნიშვნელობის გამოთვლა;



2. თუ მნიშვნელობა ჭეშმარიტია, მაშინ შესრულდება ციკლის ტანი, წინააღმდეგ შემთხვევაში - გადავდივართ ციკლის შემდეგ მდგომ ოპერატორზე.

ციკლის სინტაქსი წინაპირობით შემდეგია:

```
while (ლოგიკური_გამოსახულება)
ინსტრუქცია;
```

მოცემულ შემთხვევაში ციკლის ტანს ინსტრუქცია წარმოადგენს. ჩვეულებრივ, ციკლის ტანი მრავალი ოპერატორისაგან შედგება. მოვიყვანოთ მაგალითი:

```
<?php
$x=0;
while ($x++<10) echo $x;
// გამოიტანს 12345678910
?>
```

აქ პირობის ოპერაციის  $x++<10$  შესრულების მიმდევრობას უნდა მიექცეს ყურადღება. ჯერ მოწმდება პირობა და მხოლოდ ამის შემდეგ იზრდება ცვლადის მნიშვნელობა. თუ ინკრემენტის ოპერაციას ცვლადის წინ დავწერთ ( $++x<10$ ), მაშინ ჯერ ცვლადის მნიშვნელობა გაიზრდება და მხოლოდ ამის შემდეგ მოხდება შედარება. შედეგად, მივიღებთ შემდეგ სტრიქონს: 123456789. ეს ციკლი შეიძლება სხვანაირადაც ჩაგვეწერა:

```
<?php
$x=0;
while ($x<10)
{
$x++; // მთვლელის გაზრდა
```

```
echo $x;  
}  
// გამოიტანს 12345678910  
?>
```

თუ ჩვენ მთვლელს echo ოპერატორის შემდეგ გავზრდით, მაშინ 0123456789 სტრიქონს მივიღებთ. ნებისმიერ შემთხვევაში ჩვენ 10 იტერაცია გვაქვს. იტერაცია არის ციკლის ტანში შესრულებული ოპერატორები.

პირობითი if ოპერატორის კონსტრუქციის მსგავსად, ციკლის while ოპერატორის ტანში შესაძლებელია ოპერატორის დაჯგუფება, თუ შემდეგ ალტერნატიულ სინტაქსს გამოვიყენებთ:

```
while (ლოგიკური_გამოსახულება):  
ინსტრუქცია;  
...  
endwhile;
```

ალტერნატიული სინტაქსის გამოყენების მაგალითი:

```
<?php  
$x = 1;  
while ($x <= 10):  
    echo $x;  
    $x++;  
endwhile;  
?>
```

## ციკლი შემდგომი პირობით do while

While ციკლისაგან განსხვავებით, ეს ციკლი გამოსახულების მნიშვნელობის შემოწმებას არა დასაწყისში, არამედ ყოველი გავლის შემდეგ ახდენს. ამგვარად, ციკლის ტანი ერთხელ მაინც სრულდება. ციკლის შემდგომი პირობით სინტაქსი ასეთია:

```
do
{
ციკლის_ტანი;
}
while (ლოგიკური_გამოსახულება);
```

ყოველი მომდევნო იტერაციის შემდეგ მოწმდება ლოგიკური გამოსახულების ჭეშმარიტება, თუ იგი ჭეშმარიტია, მართვა კვლავ ციკლის დასაწყისს გადაეცემა, ხოლო წინააღმდეგ შემთხვევაში ციკლი წყდება.

PHP-ში do-while ციკლის ალტერნატიული სინტაქსი გათვალისწინებული არ არის.

მაგალითი:

```
<?php
$x = 1;
do {
    echo $x;
} while ($x++<10);
?>
```

განხილული სცენარი გამოიტანს: 12345678910

## ციკლი მთვლელობით for

ციკლი მთვლელობით ციკლის ტანის რამდენიმეჯერ შესასრულებლად გამოიყენება. for ციკლის მეშვეობით შეიძლება (აუცილებელიცაა) ისეთი კონსტრუქცია შეიქმნას, რომელიც არატრივიალურ მოქმედებებს შეასრულებს.

for ციკლის სინტაქსი შემდეგია:

```
for (მაინიციალიზებული_ბრძანება; ციკლის_პირობა;  
იტერაციის_შემდგომი_ბრძანება) { ციკლის_ტანი; }
```

for ციკლი მუშაობას მაინიციალიზებული\_ბრძანების შესრულებით იწყებს. ბრძანების მონაცემები მხოლოდ ერთხელ სრულდება. ამის შემდეგ ციკლის\_პირობა მოწმდება, თუ იგი ჭეშმარიტია (true), მაშინ ციკლის\_ტანი შესრულდება. მას შემდეგ, რაც ციკლის ტანის ბოლო ოპერატორი შესრულდება, სრულდება იტერაციის\_შემდგომი\_ბრძანება. შემდეგ ისევ მოწმდება ციკლის\_პირობა. თუ იგი ისევ ჭეშმარიტია (true), მაშინ ისევ იწყება თავიდან და ა. შ.

```
<?php  
for ($x=0; $x<10; $x++) echo $x;  
?>
```

მოცემული სცენარი გამოიტანს: 0123456789 სტრიქონს.  
არის 12345678910 სტრიქონის გამოტანის ვარიანტიც:

```
<?php  
for ($x=0; $x++<10;) echo $x;  
// გამოიტანს 12345678910  
?>
```

მოცემულ მაგალითში მთვლელის გაზრდა ლოგიკური გამოსახულების შემოწმების დროს ვუზრუნველყავით. ამ შემთხვევაში ჩვენ იტერაციის შემდგომი ბრძანება აღარ გვჭირდება.

თუ ციკლში რამდენიმე ბრძანების შესრულებაა საჭირო, მაშინ ისინი, ეს ბრძანებები და ციკლის პირობა ერთმანეთისაგან წერტილ-მძიმით უნდა გამოვყოთ, თვით ბრძანებები კი - მძიმეებით:

```
<?php
for ($x=0, $y=0; $x<10; $x++, $y++) echo $x;
// გამოიტანს 0123456789
?>
```

ქვემოთ მოვიყვანოთ for ციკლში რამდენიმე ბრძანების გამოყენების უფრო პრაქტიკული მაგალითი:

```
<?php
for($i=0,$j=0,$k="წერტილი"; $i<10; $j++, $i+=$j) { $k=$k.". "; echo $k; }
// გამოიტანს წერტილი.წერტილი..წერტილი...წერტილი....
?>
```

განხილული მაგალითის რეალიზება while ოპერატორითაც შეიძლება, მაგრამ ეს არ იქნება ისე მოხერხებული და ლაკონური.

for ციკლისათვისაც არსებობს ალტერნატიული სინტაქსი:

```
for (მანიციალიზებული_ბრძანება; ციკლის_პირობა;
იტერაციის_შემდგომი_ბრძანება): ოპერატორები;
endfor;
```

## მასივების გადავსების ციკლი foreach

PHP4-ში გამოჩნდა კიდევ ერთი სპეციალური ტიპის ციკლი - foreach. მოცემული ციკლი სპეციალურად მასივების გადასარჩევადაა განკუთვნილი.

ციკლის სინტაქსი შემდეგნაირად გამოიყურება:

```
foreach (მასივი as $გასაღები=>$მნიშვნელობა)  
ბრძანებები;
```

აქ ბრძანებები მასივის თითოეული ელემენტისათვის ციკლურად სრულდება, ამასთან, შემდგომი გასაღები=>მნიშვნელობა წყვილი \$გასაღები და \$მნიშვნელობა ცვლადებში გაჩნდება. მოვიყვანოთ foreach ციკლის მუშაობის მაგალითი:

```
<?php  
$names["ბერიძე"]="ლამა";  
$names["მიქელაძე"]="ნიკა";  
$names["გელოვანი"]="კახა";  
$names["კვესელავა"]="გიორგი";  
foreach ($names as $key => $value) {  
echo "<b>$value $key</b><br>";  
}  
?>
```

განხილული სცენარი გამოიტანს:

ლაშა ბერიძე  
ნიკა მიქელაძე  
კახა გელოვანი  
გიორგი კვესელავა

Foreach ციკლს სხვა ფორმაც აქვს, რომლის გამოყენება შეიძლება, მაშინ, როდესაც მომდევნო ელემენტის გასაღების მნიშვნელობა არ გვაინტერესებს. ეს ასე გამოიყურება:

```
foreach (მასივი as $მნიშვნელობა)  
    ბრძანებები;
```

ამ შემთხვევაში მხოლოდ მასივის მომდევნო ელემენტის მნიშვნელობაა მისაწვდომი. ეს გამოსადეგი შეიძლება იყოს, მაგალითად, მასივ-სიებთან მუშაობის დროს.

```
<?php  
$names[]="ლაშა";  
$names[]="ნიკა";  
$names[]="კახა";  
$names[] = "გიორგი";  
foreach ($names as $value) {  
    echo "<b>$value</b><br>";  
}  
?>
```

ამ შემთხვევაში შედეგს შემდეგი სახე ექნება:

ლაშა  
ნიკა  
კახა  
გიორგი

ყურადღება: Foreach ციკლი საწყის მასივზე კი არ ახდენს ცვლილებებს, არამედ მის ასლზე. ეს ნიშნავს, რომ ნებისმიერი ცვლილება, რომელიც მასივში ხორციელდება, ციკლის ტანიდან „არ ჩანს“. რაც საშუალებას გვაძლევს, მაგალითად, მასივის სახით არა მარტო ცვლადი გამოვიყენოთ, არამედ რაიმე ფუნქციის მუშაობის შედეგი, რომელიც მასივს აბრუნებს (ამ შემთხვევაში ფუნქცია მხოლოდ ერთხელ იქნება გამოძახებული - ციკლის დაწყებამდე, ხოლო შემდეგ მუშაობა დაბრუნებული მნიშვნელობის ასლთან გაგრძელდება).

### **კონსტრუქცია break**

ხშირად, იმისათვის, რომ რაიმე რთული ციკლის ლოგიკა გავამარტივოთ, მოსახერხებელია შემდეგი იტერაციის მსვლელობის დროს მისი შეწყვეტის საშუალება გვქონდეს (მაგალითად, რაიმე განსაკუთრებული პირობის შესრულების დროს). სწორედ ამისთვის არსებობს კონსტრუქცია break, რომელიც ციკლიდან გამოსვლას მცის ასრულებს. იგი შეიძლება ერთ არააუცილებელ პარამეტრთან ერთად იყოს მოცემული - ეს არის რიცხვი, რომელიც მიუთითებს, თუ ერთმანეთში ჩალაგებული რომელი ციკლიდან უნდა განხორციელდეს გამოსვლა. ჩუმათობის პრინციპით



გამოიყენება რიცხვი 1 ანუ მიმდინარე ციკლიდან გამოსვლა. break კონსტრუქციის სინტაქსია:

```
break; // ჩუმათობის პრინციპით  
break (ციკლის_ნომერი); // ერთმანეთში ჩალაგებული  
ციკლისათვის (შესაწყვეტი ციკლის ნომერი)
```

მაგალითი:

```
<?php  
$x=0;  
while ($x++<10) {  
if ($x==3) break;  
echo "<b>იტერაცია $x</b><br>";  
}  
// როდესაც $x უდრის 3, ციკლი შეწყდება  
?>
```

განხილული სცენარი გამოიტანს და შემდეგ შეწყდება ციკლი:

```
იტერაცია 1  
იტერაცია 2
```

თუ რომელიმე განსაზღვრული (ჩალაგებული) ციკლის შეწყვეტა გვსურს, მაშინ break კონსტრუქციას უნდა მივაწოდოთ

პარამეტრი - ციკლის ნომერი, მაგალითად, break (1). ციკლების ნუმერაცია შემდეგნაირად გამოიყურება:

```
for (...) // მესამე ციკლი
{
  for (...) // მეორე ციკლი
  {
    for (...) // პირველი ციკლი
    {
    }
  }
}
```

### კონსტრუქცია continue

კონსტრუქცია continue, როგორც break ციკლის კონსტრუქციასთან ერთად „წყვილში“ მუშაობს. იგი მომენტალურად ასრულებს ციკლის მიმდინარე იტერაციას და გადადის ახალზე. აქაც შეიძლება მითითებული იყოს ერთმანეთში ჩალაგებული ერთი ან რამდენიმე ციკლის ნომერი.

ძირითადად continue კოდი ფიგურული ფრჩხილების ეკონომიის საშუალებას იძლევა და მისი წაკითხვის მოხერხებულობას ზრდის. ეს განსაკუთრებით საჭირო ხდება ციკლიტრებში, როდესაც გარკვეული რაოდენობის ობიექტები უნდა გადაირჩეს და მათ შორის ამოირჩეს ის, რომელიც გარკვეულ პირობას აკმაყოფილებს. ქვემოთ მოგვყავს continue კონსტრუქციის გამოყენების მაგალითი:

```
<?php
$x=0;
```

```
while ($x++<5) {  
if ($x==3) continue;  
echo "<b>იტერაცია $x</b><br>";  
}  
// ციკლი მხოლოდ მესამე იტერაციაზე შეწყდება და შემდეგ ისევ  
გაგრძელდება  
?>
```

განხილული სკრიპტი გამოიტანს:

```
იტერაცია 1  
იტერაცია 2  
იტერაცია 4  
იტერაცია 5
```

გონივრულად გამოყენებული break და continue კონსტრუქციის კოდები ერთმანეთში ჩალაგებული else ბლოკების წაკითხვას მნიშვნელოვნად ამარტივებს.

## ამორჩევის კონსტრუქციები

ხშირად, მიმდევრობით განლაგებული რამდენიმე if-else კონსტრუქციის მაგივრად, ამორჩევის სპეციალური switch-case კონსტრუქციის გამოყენება მიზანშეწონილი. მოცემული კონსტრუქცია მითითებული გამოსახულების მნიშვნელობის მიხედვით არჩევს მოქმედებას. switch-case კონსტრუქცია რაღაცით მოგვაგონებს if-else კონსტრუქციას, რომელიც არსით მისი ანალოგია. ამორჩევის კონსტრუქციის გამოყენება მიზანშეწონილია მაშინ,

როდესაც სავარაუდო პასუხი ბევრია, მაგალითად, ხუთს აღმატება, და ამასთან, თითოეული ვარიანტისათვის სხვადასხვა სპეციფიკური მოქმედება უნდა შესრულდეს. ამ შემთხვევაში, მართლაც if-else კონსტრუქციის გამოყენება მოუხერხებელი იქნება.

switch-case კონსტრუქციის სინტაქსი შემდეგია:

```
switch (გამოსახულება) {
case მნიშვნელობა1: ბრძანება1; [break;]
case მნიშვნელობა2: ბრძანება2; [break;]
...
case მნიშვნელობაN: ბრძანებაN; [break;]
[default: ბრძანება_ჩუმათობის_პრინციპით; [break]]
}
```

switch-case კონსტრუქციის მუშაობის პრინციპი ასეთია:

1. გამოითვლება გამოსახულების მნიშვნელობა;

2. დათვალიერდება მნიშვნელობები. დავუშვათ, პირველ ბიჯზე გამოთვლილი გამოსახულების მნიშვნელობაა მნიშვნელობა*i*. თუ კონსტრუქციაში არ არის გამოყენებული ოპერატორი break, მაშინ შესრულდება ბრძანებები *i*, *i+1*, *i+2*, ... , *N*, წინააღმდეგ შემთხვევაში (თუ გამოყენებულია ოპერატორი break) კი - მხოლოდ ბრძანება ნომრით *i*.

3. თუ გამოთვლილი გამოსახულების მნიშვნელობა არც ერთ მნიშვნელობას არ დაემთხვა, მაშინ სრულდება default ბლოკი (თუ ის მითითებულია).

მოვიყვანოთ switch-case კონსტრუქციის გამოყენების მაგალითი:

```
<?php
$x=1;
```

```

// ვიყენებთ if-else
if ($x == 0) {
    echo "x=0<br>";
} elseif ($x == 1) {
    echo "x=1<br>";
} elseif ($x == 2) {
    echo "x=2<br>";
}
// ვიყენებთ switch-case
switch ($x) {
case 0:
    echo "x=0<br>";
    break;
case 1:
    echo "x=1<br>";
    break;
case 2:
    echo "x=2<br>";
    break;
}
?>

```

განხილული სცენარი ორჯერ გამოიტანს x=1. ქვემოთ განვიხილოთ კიდევ ერთი მაგალითი:

```

<?php
$x="ვაშლი";
switch ($x) {
case "ვაშლი":
    echo "ეს ვაშლია";

```

```

break;
case "მსხალი":
    echo "ეს მსხალია";
    break;
case "საზამთრო":
    echo "ეს საზამთროა";
    break;
}
?>

```

მოცემული სკრიპტი გამოიტანს ტექსტს „ეს ვაშლია“.

თუ არ არის გამოყენებული ოპერატორი break, მაშინ სკრიპტს ექნება შემდეგი სახე:

```

<?php
$x=0;
switch ($x) {
case 0:
    echo "x=0<br>";
case 1:
    echo "x=1<br>";
case 2:
    echo "x=2<br>";
}
// break ოპერატორის გამოყენების გარეშე გამოიტანს
// x=0
// x=1
// x=2
?>

```

Case-სათვის ოპერატიული სია შეიძლება გამოტოვებული იყოს, ამ შემთხვევაში მართვა უბრალოდ შემდეგ case კონსტრუქციამდე ოპერატიულ სიას გადაეცემა:

```
<?php
switch ($x) {
case 0:
case 1:
case 2:
    echo "x ნაკლებია 3-ზე, მაგრამ არ არის უარყოფითი";
    break;
case 3:
    echo "x=3";
}
?>
```

მოვიყვანოთ default ბლოკის გამოყენების მაგალითი:

```
<?php
$x=3;
switch ($x) {
case 0:
    echo "x=0";
    break;
case 1:
    echo "x=1";
    break;
case 2:
    echo "x=2";
    break;
```

```
default:
    echo "x არ უდრის 0, 1 ან 2";
}
?>
```

ვინაიდან მოცემული სკრიპტში ცვლადი  $x=3$ , შედეგს ექნება შემდეგი სახე:

```
x არ უდრის 0, 1 ან 2
```

switch-case კონსტრუქციას ასევე აქვს ალტერნატიული სინტაქსი:

```
switch(გამოსახულება):
case მნიშვნელობა1: ბრძანება1; [break;]
...
case მნიშვნელობაN: ბრძანებაN; [break;]
[default: ბრძანება_ჩუმათობის_პრინციპით; [break;]]
endswitch;
```

ქვემოთ მოგვყავს switch-case კონსტრუქციის ალტერნატიული სინტაქსის გამოყენების მაგალითი:

```
<?php
$x=3;
switch ($x):
```



```
case 0:
    echo "x=0";
    break;
case 1:
    echo "x=1";
    break;
case 2:
    echo "x=2";
    break;
default:
    echo "x არ უდრის 0, 1 ან 2";
endswitch;
?>
```

როგორც მიხვდით, ვინაიდან  $x=3$  მოცემული სკრიპტი იგივე შედეგს მოგვცემს.

## მნიშვნელობის დაბრუნების კონსტრუქცია

### კონსტრუქცია return

return კონსტრუქცია, უმეტეს შემთხვევაში, მომხმარებლის ფუნქციიდან აბრუნებს მნიშვნელობას, როგორც ფუნქციური მოთხოვნის პარამეტრს. return კონსტრუქციის გამოძახების შემთხვევაში მომხმარებლის ფუნქციის შესრულება წყდება, ხოლო return კონსტრუქცია განსაზღვრულ მნიშვნელობას აბრუნებს.

თუ return კონსტრუქცია გლობალური არიდან იქნება გამოძახებული (არა მომხმარებლის ფუნქციიდან), მაშინ სკრიპტი ასევე დაამთავრებს მუშაობას, ხოლო return კონსტრუქცია ასევე დააბრუნებს განსაზღვრულ მნიშვნელობას.

დაბრუნებული მნიშვნელობა შეიძლება იყოს ნებისმიერი ტიპის, მათ შორის სია ან ობიექტი. მნიშვნელობის დაბრუნება იწვევს ფუნქციის შესრულების დამთავრებას და მართვა კოდის იმ სტრიქონს გადაეცემა, საიდანაც ეს ფუნქცია იყო გამოძახებული.

ქვემოთ მოყვანილია integer ტიპის მნიშვნელობის დასაბრუნებლად return კონსტრუქციის გამოყენების მაგალითი:

```
<?php
function retfunct()
{
    return 7;
}
echo retfunct(); // გამოიტანს '7'.
?>
```

return კონსტრუქციის მიერ მასივის დაბრუნების მაგალითი:

```
<?php
function numbers()
{
    return array (0, 1, 2);
}
list ($zero, $one, $two) = numbers();
echo $zero;
echo $one;
echo $two;
// გამოიტანს '012'
?>
```

იმისათვის, რომ ფუნქციამ დააბრუნოს შედეგები ბმულით, აუცილებელია & ოპერატორის გამოყენება როგორც ფუნქციის

აღწერის, ასევე ცვლადისათვის დასაბრუნებელი მნიშვნელობის მინიჭების დროს:

```
<?php
function &returns_reference()
{
    return $someref;
}

$newref =& returns_reference();
?>
```

როგორც ვხედავთ, კონსტრუქცია ძალზე მოსახერხებელია მომხმარებლის ფუნქციებში გამოსაყენებლად.

# ფუნქციები

PHP-ს ე. წ. ჩაშენებული ფუნქციების სრულყოფილი ნაკრები აქვს. ამ ფუნქციების გარდა PHP თავის მომხმარებელს საშუალებას აძლევს თავისი საკუთარი ფუნქციები შექმნას. PHP ფუნქცია გულისხმობს რაიმე ამოცანის გადასაწყვეტად ჩაწერილი ოპერაციების მიმდევრობას, რომელსაც შეიძლება გარკვეული სახელით - ფუნქციის სახელით - მიემართოთ, როგორც ბლოკს.

შეიძლება ყველასთვის კარგად ნაცნობი ეკრანზე გამოტანის ოპერაცია echo განიხილო როგორც ფუნქცია. ზოგადად ფუნქციის გამოძახება ხდება ფუნქციის სახელით, რომელსაც მრგვალ ფრჩხილებში ჩასმული პარამეტრების სია მოსდევს.

## ფუნქციის სახელი (პარამეტრი1, . . . )

ზოგიერთ ფუნქციას პარამეტრები არ სჭირდება. ამ დროს ფრჩხილებში არაფერი არ იწერება. რამდენიმე პარამეტრის არსებობის შემთხვევაში ისინი ერთმანეთისაგან მძიმით გამოიყოფიან.

PHP-ის ფუნქციების შესახებ ინფორმაცია შეიძლება მოიპოვოთ [www.php.net](http://www.php.net) საიტზე (იხ. მთავარი გვერდის documentation ჩანართი). PHP-ში ჩაშენებული ფუნქციების საერთო რაოდენობა რამდენიმე ასეულის ტოლია, ამიტომ მიზანშეწონილია მათი კატეგორიებად დაყოფა. დასაწყისში ალბათ საკმარისი იქნება შემდეგი ჯგუფის ფუნქციების გაცნობა:

- მათემატიკური ფუნქციები;
- სტრიქონული ფუნქციები;
- მასივებთან მუშაობის ფუნქციები;
- ფაილებთან სამუშაო ფუნქციები;
- თარიღისა და დროის ფუნქციები;
- საფოსტო ფუნქციები;
- გამოსახულებებთან სამუშაო ფუნქციები;

- MySQL-ის ფუნქციები (ამ ფუნქციებს გავეცნობით მონაცემთა ბაზების განხილვის შემდეგ, ცალკე თავში).

ფუნქციების სტრუქტურული კლასიფიკაცია გაცილებით ვრცელია და თითოეულ ჯგუფშიც ბევრად მეტი ფუნქცია შედის, ვიდრე ამ სახელმძღვანელოში იქნება განხილული, მაგრამ ჩვენ მხოლოდ ძირითადი და შედარებით პოპულარული ფუნქციების განხილვით შემოვიფარგლეთ.

## PHP-ის მათემატიკური ფუნქციები

### მათემატიკური კონსტანტების ცხრილი

კონსტანტა	მნიშვნელობა	აღწერა
M_PI	3.14159265358979323846	რიცხვი $\pi$
M_E	2.7182818284590452354	ეილერის რიცხვი $e$
M_LOG2E	1.4426950408889634074	$\log_2 e$
M_LOG10E	0.43429448190325182765	$\lg e$
M_LN2	0.69314718055994530942	$\ln 2$
M_LN10	2.30258509299404568402	$\ln 10$
M_PI_2	1.57079632679489661923	$\pi/2$
M_PI_4	0.78539816339744830962	$\pi/4$
M_1_PI	0.31830988618379067154	$1/\pi$
M_2_PI	0.63661977236758134308	$2/\pi$
M_SQRTPI	1.77245385090551602729	$\sqrt{\pi}$
M_2_SQRTPI	1.12837916709551257390	$2/\sqrt{\pi}$
M_SQRT2	1.41421356237309504880	$\sqrt{2}$
M_SQRT3	1.73205080756887729352	$\sqrt{3}$
M_SQRT1_2	0.70710678118654752440	$1/\sqrt{2}$
M_LNPI	1.14472988584940017414	$\ln \pi$
M_EULER	0.57721566490153286061	ეილერის მუდმივა

## მათემატიკური ფუნქციები

abs - რიცხვის მოდული;

acos - არკკოსინუსი;

acosh - ინვერსიული ჰიპერბოლური კოსინუსი;

asin - არკსინუსი;

asinh - ინვერსიული ჰიპერბოლური სინუსი;

atan2 - ორი ცვლადის არკტანგენსი (ამ ფუნქციას აქვს სახე atan2(x,y), შედეგად აბრუნებს x/y განაყოფის არკტანგენს რადიანებით);

atan - არკტანგენსი;

atanh - ინვერსიული ჰიპერბოლური ტანგენსი;

base\_convert - რიცხვი გადაჰყავს ერთი ბაზიდან მეორეში;

bindec - რიცხვი გადაჰყავს ათვლის ორობითი სისტემიდან ათობითში;

ceil - წილადს ამრგვალებს მეტობით;

cos - კოსინუსი;

cosh - ჰიპერბოლური კოსინუსი;

decbin - რიცხვის გადაყვანა ათვლის ათობითი სისტემიდან ორობითში;

dechex - რიცხვის გადაყვანა ათვლის ათობითი სისტემიდან თექვსმეტობითში;

decoct - რიცხვის გადაყვანა ათვლის ათობითი სისტემიდან რვაობითში;

deg2rad - გრადუსებით გამოსახული კუთხის რადიანებით გამოსახვა;

exp - გამოთვლის ექსპონენტა e (ნატურალური ლოგარითმის ფუძე);

expm1 - აბრუნებს  $\exp(\text{რიცხვი})-1$ , გამოთვლები ზუსტია, მაშინაც კი, როცა რიცხვის მნიშვნელობა ნულს უახლოვდება;

floor - წილადს ამრგვალებს ნაკლებობით;

fmod - აბრუნებს გაყოფის შედეგად მიღებულ ნაშთს;

getrandmax - მაქსიმალურად შესაძლებელი შემთხვევითი რიცხვი;

hexdec - რიცხვის გადაყვანა ათვლის თექვსმეტობითი სისტემიდან ათობით სისტემაში;

hypot - მართკუთხა სამკუთხედში ჰიპოტენუზის სიგრძის გამოთვლა;

is\_finite - განსაზღვრავს, მოცემული მნიშვნელობა არის თუ არა დასაშვები საბოლოო რიცხვი;

is\_infinite - განსაზღვრავს, მნიშვნელობა უსასრულობაა თუ არა;

is\_nan - განსაზღვრავს, მნიშვნელობა არის თუ არა რიცხვი;

lcg\_value - გაერთიანებული წრფივი congruential გენერატორი;

log10 - ლოგარითმი 10-ის ფუძით;

log1p - შედეგად აბრუნებს  $\log(1 + \text{რიცხვი})$ , გამოთვლები ზუსტია, მაშინაც კი როცა რიცხვის მნიშვნელობა ნულს უახლოვდება;

log - ნატურალური ლოგარითმი;

max - უდიდესი მნიშვნელობა;

min - უმცირესი მნიშვნელობა;

mt\_getrandmax - აჩვენებს რიცხვის უდიდეს შესაძლო შემთხვევით მნიშვნელობას;

mt\_rand - საუკეთესო შემთხვევითი რიცხვის გენერირება;

mt\_srand - შემთხვევითი რიცხვის საუკეთესო გენერატორის მომზადება;

octdec - რიცხვის გადაყვანა ათვლის რვაობითი სისტემიდან ათვლის ათობით სისტემაში;

pi - რიცხვი  $\pi$ ;

pow - ექსპონენციალური გამოსახულება;  
rad2deg - რადიანებით გამოსახული რიცხვის გრადუსებით გამოსახვა;  
rand - შემთხვევითი რიცხვების გენერირება;  
round - float ტიპის რიცხვების დამრგვალება;  
sin - სინუსი;  
sinh - ჰიპერბოლური სინუსი;  
sqrt - კვადრატული ფესვი რიცხვიდან;  
srand - ფსევდოშემთხვევითი რიცხვების გენერატორის საწყისი რიცხვის შეცვლა;  
tan - ტანგენსი;  
tanh - ჰიპერბოლური ტანგენსი.

## სტრიქონული ფუნქციები და ოპერატორები

### სტრიქონების შედარების ოპერატორები

== და != შედარების ოპერატორის გამოყენება სტრიქონების შესადარებლად არ არის რეკომენდებული, ვინაიდან იგი ტიპების გარდაქმნას მოითხოვს. მაგალითი:

```
<?php
$x=0;
$y=1;
if ($x == "") echo "<p>x - ცარიელი სტრიქონია</p>";
if ($y == "") echo "<p>y - ცარიელი სტრიქონია</p>";
// გამოიტანს:
// x - ცარიელი სტრიქონია
?>
```

მოცემული სკრიპტი გვატყობინებს, რომ \$x - ცარიელი სტრიქონია. ეს იმასთანაა დაკავშირებული, რომ ცარიელი



სტრიქონი (""), უპირველეს ყოვლისა, ნულად აღიქმება, ხოლო ამის შემდეგ - როგორც ცარიელი. PHP-ში ოპერანდები ერთმანეთს, როგორც სტრიქონები მხოლოდ იმ შემთხვევაში შედარდება, თუ ორივე სტრიქონია. წინააღმდეგ შემთხვევაში ისინი ერთმანეთს შედარდებიან როგორც რიცხვები. ამასთან, ნებისმიერი სტრიქონი, რომელიც PHP-ს რიცხვად ვერ გადაჰყავს (მათ შორის რიცხვი და ცარიელი სტრიქონი), აღიქმება, როგორც 0.

სტრიქონების შედარების მაგალითი:

```
<?php
$x="სტრიქონი";
$y="სტრიქონი";
$z="სტროფი";
if ($x == $z) echo "<p>სტრიქონი X უდრის სტრიქონ Z-ს</p>";
if ($x == $y) echo "<p>სტრიქონი X უდრის სტრიქონ Y-ს</p>";
if ($x != $z) echo "<p>სტრიქონი X არ უდრის სტრიქონ Z-ს</p>";
?>
```

გაუგებრობის თავიდან ასაცილებლად რეკომენდებულია სტრიქონების შედარების დროს გამოვიყენოთ ეკვივალენტობის ოპერატორი. ეკვივალენტობის ოპერატორი საშუალებას იძლევა ყოველთვის კორექტულად შედარდეს სტრიქონები, ვინაიდან სიდიდეებს ადარებს როგორც მნიშვნელობის, ისე ტიპის მიხედვით.

```
<?php
$x="სტრიქონი";
$y="სტრიქონი";
$z="სტროფი";
if ($x === $z) echo "<p>სტრიქონი X უდრის სტრიქონ Z-ს</p>";
```

```
if ($x === $y) echo "<p>სტრიქონი X უდრის სტრიქონ Y-ს</p>";  
if ($x !== $z) echo "<p>სტრიქონი X არ უდრის სტრიქონ Z-ს</p>";  
?>
```

ორივე სკრიპტის მუშაობის შედეგად მივიღებთ:

```
სტრიქონი X უდრის სტრიქონ Y-ს  
სტრიქონი X არ უდრის სტრიქონ Z-ს
```

## სტრიქონებთან სამუშაო ფუნქციები

PHP-ში სტრიქონებთან სამუშაოდ ბევრი ფუნქცია გამოგვადგება. განვიხილოთ ზოგიერთი მათგანი.

### ბაზური სტრიქონული ფუნქცია

**ფუნქცია strlen(string \$st)**

ერთ-ერთი ყველაზე გამოსადეგი ფუნქციაა. უბრალოდ, შედეგად სტრიქონის სიგრძეს ანუ მასში შემავალი სიმბოლოების რაოდენობას აბრუნებს. სტრიქონი შეიძლება ნებისმიერ სიმბოლოს შეიცავდეს, მათ შორის ნულოვანი კოდით (რაც C ენაში აკრძალულია). მაგალითი:

```
$x = "Hello!";  
echo strlen($x); // გამოიტანს 6
```

## ფუნქცია `strpos(string $where, string $what[, int $fromwhere=0])`

`$where` სტრიქონში `$what` ქვესტრიქონს (სიმბოლოთა მიმდევრობა) ეძებს და წარმატების შემთხვევაში შედეგად აბრუნებს ამ ქვესტრიქონის პოზიციას (ინდექსს) მოცემულ სტრიქონში. `$fromwhere` არააუცილებელი პარამეტრია, გამოიყენება იმ შემთხვევაში, როდესაც შედარებას სხვა პოზიციიდან ვიწყებთ. თუ ქვესტრიქონის პოვნა ვერ მოხდა, მაშინ შედეგი `false` იქნება. მაგალითი:

```
echo strpos("Hello","el"); // გამოიტანს 1-ს
```

კიდევ ერთი მაგალითი:

```
if (strpos("Norway","rwa") !== false) echo "Norway სტრიქონშიარის  
rwa სტრიქონი";  
// შედარების დროს ეკვივალენტობის ოპერატორები (===) (!==)  
გამოიყენეთ.
```

## ფუნქცია `substr(string $str, int $start [,int $length])`

მოცემული ფუნქციაც ძალიან ხშირად გამოიყენება. მისი დანიშნულებაა - შედეგად `$str` სტრიქონის `$start` პოზიციიდან დაწყებული და `$length` სიგრძის ნაწილი დააბრუნოს. თუ `$length` მოცემული არ არის, მაშინ მოიაზრება `$str` სტრიქონის ნაწილი `$start` პოზიციიდან დაწყებული სტრიქონის ბოლომდე. თუ `$start`-ის მნიშვნელობა სტრიქონის სიგრძეს აღემატება ან `$length` ნულის ტოლია, მაშინ შედეგად ცარიელი სტრიქონი დაბრუნდება. ამის გარდა, ამ ფუნქციას კიდევ სხვა დანიშნულების შესრულებაც შეუძლია. მაგალითად, თუ `$start`-ს უარყოფით მნიშვნელობას მივანიჭებთ, მაშინ ინდექსის ათვლა `$str`-ის ბოლოდან დაიწყება. (მაგალითად, -1 აღნიშნავს სტრიქონის ბოლო სიმბოლოს). `$length` პარამეტრიც შეიძლება იყოს უარყოფითი. მაგალითად,

```
$str = "Programmer";  
echo substr($str,0,2); // გამოიტანს Pr  
echo substr($str,-3,3); // გამოიტანს mer
```

### ფუნქცია `strcmp(string $str1, string $str2)`

ორ სტრიქონს ერთმანეთს სიმბოლოებად (უფრო სწორად ბაიტურად) ადარებს და შედეგად აბრუნებს: 0-ს, თუ სტრიქონები ზუსტად ემთხვევა ერთმანეთს; -1-ს, თუ `$str1` სტრიქონი ლექსიკოგრაფულად `$str2` სტრიქონზე ნაკლებია; და 1, პირიქით, თუ `$str1` „მეტა“ `$str2`-ზე. ვინაიდან, შედარება ბაიტურად მიდის, ამიტომ სიმბოლოთა რეგისტრი შედეგზე მოქმედებს.

### ფუნქცია `strcasecmp(string $str1, string $str2)`

იგივეა, რაც `strcmp()`, ოღონდ მისი მუშაობის დროს სიმბოლოთა რეგისტრი არ გაითვალისწინება. მაგალითად, ამ ფუნქციის მიხედვით „ab“ და „AB“ ერთმანეთის ტოლია.

## ტექსტის ბლოკებთან სამუშაო ფუნქციები

ქვემოთ ჩამოთვლილი ფუნქციების გამოყენება მოსახერხებელია მაშინ, როდესაც ერთი და იგივე ოპერაციები სტრიქონულ ცვლადებში მოცემულ მრავალსტრიქონიანი ტექსტის ბლოკებთანაა ჩასატარებელი.

### ფუნქცია `str_replace(string $from, string $to, string $str)`

`$str` სტრიქონში შემავალ `$from` (რეგისტრის გათვალისწინებით) ქვესტრიქონს `$to` ქვესტრიქონით ცვლის და აბრუნებს შედეგს. ამასთან, მესამე პარამეტრის სახით მოცემული საწყისი სტრიქონი არ იცვლება. ეს ფუნქცია უფრო სწრაფად მუშაობს, ვიდრე PHP-ის რეგულარულ გამოსახულებებთან მუშაობისათვის განკუთვნილი `ereg_replace()` ფუნქცია. მაგალითად, ასე შეიძლება

სტრიქონის გადაყვანის ყველა სიმბოლო შევცვალოთ მისი HTML-ის ეკვივალენტური <br> ტეგით:

```
$st=str_replace("\n","<br>\n",$str)
```

როგორც ვხედავთ, ის, რომ <br>\n სტრიქონში თუ კვლავ არის სტრიქონის გადაყვანის სიმბოლო, ფუნქციის მუშაობაზე არავითარ გავლენას არ ახდენს, ანუ ფუნქცია სტრიქონში მხოლოდ ერთჯერად გავლას ახდენს. აღწერილი ამოცანის გადასაწყვეტადაც nl2br() ფუნქცია გამოიყენება, რომელიც ცოტათი უფრო სწრაფად მუშაობს.

**ფუნქცია stringnl2br(string \$string)**

ახალი სტრიქონის ყველა \n სიმბოლოს <br>\n სიმბოლოთი შეცვლის და შედეგს დააბრუნებს. საწყისი სტრიქონი არ შეიცვლება.

**ფუნქცია WordWrap(string \$str, int \$width, string \$break)**

ეს ფუნქცია, რომელიც პირველად PHP 4-ში გამოჩნდა, ძალზე გამოსადეგია, მაგალითად, წერილის ტექსტის დაფორმატების დროს ადრესატისათვის mail() საშუალებით მისი ავტომატური გაგზავნის წინ. იგი \$str ტექსტის ბლოკს რამდენიმე სტრიქონად დაყოფს, რომლებიც \$break სიმბოლოებით დამთავრდება, ისე, რომ ერთ სტრიქონში \$width სიმბოლოზე მეტი სიმბოლო არ აღმოჩნდეს. დაყოფა სიტყვების საზღვრის მიხედვით მოხდება, ისე, რომ ტექსტი თავისუფლად იკითხებოდეს. მიღებული სტრიქონი დაბრუნდება \$break-ში მითითებული სტრიქონის გადაყვანის სიმბოლოს ჩამატებით. მაგალითი:

```
<?php
$str = "ეს არის ელექტრონული წერილის ტექსტი, რომელიც
ადრესატს უნდა გაეგზავნოს ...";
// ტექსტი დავყოთ 20-20 სიმბოლოებად
```

```
$str = WordWrap ($str, 20, "<br>");  
echo $str;  
// გამოიტანს:  
/* ეს არის ელექტრონული  
წერილის ტექსტი,  
რომელიც ადრესატს  
უნდა გაეგზავნოს ... */  
?>
```

### ფუნქცია strip\_tags (string \$str [, string \$allowable\_tags])

სტრიქონებთან მომუშავე კოდევ ერთი გამოსადეგი ფუნქცია. ეს ფუნქცია სტრიქონიდან ყველა ტეგს ამოშლის და შედეგს აბრუნებს. \$allowable\_tags პარამეტრში შეიძლება ის ტეგები ჩამოვთვალოთ, რომელთა ამოშლაც სტრიქონიდან არ გვინდა. ისინი ინტერვალის გარეშე უნდა იყოს ჩამოთვლილი. მაგალითი:

```
$stripped = strip_tags ($str); // სტრიქონიდან (ტექსტიდან) ყველა  
html-ტეგს წაშლის  
$stripped = strip_tags($str, "<head><title>"); // ყველა html-ტეგს  
წაშლის, გარდა <head> და <title> ტეგისა
```

### ცალკეულ სიმბოლოებთან სამუშაო ფუნქციები

PHP-ში, როგორც დაპროგრამების სხვა ენებში, შესაძლებელია სტრიქონის ცალკეულ სიმბოლოებთან მუშაობა.

სტრიქონის ცალკეულ სიმბოლოებს შეიძლება მისი ინდექსით მივმართოთ:

```
$str = "HTML";  
echo $str[0]; // გამოიტანს 'H'
```

### ფუნქცია `chr(int $code)`

მოცემული ფუნქცია გამოიტანს სიმბოლოსაგან შედგენილ სტრიქონს, რომლის კოდია `$code`. მაგალითი:

```
echo chr(75); //გამოიტანს K
```

### ფუნქცია `ord($char)`

მოცემული ფუნქცია გამოიტანს `$char` სიმბოლოს კოდს. მაგალითი:

```
echo ord('A'); // გამოიტანს 65 - 'A' სიმბოლოს კოდს
```

## ჰარების წასაშლელი ფუნქცია

ხშირად ისეთი მომხმარებელი გვხვდება, რომ ტექსტში ზედმეტ ჰარებს (ინტერვალი) სვამს, ან სხვადასხვა შეცდომას უშვებს. ზედმეტი ჰარებისაგან თავის დაცვა ძალზე მარტივია და PHP-ში ამისათვის სპეციალური ფუნქცია არსებობს. ეს ფუნქცია, მისთვის მიწოდებული სტრიქონების მოცულობისაგან დამოუკიდებლად, ძალზე სწრაფად მუშაობს.

### ფუნქცია `trim(string $str)`

შედეგად `$str` სტრიქონის ასლს აბრუნებს, რომელშიც წაშლილია საწყისი და საბოლოო ჰარები. აქ ჰარები გულისხმობს: სიმბოლო ჰარს " ", სტრიქონის გადაყვანის სიმბოლო `\n`-ს, სტრიქონის თავში დაბრუნების სიმბოლო (`<Home>` კლავიში) `\r`-ს და ტაბულაციის სიმბოლო `\t`-ს. მაგალითად, `trim("test\n")` ფუნქცია შედეგად დააბრუნებს "test" სტრიქონს. მისი გამოყენება შეიძლება ყველგან, სადაც შეცდომითი ჰარის არსებობის სულ მცირე ეჭვი მაინც გვაქვს, ვინაიდან ეს ფუნქცია ძალიან სწრაფად მუშაობს.

### ფუნქცია `ltrim(string $st)`

იგივეა, რაც trim(), ოღონდ მხოლოდ საწყის ჰარებს შლის, ხოლო საბოლოოს ხელს არ ახლებს. შედარებით იშვიათად გამოიყენება.

### **ფუნქცია chop(string \$st)**

მხოლოდ საბოლოო ჰარებს შლის, ხოლო საწყისს ჰარებს ხელს არ ახლებს.

## **სიმბოლოთა გარდაქმნის ფუნქციები**

Web-დაპროგრამება - ერთ-ერთი იმ სფეროთაგანია, რომელშიც მუდმივად გვიხდება სტრიქონებით მანიპულირება: მათი გაწყვეტა, ჰარების დამატება და წაშლა, სხვადასხვა კოდირების შეცვლა, და ბოლოს, URL- კოდირება და დეკოდირება. PHP-ში ყოველივე ამის ხელით რეალიზება, სწრაფქმედების თვალსაზრისით, უბრალოდ შეუძლებელია. ამიტომაც, არსებობს შესაბამისი ფუნქციები.

### **ფუნქცია strtr(string \$str, string \$from, string \$to)**

მართალია, ეს ფუნქცია არც ისე ხშირად გამოიყენება, მაგრამ ზოგჯერ ძალზე გამოსადეგია. იგი \$str სტრიქონის ყველა იმ სიმბოლოს, რომელიც \$from სტრიქონშიც გვხვდება ცვლის მისი შესაბამისი „მეწყვილე“ სიმბოლოთი \$to სტრიქონიდან.

შემდეგი რამდენიმე ფუნქცია სწრაფი URL-კოდირებისა და დეკოდირებისათვისაა განკუთვნილი.

URL-კოდირება აუცილებელია მონაცემთა ინტერნეტით გადასაცემად. მაგალითად, ასეთი კოდირება მიზანშეწონილია, თუ თქვენ სკრიპტის პარამეტრის სახით ქართულენოვან ინფორმაციას გადასცემთ. ასევე მსგავსი კოდირება შეიძლება საჭირო გახდეს ფაილის გადაცემის დროს, რათა ზოგიერთი სერვერის მიერ



8 ბიტანი კოდირების მხარდაჭერის არ ქონის გამო არ მოხდეს კოლიზია. აი ეს ფუნქციებიც:

### **ფუნქცია `urlencode(string $str)`**

ფუნქცია `$str` სტრიქონის URL-კოდირებას ახდენს და აბრუნებს შედეგს. ამ ფუნქციის გამოყენება მოსახერხებელია მაშინ, როდესაც მაგალითად, რომელიმე სცენარში დინამიკური ბმულის `<a href=...>` ფორმირება გვსურს, მაგრამ არ ვიცით მისი პარამეტრები მხოლოდ ალფაბეტურ-ციფრულ სიმბოლოებს შეიცავს თუ არა. ამ შემთხვევაში ეს ფუნქცია ასე უნდა გამოვიყენოთ:

```
echo "<a href=/script.php?param=".urlencode($UserData);
```

ახლა, თუ `$UserData` ცვლადი `=`, `&` სიმბოლოებს ან ჰარებს რომც შეიცავდეს, სცენარს მაინც კორექტული მონაცემები გადაეცემა.

### **ფუნქცია `urldecode(string $st)`**

ეს ფუნქცია სტრიქონის URL-დეკოდირებას ახდენს. იგი იშვიათად გამოიყენება, ვინაიდან PHP შემოსული მონაცემების ავტომატურ დეკოდირებას ახდენს.

### **ფუნქცია `rawurlencode(string $st)`**

თითქმის `urlencode()` ფუნქციის ანალოგიურია, მაგრამ ჰარებს `+-ად` კი არ გარდაქმნის, როგორც ეს ფორმებიდან მონაცემების გადაცემის დროს ხდება, არამედ აღიქმება, როგორც არაალფაბეტურ-ციფრული სიმბოლოები.

### **ფუნქცია `rawurldecode(string $st)`**

`urldecode()` ფუნქციის ანალოგიურია, მაგრამ `+-ს`, როგორც ჰარს არ აღიქვამს.

### **ფუნქცია `htmlspecialchars(string $str)`**

ეს ფუნქცია, ჩვეულებრივ, echo-სთან კომბინაციაში გამოიყენება. მისი ძირითადი დანიშნულებაა გარანტირება იმისა, რომ გამოსატანი სტრიქონის არც ერთი მონაკვეთი ტეგად არ აღქმება.

ზოგიერთ სიმბოლოს (როგორცაა ამპერსანტი, ბრჭყალები და „მეტობისა“ და „ნაკლებობის“ ნიშნები) სტრიქონში მისი HTML-ის შესაბამისი კოდებით ცვლის ისე, რომ გვერდზე ჩვეულებრივად გამოიყურებოდეს.

### **ფუნქცია StripSlashes(string \$str)**

\$str სტრიქონში დახრილი ხაზებს მისი ეკვივალენტური სიმბოლოებით ცვლის. ეს ეხება შემდეგ სიმბოლოებს: ", ', \.

### **ფუნქცია AddSlashes(string \$str)**

დახრილ ხაზებს მხოლოდ ', " და \ სიმბოლოების წინ სვამს.

## **რეგისტრის შესაცვლელი ფუნქცია**

ზოგჯერ შეიძლება დაგვჭირდეს ზოგიერთი სტრიქონის ზედა რეგისტრში გადაყვანა, ანუ დიდი ასოებით ჩაწერა. ძირითადად, ამ მიზნით strtolower() ფუნქციის გამოყენებაც შეიძლება, მაგრამ იგი შედარებით ნელა მუშაობს. PHP-ში არის ფუნქციები, რომლებიც სპეციალურად ამ მიზნისათვის გამოიყენება. აი ისინიც:

### **ფუნქცია strtolower(string \$str)**

სტრიქონს ქვედა რეგისტრში გადაიყვანს.

მარტივ პროგრამებში, ასევე იმ შემთხვევაში თუ არა ვართ დარწმუნებული მხარს უჭერს თუ არა მოცემული ოპერაციული სისტემა შესაბამის ლოკალს, უფრო მარტივი იქნება სიმბოლოების გადასაყვანად strtolower() ფუნქცია გამოვიყენოთ:

```
$st=strtr($st, "ABCDEFGHIJKLMNOPQRSTUVWXYZ",  
"abcdefghijklmnopqrstuvwxyz");
```

მოცემული საშუალების მთავარი ღირსებაა ის, რომ კოდირების პრობლემის შემთხვევაში სცენარის ქმედითუნარიანობის აღდგენა ძალზე მარტივად მოხდება: საჭიროა იგი იმავე კოდირებით გარდავექმნათ, რომლითაც დოკუმენტები სერვერზე ინახება.

### **ფუნქცია strtoupper(string \$str)**

სტრიქონი ზედა რეგისტრში გადაჰყავს.

### **ლოკალის მომართვა (ლოკალური მონაცემების მომართვა)**

ლოკალს ვუწოდებთ სისტემის ლოკალური მონაცემების ერთობლიობას, როგორცაა თარიღისა და დროის ფორმატი, ენა, კოდირება.

ლოკალის მომართვა დიდადაა დამოკიდებული ოპერაციულ სისტემაზე.

ლოკალის მომართვისათვის ფუნქცია SetLocale() გამოიყენება:

### **ფუნქცია SetLocale(string \$category, string \$locale)**

ეს ფუნქცია მომართავს მიმდინარე ლოკალს, რომელთანაც სიმბოლოების რეგისტრის გარდაქმნის, თარიღისა და დროის გამოტანის და სხვ. ფუნქციები იმუშავებს. ანუ ზოგადად, თითოეული კატეგორიის ფუნქციისათვის ლოკალი ცალკე უნდა განისაზღვროს და იგი სხვადასხვანაირად გამოიყურება. სახელდობრ, SetLocale() ფუნქციის გამოძახება, თუ რომელი

კატეგორიის ფუნქციას შეეხება, \$category პარამეტრით იქნება მოცემული. მან შეიძლება შემდეგი სტრიქონული მნიშვნელობები მიიღოს:

LC\_CTYPE - ზედა/ქვედა რეგისტრში გადაყვანის ფუნქციისთვის მითითებული ლოკალის აქტივაციას ახდენს;

LC\_NUMERIC - წილადი რიცხვების დაფორმატების ფუნქციისთვის (კერძოდ, რიცხვებში მთელი და წილადი ნაწილების გამყოფ სიმბოლოს განსაზღვრავს) ლოკალის აქტივაციას ახდენს;

LC\_TIME - თარიღისა და დროის გამოტანის ფორმატს განსაზღვრავს ჩუმათობის პრინციპით;

LC\_ALL - ყველა ზემოთ ჩამოთვლილ რეჟიმს აყენებს.

რაც შეეხება \$locale პარამეტრს, როგორც ცნობილია, სისტემაში დაყენებულ თითოეულ ლოკალს აქვს თავისი უნიკალური სახელი, რომლითაც მას შეიძლება მიმართო. სწორედ, ამ პარამეტრით ხდება ამ ფაქტის ფიქსირება. ამ წესიდან ორი გამონაკლისი არსებობს. პირველი, თუ \$locale სიდიდე "" ცარიელი სტრიქონის ტოლია, მაშინ დაყენდება ის ლოკალი, რომელიც გლობალური ცვლადის სახელშია მითითებული და რომელიც \$category კატეგორიაში მითითებულ სახელს ემთხვევა. მეორე, თუ პარამეტრში მითითებულია 0, მაშინ დგება არა ახალი ლოკალი, არამედ, უბრალოდ, მითითებული რეჟიმისათვის მიმდინარე ლოკალის სახელი დაბრუნდება.

სამწუხაროდ, ლოკალის სახელები ოპერაციული სისტემის მომართვის დროს უნდა დადგეს და ამისათვის რაიმე სტანდარტები არ არსებობს.

სინამდვილეში, ლოკალებთან მუშაობა ძალზე არაპროგნოზირებადია და ოპერაციული სისტემების მიერ ძნელად გადასატანი ქმედებაა, ამიტომ, თუ სცენარები ძალიან დიდი არ არის, მაშინ

უმჯობესია როგორმე შემოვლითი გზები მოვძებნოთ, მაგალითად, გამოვიყენოთ `strstr()` ფუნქცია და არა ლოკალი.

## გამოტანის ბუფერის გასუფთავების ფუნქცია

### ფუნქცია `flush()`

ამ ფუნქციას სტრიქონთან მუშაობასთან პირდაპირი კავშირი არა აქვს, მაგრამ სხვა ფუნქციებიდანაც შორს დგას.

`echo`-ს გამოყენების დროს მონაცემები კლიენტს პირდაპირ არასდროს არ გადაეცემა. იგი ჯერ სპეციალურ ბუფერში გროვდება, რათა შემდეგ მათი ტრანსპორტირება ერთად მოხდეს. გადაცემა ასე უფრო დაჩქარდება.

მაგრამ, ზოგჯერ მომხმარებლისათვის ბუფერიდან ინფორმაციის ვადაზე ადრე გაგზავნის საჭიროება დგება, მაგალითად, თუ რაიმე ინფორმაციის რეალურ დროში მიწოდება ხდება (ასე ხდება ჩეთებში მუშაობა). ზუსტად, აქ გამოგადგებათ `flush()` ფუნქცია, რომელიც ბუფერის შიგთავსს მომხმარებლის ბრაუზერში მაშინვე გადააგზავნის.

## სტრიქონულ ტიპად გარდაქმნა

თქვენ შეგიძლიათ მნიშვნელობები სტრიქონულ ტიპად გარდაქმნათ, თუ (`string`) აღწერას ან `strval()` ფუნქციას გამოიყენებთ.

გამოსახულებებში, სადაც აუცილებელია სტრიქონის გამოყენება, გარდაქმნა ავტომატურად ხდება. ეს მაშინ, როდესაც `echo()` ან `print()` ფუნქციებს ვიყენებთ ან, როდესაც ცვლადის მნიშვნელობას სტრიქონს ვადარებთ. შეიძლება აგრეთვე, ფუნქცია `settype()` გამოვიყენოთ. მისი სინტაქსია:

**ფუნქცია `settype (mixed var, string type)`**

var ცვლადს type ტიპს ანიჭებს. type-ის შესაძლო მნიშვნელობებია:

- "boolean" (ან PHP 4.2.0-დან დაწყებული, "bool")
- "integer" (ან "int")
- "float" (ან "double")
- "string"
- "array"
- "object"
- "null" (PHP 4.2.0-დან დაწყებული)

წარმატების შემთხვევაში შედეგად აბრუნებს TRUE-ს; წინააღმდეგ შემთხვევაში FALSE. მაგალითად:

```
$foo = "5bar"; // string
$bar = true; // boolean
settype($foo, "integer"); // $foo გახდება 5 (integer)
settype($bar, "string"); // $bar გახდება "1" (string)
```

ბულის ტიპის (boolean) მნიშვნელობა TRUE გარდაიქმნება სტრიქონად "1", ხოლო მნიშვნელობა FALSE, როგორც " "(ცარიელი სტრიქონი). ამ წესით მნიშვნელობების გარდაქმნა ორივე მიმართულებით შეიძლება - ბულის ტიპიდან სტრიქონად და პირიქით.

მთელი (integer) ან მცოცავმძიმანი რიცხვი (float) სტრიქონად გარდაიქმნება ისე, რომ სტრიქონი მოცემულ რიცხვში გამოყენებული ციფრებისაგან (მცოცავმძიმანი რიცხვის შემთხვევაში კი - ხარისხის მაჩვენებლის ჩათვლით) იქნება შედგენილი.

მასივი ყოველთვის გარდაიქმნება "Array" ტიპის სტრიქონად, ისე, რომ echo() ან print() გამოყენების შემთხვევაში (array) მასივის შიგთავსის გასაცნობად შიგთავსს მასში ერთიანად ვერ ავსახავთ.

ერთი ელემენტი რომ ვნახოთ, ამისათვის საჭიროა `echo $arr['foo']` ბრძანების მსგავსი ბრძანება გამოვიყენოთ.

ობიექტი ყოველთვის "Object" სტრიქონად გარდაიქმნება. თუ მოთხოვნილი ობიექტის კლასის სახელის მიღება გვსურს, `get_class()` ფუნქცია უნდა გამოვიყენოთ.

რესურსი ყოველთვის შემდეგი სტრუქტურის სტრიქონად გარდაიქმნება: "Resource id #1", სადაც 1 რესურსის (resource) უნიკალური ნომერია, რომელსაც მას PHP შესრულების მომენტში ანიჭებს. თუ გსურთ რესურსის ტიპის მიღება, მაშინ `get_resource_type()` ფუნქცია უნდა გამოვიყენოთ.

NULL ყოველთვის ცარიელ სტრიქონად გარდაიქმნება.

მასივის, ობიექტის ან რესურსის გამოტანა მათი მნიშვნელობების შესახებ არავითარ სასარგებლო ინფორმაციას არ იძლევა. მნიშვნელობების გამართვისათვის გამოტანის უფრო მისაღები საშუალებაა `print_r()` და `var_dump()` ფუნქციების გამოყენება.

PHP-ს მნიშვნელობების გარდაქმნა შესაძლებელია მისი მუდმივად შენახვის მიზნითაც. ამ მეთოდს სერიალიზაცია ეწოდება და შეიძლება `serialize()` ფუნქციის მეშვეობით შესრულდეს. ამის გარდა, თუ დაყენებულ PHP-ში გვაქვს WDDX-ს მხარდაჭერა, მაშინ PHP-ს მნიშვნელობების სერიალიზაცია შეიძლება XML სტრუქტურაში მოვახდინოთ.

## მონაცემთა ტიპის მინიჭება და ინდიკაცია

### ფუნქცია `settype (mixed var, string type)`

ზემოთ უკვე ავღნიშნეთ, რომ PHP-ში ცვლადის ტიპის წინასწარ აღწერა არ არის აუცილებელი. ცვლადზე მინიჭებული

მნიშვნელობის მიხედვით ინტერპრეტატორი მის ტიპს თავად განსაზღვრავს. მაგრამ ზოგჯერ ცვლადის ტიპის ცხადად განსაზღვრა საჭირო. ამისათვის, იგივე `settype()` ფუნქცია გამოიყენება.

### **ფუნქცია `gettype (mixed var)`**

ზოგჯერ საჭირო ხდება ცვლადის ტიპის გაგება, რისთვისაც სპეციალური ფუნქცია `gettype (mixed var)` გამოიყენება. მისი პარამეტრის სახით იმ ცვლადის სახელი მიეთითება, რომლის ტიპის დადგენაც გვინტერესებს. ამ სკრიპტის შესრულების შედეგად მიიღება შემდეგი პასუხები: ან `boolean`, ან `integer`, ან `double`, ან `string` იმისდა მიხედვით, თუ რომელი ტიპის მონაცემთან გვაქვს იმ მომენტში საქმე.

### **სტრიქონის რიცხვად გარდაქმნა**

თუ სტრიქონი აღიქმება, როგორც რიცხვითი მნიშვნელობა, მაშინ მისი საშედეგო მნიშვნელობა და ტიპი შემდეგნაირად განისაზღვრება:

სტრიქონი აღქმული იქნება, როგორც `float`, თუ იგი შეიცავს '.', 'e', 'E' სიმბოლოთაგან ერთ-ერთს. წინააღმდეგ შემთხვევაში იგი აღიქმება, როგორც მთელი რიცხვი.

მნიშვნელობა სტრიქონის საწყისი ნაწილით განისაზღვრება. თუ სტრიქონი მართალი რიცხვითი მნიშვნელობით იწყება, მაშინ ეს მნიშვნელობა იქნება გამოყენებული. წინააღმდეგ შემთხვევაში მნიშვნელობა იქნება 0 (ნული). მართალი რიცხვითი მნიშვნელობა - ეს არის ერთი ან მეტი ციფრი (შეიძლება ათობით წერტილს შეიცავდეს), სურვილის მიხედვით რიცხვის ნიშნით, ასევე არააუცილებელი ხარისხის მაჩვენებლით. ხარისხის მაჩვენებელი ეს არის 'e' ან 'E', რომელსაც მოსდევს ერთი ან მეტი ციფრი.



```

<?php
$foo = 1 + "10.5";
echo "\$foo=$foo; ტიპი: " . gettype ($foo) . "<br />\n";
$foo = 1 + "-1.3e3";
echo "\$foo=$foo; ტიპი: " . gettype ($foo) . "<br />\n";
$foo = 1 + "bob-1.3e3";
echo "\$foo=$foo; ტიპი: " . gettype ($foo) . "<br />\n";
$foo = 1 + "bob3";
echo "\$foo=$foo; ტიპი: " . gettype ($foo) . "<br />\n";
$foo = 1 + "10 Small Pigs";
echo "\$foo=$foo; ტიპი: " . gettype ($foo) . "<br />\n";
$foo = 4 + "10.2 Little Piggies";
echo "\$foo=$foo; ტიპი: " . gettype ($foo) . "<br />\n";
$foo = "10.0 pigs " + 1;
echo "\$foo=$foo; ტიპი: " . gettype ($foo) . "<br />\n";
$foo = "10.0 pigs " + 1.0;
echo "\$foo=$foo; ტიპი: " . gettype ($foo) . "<br />\n";
$foo = "Text";
echo "\$foo=$foo; ტიპი: " . gettype ($foo) . "<br />\n";
$foo = TRUE;
echo "\$foo=$foo; ტიპი: " . gettype ($foo) . "<br />\n";
?>

```

შედეგად შემდეგ პასუხს მივიღებთ:

```
Sfoo=11.5; ტიპი: double
Sfoo=-1299; ტიპი: double
Sfoo=1; ტიპი: integer
Sfoo=1; ტიპი: integer
Sfoo=11; ტიპი: integer
Sfoo=14.2; ტიპი: double
Sfoo=11; ტიპი: double
Sfoo=11; ტიპი: double
Sfoo=Text; ტიპი: string
Sfoo=1; ტიპი: boolean
```

## მასივებთან სამუშაო ფუნქციები

განვიხილოთ მასივებთან სამუშაო ზოგიერთი ხშირად გამოსაყენებელი ფუნქცია.

### ფუნქცია list()

დავუშვათ, გვაქვს სამი ელემენტისაგან შემდგარი მასივი:

```
$names[0]="ალექსანდრე";
$names[1]="ნიკოლოზი";
$names[2]="იაკობი";
```

დავუშვათ, რაღაც მომენტში ამ მასივის ელემენტების მნიშვნელობის შესაბამისად სამი ცვლადისათვის \$alex, \$nick, \$yakov გადაცემა დაგვჭირდა. ეს, შეიძლება ასე გავაკეთოთ:

```
$alex = $names[0];
$nick = $names[1];
$yakov = $names[2];
```

თუ მასივი დიდია, მაშინ მასივის ელემენტების მინიჭების ეს მეთოდი არც ისე მოსახერხებელია. ამისათვის, უფრო რაციონალური მეთოდი არსებობს - list() ფუნქციის გამოყენება:

```
list ($alex, $nick, $yakov) = $names;
```

თუ ჩვენ მხოლოდ "ნიკოლოზი" და "იაკობი" გვჭირდება, მაშინ ეს შეიძლება ასე გაკეთდეს:

```
list (, $nick, $yakov) = $names;
```

### ფუნქცია array()

ფუნქცია array() სპეციალურად მასივის შესაქმნელად გამოიყენება. ამასთან, იგი ცარიელი მასივის შექმნის საშუალებასაც იძლევა. ქვემოთ array() ფუნქციის გამოყენების მაგალითებია მოყვანილი:

```
<?php
// ცარიელ მასივს ქმნის:
$arr = array();

// სამ ელემენტთან სიას ქმნის. ინდექსი ნულით იწყება:
$arr2 = array("ბერიძე", "კახაძე", "მელაძე");

// სამ ელემენტთან ასოციურ მასივს ქმნის:
$arr3 = array("ბერიძე" => "ზაზა", "კახაძე" => "ნიკა", "მელაძე" => "ლევან
ი");

// მრავალგანზომილებიან ასოციურ მასივს ქმნის:
$arr4 = array("name" => "ბერიძე", "age" => "24", "email" => "beridze@gmail.com");
$arr4 = array("name" => "კახაძე", "age" => "34", "email" => "kaxadze@gmail.com");
$arr4 = array("name" => "მელაძე", "age" => "47", "email" => "meladze@gmail.com");
```

```
.com");  
?>
```

## ოპერაციები მასივებზე

### მასივის ელემენტების მოწესრიგება (დახარისხება)

დავიწყოთ ყველაზე მარტივით - მასივის მოწესრიგებით. ამისათვის PHP-ში მრავალი ფუნქცია არსებობს. მათი საშუალებით ასოციური მასივის და სიების მოწესრიგება შეიძლება ზრდადობით ან კლებადობით, ან ისეთი მიმდევრობით, როგორც მომხმარებელს ესაჭიროება.

### მასივის ელემენტების მნიშვნელობის მიხედვით

#### მოწესრიგება `asort()` და `arsort()` ფუნქციების გამოყენებით:

`asort()` ფუნქცია მასში პარამეტრად მითითებული მასივის ელემენტებს აწესრიგებს ალფაბეტის (თუ ეს სტრიქონია) ან ზრდადობის (რიცხვებისათვის) მიხედვით. ამასთან, გასაღებსა და მის შესაბამის ელემენტს შორის კავშირი შენარჩუნებული იქნება. მაგალითად:

```
<?php  
$A=array("a"=>"Zero","b"=>"Weapon","c"=>"Alpha","d"=>"Processor");  
asort($A);  
foreach($A as $k=>$v) echo "$k=>$v ";  
?>
```

მოცემული სკრიპტი მასივის ელემენტების მიმდევრობას შეცვლის, მაგრამ დამოკიდებულება გასაღები => მნიშვნელობა არ დაირღვევა.

```
c=>Alpha d=>Processor b=>Weapon a=>Zero
```

ფუნქცია `arsort()` იგივე მოქმედებას ასრულებს, ოღონდ მოწესრიგება კლებადობის მიხედვით ხდება.

**მასივის ელემენტების გასაღების მიხედვით მოწესრიგება**  
**`ksort()` და `krsort()` ფუნქციების გამოყენებით:**

ფუნქცია `ksort()` პრაქტიკულად `arsort()` ფუნქციის იდენტურია, მხოლოდ ერთი განსხვავებით, იგი მასივის ელემენტს აწესრიგებს გასაღების მიხედვით (ზრდადობით). მაგალითად:

```
$A=array("d"=>"Zero", "c"=>"Weapon", "b"=>"Alpha", "a"=>"Processor");  
ksort($A);  
for(Reset($A); list($k,$v)=each($A); echo "$k=>$v ";
```

```
a=>Processor b=>Alpha c=>Weapon d=>Zero
```

მასივის ელემენტების გასაღების მიხედვით კლებადობით დალაგებას `krsort()` ფუნქცია ახდენს.

## მასივის ელემენტების გასაღების მიხედვით მოწესრიგება uksort() ფუნქციის გამოყენებით:

მასივის ელემენტების მოწესრიგება ხშირად უფრო რთული კრიტერიუმების მიხედვით შეიძლება. მაგალითად, დავუშვათ \$Files-ში ინახება საქალაქებში არსებული ფაილებისა და ქვესაქალაქების სია. შეიძლება დაგვჭირდეს ამ სიის არა მარტო ლექსიკოგრაფიული მიმდევრობით გამოტანა, არამედ ისე, რომ საქალაქების სია ფაილების სიას უსწრებდეს. ამ შემთხვევაში უნდა გამოვიყენოთ uksort() ფუნქცია, რომლის წინაც ორი პარამეტრით შედარების ფუნქცია უნდა ჩაიწეროს, როგორც ამას uksort() მოითხოვს.

```
<?php
// ამ ფუნქციამ უნდა შეადაროს $f1 და $f2 მნიშვნელობები და
შედეგად დააბრუნოს:
// -1, თუ $f1<$f2,
// 0, თუ $f1==$f2
// 1, თუ $f1>$f2
// <და> გულისხმობს გამოსატან სიაში ამ სახელების მიმდევრობას
function FCmp($f1,$f2)
{ // საქალაქადე უსწრებს ფაილს
if(is_dir($f1) && !is_dir($f2)) return -1;
// ფაილი ყოველთვის მოსდევს საქალაქებს
if(!is_dir($f1) && is_dir($f2)) return 1;
// თუ არა ლექსიკოგრაფიული შედარება ხდება
if($f1<$f2) return -1; elseif($f1>$f2) return 1; else return 0;
}
// დავუშვათ $Files შეიცავს მასივს გასაღებით - ფაილის
სახელებით
```

```
// მიმდინარე საქალაქო დავალაგოთ.  
uksort($Files,"FCmp"); // „ბმულით“ გადავცემთ მოწესრიგების  
ფუნქციას  
?>
```

რა თქმა უნდა, შენარჩუნებული იქნება გასაღებსა და uksort() ფუნქციის მნიშვნელობებს შორის კავშირი.

### **მასივის ელემენტების მნიშვნელობების მიხედვით მოწესრიგება uasort() ფუნქციის გამოყენებით**

ფუნქცია uasort() ძალიან ჰგავს uksort() ფუნქციას, იმ განსხვავებით, რომ მომხმარებლის მიერ შექმნილ დალაგების ფუნქციაში მასივის მომდევნო მნიშვნელობები გამოიყენება და არა გასაღებები. ამასთან, კავშირი გასაღები => მნიშვნელობა წყვილებს შორის ინახება.

### **მასივის გადაბრუნება array\_reverce() ფუნქციის გამოყენებით**

ფუნქცია შედეგად აბრუნებს მასივს, რომლის ელემენტები პარამეტრში მითითებული მასივის ელემენტების საპირისპირო მიმდევრობითაა დალაგებული. ამასთან, კავშირი გასაღებებსა და მნიშვნელობებს შორის არ იკარგება. მაგალითად, იმის მაგივრად, რომ მოვახდინოთ მასივის დალაგება საპირისპირო მიმართულებით arsort() ფუნქციის მეშვეობით, შეგვიძლია იგი ჯერ დავალაგოთ და შემდეგ გადავაბრუნოთ:

```
$A=array("a"=>"Zero","b"=>"Weapon","c"=>"Alpha","d"=>"Processor");  
asort($A);  
$A=array_reverse($A);
```

რა თქმა უნდა, მითითებული მიმდევრობა უფრო დიდხანს მუშაობს, ვიდრე `arsort()` ფუნქციის ერთხელ გამოძახება.

### სიის მოწესრიგება (დალაგება) `sort()` და `rsort()` ფუნქციების მეშვეობით

ეს ორი ფუნქცია პირველ რიგში სიების დასალაგებლად არის განკუთვნილი.

ფუნქცია `sort()` სიას დაალაგებს ზრდადობის მიხედვით, ხოლო `arsort()` ფუნქცია - კლებადობის მიხედვით. ქვემოთ მოყვანილია ფუნქცია `sort()`-ის მაგალითი:

```
<?php
$A=array("40", "20", "10", "30");
sort($A);
for($i=0; $i<count($A); $i++) echo "$A[$i]".<br>";
?>
```

```
10
20
30
40
```

### სიის გადანაცვლება `shuffle()` ფუნქციების მეშვეობით

ფუნქცია "გადანაცვლებს" სიის ელემენტებს ისე, რომ მათი მნიშვნელობების განაწილება შემთხვევითი იყოს. მაგალითად:

```
$A=array(10,20,30,40,50);
shuffle($A);
foreach($A as $v) echo "$v ";
```



კოდის მოცემული ფრაგმენტი 10, 20, 30, 40 და 50 რიცხვებს შემთხვევითი მიმდევრობით გამოიტანს.

```
50
10
40
20
30
```

თუ ამ ფრაგმენტს რამდენიმეჯერ შევასრულებთ, ვნახავთ, რომ გამოტანილი რიცხვების მიმდევრობა ყოველთვის ერთი და იგივე იქნება. ეს იმითაა განპირობებული, რომ ეს ფუნქცია შემთხვევითი რიცხვების სტანდარტულ გენერატორს იყენებს. ეს რომ არ მოხდეს, ამისათვის საჭიროა ყოველი გაშვების წინ `srand()` ფუნქციის მეშვეობით შემთხვევითი რიცხვების სტანდარტული გენერატორის ინიციალიზება.

### ოპერაციები მასივის გასაღებებსა და მნიშვნელობებზე

#### ფუნქცია `array_flip(array $arr)`

`array_flip(array $arr)` ფუნქცია მთელ მასივს გაივლის და მის გასაღებებსა და მნიშვნელობებს ადგილებს შეუცვლის. `$arr` საწყისი მასივი არ იცვლება, მხოლოდ საშედეგო მასივი ბრუნდება.

რა თქმა უნდა, თუ მასივში რამდენიმე ელემენტს ერთი და იგივე მნიშვნელობა აქვს, მაშინ გაითვალისწინება მხოლოდ მისი ბოლო მნიშვნელობა:

```
$A=array("a"=>"aaa", "b"=>"aaa", "c"=>"ccc");
$A=array_flip($A);
// ახლა შედეგი იქნება $A===array("aaa"=>"b", "ccc"=>"c");
```

### ფუნქცია `array_keys(array $arr [,mixed $SearchVal])`

`array_keys()` ფუნქცია აბრუნებს სიას, რომელიც `$arr` მასივის ყველა გასაღებს შეიცავს. თუ მასში მითითებულია `$SearchVal` არააუცილებელი პარამეტრი, მაშინ იგი მხოლოდ იმ გასაღებებს დააბრუნებს, რომელთაც `$SearchVal` მნიშვნელობები შეესაბამება.

ფაქტობრივად, ეს ფუნქცია მოცემული მეორე პარამეტრით, `array_flip(array $arr)` ოპერატორის შებრუნებულ ფუნქციას წარმოადგენს.

### ფუნქცია `in_array(mixed $val, array $arr)`

`in_array()` ფუნქცია შედეგად აბრუნებს TRUE-ს, თუ `$arr` მასივში `$val` მნიშვნელობის ელემენტი არის.

### ფუნქცია `array_count_values($List)`

ფუნქცია ითვლის, რამდენჯერ გვხვდება მოცემული მნიშვნელობა ამ სიაში და შედეგად აბრუნებს ასოციურ მასივს, რომლის გასაღები არის სიის ელემენტები, ხოლო მნიშვნელობა - ამ ელემენტების გამეორების რაოდენობა. სხვა სიტყვებით რომ ვთქვათ, `array_count_values()` ფუნქცია `$List` სიაში ამა თუ იმ მნიშვნელობათა სიხშირეს განსაზღვრავს. მაგალითი:

```
$List=array(1, "hello", 1, "world", "hello");
array_count_values($array);
// შედეგად აბრუნებს array(1=>2, "hello"=>2, "world"=>1)
```

## მასივების შერწყმა

მასივების შერწყმა (კონკატენაცია) არის მასივის შექმნის ოპერაცია, რომელიც რამდენიმე სხვა მასივის ელემენტებისაგან შედგება. მასივების შერწყმა ძალზე საშიში ოპერაციაა, ვინაიდან შერწყმის შედეგები თავისებურ ლოგიკას ექვემდებარება, რის გამოც შეიძლება ზოგიერთი მონაცემი დაიკარგოს. მასივების შერწყმის რეალიზება "+" ოპერატორის ან `array_merge()` ფუნქციის მეშვეობით ხდება. სიების შერწყმა აუცილებლად მხოლოდ `array_merge()` ფუნქციის მეშვეობით უნდა განხორციელდეს.

დავუშვათ გვაქვს ორი მასივი:

```
$A = array("1"=>"პირველი", "2"=>"მეორე");  
$B = array("1"=>"მესამე", "2"=>"მეოთხე");
```

ახლა, მოვახდინოთ ამ ორი მასივის ერთ მასივად შერწყმა \$C:

```
$C = $A + $B;
```

მასივებისათვის "+" ოპერატორი კომპუტატიური არ არის. ეს ნიშნავს, რომ `$A + $B` არ უდრის `$B + $A`-ს.

განხილული მაგალითის ფარგლებში მივიღებთ შემდეგი სახის \$C მასივს:

```
"1"=>"პირველი", "2"=>"მეორე", "3"=>"მესამე", "4"=>"მეოთხე"
```

ხოლო `$B + $A`-ს შედეგად - ასეთი მასივის:

```
"1"=>"მესამე", "2"=>"მეოთხე", "3"=>"პირველი", "4"=>"მეორე"
```

სიების შერწყმის დროს ასეთი მეთოდი არ მუშაობს. ავხსნათ ეს მაგალითის გამოყენებით. დავუშვათ გვაქვს ორი მასივი:

```
$A = array(10,11,12);  
$B = array(13,14,15);
```

\$A და \$B (\$A + \$B) სიების შერწყმის შედეგად მივიღებთ: 10, 11, 12. ეს კი სულაც არ არის ის შედეგი, რომლის მიღებაც გვინდოდა. ეს დაკავშირებულია იმასთან, რომ ერთნაირი ინდექსის მქონე სიების შერწყმის დროს, საშედეგო მასივში რჩება პირველი მასივის ელემენტები, ამასთან იმავე ადგილზე. ასეთი შემთხვევის დროს აუცილებელია array\_merge() ფუნქციის გამოყენება.

### ფუნქცია array\_merge()

ფუნქცია array\_merge()-ის ყველა იმ ნაკლის აღმოფხვრა შეუძლია, რომელიც "+" ოპერატორს გააჩნია. კერძოდ, იგი მის არგუმენტებში ჩამოთვლილი მასივების ერთ დიდ მასივად შერწყმას და შედეგის დაბრუნებას ახდენს. თუ მასივში ერთნაირი გასაღები გვხვდება, მაშინ შედეგში იმ მასივის წყვილი გასაღები => მნიშვნელობა განთავსდება, რომელიც არგუმენტების სიაში მარჯვნივაა განთავსებული. მაგრამ ეს რიცხვით გასაღებებს არ ეხება: ასეთი გასაღების მქონე ელემენტები ნებისმიერ შემთხვევაში საშედეგო მასივის ბოლოს განთავსდება.

ქვემოთ ორი სიის შერწყმის მაგალითია მოყვანილი:

```
$L1=array(100,200,300);  
$L2=array(400,500,600);  
$L=array_merge($L1,$L2);  
// შედეგი $L===array(100,200,300,400,500,600);
```

### მასივის ნაწილის მიღება

მასივის ნაწილის მისაღებად გამოიყენება array\_slice() ფუნქცია.

ფუნქცია array\_slice(array \$Arr, int \$offset [, int \$len])

ეს ფუნქცია, \$offset-ის მომდევნო ნომრიდან დაწყებული და \$len სიგრძის (თუ ეს პარამეტრი არ არის მითითებული, მაშინ მასივის ბოლომდე) გასაღები => მნიშვნელობა სახით ასოცირებული მასივის ნაწილს აბრუნებს. \$offset და \$len პარამეტრები შეიძლება იყოს როგორც დადებითი, ასევე უარყოფითი (ამ შემთხვევაში ათვლა მასივის ბოლოდან იწყება) ანალოგიური ფუნქციაა substr(). მაგალითი:

```
$input = array ("a", "b", "c", "d", "e");  
$output = array_slice ($input, 2); // "c", "d", "e"  
$output = array_slice ($input, 2, -1); // "c", "d"  
$output = array_slice ($input, -2, 1); // "d"  
$output = array_slice ($input, 0, 3); // "a", "b", "c"
```

## მასივის ელემენტების ჩამატება და წაშლა

ჩვენ უკვე რამდენიმე ოპერატორს ვიცნობთ, რომლის საშუალებითაც მასივის ელემენტების ჩამატება ან წაშლა შეიძლება. მაგალითად, ოპერატორი [ ] (ცარიელი კვადრატული ფრჩხილები) მასივის ბოლოს დაუმატებს ელემენტს, რომელსაც რიცხვით კოდს მიანიჭებს, ხოლო Unset() ოპერატორი გასაღებით მასთან ერთად საჭირო ელემენტს წაშლის. PHP ენა სხვა მრავალ ფუნქციასაც უჭერს მხარს, რომელთა გამოყენება ზოგჯერ უფრო მოსახერხებელია.

**ფუნქცია array\_push(alist &\$Arr, mixed \$var1 [, mixed \$var2, ...])**

ეს ფუნქცია \$Arr სიას \$var1, \$var2 და სხვ. ელემენტებს უმატებს. იგი მათ რიცხვით ინდექსებს ანიჭებს - ზუსტად ისე, როგორც ეს [ ] სტანდარტული ფუნქციისათვის ხდება. თუ თქვენ მხოლოდ ერთი ელემენტის დამატება გჭირდებათ, მაშინ, რა თქმა უნდა, ეს ოპერატორი უნდა გამოიყენოთ:

```
array_push($Arr,1000); // ვიძახებთ ფუნქციას...
$Arr[]=1000; // იგივეს შესრულება, ოღონდ მოკლედ
```

ყურადღება მიაქციეთ იმას, რომ array\_push() ფუნქცია მასივს აღიქვამს, როგორც სტეკს და ელემენტს ყოველთვის ბოლოში უმატებს. იგი მასივში ელემენტების ახალ რიცხვს აბრუნებს.

### ფუნქცია array\_pop(list&\$Arr)

array\_pop() ფუნქცია array\_push() ფუნქციის საპირისპირო ფუნქციას წარმოადგენს, სიის ბოლო ელემენტს წაშლის და აბრუნებს ახალ \$Arr მასივს. ამ ფუნქციის მეშვეობით შეგვიძლია ავავოთ კონსტრუქცია, რომელიც სტეკს მოგვაგონებს. თუ \$Arr სია ცარიელი იყო, მაშინ ეს ფუნქცია ცარიელ სტრიქონს დააბრუნებს.

```
<?php
$stack = array("ფორთოხალი", "ბანანი", "ვაშლი", "ჟოლო");
$fruit = array_pop($stack);
print_r($stack);
?>
```

```
Array ( [0] => ფორთოხალი [1] => ბანანი [2] => ვაშლი )
```

### ფუნქცია array\_unshift(list&\$Arr, mixed \$var1 [, mixed \$var2, ...])

array\_unshift() ფუნქცია array\_push() ფუნქციას ძალიან ჰგავს, ოღონდ ჩამოთვლილ ელემენტებს უმატებს არა ბოლოში, არამედ მასივის დასაწყისში. ამასთან, \$var1, \$var2 და ა. შ. მიმდევრობა იგივე დარჩება ანუ მასივის ელემენტები სიას მარცხნიდან ემატება. სიის

ახალ ელემენტებს, ჩვეულებრივ, 0-დან დაწყებული რიცხვითი ინდექსები მიენიჭება; ამასთან, მასივის ძველი ელემენტების რიცხვითი ინდექსები შეიცვლება (უმეტეს შემთხვევაში მათი მნიშვნელობა ჩამატებული ელემენტების რაოდენობით გაიზრდება). ფუნქცია მასივის ახალ ზომას აბრუნებს. ქვემოთ მოყვანილი ამ ფუნქციის გამოყენების მაგალითი:

```
$A=array(10,"a"=>20,30);  
array_unshift($A,"!","?");  
// მიიღება $A===array(0=>"!", 1=>"?", 2=>10, a=>20, 3=>30)
```

### ფუნქცია **mixed array\_shift(list &\$Arr)**

`mixed array_shift()` ფუნქცია `$Arr` მასივის პირველ ელემენტს წაშლის და დააბრუნებს მიღებულ მასივს. იგი `array_pop()` ფუნქციას ძალიან ჰგავს, ოღონდ პირველ ელემენტს იღებს და არა ბოლოს აგრეთვე ყველა დარჩენილი ელემენტის რიცხვითი ინდექსების კორექტირება ხდება.

### ფუნქცია **array\_unique(array \$Arr)**

`array_unique()` ფუნქცია აბრუნებს მასივს, რომელიც შედგება `$Arr` მასივის ყველა უნიკალური ელემენტისა და მათი გასაღებისაგან. საშუალოდ მასივში ის წყვილები გასაღები => მნიშვნელობა განთავსდება, რომლებიც პირველად შეგვხვდება:

```
$input=array("a" => "მწვანე", "წითელი", "b" => "მწვანე", "ლურჯი",  
"წითელი");  
$result=array_unique($input);  
// მიიღება $result===array("a"=>"მწვანე", "წითელი", "ლურჯი");
```

### ფუნქცია **array\_splice(array &\$Arr, int \$offset [, int \$len] [, int \$Repl])**

array\_splice() ფუნქცია, ისევე როგორც array\_slice() ფუნქცია, \$offset ინდექსიდან დაწყებული \$len მაქსიმალური სიგრძის \$Arr მასივის ქვემასივს აბრუნებს, მაგრამ, ამასთანავე, იგი მოცემულ ელემენტებს \$Repl მასივში არსებული ელემენტებით ცვლის (თუ \$Repl მასივი არ არის მითითებული, მაშინ ამ ელემენტებს შლის). \$offset და \$len პარამეტრები შეიძლება უარყოფითიც იყოს, მაშინ ათვლა ბოლოდან იწყება. ქვემოთ ზოგიერთი მოყვანილია მაგალითი:

```
<?php
$input=array("წითელი", "მწვანე", "ლურჯი", "ყვითელი");
array_splice($input,2);
// მიიღება $input===array("წითელი", "მწვანე")

array_splice($input,1,-1);
// მიიღება $input===array("წითელი", "ყვითელი")

array_splice($input, -1, 1, array("შავი", "შინდისფერი"));
// მიიღება $input===array("წითელი", "მწვანე", "ლურჯი", "შავი", "ში
ნდისფერი")

array_splice($input, 1, count($input), "ნარინჯისფერი");
// მიიღება $input===array("წითელი", "ნარინჯისფერი")
?>
```

ბოლო მაგალითი გვიჩვენებს, რომ \$Repl პარამეტრის სახით ერთელემენტაანი მასივის ნაცვლად, ჩვეულებრივი სტრიქონული მნიშვნელობა შეგვიძლია მივუთითოთ.

### რიცხვთა დიაპაზონის - სიის შექმნა



## ფუნქცია range(number low, number high [, number step])

range() ფუნქცია ძალზე მარტივია. იგი ქმნის სიებს, რომელთა ელემენტები low-დან high-ის ჩათვლით მთელი რიცხვებია. თუ low > high, მაშინ მიმდევრობა კლებადი იქნება. step პარამეტრი PHP 5.0.0 ვერსიაში იყო დამატებული.

თუ step პარამეტრი მითითებულია, მაშინ იგი მიმდევრობის ელემენტებს შორის გამოყენებული იქნება, როგორც ინკრემენტი. step პარამეტრი ყოველთვის დადებითი რიცხვი უნდა იყოს. თუ იგი მითითებული არ არის, მაშინ ჩუმათობის პრინციპით 1-ის ტოლად იგულისხმება.

```
<?php
// array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
foreach (range(0, 12) as $number) {
    echo $number. " ";
}
echo "<br>";
// array(0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
foreach (range(0, 100, 10) as $number) {
    echo $number. " ";
}
echo "<br>";
// array(0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
foreach (range(100, 0, 10) as $number) {
    echo $number. " ";
}
echo "<br>";
// სიმბოლოების მიმდევრობის გამოყენება დამატებული იყო 4.1.0
ვერსიაში
// array('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i');
```

```

foreach (range('a', 'i') as $letter) {
    echo $letter." ";
}
echo "<br>";
// array('c', 'b', 'a');
foreach (range('c', 'a') as $letter) {
    echo $letter." ";
}
?>

```

ამ სკრიპტის მუშაობის შედეგს შემდეგი სახე ექნება:

```

0 1 2 3 4 5 6 7 8 9 10 11 12
0 10 20 30 40 50 60 70 80 90 100
100 90 80 70 60 50 40 30 20 10 0
a b c d e f g h i
c b a

```

## მასივის ელემენტების მთვლელობა

**ფუნქცია count ( mixed var [, int mode] )**

count() ფუნქცია მასივის ელემენტების დასათვლელად გამოიყენება. თუ var პარამეტრი მასივი ან ობიექტი არ არის, მაშინ შედეგი 1-ს ტოლი იქნება, გარდა იმ შემთხვევისა, როდესაც var პარამეტრი NULL-ია, ამ შემთხვევაში შედეგი იქნება 0.

დამატებითი პარამეტრი mode PHP 4.2.0 ვერსიაში იყო დამატებული. თუ დამატებით პარამეტრში მითითებულია COUNT\_RECURSIVE (ან 1), მაშინ count() ფუნქცია მასივის ელემენტებს რეკურსიულად დათვლის. ეს განსაკუთრებით მრავალგანზომილებიანი მასივის ყველა ელემენტის დასათვლელადაა

მოსახერხებელი. თუ mode არ არის მითითებული, ან იგი 0-ის ტოლია, მაშინ count() ფუნქცია უსასრულო რეკურსიას ვერ ამჩნევს.

count() ფუნქციის გამოყენების მაგალითი:

```
<?php
$arr[]=5;
$arr[]=4;
$arr[]=8;
$arr[]=3;
$arr[]=8;
echo "<h2>მასივის ელემენტთა რაოდენობა: ".count($arr)."</h2>";
// გამოიტანს: მასივის ელემენტთა რაოდენობა: 5
?>
```

**მასივის ელემენტთა რაოდენობა: 5**

ქვემოთ count() ფუნქციის რეკურსიული გამოყენების მაგალითია მოყვანილი:

```
<?php
$food = array('ხილი' => array('ფორთოხალი', 'ბანანი', 'ვაშლი'),
              'ბოსტნეული' => array('სტაფილო', 'კარტოფილი', 'ჭარხალი'
));

// რეკურსიული count
echo count($food, COUNT_RECURSIVE); // output 8

// ჩვეულებრივი count
```

```
echo count($food);           // output 2

?>
```

## მასივისა და მისი ელემენტების წაშლა

### ფუნქცია unset()

თუ გსურთ მასივის მთლიანად წაშლა ან წაშლა წყვილისა გასაღები => მნიშვნელობა, შეგიძლიათ გამოიყენოთ unset() ფუნქცია. მოვიყვანოთ კონკრეტული მაგალითი:

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56; // სკრიპტის ამ ადგილზე ეს ეკვივალენტურია
$arr[13] = 56;

$arr["x"] = 42; // ეს მასივს უმატებს ახალ ელემენტს გასაღებით "x"

unset($arr[5]); // ეს წაშლის მასივიდან ელემენტს

unset($arr); // ეს წაშლის მთელ მასივს

?>
```

## მასივებთან მუშაობის ზოგიერთი თავისებურება

## მასივად გარდაქმნა (ტიპი array)

თუ ნებისმიერი ტიპისათვის: integer, float, string, boolean და resource, მნიშვნელობას მასივად გარდაქმნით, მიიღებთ ერთელემენტის მასივს, რომლის გასაღები იქნება რიცხვი ინდექსით 0.

თუ მასივად გარდაქმნით ობიექტს (object), მაშინ მასივის ელემენტებად ამ ობიექტის თვისებას (ცვლადი-წევრები) მიიღებთ. გასაღები კი იქნება ცვლადი-წევრების სახელები.

თუ მასივად გარდაქმნით მნიშვნელობა NULL-ს, შედეგად მიიღებთ ცარიელ მასივს

### მასივებთან მუშაობის ზოგიერთი პრაქტიკული მაგალითი

```
<?php
$a = array( 'ფერი' => 'ქარვისფერი',
           'გემო' => 'ტკბილი',
           'ფორმა' => 'მრგვალი',
           'სახელი' => 'ყურძენი',
           4          ); // გასაღები იქნება 0

// სრულად შეესაბამება
$a['ფერი'] = 'ქარვისფერი';
$a['გემო'] = 'ტკბილი';
$a['ფორმა'] = 'მრგვალი';
$a['სახელი'] = 'ყურძენი';
$a[] = 4; // გასაღები იქნება 0

$b[] = 'a';
$b[] = 'b';
$b[] = 'c';
// ქმნის მასივს array(0 => 'a' , 1 => 'b' , 2 => 'c'),
```

```
// ან უბრალოდ array('a', 'b', 'c')
?>
```

კიდევ ერთი პრაქტიკული მაგალითი:

```
<?php
// მასივი, როგორც რუკა (თვისებები)
$map = array( 'version' => 4,
              'OS'     => 'Linux',
              'lang'   => 'English',
              'short_tags' => true
            );

// მხოლოდ რიცხვითი გასაღები
$array = array( 7,
               8,
               0,
               156,
               -10
            );

// ეს იგივეა რაც array(0 => 7, 1 => 8, ...)

$switching = array( 10, // გასაღები = 0
                   5  => 6,
                   3  => 7,
                   'a' => 4,
                   11, // გასაღები = 6 (რადგან მაქსიმალური
რიცხვითი ინდექსი იყო 5)
                   '8' => 2, // გასაღები = 8 (რიცხვი!)
                   '02' => 77, // გასაღები = '02'
                   0  => 12 // მნიშვნელობა 10 შეიცვლება 12-ით
```

```

);

// ცარიელი მასივი
$empty = array();
?>

კოლექცია:
<?php
$colors = array('წითელი', 'ლურჯი', 'მწვანე', 'ყვითელი');
foreach ($colors as $color) {
    echo "თქვენ მოგწონთ $color ფერი?";
}
?>

```

განხილული სკრიპტის შედეგი იქნება:

```

თქვენ მოგწონთ წითელი ფერი?
თქვენ მოგწონთ ლურჯი ფერი?
თქვენ მოგწონთ მწვანე ფერი?
თქვენ მოგწონთ ყვითელი ფერი?

```

შემდეგი მაგალითი ქმნის მასივს, რომელიც ერთიანით იწყება:

```

<?php
$firstquarter = array(1 => 'იანვარი', 'თებერვალი', 'მარტი');
print_r($firstquarter);

?>

```

მოცემული სკრიპტის შედეგი იქნება:

Array

```
(  
  [1] => 'იანვარი'  
  [2] => 'თებერვალი'  
  [3] => 'მარტი'  
)
```

მასივის შევსების მაგალითი:

```
<?php  
// მასივს ავსებს დირექტორიის ყველა ელემენტით  
$handle = opendir('.');  
while (false !== ($file = readdir($handle))) {  
  $files[] = $file;  
}  
closedir($handle);  
?>
```

მოწესრიგებული მასივები. სორტირების სხვადასხვა ფუნქციის გამოყენებით შეგიძლიათ ელემენტთა მიმდევრობა შეცვალოთ. ასევე შეიძლება count () ფუნქციის გამოყენებით მასივის ელემენტთა დათვლა.

რეკურსიული და მრავალგანზომილებიანი მასივები:

```
<?php  
$fruits = array ( "ხილი" => array ( "a" => "ფორთოხალი",  
                                     "ბ" => "ბანანი",  
                                     "ც" => "ვაშლი"  
                                   ),  
  "რიცხვი" => array ( 1,  
                     2,
```



```

        3,
        4,
        5,
        6
    ),
    "ჭრილი" => array (
        "პირველი",
        5 => "მეორე",
        "მესამე"
    )
);

```

// წინა მასივების მნიშვნელობებთან წვდომის რამდენიმე მაგალითი

```

echo $fruits["ჭრილი"][5]; // დაბეჭდავს "მეორე"
echo $fruits["ხილი"]["a"]; // დაბეჭდავს "ფორთოხალი"
unset($fruits["ჭრილი"][0]); // წაშლის "პირველი"

```

```

// ახალ მრავალგანზომილებიან მასივს შექმნის
$juices["ვაშლი"]["მწვანე"] = "კარგია";
?>

```

ყურადღება მიაქციეთ, რომ მასივის მინიჭების დროს ყოველთვის ხდება მონაცემის კოპირება. მასივის ბმულით კოპირების შემთხვევაში უნდა გამოიყენოთ ბმულის ოპერატორი:

```

<?php
$arr1 = array(2, 3);
$arr2 = $arr1;
$arr2[] = 4; // $arr2 შეიცვალა,
// $arr1 ძველებურად array(2,3)

```

```
$arr3 = &$arr1;  
$arr3[] = 4; // ახლა $arr1 და $arr3 ერთმანეთის ეკვივალენტურია  
?>
```

ჩვენ განვიხილეთ მასივებთან მუშაობის ძირითადი შესაძლებლობები.

## ფაილებთან სამუშაო ფუნქციები

ფაილებთან მუშაობა 3 ეტაპისაგან შედგება:

- ფაილის გახსნა;
- მონაცემთა დამუშავება (წაკითხვა, ჩაწერა);
- ფაილის დახურვა.

### ფაილის გახსნა

#### ფუნქცია fopen()

fopen() ფუნქციით იხსნება ფაილი. მისი სინტაქსია:

```
resource fopen ( string filename, string mode [, bool use_include_path [, resource zcontext]])
```

მისი საშუალებით აუცილებლად ორი პარამეტრი გადაიცემა: პირველი - filename (ფაილის სახელი - სტრიქონი) და მეორე mode (რეჟიმი - ისიც სტრიქონი). ფუნქცია შესრულების შედეგად რესურსის ტიპის მნიშვნელობას აბრუნებს. შემდეგში მას ფაილებთან მომუშავე სხვა ფუნქციები გამოიყენებენ.

თუ მითითებული ფაილი მიმდინარე საქალაქდება, მაშინ მისი სახელის (მასზე მიმართვის გზის მითითების გარეშე) მითითებაა საკმარისი. თუ ფაილი სხვა ადგილზე მდებარეობს, მაშინ სრული მიმართვის გზის მითითება აუცილებელია.

მიმდინარე საქალაქის შესაცვლელად გამოიყენება ფუნქცია `chdir()`. მას აუცილებლად უნდა მივაწოდოთ იმ საქალაქის სახელი, რომელიც გვინდა მიმდინარე გახდეს. PHP არსებულ მიმდინარე კატალოგს შეცვლის ამ ფუნქციაში მითითებული კატალოგით. შედეგად ეს ფუნქცია დააბრუნებს TRUE-ს, თუ მოქმედება წარმატებით დამთავრდა, ან FALSE-ს თუ რაიმე შეცდომა წარმოიშვა. თუ კატალოგის შეცვლა ვერ მოხერხდა, ფუნქცია დააბრუნებს FALSE-ს.

იმისათვის, რომ გავიგოთ მოცემულ მომენტში რომელია მიმდინარე კატალოგი გამოიყენება ფუნქცია `getcwd()`.

`fopen()` სახელდებულ რესურსს `filename` არგუმენტში მითითებულ ნაკადს მიამაგრებს. თუ `filename` "scheme://..." ფორმითაა გადაცემული, მაშინ იგი თვლის, რომ იგი URL-მისამართია და PHP ამ სქემის შესაბამისი პროტოკოლის დამმუშავებლის ძებნას დაიწყებს. თუ ასეთი რამ ვერ აღმოაჩინა, მაშინ PHP მოგვცემს შენიშვნას, რათა ჩვენ სკრიპტში მოსალოდნელი პოტენციური პრობლემა განვსაზღვროთ, ხოლო თვითონ გააგრძელებს მუშაობას ისე, თითქოს `filename`-ში ჩვეულებრივი ფაილის სახელი იყოს მითითებული.

თუ PHP-მ გადაწყვიტა, რომ `filename` ლოკალურ ფაილზე მიუთითებს, მაშინ ის ეცდება ამ ფაილისაკენ ნაკადი გახსნას. PHP-ს ამ ფაილზე წვდომა უნდა ჰქონდეს, მაგრამ უნდა დავრწმუნდეთ, რომ ფაილზე წვდომის უფლება ამის ნებას გვაძლევს. იმისდა

მიხედვით მომხმარებელმა ჩართო უსაფრთხო რეჟიმი<sup>1</sup>, თუ `open_basedir`-ს<sup>2</sup> დამატებითი შეზღუდვები ედება.

თუ PHP-მ გადაწყვიტა, რომ `filename` დარეგისტრირებულ პროტოკოლზე მიუთითებს და ეს პროტოკოლი დარეგისტრირებულია, როგორც ქსელური URL-მისამართი, მაშინ PHP შეამოწმებს `allow_url_fopen` დირექტივის მდგომარეობას. თუ ის გამორთულია, მაშინ PHP გასცემს გაფრთხილებას და `fopen` ფუნქციის გამოძახება წარუმატებლად დამთავრდება.

---

<sup>1</sup> უსაფრთხო რეჟიმი PHP-ში ერთობლივად გამოყენებად სერვერზე უსაფრთხოების პრობლემის გადაჭრის მცდელობაა. მიუხედავად იმისა, რომ ამ პრობლემის PHP-ს დონეზე გადაჭრა კონცეპტუალურად არასწორია, ვინაიდან დღესდღეობით ვებ-სერვერისა და ოპერაციული სისტემის დონის ალტერნატივა არ არსებობს, მრავალი მომხმარებელი, განსაკუთრებით კი პროვიდერები, კონკრეტულად ამ რეჟიმს იყენებენ.

<sup>2</sup> საქალაქდეთა იერარქიული ხის სიაში იმ მითითებული ფაილების სიას ზღუდავს, რომელიც შეიძლება PHP-ში იყოს გახსნილი, მიუხედავად იმისა, უსაფრთხო რეჟიმი გამოიყენება თუ არა.

ყოველთვის, როდესაც, მაგალითად, სკრიპტი `fopen()` ან `gzopen()` ფუნქციების საშუალებით ფაილის გახსნას ეცდება, ფაილის ადგილმდებარეობა მოწმდება. იმ შემთხვევაში, როდესაც მითითებული ფაილი საქალაქდეთა იერარქიული ხის სიაში არ არის, მაშინ PHP ფაილის გახსნაზე უარს ამბობს. ყველა სიმბოლური ბმული ამოიცინობა და გარდაიქმნება, რის გამოც სიმბოლური ბმულის მეშვეობით ამ შეზღუდვისთვის გვერდის ავლა შეუძლებელია.

სპეციალური მნიშვნელობა მიუთითებს, რომ მიმდინარედ ითვლება საქალაქდე, რომელშიც მოცემული სკრიპტი არის განთავსებული. ამ შემთხვევაში ფრთხილად უნდა ვიყოთ, რადგან სკრიპტის სამუშაო საქალაქდე `chdir()` ფუნქციის საშუალებით მარტივად შეიძლება შეცვალოთ.

mode პარამეტრი იმ წვდომის ტიპზე მიუთითებს, რომელსაც მომხმარებელი ნაკადიდან მოითხოვს. იგი შეიძლება ქვემოთ ჩამოთვლილთაგან ერთ-ერთი იყოს:

**ცხრილი:** fopen() ფუნქციის mode პარამეტრის შესაძლო რეჟიმების სია

mode	აღწერა
'r'	ფაილს გახსნის მხოლოდ წასაკითხად; მაჩვენებელს დააყენებს ფაილის დასაწყისში.
'r+'	ფაილს გახსნის წასაკითხად და ჩასაწერად; მაჩვენებელს დააყენებს ფაილის დასაწყისში.
'w'	ფაილს გახსნის მხოლოდ ჩასაწერად; მაჩვენებელს დააყენებს ფაილის დასაწყისში და ჩამოაჭრის ფაილს ნულოვან სიგრძემდე. თუ ფაილი არ არსებობს - ცდილობს მის შექმნას.
'w+'	ფაილს გახსნის წასაკითხად და ჩასაწერად; მაჩვენებელს დააყენებს ფაილის დასაწყისში და ჩამოაჭრის ფაილს ნულოვან სიგრძემდე. თუ ფაილი არ არსებობს - ცდილობს მის შექმნას.
'a'	ფაილს გახსნის მხოლოდ ჩასაწერად; მაჩვენებელს დააყენებს ფაილის ბოლოში. თუ ფაილი არ არსებობს - ცდილობს მის შექმნას.
'a+'	ფაილს გახსნის წასაკითხად და ჩასაწერად; მაჩვენებელს დააყენებს ფაილის ბოლოში. თუ ფაილი არ არსებობს - ცდილობს მის შექმნას.

mode	აღწერა
'x'	<p>ფაილს გახსნის მხოლოდ ჩასაწერად; მაჩვენებელს დააყენებს ფაილის დასაწყისში. თუ ფაილი უკვე არსებობს, მაშინ ფუნქციის გამოძახება წარუმატებლად დამთავრდება, შედეგად დააბრუნებს FALSE-ს და გაფრთხილებას მოგვცემს (სკრიპტი მუშაობას არ შეწყვეტს). თუ ფაილი არ არსებობს - ცდილობს მის შექმნას. ეს ოფცია მხოლოდ ლოკალური ფაილებისათვის მუშაობს.</p>
'x+'	<p>ფაილს შექმნის და გახსნის წასაკითხად და ჩასაწერად; მაჩვენებელს დააყენებს ფაილის დასაწყისში. თუ ფაილი უკვე არსებობს, მაშინ ფუნქციის გამოძახება წარუმატებლად დამთავრდება, შედეგად დააბრუნებს FALSE-ს და გაფრთხილებას მოგვცემს (სკრიპტი მუშაობას არ შეწყვეტს). თუ ფაილი არ არსებობს - ცდილობს მის შექმნას. ეს ოფცია მხოლოდ ლოკალური ფაილებისათვის მუშაობს.</p>

**შენიშვნა:** 1. სტრიქონის დამთავრების შესახებ სხვადასხვა ოპერაციულ სისტემას სხვადასხვა შეთანხმება აქვს. როდესაც იწერება ტექსტი და მათ შორის უნდა ჩაისვას სტრიქონის დამთავრების სიმბოლო, მომხმარებელმა უნდა გამოიყენოს თავისი ოპერაციული სისტემის შესაბამისი სწორი სიმბოლო (სიმბოლოები). Unix-ის ოჯახის სისტემები სტრიქონის დამთავრების სიმბოლოდ იყენებს \n-ს, Windows-ის სისტემები -\r\n-ს, Macintosh-ის - \r-ს.

თუ მომხმარებელი ფაილების რედაქტირების დროს სტრიქონის დასამთავრებლად არასწორ სიმბოლოს იყენებს,

შეიძლება გახსნის შემდეგ ეს ფაილები „სასაცილოდ გამოიყურებოდეს“.

Windows-ი მომხმარებელს სთავაზობს ტექსტური ტრანსლაციის რეჟიმის ('t') ალამს, რომელიც ფაილებთან მუშაობის დროს \n-ი ავტომატურად გადაიყვანს ახალ სტრიქონზე. ასევე შეიძლება 'b'-ის გამოყენება იმისათვის, რომ იძულებით ჩაირთოს ბინარული (ორობითი) რეჟიმი, რომელშიც მონაცემები არ გარდაიქმნება. იმისათვის, რომ ეს რეჟიმები გამოიყენოთ mode პარამეტრში, ბოლო სიმბოლოს სახით 'b' ან 't' მიუთითეთ.

ვინაიდან ტრანსლაციის ალმის დაყენება ჩუმათობის პრინციპით SAPI-ზე და PHP-ს გამოყენებულ ვერსიაზეა დამოკიდებული, პორტირებიდან გამომდინარე რეკომენდებულია ცხადად იყოს მითითებული ზემოთ ნახსენები ალამი. თუ მომხმარებელი ტექსტურ ფაილებთან მუშაობს უნდა გამოიყენოს რეჟიმი 't' და სკრიპტში სტრიქონის ბოლოს აღსანიშნავად \n. წინააღმდეგ შემთხვევაში სასურველია გამოიყენოს ალამი 'b'.

თუ ბინარულ ფაილებთან მუშაობის დროს ცხადად არ იქნება 'b' ალამი მითითებული, შეიძლება მონაცემების უცნაური დამახინჯება მოხდეს, მათ შორის გამოსახულების ფაილების და უცნაური პრობლემები \r\n სიმბოლოებთან.

2. პორტირების მიზეზებიდან გამომდინარე fopen() ფუნქციით ფაილის გახსნის შემთხვევაში კატეგორიულად მოვითხოვთ, რომ გამოყენებული იყოს 'b' ალამი.

მესამე არააუცილებელი use\_include\_path პარამეტრი შეიძლება იყოს 1 ან TRUE. თუ ფაილის გახსნა ვერ მოხერხდა, ფუნქცია დააბრუნებს FALSE-ს და მოხდება შეცდომის გენერირება (სკრიპტი

მუშაობას არ შეწყვეტს). მუშაობის გასაგრძელებლად მომხმარებელს შეუძლია გამოიყენოს სიმბოლო „@“.

```
<?php
$handle = fopen("/home/rasmus/file.txt", "r");
$handle = fopen("/home/rasmus/file.gif", "wb");
$handle = fopen("http://www.example.com/", "r");
$handle = fopen("ftp://user:password@example.com/somefile.txt", "w");
?>
```

თუ ფაილების წაკითხვის ან ჩაწერის დროს რაიმე პრობლემებს აწყდებით და სერვერული მოდულის სახით PHP-ს იყენებთ, უნდა დარწმუნდეთ, რომ თქვენ მიერ გამოყენებულ ფაილებთან და საქალაქდებთან სერვერის პროცესს წვდომა აქვს.

Windows-ის პლატფორმაზე მუშაობის დროს არ უნდა დაგვავიწყდეს, ფაილზე მიმართვის გზის საპირისპიროდ დახრილი ყველა ხაზის ეკრანირება.

```
<?php
$handle = fopen("c:\\data\\info.txt", "r");
?>
```

### ფაილის დამუშავება

ფაილის დამუშავება მის წაკითხვას და/ან ჩაწერას გულისხმობს. განვიხილოთ რამდენიმე ფუნქცია, რომლებიც ფაილს კითხულობენ.

#### ფუნქცია fgets

ფუნქცია მითითებული ან ნაკლები რაოდენობის (თუ ადრე სტრიქონის ბოლოს ან ფაილის ბოლოს შეხვდა) ბაიტს კითხულობს. მისი სინტაქსია:



### **string fgets ( resource handle [, int length ] )**

პირველი პარამეტრი რესურსის მაჩვენებელია, ზუსტად ის, რომელიც შესრულების შემდეგ fopen() ფუნქციამ დააბრუნა. მეორე არააუცილებელი პარამეტრი არის იმ ბაიტების რაოდენობა, რომელიც უნდა წაიკითხოს.

ფუნქცია შედეგად აბრუნებს length - 1 ბაიტი სიგრძის ფაილის დესკრიპტორიდან წაკითხულ სტრიქონს, რომელზედაც handle პარამეტრი უთითებს. კითხვა მაშინ მთავრდება, როდესაც წაკითხული ბაიტების რაოდენობა length-ს გაუტოლდება, სტრიქონის ბოლოს (რომელიც დაბრუნებულ მნიშვნელობაში შედის) ან ფაილის ბოლოს (რომელიც პირველი შეხვედბა) მიღწევის შემთხვევაში. თუ სიგრძე მითითებული არ არის, მაშინ ჩუმათობის პრინციპით მისი მნიშვნელობა 1 კილობაიტის ანუ 1024 ბაიტის ტოლია.

შეცდომის წარმოქმნის შემთხვევაში ფუნქცია FALSE-ს აბრუნებს.

ფაილის მაჩვენებელი კორექტული უნდა იყოს და იმ ფაილზე უნდა მიუთითებდეს, რომელიც fopen() ან fsockopen() ფუნქციების მეშვეობით წარმატებით გაიხსნა.

ქვემოთ მოყვანილია მარტივი მაგალითი:

```
<?php
$handle = fopen("/tmp/inputfile.txt", "r");
while (!feof($handle)) {
    $buffer = fgets($handle, 4096);
    echo $buffer;
}
fclose($handle);
?>
```

ფუნქცია `file_get_contents()` - არგუმენტად ფაილის სახელს იღებს და შედეგად მის შიგთავსს ერთ სტრიქონად აბრუნებს.

ფუნქცია `file()` - არგუმენტად ფაილის სახელს იღებს და შედეგად მის შიგთავსს მასივის სახით აბრუნებს.

ბოლო ორი ფუნქციის გამოყენების დროს `fopen()` ფუნქციით ფაილის გახსნა საჭირო არ არის, ისინი ამას თვითონ შეასრულებენ.

მონაცემების ფაილში ჩასაწერად გამოიყენება შემდეგი ფუნქციები:

### ფუნქცია `fputs()`

პირველი არგუმენტი რესურსზე მიუთითებს, ხოლო მეორე მასში ჩასაწერი სტრიქონია.

### ფუნქცია `file_put_contents()`

არგუმენტებად იღებს ფაილისა და სტრიქონის სახელებს, რომლებიც მოცემულ ფაილში უნდა ჩაიწეროს.

### ფუნქცია `fclose()`

ფუნქცია ხურავს ფაილს. მას არგუმენტად უნდა გადავცეთ რესურსზე მაჩვენებელი.

ამ ფუნქციების მუშაობას გავცნოთ მაგალითზე. ვთქვათ, `/usr/tmp` კატალოგის `f1.txt` ფაილში რაღაც რიცხვები ინახება, ყოველ სტრიქონში თითო რიცხვი. ჩვენ უნდა წავიკითხოთ ეს რიცხვები, გავზარდოთ და `f2.txt` ფაილში ჩავწეროთ.

```
<?PHP
chdir('/usr/tmp');
$src = fopen('f1.txt', 'r'); // 'r' ფუნქციას მიუთითებს, რომ უნდა
გახსნას ფაილი წასაკითხად
$dst = fopen('f2.txt', 'w'); // 'w' ფუნქციას მიუთითებს, რომ უნდა
```

```

გახსნას ფაილი ჩასაწერად
while ( !feof($src) ) {
    $line = fgets($src, 16);
    $line++;
    fputs($dst, $line);
}
fclose($dst);
fclose($src);
?>

```

აქ შეგვხვდა ახალი ფუნქცია feof(), რომელიც შესრულების შედეგად აბრუნებს true, თუ მომდევნო ოპერაციის შედეგად ფაილის ბოლოა მიღწეული.

## საქალაქდესთან სამუშაო ფუნქციები

PHP-ში როგორც ფაილს, ისე საქალაქდესაც სჭირდება გახსნა, ხოლო მუშაობის დამთავრების შემდეგ დახურვა. ამას opendir(\$path) და closedir(\$res) ფუნქციები აკეთებენ.

### ფუნქცია opendir

**opendir ( string path )**

შესრულების შედეგად აბრუნებს კატალოგის (საქალაქდეს) დესკრიპტორს, ფუნქციების closedir(), readdir() (ეს ფუნქცია შედეგად აბრუნებს კატალოგის რიგით მომდევნო ელემენტის სახელს. ელემენტების სახელები ფაილური სისტემაზე დამოკიდებული რიგის მიხედვით ბრუნდება) და rewinddir()-ის (კატალოგების ნაკადს გადატვირთავს, ისე, რომ გადაცემული პარამეტრი კატალოგის დასაწყისზე მიუთითებდეს) მეშვეობით შემდგომი გამოყენების მიზნით.

თუ კატალოგზე მითითებული მიმართვის გზა არ არსებობს ან რაიმე შეზღუდვის, ან ფაილური სისტემის შეცდომის გამო ამ კატალოგის გახსნა შეუძლებელია, opendir() ფუნქცია შედეგად აბრუნებს FALSE-ს და შეცდომის შესახებ შეტყობინების გენერირება მოხდება ისე, რომ სკრიპტი მუშაობას არ შეწყვეტს. ქვემოთ მოყვანილია opendir() ფუნქციის გამოყენების მაგალითი:

```
<?php
$dir = "/tmp/";
// გაიხსნას წინასწარ ცნობილი არსებული კატალოგი და დაიწყოს
მისი წაკითხვა if (is_dir($dir)) {
    if ($dh = opendir($dir)) {
        while (($file = readdir($dh)) !== false) {
            print "Файл: $file : тип: " . filetype($dir . $file) . "\n";
        }
        closedir($dh);
    }
}
?>
```

PHP 4.3.0 ვერსიიდან დაწყებული პარამეტრი „მიმართვის გზა“ შეიძლება ნებისმიერი URL-მისამართი იყოს, რომელზე მიმართვაც მისი ფაილებისა და კატალოგების სიას მოგვცემს. მოცემული წესი მხოლოდ file:// url-დამკომპლექტებლებთან მუშაობდა, ხოლო PHP 5.0.0-დან მას ftp://-ც დაემატა.

**ფუნქცია closedir**

**closedir ( resource dir\_handle )**

კატალოგთან და `dir_handle` გადაცემულ პარამეტრთან დაკავშირებულ ნაკადს დახურავს. ამ ფუნქციის გამოყენების წინ ეს ნაკადი აუცილებლად `opendir()` ფუნქციის მიერ უნდა გაიხსნას.

### ფუნქცია `scandir`

**`scandir ( string directory [, integer sorting_order] )`**

მას პარამეტრად უნდა მივაწოდოთ კატალოგზე მიმართვის გზა და შედეგად დააბრუნებს კატალოგის ყველა ელემენტის სიას მასივის სახით. ამ დროს `opendir()` ფუნქციით საქალაქის გახსნა საჭირო არ არის. თუ `directory` პარამეტრი კატალოგი არ არის, მაშინ შედეგად ლოგიკური მნიშვნელობა `FALSE` დაბრუნდება და მოხდება შეცდომის გამოტანა, სკრიპტი კი განაგრძობს მუშაობას.

ჩუმათობის პრინციპით სიის მოწესრიგება ალფაბეტის მიხედვით ზრდადობით მოხდება. თუ არააუცილებელი `sorting_order` პარამეტრია მითითებული და უდრის 1-ს, მაშინ სიის მოწესრიგება ალფაბეტის მიხედვით კლებადობით მოხდება. ქვემოთ მოყვანილია ამ ფუნქციის გამოყენების მარტივი მაგალითი:

```
<?php
$dir = '/tmp';
$files1 = scandir($dir);
$files2 = scandir($dir, 1);

print_r($files1);
print_r($files2);

/* შედეგი დაახლოებით ასეთი იქნება:
Array
(
    [0] => .
```

```

[1] => ..
[2] => bar.php
[3] => foo.txt
[4] => somedir
)
Array
(
    [0] => somedir
    [1] => foo.txt
    [2] => bar.php
    [3] => ..
    [4] => .
)
*/
?>

```

საკვალადეებთან სამუშაოდ PHP-ს აგრეთვე, ჩაშენებული კლასი აქვს. მას `dir` ჰქვია. იგი მასზე მიმართვის გზას, დესკრიპტორს, აგრეთვე დესკრიპტორის წაკითხვის, დახურვის და გადატვირთვის მეთოდებს შეიცავს. მათი გამოყენების წესს მოგვიანებით გავეცნობით.

## ფუნქცია `readdir`

**`readdir ( resource dir_handle )`**

შედეგად აბრუნებს კატალოგის რიგით მომდევნო ელემენტის სახელს. ელემენტის სახელები ფაილურ სისტემაზე დამოკიდებულებით ბრუნდება.

ქვემოთ მოყვანილ მაგალითში ყურადღება უნდა მიექცეს `readdir()` ფუნქციით დაბრუნებული მნიშვნელობის შემოწმების საშუალებას. ამ მაგალითში მნიშვნელობის `FALSE`-თან იგივეურად

შედარება ხდება, ვინაიდან კატალოგის ნებისმიერი ელემენტი, რომლის სახელი შეიძლება, როგორც FALSE გამოისახოს, დაასრულებს ციკლს (მაგალითად, ელემენტი სახელით „0“). ქვემოთ მოყვანილ მაგალითში კატალოგის ყველა ფაილის სიის გამოტანა უნდა განხორციელდეს:

```
<?php
if ($handle = opendir('/path/to/files')) {
    echo "კატალოგის დესკრიპტორი: $handle\n";
    echo "ფაილები:\n";
    /* კატალოგების ელემენტების წაკითხვის ეს მეთოდი არის
სწორი */
    while (false !== ($file = readdir($handle))) {
        echo "$file\n";
    }
    /* წაკითხვის ეს მეთოდი არასწორია */
    while ($file = readdir($handle)) {
        echo "$file\n";
    }
    closedir($handle);
}
?>
```

მეორე მაგალითში readdir() ფუნქცია აბრუნებს ელემენტებს, რომელთა სახელებია f1 და f2:

```
<?php
if ($handle = opendir('f')) {
    while (false !== ($file = readdir($handle))) {
        if ($file != "f1" && $file != "f2") {
            echo "$file\n";
        }
    }
}
```

```

    }
}
closedir($handle);
}
?>

```

readdir() ფუნქციის მეშვეობით გამოვიტანთ კატალოგის მომდევნო ელემენტის სახელს. ახლა შეგვიძლია გავიგოთ რა ელემენტია ეს - კიდევ ერთი საქაღალდე, ფაილი თუ ბმული. ამისათვის შესაბამისი ფუნქციები გამოიყენება.

### ფუნქცია is\_dir

**is\_dir ( string filename )**

შედეგად აბრუნებს TRUE-ს, თუ ეს ფაილი არსებობს და ის საქაღალდეს წარმოადგენს. თუ filename ფაილის სახელია, მაშინ იგი მიმდინარე საქაღალდის მიმართ შემოწმდება.

```

<?
var_dump(is_dir('a_file.txt')) . "\n";
var_dump(is_dir('bogus_dir/abc')) . "\n";
var_dump(is_dir('..')); //ერთი საქაღალდით მაღლა
?>
// ამ მაგალითის შედეგი იქნება
bool(false)
bool(false)
bool(true)

```

### ფუნქცია is\_file

**is\_file ( string filename )**



შედეგად აბრუნებს TRUE-ს, თუ ეს ფაილი არსებობს და ის ჩვეულებრივი ფაილია.

### ფუნქცია `is_link`

`is_link ( string filename )`

შედეგად აბრუნებს TRUE-ს, თუ ეს ფაილი არსებობს და ის სიმბოლური ბმულია.

### ფუნქცია `filetype`

`filetype ( string filename )`

შედეგად აბრუნებს ფაილის ტიპს. მისი შესაძლო მნიშვნელობებია `fifo`, `char`, `dir`, `block`, `link`, `file` და `unknown`.

შეცდომის წარმოქმნის შემთხვევაში შედეგად აბრუნებს FALSE-ს. ფაილის შესახებ ინფორმაციის მოპოვების წარუმატებელი მცდელობის შემთხვევაში ან თუ ფაილის ტიპი უცნობია `filetype()` ფუნქცია `E_NOTICE` შეტყობინებას გამოიწვევს. ქვემოთ მოცემულია `filetype()` ფუნქციის გამოყენების მაგალითი:

```
<?php
echo filetype('/etc/passwd'); // გამოიტანს file
echo filetype('/etc/');      // გამოიტანს dir
?>
```

### ფუნქცია `clearstatcache()`

`clearstatcache()` ფუნქცია ფაილების კემ-მდგომარეობას ასუფთავებს. თუ მომხმარებელი თავისი სკრიპტით ფაილებში რაიმე ცვლილებებს ახდენს, ხოლო შემდეგ ამ ცვლილებებზე დაყრდნობით მუშაობს, მაშინ ცვლილებების განხორციელების შემდეგ აუცილებლად უნდა მოხდეს ამ ფუნქციის გამოძახება. წინააღმდეგ შემთხვევაში ეს ცვლილებები შეიძლება შეუმჩნეველი

დარჩეს. მაგალითად, ვთქვათ მომხმარებელი ფაილში წერს მანამ, ვიდრე მისი ზომა არ გახდება 2 მბ. ყოველი ჩაწერის შემდეგ შემოწმების მიუხედავად, შეიძლება ფაილის ზომამ 2 გბ-ს მიაღწიოს, მაგრამ PHP მას ისევ ძველ ზომებში ხედავს. თუ clearstatcache() ფუნქციას გამოვიყენებთ, მაშინ ყველაფერი ნორმაში ჩადგება.

განვიხილოთ მცირე მაგალითი: გავხსნათ საქალაქდე /usr/home/mydir და ვნახოთ რა არის მასში.

```
<?PHP
$dir_hndl = opendir('/usr/home/mydir');
while (false !== ($name = readdir($dir_hndl)))
    if ( $name == '..' ) {
        echo 'Parent directory<br>';
        continue;
    }
    elseif ( $name == '.' ) {
        echo 'Current directory<br>';
        continue;
    }
    if ( is_dir($name) ) echo $name.' is a dir<br>';
    elseif ( is_file($name) ) echo $name. ' is a filr';
    else echo $name. ' რა შეიძლება ეს იყოს?<br> '
}
closedir($dir_hndl)
?>
```

ახლა მოკლედ დანარჩენი ძირითადი ფუნქციების შესახებ:  
stat() - მასივის სახით იღებს ინფორმაციას ფაილის შესახებ (შექმნის თარიღი, მფლობელი).

lstat() - იღებს ინფორმაციას ფაილის ან სიმბოლური ბმულის შესახებ.

file\_exists() - ამოწმებს, არსებობს თუ არა ფაილი ან საქალაქ-დე.

is\_writable() - ამოწმებს, ფაილში ჩაწერის შესაძლებლობას.

is\_readable() - ამოწმებს, ფაილიდან წაკითხვის შესაძლებლობას.

is\_executable() - არკვევს, ფაილი შესრულებადია თუ არა.

filectime(), fileatime(), filemtime(), fileinode(), filegroup(), fileowner(), filesize(), filetype(), fileperms() - შედეგად აბრუნებს ინფორმაციას ფაილზე. ინფორმაციის სახეობა ფუნქციის სახელიდან გამომდინარეობს.

## თარიღისა და დროის ფუნქციები

ეს ფუნქციები საშუალებას გვაძლევს მიმდინარე დრო გავიგოთ იმ სერვერზე, რომელზეც მოცემული სკრიპტი სრულდება. ამის გარდა, შეიძლება დროის სხვადასხვა ფორმატში წარმოდგენა, დროის ორ სხვადასხვა მომენტს შორის სხვაობის გაგება, ამა თუ იმ დღეს დღისა და ღამის ხანგრძლივობა, მზის ამოსვლის დრო და სხვა.

გავეცნოთ ამ ფუნქციებიდან ძირითადს:

### ფუნქცია `checkdate ()`

აღნიშნული ფუნქციის ფორმატია:

**`checkdate ( int $month, int $day, int $year )`**

შესრულების შედეგად დააბრუნებს TRUE-ს, თუ არგუმენტებით მიწოდებული თარიღი სწორია; წინააღმდეგ შემთხვევაში შედეგი იქნება FALSE. თარიღი ითვლება სწორად, თუ:

\$year - წელი არის დიაპაზონში 1-დან 32767-ის ჩათვლით;

\$month - თვე არის დიაპაზონში 1-დან 12-ის ჩათვლით;

\$day - რიცხვის მნიშვნელობა დასაშვებია \$month არგუმენტით მოცემული თვის მიხედვით, ამასთან, გათვალისწინებული იქნება ისიც, რომ \$year არგუმენტით მოცემული წელი შეიძლება ნაკიანი იყოს. მაგალითად:

```
<?php
var_dump(checkdate(12, 31, 2000));
var_dump(checkdate(2, 29, 2001));
?>

// შედეგი იქნება

bool(true)

bool(false)
```

## ფუნქცია date\_parse ()

აღნიშნული ფუნქციის ფორმატია:

**date\_parse ( string \$date )**

შესრულების შედეგად დააბრუნებს \$date-ს დროის შესახებ ასოცირებულ მასივს. ეს მასივი შეიცავს წელს, თვეს, რიცხვს, საათს, წუთს, წამს, წამის მეათედს, გაფრთხილებათა რაოდენობას, გაფრთხილებათა მასივს, შეცდომათა რაოდენობას, შეცდომების მასივს, ადგილობრივ დროს. მაგალითად:

```
<?php
print_r(date_parse("2014-09-12 10:15:00.5"));
?>

// ამ სკრიპტის შესრულების შედეგია:
```

```

Array (
  [year] => 2014
  [month] => 09
  [day] => 12
  [hour] => 10
  [minute] => 15
  [second] => 0
  [fraction] => 0.5
  [warning_count] => 0
  [warnings] => Array()
  [error_count] => 0
  [errors] => Array()
  [is_localtime] =>
)

```

## ფუნქცია `date_sun_info ()`

აღნიშნული ფუნქციის ფორმატია:

**`date_sun_info ( int $time, float $latitude, float $longitude )`**

შესრულების შედეგად დააბრუნებს მასივს, რომლის ელემენტებიც შეიცავენ ინფორმაციას მზის ამოსვლისა და ჩასვლის დროის, დღის დაწყებისა და დამთავრების დროის, საზღვაო დღის დაწყებისა და დამთავრების დროის, ასტრონომიული დღის დაწყებისა და დამთავრების დროის შესახებ.

`date_sun_info ()` ფუნქციის არგუმენტებია `$time` - დრო, `$latitude` - გრძედი, `$longitude` - განედი. მაგალითად:

```

<?php
$sun_info = date_sun_info(strtotime("2014-12-12"), 31.7667, 35.2333);
foreach ($sun_info as $key => $val) {

```

```

    echo "$key: " . date("H:i:s", $val) . "\n";
}
?>
// ამ სკრიპტის შესრულების შედეგი იქნება:
sunrise: 05:52:11
sunset: 15:41:21
transit: 10:46:46
civil_twilight_begin: 05:24:08
civil_twilight_end: 16:09:24
nautical_twilight_begin: 04:52:25
nautical_twilight_end: 16:41:06
astronomical_twilight_begin: 04:21:32
astronomical_twilight_end: 17:12:00

```

## ფუნქცია time ()

აღნიშნული ფუნქციის ფორმატია:

**time ( void )**

შედეგად აბრუნებს Unix-ის (The Unix Epoch, 1 იანვარი 1970, 00:00:00 GMT) ეპოქის დასაწყისიდან მიმდინარე დრომდე გასული წამების რაოდენობას.

```

<?php
$nextWeek = time() + (7 * 24 * 60 * 60);
    // 7 დღე; 24 საათი; 60 წუთი; 60 წამი
echo 'ახლა:      '. date('Y-m-d') . "\n";
echo 'შემდეგი კვირა: '. date('Y-m-d', $nextWeek) . "\n";
// ან strtotime() ფუნქციის მეშვეობით:
echo 'შემდეგი კვირა: '. date('Y-m-d', strtotime('+1 week')) . "\n";
?>

```

```
// ამ სკრიპტის შესრულების შედეგი იქნება:
```

```
ახლა:      2014-09-30
```

```
შემდეგი კვირაა: 2014-10-06
```

```
შემდეგი კვირაა: 2014-10-06
```

## ფუნქცია date ()

აღნიშნული ფუნქციის ფორმატია:

```
date ( string $format [, int $ timestamp ] )
```

შესრულების შედეგად დააბრუნებს format არგუმენტის შესაბამისად დაფორმატებულ დროს, რომლისთვისაც timestamp არგუმენტით მოცემულ დროის ჭდეს, ან, თუ timestamp არგუმენტი მოცემული არ არის, მაშინ მიმდინარე სისტემურ დროს გამოიყენებს.

format არის სტრიქონი, რომელიც დაფორმატების სიმბოლოებს ან ჩვეულებრივ სიმბოლოებს შეიცავს. ჩვეულებრივი სიმბოლოები გამოაქვს როგორც არის ისე, ხოლო დაფორმატების სიმბოლოები შესაბამისი მნიშვნელობებით იცვლება:

სიმბოლო format სტრიქონში	აღწერა	დაბრუნებული მნიშვნელობის მაგალითი
a	Ante meridiem ან Post meridiem ქვედა რეგისტრში	am ან pm
A	Ante meridiem ან Post meridiem ზედა რეგისტრში	AM ან PM
B	დრო Swatch Internet-ის სტანდარტით	000-დან 999-მდე

სიმბოლო format სტრიქონში	აღწერა	დაბრუნებული მნიშვნელობის მაგალითი
c	თარიღი ISO 8601-ის ფორმატში (დამატებულია PHP 5-ში)	2014-02- 12T15:19:21+00:00
d	თვის რიცხვი, 2 ციფრი ნიშნადი ნულით	01-დან 31-მდე
D	კვირის დღეების შემოკლებული დასახელება, 3 სიმბოლო	Mon-დან Sun-მდე
F	თვის სრული დასახელება, მაგალითად, January ან March	January-დან December-მდე
g	საათი 12 საათიანი ფორმატით, ნიშნადი ნულების გარეშე	1-დან 12-მდე
G	საათი 24 საათიანი ფორმატით, ნიშნადი ნულების გარეშე	0-დან 23-მდე
h	საათი 12 საათიანი ფორმატით, ნიშნადი ნულებით	01-დან 12-მდე
H	საათი 24 საათიანი ფორმატით, ნიშნადი ნულებით	00-დან 23-მდე
i	წუთები ნიშნადი ნულებით	00-დან 59-მდე
I (მთავრული i)	ზაფხულის დროის მაჩვენებელი	1, თუ თარიღი ზაფხულის დროს



სიმბოლო format სტრუქტურაში	აღწერა	დაბრუნებული მნიშვნელობის მაგალითი
		შეესაბამება, წინა- აღმდეგ შემთხვევაში 0 otherwise.
j	თვის რიცხვი, ნიშნადი ნულების გარეშე	1-დან 31-მდე
l (კატარა 'L')	კვირის დღეების სრული სახელწოდება	Sunday-დან Saturday- მდე
L	ნაკიანი წლის მაჩვენებელი	1, თუ წელი ნაკიანი, თუ არა 0.
m	თვის რიგითი ნომერი ნიშნადი ნულებით	01-დან 12-მდე
M	თვეების შემოკლებული დასახელება, 3 სიმბოლო	Jan-დან Dec-მდე
n	თვის რიგითი ნომერი ნიშნადი ნულების გარეშე	1-დან 12-მდე
O	გრინვიჩის დროსთან სხვაობა	მაგალითად, +0400
r	თარიღი RFC 2822 ფორმატში	მაგალითად, Thu, 22 Dec 2010 16:01:07 +0400
s	წამები ნიშნადი ნულებით	00-დან 59-მდე
S	თვის რიგითი ნომრის ინგლისური სუფიქსი, 2 სიმბოლო	st, nd, rd ან th. გამოიყენება c j-სთან ერთად

სიმბოლო format სტრიქონში	აღწერა	დაბრუნებული მნიშვნელობის მაგალითი
t	თვეში დღეების რაოდენობა	28-დან 31-მდე
T	დროითი ზონა სერვერზე	მაგალითად, EST, MDT ...
U	Unix-ის (The Unix Epoch, 1 იანვარი 1970, 00:00:00 GMT) ეპოქის დასაწყისიდან გასული წამების რაოდენობას.	იხ. აგრეთვე time()
w	კვირის დღის რიგითი ნომერი	0-დან (კვირა)6-მდე (შაბათი)
W	წლის კვირის რიგითი ნომერი ISO-8601-ს ფორმატის მიხედვით, კვირის პირველი დღე ორშაბათია (დამატებულია PHP 4.1.0)	მაგალითად, 42 (წლის 42-ე კვირა)
Y	წლის რიგითი ნომერი, 4 ციფრი	მაგალითად, 1999, 2014
y	წლის რიგითი ნომერი, 2 ციფრი	მაგალითად, 99, 14
z	წელიწადში დღის რიგითი ნომერი (ნუმერაცია 0-დან იწყება)	0-დან 365-მდე

სიმბოლო format სტრუქტურაში	აღწერა	დაბრუნებული მნიშვნელობის მაგალითი
Z	დროითი ზონის წანაცვ- ლება წამებით. UTC-დან დასავლეთის დროითი ზონისათვის უარყოფი- თია, ხოლო აღმოსავლე- თით - დადებითი	-43200-დან 43200-მდე

## ელექტრონული ფოსტის ფუნქციები

### ფუნქცია mail ()

აღნიშნული ფუნქციის ფორმატია:

```
mail ( string $to , string $subject , string $message [, stri-
ng $additional_headers [,
string $additional_parameters ] ] )
```

იგი აგზავნის ელექტრონულ ფოსტას. მისი პარამეტრებია:

to - წერილის მიმღები ან მიმღებები. ამ პარამეტრის ფორმატი უნდა შეესაბამებოდეს RFC 2822 (ინტერნეტ შეტყობინების ფორმატი) სტანდარტს. რამდენიმე მიმღების მითითების შემთხვევაში მისამართები ერთმანეთისაგან მძიმით გამოიყოფა. მისი ზოგიერთი მაგალითია:

- user@gmail.com
- user@gmail.com, anotheruser@yahoo.com
- User <user@gmail.com>
- User <anotheruser@yahoo.com>, Another User <user@gmail.com>

subject - გასაგზავნი წერილის თემა. წერილის თემა უნდა შეესაბამებოდეს RFC 2047 (მრავალფუნქციური ინტერნეტ ფოსტის გაფართოებები) სტანდარტს.

message - გასაგზავნი შეტყობინება. თითოეული სტრიქონი ერთმანეთისაგან გამოყოფილი უნდა იყოს LF (\n) სიმბოლოთი. სტრიქონების სიგრძე 70 სიმბოლოს არ უნდა აღემატებოდეს.

*გაფრთხილება: (მხოლოდ Windows-ისათვის) თუ PHP მონაცემებს SMTP-სერვერს გადასცემს და სტრიქონის დასაწყისში დგას წერტილი, მაშინ ის წაიშლება. ეს რომ თავიდან ავიცილოთ ამისათვის საჭიროა ასეთი წერტილი ორი წერტილით შევცვალოთ.*

```
<?php
$text = str_replace("\n.", "\n..", $text);
?>
```

additional\_headers (არააუცილებელი) სტრიქონი, რომელიც დამატებით ჩაისმება სათაურით გაგზავნილი წერილის ბოლოს. ჩვეულებრივ, გამოიყენება სათაურების (From, Cc, and Bcc) დასამატებლად. დამატებითი სათაურები ერთმანეთისაგან CRLF (\r\n) უნდა იყოს გამოყოფილი.

additional\_parameters (არააუცილებელი) პარამეტრი შეიძლება დამატებითი აღმების გადასაცემად იქნეს გამოყენებული ბრძანების სტრიქონის არგუმენტების სახით, წერილის გაგზავნის კონფიგურირების პროგრამისათვის, sendmail\_path დირექტივის მითითებით.

ფუნქცია შესრულების შედეგად აბრუნებს TRUE-ს, თუ წერილი გასაგზავნადაა მიღებული, წინააღმდეგ შემთხვევაში - FALSE-ს.

აქვე უნდა აღინიშნოს, რომ თუ წერილი გადასაცემდაა მიღებული, სულაც არ ნიშნავს, რომ ის ადრესატმა მიიღო.

მაგალითი 1. mail() ფუნქციის გამოყენება ჩვეულებრივი წერილის გასაგზავნად:

```
<?php
// შეტყობინება
$message = "Line 1\nLine 2\nLine 3";

// იმ შემთხვევაში, თუ წერილის რომელიმე სტრიქონის სიგრძე 70
სიმბოლოს აღემატება,

// მაშინ გამოიყენება wordwrap()
$message = wordwrap($message, 70);

// გაგზავნა
mail('caffeinated@example.com', 'My Subject', $message);
?>
```

მაგალითი 2. წერილის დამატებითი სათაურებით გაგზავნა. ჩვეულებრივი სათაურების დამატება, რომელიც საფოსტო აგენტს From და Reply-To მისამართს ატყობინებს:

```
<?php
$to = 'nobody@gmail.com';
$subject = 'the subject';
$message = 'hello';
$headers = 'From: webmaster@yahoo.com' . "\r\n" .
'Reply-To: webmaster@yahoo.com' . "\r\n" .
'X-Mailer: PHP/' . phpversion();
```

```
mail($to, $subject, $message, $headers);  
?>
```

მაგალითი 3. ბრძანების სტრიქონის დამატებითი არგუმენტებით წერილის გაგზავნა. `additional_parameters` პარამეტრი შეიძლება პროგრამისათვის დამატებითი პარამეტრების გადასაცემად იყოს გამოყენებული, წერილების გასაგზავნად `sendmail_path` დირექტივის მეშვეობით.

```
<?php  
mail('nobody@example.com', 'the subject', 'the message', null,  
    '-fwebmaster@example.com');  
?>
```

## GD - გამოსახულებასთან მუშაობა

PHP-ს შესაძლებლობები HTML-კოდის შექმნით არ შემოისაზღვრება. PHP სხვადასხვა ფორმატის, მათ შორის gif, png, jpg, wbmp და xpm, გამოსახულების შექმნისა და მანიპულირებისათვისაც შეიძლება იქნეს გამოყენებული. ამის გარდა, PHP შეუძლია გამოსახულება პირდაპირ ბრაუზერში გამოიტანოს. გამოსახულებასთან სამუშაოდ მომხმარებელს PHP ესაჭიროება GD გრაფიკულ ბიბლიოთეკასთან ერთად. იმის მიხედვით, თუ გამოსახულების რომელ ფორმატთან მუშაობს მომხმარებელი, GD და PHP შეიძლება სხვა ბიბლიოთეკებზეც იყოს დამოკიდებული.

JPEG, GIF, PNG, SWF, TIFF და JPEG2000 ფორმატის გამოსახულების ზომების მისაღებად PHP-ის სტანდარტული ფუნქცია არსებობს.

JPEG და TIFF გამოსახულებების სათაურებში შენახულ ინფორმაციასთან მუშაობა შესაძლებელია EXIF მოდელით. ამგვარად, ციფრული ფოტოაპარატებით გენერირებული მეტა-მონაცემების წაკითხვა შესაძლებელია. EXIF მოდელი GD გრაფიკულ ბიბლიოთეკას არ საჭიროებს. აქვე უნდა შევნიშნოთ, რომ არც `getimagesize()` ფუნქცია არ საჭიროებს GD გრაფიკულ ბიბლიოთეკას.

GD გრაფიკული ბიბლიოთეკის არსებობის შემთხვევაში, მომხმარებელს გამოსახულების შექმნა და შეცვლა შეუძლია.

გამოსახულების ფორმატი, რომელთანაც შესაძლებელია მუშაობა, დამოკიდებულია დაყენებულ GD-ის ვერსიასა და სხვა ბიბლიოთეკებზე, რომელთაც შეიძლება ამა თუ იმ ფორმატზე წვდომის მისაღებად GD-იმ მიმართოს. GD-1.6-ზე ქვემოთ GD-ის ვერსიები GIF გამოსახულებას მხარს უჭერს, მაგრამ მხარს არ უჭერს PNG-ს, მაშინ როდესაც GD-1.6-ის ზემოთ და GD-2.0.28-ის ქვემოთ

მხარს უჭერს PNG-ს, მაგრამ მხარს არ უჭერს GIF-ს. GIF-ის მხარდაჭერა კვლავ GD-2.0.28 ვერსიაში აღდგა.

*შენიშვნა: PHP 4.3 ვერსიიდან დაწყებული GD ბიბლიოთეკა PHP-ში ჩაშენებულია. ამ ჩაშენებულ ვერსიას რამდენიმე დამატებითი შესაძლებლობა აქვს. სახელდობრ, რეკომენდებულია ამ ჩაშენებული ვერსიის გამოყენება, ვინაიდან მისი კოდი უფრო სტაბილურია და მომხმარებლის მიერ უფრო მხარდაჭერილი.*

### გამოსახულების დასამუშავებელი ფუნქციები

PHP-ში გამოსახულების დასამუშავებელი ფუნქციები შეიძლება ორ კატეგორიად დაიყოს. ფუნქციები, რომლებიც მუშაობენ ფაილებთან და ფუნქციები, რომლებიც მუშაობენ გამოსახულებებთან მეხსიერებაში (რესურსთან).

აქ ყველას აღწერა შეუძლებელია. განვიხილოთ ზოგიერთი მათგანი.

#### ფუნქცია `imagecreatetruecolor ()`.

ამ ფუნქციის დანიშნულებაა ახალი ფერადი გამოსახულების შექმნა. იგი გამოსახულების წარმოდგენისა და შენახვის ისეთ მეთოდებს იყენებს, რომლებიც RGB ფორმატში ბევრი ფერის, ნახევარტონისა და სხვადასხვა შეფერილობის გამოყენების საშუალებას მოგვცემს. როგორც ცნობილია RGB ფორმატი ეს არის სამი საბაზო წითელი (Red), მწვანე (Green) და ლურჯი (Blue) ფერის გამოყენებით ნებისმიერი ფერის მიღების საშუალება. მისი სინტაქსია:

```
imagecreatetruecolor ( int $width , int $height )
```



`imagecreatetruecolor ()` ფუნქცია არგუმენტად ორ მთელ რიცხვს იღებს. ეს არის ახალი გამოსახულების სიგრძე და სიგანე. ამ ფუნქციის შესრულების შედეგი შავი მართკუთხედი, ზომით იქნება  $\$width \times \$height$ .

თუ მომხმარებელს სხვა ფერის მართკუთხედი სჭირდება, მაშინ მას მომდევნო ფუნქციის გამოყენება მოუხდება.

**ფუნქცია `imagecolorallocate ()`.**

ეს ნიშნავს, რომ ამის შემდეგ გამოსახულებასთან ვიმუშავებთ უკვე მეხსიერებაში. ამის შემდეგ ეს იდენტიფიკატორი გადაეცემა უკვე `imagecolorallocate ()` ფუნქციას, რომელიც ფერების წარმოდგენის გენერირებას იმ სახით ახდენს, რა სახითაც სურათზე შემდგომი მუშაობის დროს გამოიყენება.

ამ ფუნქციის პირველი გამოძახება გამოსახულებას ფონს მისცემს.

**`imagecolorallocate ( resource $image , int $red , int $green , int $blue )`**

ამ ფუნქციის `resource $image` პირველი პარამეტრი, სწორედ ის იდენტიფიკატორია, რომელიც `imagecreatetruecolor ()` ფუნქციის საშუალებით მივიღეთ. ეს პარამეტრი იმისთვისაა საჭირო, რომ ფერის წარმოდგენა გამოსახულების ფორმატს შეესაბამებოდეს.

შემდეგი სამი პარამეტრი კი ჩვენთვის ცნობილია. ის არის რიცხვი 0-დან 255-მდე, რომელიც ფერს RGB სისტემაში გამოსახავს.

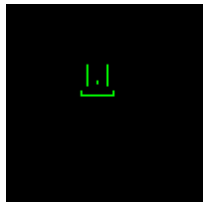
შავი ფერის მისაღებად სამი 0-იანი გვჭირდება, ხოლო თეთრი ფერისათვის სამჯერ 255.

ამგვარად, შექმნილია გამოსახულება, მომზადებულია პალიტრა, ახლა უკვე საჭიროა შემოქმედებითი პროცესის დაწყება. გამოვიყენოთ `ImageLine ()` ფუნქცია და ამ მართკუთხედში

გავავლოთ ხაზები, ხოლო თვით ფუნქცია მოგვიანებით განვიხილოთ.

ქვემოთ მოყვანილია ამ ფუნქციების გამოყენების მაგალითი:

```
<?PHP
// გამოსახულების შექმნა ზომით 100*100
$img = imagecreatetruecolor(100, 100);
// ზოგიერთი ფერის გამოყოფა
$red = imagecolorallocate($img, 255, 0, 0);
$green = imagecolorallocate($img, 0, 255, 0);
$blue = imagecolorallocate($img, 0, 0, 255);
// დახაზოს რამდენიმე მონაკვეთი
imageline($img, 40, 30, 40, 40, $green);
imageline($img, 50, 30, 50, 40, $green);
imageline($img, 45, 38, 45, 39, $green);
imageline($img, 37, 45, 53, 45, $green);
imageline($img, 37, 43, 37, 45, $green);
imageline($img, 53, 43, 53, 45, $green);
// გამოსახულების ბრაუზერში გამოტანა
header("Content-type: image/png");
imagepng($img);
// მესხიერების გათავისუფლება
imagedestroy($img);
?>
```



## ფუნქცია ImageLine ()

ამ ფუნქციის მეშვეობით ხდება წრფის მონაკვეთის გავლება. მისი ფორმატია:

**imageline ( resource \$image , int \$x1 , int \$y1 , int \$x2 , int \$y2 ,  
int \$color )**

\$image არგუმენტით ფუნქცია მებსიერებაში მიმართავს imageCreateTrueColor () ფუნქციით და ფერის იდენტიფიკატორი imageColorAllocate () ფუნქციით შექმნილ გამოსახულებას. ხოლო \$x1 არის წრფის მონაკვეთის საწყისი წერტილის x-კოორდინატი, \$y1 - საწყისი წერტილის y -კოორდინატი, \$x2 - ბოლო წერტილის x-კოორდინატი, \$y2 - ბოლო წერტილის y-კოორდინატი, \$color - ხაზის ფერი, რომელიც ფერის იდენტიფიკატორი imageColorAllocate () ფუნქციითა შექმნილი.

მოცემული ფუნქცია მოქმედების წარმატებით დამთავრების შემთხვევაში შედეგად აბრუნებს TRUE-ს, ხოლო შეცდომის შემთხვევაში FALSE-ს.

ზუსტად ასე მუშაობს გამოსახულებასთან ყველა ფუნქცია.

## ფუნქცია imagecopyresampled ()

ამ ფუნქციის მეშვეობით ხდება გამოსახულებიდან მისი ნაწილის ამოჭრა. მისი ფორმატია:

**imagecopyresampled ( resource \$dst\_image , resource  
\$src\_image , int \$dst\_x , int \$dst\_y , int \$src\_x , int \$src\_y ,  
int \$dst\_w , int \$dst\_h , int \$src\_w , int \$src\_h )**

ეს ფუნქცია ერთი სურათიდან მართკუთხედს ამოჭრის, ცვლის მის ზომებს და სხვა სურათში სვამს. ეს ერთ-ერთი ყველაზე მრავალრიცხოვანარგუმენტიანი ფუნქციაა.

`$src_image`, `$dst_image` - პირველი ორი არგუმენტი გვიჩვენებს საიდან უნდა ამოჭრას და სად ჩასვას. ორივე არგუმენტი მიუთითებს გამოსახულებაზე მეხსიერებაში.

`$dst_x`, `$dst_y`, `$src_x`, `$src_y` - ეს არის მართკუთხედის მარცხენა ზედა კუთხის კოორდინატები ახალ და ძველ გამოსახულებებზე.

`$dst_w`, `$dst_h`, `$src_w`, `$src_h` - ეს არის ამ მართკუთხედის სიმაღლე და სიგანე შესაბამისად ახალ და ძველ გამოსახულებებში.

სხვა სიტყვებით რომ ვთქვათ, ფუნქცია მოძებნის `$src_image` სურათს, მასში ამოჭრის მართკუთხედს, რომლის მარცხენა ზედა კუთხე განთავსებულია წერტილში კოორდინატებით `$src_x`, `$src_y` და რომლის სიმაღლე და სიგანეა `$src_w`, `$src_h`. ამის შემდეგ, ამოჭრილ ნაწილს ფუნქცია უცვლის სიმაღლეს და სიგანეს ისე, რომ იგი შეესაბამებოდეს `$dst_w`, `$dst_h` რიცხვებს და მიღებულ მართკუთხედს `$dst_image` გამოსახულებაში ჩასვამს ისე, რომ მართკუთხედის ზედა მარცხენა წვერო განთავსებული იყოს წერტილში კოორდინატებით `$dst_x`, `$dst_y`.

**ფუნქციები `imagecreatefromgif ()`, `imagecreatefrompng ()`, `imagecreatefromjpeg ()`**

ეს ფუნქციები ფაილებიდან ან URL-მისამართიდან ახალ გამოსახულებას ქმნიან. მათი ფორმატია:

**`imagecreatefromgif (string $filename)`**

**`imagecreatefrompng (string $filename)`**

**`imagecreatefromjpeg (string $filename)`**

ეს ფუნქციები შედეგად აბრუნებენ მოცემული `$filename` ფაილიდან მიღებული გამოსახულების იდენტიფიკატორს.

აქვე ყურადღება გვინდა გავამახვილოთ იმაზე, რომ არსებობს გამოსახულებათა სხვა ფორმატებიც და მათთვის სხვა, მათი შესაბამისი, ფუნქციები გამოიყენება.

### **ფუნქცია imagegif ()**

ამ ფუნქციის საშუალებით ინახება დამუშავებული გამოსახულება. Imagegif () ფუნქციას გამოსახულება ბრაუზერში ან ფაილში გამოჰყავს. მისი სინტაქსია:

**imagegif (resource \$image [, string \$filename])**

imagegif () ფუნქცია image გამოსახულებიდან GIF-ფაილს ქმნის. \$image არგუმენტს imagecreate () ფუნქცია აბრუნებს.

გამოსახულების ფორმატი იქნება GIF87a, თუ imagecolortransparent () (გამოსახულების ფერს განსაზღვრავს, როგორც გამჭირვალეს) ფუნქციის საშუალებით გამოსახულება არ გახდა გამჭირვალე - მაშინ ფორმატი იქნება GIF89a.

\$filename არგუმენტი აუცილებელი არ არის. თუ ის მითითებულია ფუნქცია გამოსახულებას მოცემულ ფაილში აგზავნის, თუ არა, მაშინ ბრაუზერში გამოჰყავს.

### **ფუნქცია imagepng ()**

ამ ფუნქციის საშუალებითაც დამუშავებული გამოსახულება ინახება. imagepng () ფუნქციის სინტაქსია:

**imagepng (resource \$image [, string \$filename])**

imagepng () ფუნქციას გამოსახულების (\$image) GD-ნაკადი PNG ფორმატით სტანდარტულად ბრაუზერში გამოჰყავს ან, თუ \$filename არგუმენტით ფაილის სახელია მოცემული, ფაილში.

## ფუნქცია imagejpeg ()

imagejpeg () ფუნქციას გამოსახულება ბრაუზერში ან ფაილში გამოჰყავს. მისი სინტაქსია:

**imagejpeg (resource \$image [, string \$filename [, int \$quality]])**

imagejpeg () ფუნქცია \$image გამოსახულებიდან JPEG-ფაილს ქმნის.

\$filename არგუმენტი აუცილებელი არ არის. იმ შემთხვევაში, თუ \$quality არგუმენტის მითითება აუცილებელია, მაშინ ცარიელი სტრიქონი (") უნდა გამოვიყენოთ.

*შენიშვნა: JPEG ფორმატის მხარდაჭერა მხოლოდ იმ შემთხვევაშია შესაძლებელი, თუ PHP-ს კომპილირება GD-1.8-სთან ან უფრო ახალ ვერსიასთან ერთად განხორციელდა.*

\$quality არგუმენტი აუცილებელი არ არის და მისი მნიშვნელობის დიაპაზონი 0-დან (ყველაზე ცუდი ხარისხი, ყველაზე მცირე მოცულობის ფაილი) 100-მდეა (საუკეთესო ხარისხი, ყველაზე დიდი მოცულობის ფაილი). ჩუმათობის პრინციპით IJG quality (დაახლოებით 75) მნიშვნელობა გამოიყენება.

მაგრამ ერთი პატარა პრობლემა არსებობს - ბრაუზერს უნდა მივუთითოთ, რომ მოცემული გამოსახულება დაამუშაოს, როგორც სურათი და არა, როგორც ჩვეულებრივი ტექსტი.

სათაურის გასაგზავნად, header () ფუნქცია გამოიყენება. სათაურის შინაარსი ასე გამოიყურება (დოკუმენტის ტიპი: სურათი/ფორმატი). მაგალითად,

```
header("Content-type: image/gif");  
header("Content-type: image/png");
```

## ფუნქცია `imagedestroy ()`

ეს ფუნქცია ასუფთავებს მეხსიერებას. მისი ფორმატია:

### **`imagedestroy (resource $image)`**

`imagedestroy ()` ფუნქცია `$image` გამოსახულებას არ შლის, მაგრამ, მასთან ასოცირებულ მეხსიერებას ათავისუფლებს. `$image` არის გამოსახულების იდენტიფიკატორი. ეს ფუნქცია განსაკუთრებით მაშინ გამოდგება, როდესაც დიდი რეზოლუციის მქონე ან პარალელურად რამდენიმე გამოსახულებასთან გვიწევს მუშაობა, ამასთან, თავისუფალი მეხსიერების დასაშვებ საზღვრებს არ უნდა გავცდეთ.

აქვე უნდა აღინიშნოს, რომ სკრიპტისათვის გამოყოფილი მეხსიერება შესრულების შემდეგ ავტომატურად თავისუფლდება.

## ფუნქცია `getimagesize ()`

ამ ფუნქციის საშუალებით გამოსახულების ზომის მიღებაა შესაძლებელი. მისი ფორმატია:

### **`getimagesize ( string $filename [, array &$imageinfo ] )`**

`getimagesize ()` ფუნქცია მოცემული გამოსახულების ზომას განსაზღვრავს და ფაილის ტიპთან და `height/width` ტექსტურ სტრიქონთან ერთად დააბრუნებს მის ზომას, რომელიც შემდეგ HTML-ის `IMG` ტეგში შეიძლება იყოს გამოყენებული, ასევე HTTP შიგთავსის შესაბამის ტიპს დააბრუნებს.

ამის გარდა, `getimagesize ()` ფუნქციას შეუძლია `$imageinfo` არგუმენტის საშუალებით გამოსახულების შესახებ დამატებითი ინფორმაცია მოგვაწოდოს.

`$imageinfo` ეს არააუცილებელი არგუმენტია, რომელიც საშუალებას იძლევა ზოგიერთი გაფართოებული მონაცემი გამოსახულების ფაილიდან ამოიღოს. დღესდღეობით, სხვადასხვა

JPG APP მარკერი ასოცირებული მასივის სახით შეიძლება მივიღოთ. ზოგიერთი პროგრამა ამ მარკერებს სურათზე ტექსტის განსათავსებლად იყენებს.

`getimagesize ()` ფუნქცია `$imageinfo` არგუმენტის მითითების შემთხვევაში შვიდელემენტიან მასივს აბრუნებს.

მასივის ელემენტებში ინდექსით 0 და 1 მოცემულია გამოსახულების სიგრძისა და სიმაღლის მნიშვნელობები.

***შენიშვნა:** ფაილის ზოგიერთი ფორმატი შეიძლება რამდენიმე სურათს ინახავდეს ან გამოსახულებას საერთოდ არ შეიცავდეს. ამ შემთხვევაში `getimagesize ()` ფუნქცია გამოსახულების ზომების დადგენას ვერ შეძლებს. ამიტომ, `getimagesize ()` ფუნქცია სურათის სიგრძისა და სიმაღლის მნიშვნელობად ნულებს დააბრუნებს.*

მასივის ელემენტი ინდექსით 2 გამოსახულების ტიპის ერთ-ერთ კონსტანტას `IMAGETYPE_XXX` შეიცავს;

მასივის ელემენტით ინდექსით 3 მოცემულია ინფორმაცია სტრიქონს გამოსახულების სიგრძისა და სიმაღლის შესახებ შემდეგი მნიშვნელობით `height="yyy" width="xxx"`, რომელიც შემდეგ შეიძლება `IMG` ტეგში იყოს გამოყენებული.

შემდეგი ელემენტი არის `mime` - რომელიც გამოსახულების `MIME`-ტიპს შეესაბამება. ეს ცნობები `Content-type` სათაურის საფუძველზე გამოსახულების კორექტული დამუშავებისათვის გამოიყენება. ქვემოთა მოყვანილია `getimagesize ()` ფუნქციისა და `MIME`-ტიპის გამოყენების მაგალითი:

```
<?php
$size = getimagesize($filename);
$fp = fopen($filename, "rb");
if ($size && $fp) {
```



```
header("Content-type: {$size['mime']}");
fpassthru($fp);
exit;
} else {
    // შეცდომაა
}
?>
```

შემდეგი ელემენტია channels, რომელიც RGB სურათისათვის 3-ის ტოლ, ხოლო CMYK სურათისათვის 4-ის ტოლ მნიშვნელობას იღებს.

და ბოლოს, ელემენტი bits, ეს არის ფერის სიღრმე - ბიტების რაოდენობა თითოეული ფერისათვის.

ზოგიერთი ტიპის გამოსახულებისათვის channels და bits პარამეტრების მნიშვნელობის არსებობა შეიძლება გაუგებარი იყოს. მაგალითად, GIF-ი პიქსელზე ყოველთვის 3 არხს იყენებს, მაგრამ ფერთა საერთო ცხრილიდან ანიმაციური GIF გამოსახულები-სათვის ფერთა სიღრმის გამოთვლა შეუძლებელია.

**შენიშვნა:** ყურადღება მიაქციეთ იმ ფაქტს, რომ JPC და JP2 გამოსახულებების ცალკეულ ნაწილებს შეიძლება ფერთა სხვადასხვა სიღრმე ჰქონდეს. ამ შემთხვევაში bits პარამეტრში ყველა აღმოჩენილ მნიშვნელობას შორის მაქსიმალური მნიშვნელობა იქნება გამოტანილი. ასევე, JP2 ფაილები შეიძლება რამდენიმე JPEG 2000 კოდურ ნაკადს შეიცავდეს. ამ შემთხვევაში, ფუნქციის მიერ ფაილში პირველად აღმოჩენილი ასეთი ნაკადის მნიშვნელობას გამოიტანს.

ფუნქციის შეცდომის შემთხვევაში გამოიტანს მნიშვნელობას FALSE.

## მომხმარებლის ფუნქცია PHP-ში

დაპროგრამების ნებისმიერ ენაში არსებობს ქვეპროგრამა. C ენაში მათ ფუნქციები ჰქვია, ასემბლერში - ქვეპროგრამები, ხოლო Pascal-ში ორი სახის ქვეპროგრამა არსებობს: პროცედურა და ფუნქცია.

ქვეპროგრამა - ეს სპეციალური სახით გაფორმებული პროგრამის ფრაგმენტია, რომელსაც პროგრამაში ნებისმიერი ადგილიდან შეიძლება მიმართო. ქვეპროგრამა არსებითად ამარტივებს პროგრამისტის მუშაობას, აუმჯობესებს საწყისი კოდის წაკითხვას, აგრეთვე ამცირებს მის მოცულობას, ვინაიდან კოდის ცალკეული ფრაგმენტების რამდენიმეჯერ ჩაწერა არ არის საჭირო.

PHP-ში ასეთ ქვეპროგრამებს მომხმარებლის ფუნქციები წარმოადგენს.

## PHP-ის მომხმარებლის ფუნქციის თავისებურებანი

ჩამოვთვალოთ PHP-ის მომხმარებლის ფუნქციის თავისებურებანი:

- პარამეტრები მისაწვდომია ჩუმათობის პრინციპით. არის იმის საშუალება, რომ ერთი და იგივე ფუნქცია გამოძახებულ იქნეს ცვალებადი რაოდენობის პარამეტრებით;
- მომხმარებლის ფუნქციამ შესრულების შედეგად შეიძლება დააბრუნოს ნებისმიერი ტიპის მონაცემი;
- ფუნქციის შიგნით ცვლადების მოქმედების არე იერარქიულია;
- არსებობს არგუმენტის სახით გადაცემული ცვლადის შეცვლის საშუალება.

მომხმარებლის ფუნქციის გამოყენების შემთხვევაში დგება ცვლადების მოქმედების არის საკითხი. ცვლადები მოქმედების არის მიხედვით იყოფა გლობალურ და ლოკალურ ცვლადებად.

გლობალური ცვლადი - ეს ის ცვლადია, რომელიც მთელი პროგრამისათვის, მათ შორის ქვეპროგრამისათვისაც (ფუნქციისათვის) მისაწვდომია.

ლოკალური ცვლადი ის ცვლადია, რომელიც ქვეპროგრამის (ფუნქციის) შიგნით არის განსაზღვრული.

PHP-სათვის ფუნქციაში ყველა გამოცხადებული და გამოყენებული ცვლადები ჩუმათობის პრინციპით ლოკალურია. ანუ, ჩუმათობის პრინციპით ფუნქციის ტანში გლობალური ცვლადის მნიშვნელობის შეცვლა შეუძლებელია.

თუ მომხმარებლის ფუნქციის ტანში გამოყენებული იქნება ცვლადი, რომლის სახელი გლობალური ცვლადის სახელის იდენტური იქნება, მაშინ გლობალურ ცვლადსა და ამ ლოკალურ ცვლადს შორის არავითარი კავშირი არ იქნება. განვიხილოთ აღნიშნული ფაქტი კონკრეტულ მაგალითზე:

```
<?php
$a = 100;

function funct() {
    $a = 70;
    echo "<h4>$a</h4>";
}
funct();
echo "<h2>$a</h2>";
?>
```

მოცემული სკრიპტი ჯერ 70-ს გამოიტანს, ხოლო შემდეგ 100-ს.

```
70  
100
```

ზემოთ მოყვანილი ნაკლოვანების თავიდან ასაცილებლად PHP-ში არსებობს სპეციალური ინსტრუქცია `global`, რომელიც მომხმარებლის ფუნქციას გლობალურ ცვლადებთან მუშაობის საშუალებას აძლევს. მოცემული პრინციპი განვიხილოთ კონკრეტულ მაგალითზე:

```
<?php  
$a = 1;  
$b = 2;  
  
function Sum()  
{  
    global $a, $b;  
  
    $b = $a + $b;  
}  
  
Sum();  
echo "<h1>$b</h1>";  
?>
```

მოცემული სკრიპტის მუშაობის შედეგად გამოტანილი იქნება „3“. ფუნქციის შიგნით \$a და \$b ცვლადების განსაზღვრა, როგორც global-ის განსაზღვრა, შემდგომში ნებისმიერ ამ ცვლადზე მიმართვა მათ გლობალურ ვერსიაზე მიუთითებს. მომხმარებლის ფუნქციით დასამუშავებელი გლობალური ცვლადების რაოდენობაზე არავითარი შეზღუდვა არ არსებობს.

3

გლობალურ ცვლადებზე წვდომის მეორე ხერხი არის, სპეციალური, PHP მასივის განმსაზღვრელის \$GLOBALS-ის გამოყენება. ასეთ შემთხვევაში ზემოთ მოყვანილი მაგალითი შეიძლება ასე გადაიწეროს:

```
<?php
$a = 1;
$b = 2;

function Sum()
{
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Sum();
echo "<h1>$b</h1>";
?>
```

შედეგი აქაც იგივე იქნება.

\$GLOBALS - ეს არის ასოცირებული მასივი, რომლის გასაღებს გლობალური ცვლადის სახელი, ხოლო მნიშვნელობას ამ ცვლადის მნიშვნელობა წარმოადგენს. ყურადღება მიაქციეთ იმას, რომ \$GLOBALS ხედვის ნებისმიერ არეში არსებობს, ეს იმით აიხსნება, რომ ეს მასივი სუპერგლობალურია.

ლოკალური და გლობალური ცვლადების გარდა PHP-ში არსებობს კიდევ ერთი ტიპის ცვლადი - სტატიკური ცვლადი.

თუ მომხმარებლის ფუნქციის ტანში გამოცხადებულია სტატიკური ცვლადი, მაშინ კომპილატორი ფუნქციის მუშაობის დამთავრების შემდეგ მის მნიშვნელობას არ წაშლის. ქვემოთ მოყვანილია სტატიკური ცვლადის შემცველი მომხმარებლის ფუნქციის მუშაობის მაგალითი:

```
<?php
function funct()
{
    static $a;
    $a++;
    echo "$a."<br>";
}
for ($i = 0; $i++<10;) funct();
?>
```

მოცემული სკრიპტის მუშაობის შედეგად მივიღებთ:

```
1
2
3
4
5
6
7
8
9
10
```

თუ ამ სკრიპტში static-ს წავშლით, მაშინ მივიღებთ შედეგს:

```
Notice: Undefined variable: a in C:\xampp\htdocs\mag21.php on line 5
1
Notice: Undefined variable: a in C:\xampp\htdocs\mag21.php on line 5
1
Notice: Undefined variable: a in C:\xampp\htdocs\mag21.php on line 5
1
```

ეს დაკავშირებულია იმასთან, რომ ფუნქციის მუშაობის დამთავრების შემდეგ \$a ცვლადის მნიშვნელობა წაიშლება და ყოველი გამოძახების დროს მოხდება მისი განულება. განულების შემდეგ მოხდება \$a ცვლადის მნიშვნელობის ინკრემენტაცია და მხოლოდ ამის შემდეგ მისი გამოტანა.

## მომხმარებლის ფუნქციის შექმნა

მომხმარებლის ფუნქცია პროგრამის (სკრიპტის) ნებისმიერ ნაწილში შეიძლება იყოს გამოცხადებული, ოღონდ მისი პირველი გამოძახების წინ. მას არ სჭირდება წინასწარი აღწერა.

მომხმარებლის ფუნქციის გამოცხადების სინტაქსი ასეთია:

```
function ფუნქციის_სახელი (არგუმენტი1[=მნიშვნელობა1],...,  
არგუმენტიN[=მნიშვნელობაN])  
{  
ფუნქციის ტანი  
}
```

ფუნქციის გამოცხადება სპეციალური სიტყვით function-ით იწყება, შემდეგ მას მოსდევს ფუნქციის სახელი, ხოლო ფუნქციის სახელს - მრგვალ ფრჩხილებში მოთავსებული არგუმენტთა სია. ფუნქციის ტანი ფიგურულ ფრჩხილებში უნდა იყოს მოთავსებული და შეიძლება ნებისმიერი რაოდენობის ოპერატორებს შეიცავდეს.

ფუნქციის სახელი შემდეგ მოთხოვნებს უნდა აკმაყოფილებდეს:

- ფუნქციის სახელი უნდა შეიცავდეს ლათინური ანბანის ასოებს;
- ფუნქციის სახელი არ უნდა შეიცავდეს ჰარს (ინტერვალს);
- მომხმარებლის ფუნქციის ყოველი სახელი უნდა იყოს უნიკალური. ამასთან, უნდა გვახსოვდეს, რომ ფუნქციის გამოცხადებისა და მიმართვის დროს რეგისტრს ყურადღება არ ექცევა. ანუ, მაგალითად, ფუნქცია funct() და FUNCT() ერთი და იგივეა;
- ფუნქციას შეიძლება მივცეთ იგივე სახელი, რაც ცვლადს აქვს, ოღონდ სახელის წინ „\$“ სიმბოლოს გარეშე.



მომხმარებლის ფუნქციის მიერ დაბრუნებული მონაცემი შეიძლება ნებისმიერი ტიპის იყოს. მომხმარებლის ფუნქციის მუშაობის შედეგის ძირითად პროგრამაში (სკრიპტში) გადასაცემად კონსტრუქცია return გამოიყენება. თუ ფუნქცია არაფერს არ აბრუნებს, მაშინ კონსტრუქცია return არ მიეთითება.

## კონსტრუქცია return

კონსტრუქცია return აბრუნებს მნიშვნელობებს, ძირითადად მომხმარებლის ფუნქციებიდან, როგორც ფუნქციური მოთხოვნის პარამეტრებს. return კონსტრუქციის გამოძახების შემთხვევაში მომხმარებლის ფუნქციის შესრულება წყდება, ხოლო return კონსტრუქცია გარკვეულ მონაცემებს აბრუნებს.

თუ return კონსტრუქცია გამოძახებული იქნება განსაზღვრის გლობალური არიდან (მომხმარებლის ფუნქციის გარეთ), მაშინ სკრიპტიც დაამთავრებს თავის მუშაობას, ხოლო return-ი გარკვეულ მნიშვნელობას დააბრუნებს.

უმეტეს შემთხვევაში, return კონსტრუქცია მომხმარებლის ფუნქციის მიერ მნიშვნელობის დასაბრუნებლად გამოიყენება.

დაბრუნებული მნიშვნელობა შეიძლება ნებისმიერი ტიპის იყოს, მათ შორის ის შეიძლება იყოს სიები და ობიექტები. მონაცემთა დაბრუნება ფუნქციის შესრულების დამთავრებას იწვევს და მართვა ისევ იმ სტრიქონს უბრუნდება, რომლიდანაც მოცემული ფუნქციის გამოძახება მოხდა.

მოვიყვანოთ return კონსტრუქციის გამოყენების მაგალითი, სადაც იგი integer ტიპის მნიშვნელობას აბრუნებს:

```
<?php
function funct() {
$number = 777;
return $number;
```

```
}  
$a = funct();  
echo "<h2>$a</h2>";  
?>
```

განხილულ მაგალითში ფუნქცია `funct`, `return` კონსტრუქციის საშუალებით რიცხვ 777-ს აბრუნებს. ფუნქციით დასაბრუნებელი მნიშვნელობა `$a` გლობალურ ცვლადს ენიჭება, ხოლო შემდეგ `echo` ოპერატორს `$a` ცვლადის მნიშვნელობა ბრაუზერში გამოჰყავს.



777

ქვემოთ მოვიყვანოთ მაგალითი, სადაც `return` კონსტრუქცია მასივს აბრუნებს:

```
<?php  
function num()  
{  
    return array (0, 1, 2);  
}  
list ($zero, $one, $two) = num();  
echo $zero;  
echo "<br>";  
echo $one;  
echo "<br>";  
echo $two;  
?>
```

```
0
1
2
```

იმისათვის, რომ ფუნქციამ შედეგი ბმულით დააბრუნოს, ოპერატორ &-ის (ამპერსანტი) გამოყენება აუცილებელია, როგორც ფუნქციის აღწერის, ისე ცვლადისათვის დასაბრუნებელი მნიშვნელობის მინიჭების დროს. მაგალითად:

```
<?php
function &r_r()
{
    return $sref;
}

$newref =&r_r();
?>
```

როგორც მაგალითებიდან გამოჩნდა, return კონსტრუქცია მომხმარებლის ფუნქციებში გამოსაყენებლად ძალზე მოსახერხებელია.

### **მომხმარებლის ფუნქციისათვის არგუმენტის გადაცემა**

ფუნქციის გამოცხადების დროს შეიძლება მიეთითოს ის პარამეტრები, რომლებიც ფუნქციას გადაეცემა, მაგალითად:

```
<?php
function funct($a, $b, ..., $z) { ... };
?>
```

func() ფუნქციის გამოძახების დროს ყველა გადასაცემი პარამეტრი უნდა მიეთითოს, ვინაიდან ისინი აუცილებელი პარამეტრებია. მომხმარებლის ფუნქციები PHP-ში შეიძლება არააუცილებელ პარამეტრებს ანუ პარამეტრებს ჩუმათობით შეიცავდეს, რომელთაც შემდეგში გავეცნობით.

### არგუმენტების გადაცემა ბმულით

ტრადიციულად, დაპროგრამების ყველა ენაში, ფუნქციის ორი სახის არგუმენტი არსებობს:

- პარამეტრი-მნიშვნელობა;
- პარამეტრი-ცვლადი.

ფუნქციას პარამეტრი-მნიშვნელობის შეცვლა არ შეუძლია, ანუ ფუნქციისათვის იგი „მხოლოდ წასაკითხად“ არის განკუთვნილი - მას მხოლოდ მისი გამოყენება შეუძლია. პარამეტრი-მნიშვნელობაში არ არის აუცილებელი ცვლადის მითითება, შეიძლება თვითონ მონაცემი მივუთითოთ, მისი სახელიც აქედან მოდის.

ჩუმათობის პრინციპით არგუმენტები ფუნქციაში მნიშვნელობის მიხედვით გადაეცემა (ეს ნიშნავს, რომ თუ არგუმენტის მნიშვნელობას ფუნქციის შიგნით შევცვლით, მაშინ მისი მნიშვნელობა ფუნქციის გარეთ მაინც ძველი დარჩება). მოვიყვანოთ მაგალითი:

```
<?php
function funct($string)
{
```

```

    echo "<h3>პარამეტრი = $string </h3>";
}

$str = 777;
funct(777);
funct($str);
?>

```

ამ პროგრამის მუშაობის შედეგი ორივე შემთხვევაში ერთი და იგივე იქნება:

```

პარამეტრი = 777
პარამეტრი = 777

```

პარამეტრი-მნიშვნელობისაგან განსხვავებით პარამეტრი-ცვლადი ფუნქციის მუშაობის პროცესში შეიძლება შეიცვალოს. აქ უკვე მნიშვნელობის გადაცემა არ შეიძლება, აუცილებლად უნდა მოხდეს ცვლადის გადაცემა. PHP-ში პარამეტრი-ცვლადის გამოცხადებისათვის ცვლადის ბმულით გადაცემა გამოიყენება.

თუ გსურთ ფუნქციას არგუმენტების მოდიფიცირების ნება დართოთ, მაშინ არგუმენტები ბმულით უნდა გადაეცეს. ამისათვის, ფუნქციის აღწერაში არგუმენტის სახელის წინ უნდა მიეთითოს ამპერსანტი (&):

```

<?php
function funct(&$string)
{
    $string .= 'ეს სტრიქონი ფუნქციის შიგნით იმყოფება.';
}

```

```

}
$str = 'ეს სტრიქონი ფუნქციის გარეთ იმყოფება. ';
funct($str);
echo $str;
?>

```

ეს სტრიქონი ფუნქციის გარეთ იმყოფება.  
ეს სტრიქონი ფუნქციის შიგნით იმყოფება.

განვიხილოთ კიდეც ერთი მაგალითი:

```

<?php
$a = 100;
/* ფუნქცია, რომელიც აბრუნებს ბმულს */
function &s () {
global $a;
// $a ცვლადზე აბრუნებს ბმულს
return $a;
}
// $b ცვლადს ვანიჭებთ ბმულს
$b = &s();
$b = 0;
echo $a; // გამოიტანს 0
?>

```

მოცემული სკრიპტი მუშაობის შედეგად გამოიტანს 0-ს, რადგანაც ერთი ცვლადის მნიშვნელობის (\$a ან \$b) შეცვლა ავტომატურად მეორის ცვლილებას იწვევს. ობიექტებსა და

ბმულებს შორის კავშირის გასაწყვეტად გამოიყენება UnSet() ოპერატორი.

### ოპერატორი unset ()

unset () ოპერატორი სპეციფიურ ცვლადებს ანადგურებს (წაშლის). აქვე უნდა აღინიშნოს, რომ PHP 3-ში unset () იყო ფუნქცია და იგი მუშაობის შედეგად აბრუნებდა TRUE-ს, მაგრამ PHP 4-დან ის გახდა ოპერატორი, რადგან შედეგად აღარაფერს აბრუნებს. შედეგის მიღების მცდელობა შეცდომას გამოიწვევს. მისი ფორმატია:

**unset (mixed var [, mixed var [, ...]])**

**შენიშვნა:** *unset () ეს ენის კონსტრუქციაა.*

```
// ერთ ცვლადს წაშლის
unset ($foo);

// მასივის ერთ ელემენტს წაშლის
unset ($bar['quux']);

// რამდენიმე ცვლადს წაშლის
unset ($foo1, $foo2, $foo3);
```

ფუნქციის შიგნით unset () ოპერატორის ქცევა წასაშლელი ცვლადის ტიპზეა დამოკიდებული.

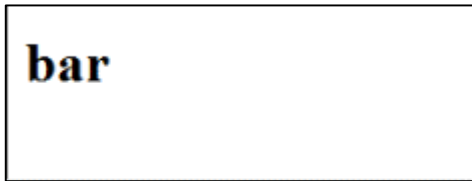
თუ გლობალური ფუნქციის წაშლა unset () ოპერატორით ფუნქციის შიგნით ხდება, მაშინ მხოლოდ ლოკალური ცვლადი წაიშლება. ცვლადის მნიშვნელობა ფუნქციის გარეთ იგივე დარჩება რაც იყო ფუნქციის გამოძახებამდე.

```
<?php
```

```
function destroy_foo() {
    global $foo;
    unset($foo);
}

$foo = 'bar';
destroy_foo();
echo "<h1>".$foo."</h1>";
?>
```

ეს მაგალითი შედეგად გამოიტანს:



თუ ბმულით გადაცემული ცვლადი ფუნქციის შიგნითაა, მაშინ მხოლოდ ლოკალური ცვლადი წაიშლება. მაგალითად:

```
<?php
function foo(&$bar) {
    unset($bar);
    $bar = "HTML";
}
$bar = 'PHP';
echo "<h1>".$bar."</h1><br>";
foo($bar);
echo "<h3>".$bar."</h3>";
?>
```

ამ მაგალითის შედეგია:



**PHP**

**PHP**

თუ static-ცვლადის წაშლა unset() ოპერატორით ფუნქციის შიგნით ხდება, მაშინ unset() ოპერატორი ამ ცვლადს და მასზე ყველა მიმართვას წაშლის.

```
<?php
function foo() {
static $a;
$a++;
echo "$a<br>";
unset($a);
}
foo();
foo();
foo();
?>
```

მაგალითი შედეგად მოგვცემს:

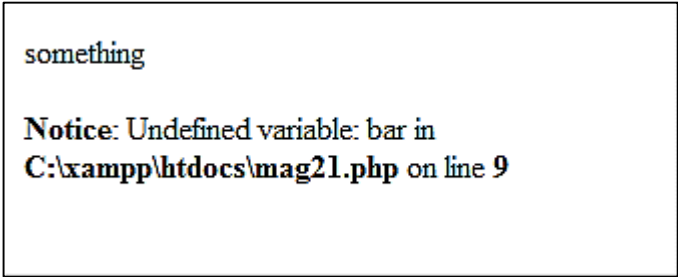
1  
2  
3

თუ ფუნქციის შიგნით გლობალური ცვლადის წაშლა გვსურს, მაშინ შეიძლება \$GLOBALS მასივი იყოს გამოყენებული:

```
<?php
function foo() {
unset($GLOBALS['bar']);
}

$bar = "something";
echo "$bar".<br>;
foo();
echo "$bar".<br>;
?>
```

შედეგი ასეთი იქნება, ვინაიდან წაიშალა გლობალური ცვლადი, ფუნქციის გამოძახების შემდეგ \$bar ცვლადის მნიშვნელობა განსაზღვრული არ არის.



### პარამეტრები ჩუმათობით

PHP-ში ფუნქციებს მათთვის გადაცემულ პარამეტრებზე დამოკიდებულებით ნებისმიერი მნიშვნელობის დაბრუნება შეუძლიათ. მაგალითად:

```
<?php
function mc($type = "ჩაი")
{
    return "გთხოვთ მომიმზადოთ ფინჯანი $type. <br>";
}
echo mc();
echo mc("ყავა");
?>
```

მოცემული სკრიპტის მუშაობის შედეგი ასეთი იქნება:

```
გთხოვთ მომიმზადოთ ფინჯანი ჩაი.
გთხოვთ მომიმზადოთ ფინჯანი ყავა.
```

პარამეტრის ჩუმათობითი მნიშვნელობა კონსტანტა უნდა იყოს.

## რეკურსიული ფუნქციები

რეკურსიული ფუნქცია - ეს ისეთი ფუნქციაა, რომელიც თავისთავს იძახებს. ასეთ გამოძახებას რეკურსიული ეწოდება. რეკურსია არსებობს პირდაპირი და არაპირდაპირი.

განვიხილოთ პირდაპირი რეკურსიული ფუნქციის მაგალითი, რომელიც  $x!$  ფაქტორიალის გამოსათვლელად გამოიყენება:

```
<?php
function factorial($x) {
if ($x === 0) return 1;
else return $x*factorial($x-1);
}
echo factorial(7);
?>
```

ამ პროგრამის (სკრიპტის) შესრულების შედეგი ასეთი იქნება:

5040

განხილულ მაგალითში მომხმარებლის ფუნქცია factorial() თავისთავს იძახებს, რაც არის პირდაპირი რეკურსია.

არაპირდაპირი რეკურსიაა, მაშინ, როდესაც პირველი ფუნქცია იძახებს მეორეს, ხოლო მეორე - პირველს.

რეკურსიული ფუნქციის შექმნის დროს აუცილებელია ფრთხილად ვიყოთ, რათა პროგრამის ჩაციკვლა არ მოხდეს.

ქვემოთ მოყვანილია რეკურსიული ფუნქციის არასწორი გამოყენების მაგალითი:

```
<?php
function factorial($x) {
if ($x === 0) return 1;
else return $x*factorial($x);
}
?>
```

შედეგად განხორციელდება ჩაციკვლა, რადგან ფუნქციაზე მიმართვის დროს არგუმენტის ცვლილება არ ხდება, რაც გამოიწვევს ამ ფუნქციის უსასრულო შესრულებას.

## პირობით განმსაზღვრელი ფუნქციები

PHP საშუალებას იძლევა ერთსა და იმავე ფუნქციაში, გარკვეული ფაქტორებიდან გამომდინარე, სხვადასხვა მოქმედება შესრულდეს. მაგალითად:

```
<?php
$phpver = phpversion();
if ($phpver[0] === "5")
{
function getversion() { return "თქვენ იყენებთ PHP 5-ს"; }
}
if ($phpver[0] === "4")
{
function getversion() { return "თქვენ იყენებთ PHP 4-ს"; }
}
if ($phpver[0] === "3")
{
function getversion() { return "თქვენ იყენებთ PHP 3-ს"; }
}
echo @getversion();
?>
```

თქვენ იყენებთ PHP 5-ს



განხილულ სკრიპტს შედეგად გამოჰყავს PHP ინტერპრეტატორის მიერ გამოყენებული ვერსია. ერთი და იმავე `getversion()` ფუნქციას `$phpver` ცვლადის მნიშვნელობის მიხედვით სხვადასხვა შედეგი შეუძლია დააბრუნოს.

## ცვლადი რაოდენობის არგუმენტები ფუნქციაში

ზოგჯერ ზუსტად არ ვიცით რამდენი პარამეტრი გადაეცემა ჩვენს ფუნქციას. სპეციალურად ასეთი შემთხვევისათვის PHP-ის შემქმნელებმა ცვლადი რაოდენობის არგუმენტის გამოყენების შესაძლებლობა გაითვალისწინეს.

ამ შესაძლებლობის გამოყენების რეალიზაცია საკმაოდ გამჭირვალეა და რამდენიმე სტანდარტული ფუნქციის გამოყენებით გამოიხატება. ეს ფუნქციებია: `func_num_args()`, `func_get_arg()` და `func_get_args()`.

### ფუნქცია `func_num_args()`

სტანდარტული ფუნქცია `func_num_args()` მუშაობის შედეგად აბრუნებს მომხმარებლის ფუნქციისათვის გადაცემული არგუმენტების რაოდენობას:

```
<?php
function funct()
{
    $numarg = func_num_args();
    echo "არგუმენტების რაოდენობა : $numarg.<br>";
}
```

```
}  
funct(1, 2, 3);  
?>
```

არგუმენტების რაოდენობა : 3

### ფუნქცია `func_get_arg()`

სტანდარტული ფუნქცია `func_get_arg()` შესრულების შედეგად მომხმარებლის ფუნქციისათვის გადაცემული არგუმენტების სიიდან აბრუნებს მითითებულ ელემენტს:

```
<?php  
function funct()  
{  
    $numargs = func_num_args();  
    echo "არგუმენტების რაოდენობა : $numargs<br>\n";  
    if ($numargs >= 2) {  
        echo "მეორე არგუმენტი : ".func_get_arg(1)."<br>\n";  
    }  
}  
  
funct(1, 2, 3);  
?>
```

არგუმენტების რაოდენობა : 3  
მეორე არგუმენტი : 2

### ფუნქცია `func_get_args ()`

სტანდარტული ფუნქცია `func_get_args()` შესრულების შედეგად აბრუნებს მომხმარებლის ფუნქციისათვის გადაცემული არგუმენტების მასივს:

```
<?php
function funct()
{
    $numargs = func_num_args();
    echo "არგუმენტების რაოდენობა : $numargs<br>\n";
    if ($numargs >= 2) {
        echo "მეორე არგუმენტი : " . func_get_arg(1) . "<br>\n";
    }
    $arg_list = func_get_args();
    for ($i = 0; $i < $numargs; $i++) {
        echo "არგუმენტი $i არის: " . $arg_list[$i] . "<br>\n";
    }
}

funct(1, 2, 3);
?>
```



არგუმენტების რაოდენობა : 3

მეორე არგუმენტი : 2

არგუმენტი 0 არის: 1

არგუმენტი 1 არის: 2

არგუმენტი 2 არის: 3

ყურადღება მიაქციეთ იმას, რომ მომხმარებლის ფუნქციის გამოცხადების დროს ფრჩხილებში არგუმენტს არ ვუთითებთ, თითქოს არავითარი არგუმენტის გადაცემას არ ვაპირებთ.

# ობიექტზე ორიენტირებული დაპროგრამების საფუძვლები

ბოლო დროს ობიექტზე ორიენტირებული დაპროგრამების (Object-Oriented Programming - OOP) იდეა, პროგრამის დაწერის კარდინალურად ახალი იდეოლოგია, სულ უფრო დიდ მოწონებით სარგებლობს პროგრამისტებს შორის.

ობიექტზე ორიენტირებული პროგრამები უფრო მარტივი და მობილურია, მათი მოდიფიცირება და თანხლება ბევრად მარტივია, ვიდრე ჩვეულებრივი, ტრადიციული პროგრამების. ამის გარდა, თვითონ ობიექტზე ორიენტირებულის იდეის კომპეტენტურად გამოყენების შემთხვევაში, შექმნილი პროგრამები ბევრად უფრო დაცულია შეცდომებისაგან.

ობიექტზე ორიენტირებული დაპროგრამება შეიქმნა იმ პროცედურული დაპროგრამების იდეოლოგიის განვითარების შედეგად, სადაც მონაცემები და მათი დასამუშავებელი ქვეპროგრამები (პროცედურები, ფუნქციები) ფორმალურად ერთმანეთთან კავშირში არ არიან.

დაპროგრამების პირველი ენა, რომელშიც ობიექტზე ორიენტირებული დაპროგრამების პრინციპები გამოიყენეს იყო სიმულა. მისი შექმნის დროს 1967 წელს რევოლუციური იდეები გაჩნდა: ობიექტები, კლასები, ვირტუალური მეთოდები და სხვა, თუმცა მაშინდელი სპეციალისტების მიერ ყოველივე ეს არ იქნა აღქმული, როგორც გრანდიოზული მოვლენა. ამის მიუხედავად, კონცეფციების უმეტესობა ალან კემისა და დენ ინგალსის მიერ Smalltalk ენაში იქნა რეალიზებული. სწორედ, იგი გახდა პირველი, ფართოდ გავრცელებული ობიექტზე ორიენტირებული დაპროგრამების ენა.

ამჟამად, გამოყენებითი დაპროგრამების ის ენებია მეტი, რომლებშიც ობიექტზე ორიენტირებული განხორციელების

მეთოდების რეალიზება ხდება, ვიდრე ის ენები, რომლებიც დაპროგრამების სხვა მეთოდებს იყენებენ. სისტემური დაპროგრამების სფეროში ჯერ კიდევ პროცედურული დაპროგრამების მეთოდები გამოიყენება და საყოველთაოდ მიღებულ დაპროგრამების ენას C წარმოადგენს. ოპერაციული სისტემის სისტემური და გამოყენებითი დონეების ურთიერთქმედების დროს ობიექტზე ორიენტირებულმა დაპროგრამებამ შესამჩნევი გავლენის მოხდენა დაიწყო. მაგალითად, ერთ-ერთ ყველაზე გავრცელებულ მულტი-პლატფორმული დაპროგრამების ბიბლიოთეკას C++ ენაზე დაწერილი ობიექტზე ორიენტირებული ბიბლიოთეკა Qt წარმოადგენს.

PHP ბოლო დრომდე ობიექტზე ორიენტირებული დაპროგრამების მხოლოდ გარკვეულ მხარდაჭერას უზრუნველყოფდა. მაგრამ, PHP 5-ის გამოსვლის შემდეგ პრაქტიკულად, ობიექტზე ორიენტირებული დაპროგრამების სრული მხარდამჭერი გახდა.

ყველაზე კარგი იქნება ობიექტზე ორიენტირებული დაპროგრამების სტრატეგიის აღწერა დაპროგრამების პროცესში მოხდეს, როგორც დანართების ფუნქციურობიდან მონაცემთა სტრუქტურებისაკენ წანაცვლება. ეს პროგრამისტს შესაქმნელ დანართებში რეალური ობიექტებისა და სიტუაციის მოდელირების საშუალებას მისცემს. ობიექტზე ორიენტირებული დაპროგრამების ტექნოლოგიას სამი მთავარი უპირატესობა გააჩნია:

- იგი გასაგებად ადვილია: ობიექტზე ორიენტირებული დაპროგრამება ყოველდღიური ობიექტების კატეგორიებით აზროვნების საშუალებას იძლევა;
- გაზრდილია საიმედოობა და მარტივია მხარდაჭერისათვის - სწორი დაპროექტება ობიექტზე ორიენტირებული პროგრამის გაფართოებისა და მოდიფიკაციის სიმარტივეს უზრუნველყოფს. მოდულური სტრუქტურა პროგრამის ცალკეულ ნაწილებში დამოუკიდებელი ცვლილებების

შეტანის საშუალებას იძლევა და ამასთან ერთად დაპროგრამების შეცდომების რისკი მინიმუმამდეა დაყვანილი;

- დამუშავების ციკლს აჩქარებს - მოდულურობა აქაც მნიშვნელოვან როლს ასრულებს, ვინაიდან ობიექტზე ორიენტირებული პროგრამის ცალკეული კომპონენტები სხვა პროგრამებში ძალზე მარტივად შეიძლება იქნეს გამოყენებული, რაც კოდის მოცულობას ამცირებს და კოპირების დროს დაშვებული შეცდომების რისკს ამცირებს.

ობიექტზე ორიენტირებული დაპროგრამების სპეციფიკა პროგრამისტის შრომის ნაყოფიერებას საკმაოდ ამაღლებს და მათ საშუალებას აძლევს უფრო მძლავრი, მასშტაბური და ეფექტური დანართები შექმნან.

ობიექტზე ორიენტირებული დაპროგრამების პრინციპებს წარმოადგენს:

ინკაპსულაცია;

პოლიმორფიზმი;

მემკვიდრეობითობა.

## **ინკაპსულაცია**

ობიექტზე ორიენტირებული დაპროგრამების მრავალი უპირატესობა მისი ერთ-ერთი ფუნდამენტური პრინციპით - ინკაპსულაციითა განპირობებული. ინკაპსულაცია არის სისტემის თვისება, რომელიც მონაცემებისა და მეთოდების მათთან მომუშავე კლასში გაერთიანებისა და მომხმარებლისაგან რეალიზაციის დეტალების დაფარვის საშუალებას იძლევა. ინკაპსულაციის საშუალებით სხვადასხვა მცირე ელემენტის უფრო მსხვილ ელემენტებში ჩართვა ხდება, რის შედეგადაც პროგრამისტი უშუალოდ ამ ობიექტთან მუშაობს. ეს პროგრამის გამარტივებას

იწვევს, რადგან მასში მეორეხარისხოვანი დეტალები გამოირიცხება.

ინკაპსულაციასა და ობიექტზე ორიენტირებულ დაპროგრამებაში „შიგა მოწყობის“ მრავალი წვრილმანი მომხმარებლისგან დაფარულია, რაც მას საშუალებას აძლევს მთელი ყურადღება კონკრეტული ამოცანის გადაწყვეტაზე გადაიტანოს. ობიექტზე ორიენტირებულ დაპროგრამებაში ამ შესაძლებლობების უზრუნველყოფა კლასებით, ობიექტებითა და მათ შორის იერარქიული კავშირების სხვადასხვა საშუალების გამოსახვით ხდება.

### **პოლიმორფიზმი**

პოლიმორფიზმი - ეს არის სისტემის თვისება ერთნაირი ინტერფეისის მქონე ობიექტები ობიექტის ტიპისა და შიგა სტრუქტურის ინფორმაციის გარეშე გამოიყენოს.

პოლიმორფიზმი საშუალებას იძლევა მსგავსი, მაგრამ ტექნიკურად სხვადასხვა ამოცანისათვის ერთი და იგივე სახელები გამოიყენოს. პოლიმორფიზმში მთავარია, რომ იგი მსგავსი ქმედებებისათვის სტანდარტული ინტერფეისის შექმნის გზით ობიექტებით მანიპულირების საშუალებას იძლევა. პოლიმორფიზმი რთული პროგრამების დაწერის მნიშვნელოვანი ხელშემწყობია.

### **მემკვიდრეობითობა**

მემკვიდრეობითობა არის სისტემის თვისება, რომელიც უკვე არსებულის საფუძველზე ახალი კლასი აღწერის საშუალებას იძლევა და ფუნქციონირებას ნაწილობრივ ან სრულად კისრულობს. კლასს, რომლისგანაც მომდინარეობს მემკვიდრეობა,

საბაზო, საბაზო ან სუპერ კლასი ეწოდება. ახალ კლასს კი მემკვიდრე ან ქვეკლასი ეწოდება.

მემკვიდრეობითობა ერთ ობიექტს საშუალებას აძლევს სხვა ობიექტის თვისება შეიძინოს, ეს ობიექტის კოპირებაში არ აურითოთ. კოპირების დროს ობიექტის ზუსტი ასლი იქმნება, მემკვიდრეობითობის დროს კი ობიექტის ზუსტი ასლი მისი იმ უნიკალური თვისებებით ივსება, რომელიც მხოლოდ წარმოქმნილი ობიექტისათვისაა დამახასიათებელი.

## ძირითადი ცნებები

აბსტრაქცია არის ობიექტის მნიშვნელოვანი მახასიათებლების ნაკრების გამოყოფისა და განხილვიდან უმნიშვნელო მახასიათებლების გამორიცხვის საშუალებანი. შესაბამისად, აბსტრაქცია - ყველა ასეთი მახასიათებლის ნაკრებია.

კლასი არის ჯერ კიდევ არ არსებული არსის (ობიექტის) ტერმინოლოგიის ენაზე აღწერილი მოდელის საწყისი კოდი. ფაქტობრივად იგი ობიექტის მოწყობას აღწერს და თავისებურ ნახაზს წარმოადგენს. ამბობენ, რომ ობიექტი ეს კლასის ეგზემპლარია. ამასთან, ზოგიერთ შესრულებად სისტემაში კლასი მონაცემთა ტიპის დინამიკური იდენტიფიცირებით პროგრამის შესრულების დროს შეიძლება რომელიმე ობიექტით იყოს წარმოდგენილი. ჩვეულებრივ, კლასების დამუშავება ისე ხდება, რომ მათი ობიექტები საგნობრივ სფეროს ობიექტებს შეესაბამებოდეს.

ობიექტი არის გამოთვლითი სისტემის სამისამართო სივრცის არსი, რომელიც კლასის ეგზემპლარის ან პროტოტიპის კოპირების დროს წარმოიშობა (მაგალითად, კომპილაციის შედეგებისა და საწყისი კოდის შესრულებაზე გაშვების შემდეგ). ობიექტი არის არსი, რომლითაც შეტყობინების გაგზავნა შეიძლება და რომელსაც

მათზე თავისი მონაცემების გამოყენებით რეაგირება შეუძლია. ობიექტი ეს კლასის ეგზემპლარია. ობიექტის მონაცემები დანარჩენი პროგრამისაგან დაფარულია. მონაცემთა დაფარვას ინკაპსულაცია ეწოდება.

პროტოტიპი არის ობიექტი-ნიმუში, რომლის მიხედვით და მსგავსებით სხვა ობიექტები იქმნება. ობიექტ-ასლმა ორიგინალ ობიექტებთან კავშირი შეიძლება შეინარჩუნს, პროტოტიპში ცვლილებების მემკვიდრეობით გადაცემა ავტომატურად მოხდეს; ეს განსაკუთრებულობა კონკრეტული ენის ფარგლებში შეიძლება განხორციელდეს.

## განმარტების სირთულე

ობიექტზე ორიენტირებულ დაპროგრამებას უკვე ორმოცწლიანი ისტორია აქვს, მაგრამ მიუხედავად ამისა, მოცემული ტექნოლოგიის მკაფიო, საყოველთაოდ მიღებული განმარტება ჯერ კიდევ არ არსებობს. ძირითადად პრინციპებმა, რომლებიც მის პირველ ობიექტურ ენებსა და სისტემებში იყო ჩადებული, შემდგომი წლების მრავალრიცხოვანი რეალიზაციის დროს მნიშვნელოვანი ცვლილებები (ან დამახინჯებანი) და დამატებები განიცადა. ამის გარდა, დაახლოებით 1980 წლის შუიდან ტერმინი „ობიექტზე ორიენტირებული“ ძალზე მოდური გახდა, რის გამოც მას ნებისმიერ ახალ მოვლენებს „აწეპებდნენ“, რათა მათი მიმზიდველობა უზრუნველყოფო.

Smalltalk ენის შემქმნელის ალან კეის აზრით, რომელსაც ობიექტზე ორიენტირებული დაპროგრამების ერთ-ერთ მამთავრად მიიჩნევენ, ობიექტზე ორიენტირებული მიდგომა შემდეგი ძირითადი პრინციპებით გამოიხატება:

1. ყველაფერი ობიექტია;

2. გამოთვლები ობიექტებს შორის მონაცემთა გაცვლით ხორციელდება, რომლის დროსაც ერთი ობიექტი მოითხოვს, რომ სხვა ობიექტმა რაიმე მოქმედება შეასრულოს. ობიექტები შეტყობინებების გაგზავნითა და მიღებით ურთიერთქმედებენ. შეტყობინება არის მოქმედების შეასრულების მოთხოვნა, შევსებული არგუმენტთა ნაკრებით, რომლებიც მოქმედების შემსრულების დროს შეიძლება გახდეს საჭირო;
3. ყოველ ობიექტს დამოუკიდებელი მეხსიერება აქვს, რომელიც სხვა ობიექტებისაგან შედგება;
4. ყოველი ობიექტი წარმომადგენელია კლასის, რომელიც ობიექტების საერთო თვისებას წარმოადგენს (მაგალითად, როგორიცაა მთელი რიცხვები ან სიები);
5. კლასში მოცემული ობიექტის ქცევა (ფუნქციურობა). შესაბამისად, ყოველ ობიექტს, რომელიც ერთი კლასის ეგზემპლარს წარმოადგენს, იგივე ქმედების შესრულება შეუძლია;
6. კლასები საერთო ფესვის მქონე ერთიან ხისმაგვარ სტრუქტურაშია გაერთიანებული, რომელსაც მემკვიდრეობითობის იერარქია ეწოდება. მეხსიერება და ქცევა, რომელიც გარკვეული კლასის ეგზემპლართანაა დაკავშირებული, იერარქიულ ხეში მის ქვემოთ მდგომი ნებისმიერი კლასისათვის ავტომატურად მისაწვდომია.

ამგვარად, პროგრამა ობიექტების ნაკრებს წარმოადგენს, რომელსაც მდგომარეობა და ქცევა გააჩნია. ობიექტები შეტყობინებების საშუალებით ურთიერთქმედებენ. ბუნებრივად განლაგდება ობიექტთა იერარქია: პროგრამა მთლიანად არის ობიექტი, რომელიც თავისი ფუნქციის შესასრულებლად მასში შემავალ



ობიექტებს მიმართავს, რომლებიც, თავის მხრივ მიმართვის გზით მოთხოვნილს პროგრამის სხვა ობიექტებზე მიმართვის გზით ასრულებენ. ცხადია, რომ მიმართვების უსასრულო რეკურსიის თავიდან აცილების მიზნით რაიმე ეტაპზე ობიექტი მასზე მიმართვის შეტყობინების ტრანსფორმირებას ახდენს და დაპროგრამების ენისა და გარემოს მიერ უზრუნველყოფილ სტანდარტულ სისტემურ ობიექტებს მიმართავს.

სტაბილურობა და მართვის სისტემა ობიექტების პასუხისმგებლობის მკაფიო განაწილებით (ყოველ ქმედებაზე გარკვეული ობიექტი აგებს პასუხს), ობიექტთაშორისი ურთიერთქმედების ინტერფეისის ცალსახა განსაზღვრით და ობიექტის შიგა სტრუქტურის გარემოდან სრული იზოლაციითაა (ინკაპსულაცია) უზრუნველყოფილი.

## კლასები და ობიექტები PHP-ში

როგორც ზემოთ აღვნიშნეთ, კლასი ობიექტზე ორიენტირებული დაპროგრამების საბაზო ცნებას წარმოადგენს. მარტივად რომ ვთქვათ, კლასი ცვლადის ერთ-ერთი თავისებური ტიპია.

კლასის ეგზემპლარი არის ობიექტი. ობიექტი არის დასამუშავებლად განკუთვნილი მონაცემებისა (თვისებების) და ფუნქციების (მეთოდების) ერთობლიობა. მონაცემებსა და მეთოდებს კლასის წევრები ეწოდებათ. საერთოდ, ობიექტს წარმოადგენს ყველაფერი, რაც ინკაპსულაციას უჭერს მხარს.

ობიექტის შიგნით მონაცემები და კოდი (კლასის წევრები) შეიძლება იყოს ღია ან დახურული. ღია მონაცემები და კლასის წევრები პროგრამის სხვა ნაწილებისათვისაც, რომლებიც ობიექტის ნაწილს არ წარმოადგენს, მისაწვდომია. ხოლო დახურული მონაცემები და კლასის წევრები მხოლოდ ამ ობიექტის შიგნითაა მისაწვდომი.

PHP-ში კლასის აღწერა class სამომხმარებლო სიტყვით იწყება:

```
class კლასი_სახელი {  
// მონაცემების - კლასის წევრების აღწერა და მათი დამუშავების  
მეთოდები  
}
```

ობიექტის გამოსაცხადებლად აუცილებლად უნდა გამოვიყენოთ ოპერატორი new:

```
ობიექტი = new კლასი_სახელი;
```

მონაცემების აღწერა var სამომხმარებლო სიტყვის მეშვეობით ხდება. მეთოდის აღწერა ჩვეულებრივი ფუნქციის მსგავსად ხდება.

მეთოდს ასევე შეიძლება გადაეცეს პარამეტრები. ქვემოთ მოყვანილია PHP-ში კლასის მაგალითი:

```
<?php
// Coor ახალი კლასის შექმნა :
class Coor {
// მონაცემები (თვისებები):
var $name;
var $addr;
// მეთოდები:
function Name() {
echo "<h3>მიხეილი</h3>";
}

}
// Coor კლასის ობიექტის შექმნა:
$object = new Coor;
?>
```

## PHP-ში კლასებთან და ობიექტებთან წვდომა

ჩვენ განვიხილეთ კლასების აღწერისა და ობიექტის შექმნის მაგალითი. ახლა აუცილებელია კლასის წევრზე წვდომა მივიღოთ, რისთვისაც PHP-ში ოპერატორი „->“ გამოიყენება. მაგალითად:

```
<?php
// Coor ახალი კლასის შექმნა :
class Coor {
// მონაცემები (თვისებები):
var $name;
```

```

// მეთოდები:
function Getname() {
echo "<h3>მიხეილი</h3>";
}

}
// Coor კლასის ობიექტის შექმნა:
$object = new Coor;
// კლასის წევრებზე წვდომას ვიღებთ:
$object->name = "ნიკოლოზი";
echo $object->name;
// გამოიტანს სახელს 'ნიკოლოზი'
// ახლა კი კლასის მეთოდზე მივიღოთ წვდომა (ფაქტობრივად
კლასის შიგნით ფუნქციაზე):
$object->Getname();
// გამოიტანს სახელს 'მიხეილი'
?>

```

იმისათვის, რომ კლასის შიგნით კლასის წევრებზე მივიღოთ წვდომა, აუცილებელია გამოვიყენოთ \$this მაჩვენებელი, რომელიც ყოველთვის მიმდინარე ობიექტს მიეკუთვნება. მოდიფიცირებული Getname() მეთოდია:

```

function Getname() {
echo $this->name;
}

```

ასეთივე წესით შეიძლება ჩაიწეროს Setname() მეთოდიც:

```
function Setname($name) {  
    $this->name = $name;  
}
```

ახლა სახელის შესაცვლელად Setname() მეთოდის გამოყენება შეიძლება:

```
$object->Setname("ლუკა");  
$object->Getname();
```

ქვემოთ კოდი სრულადაა მოცემული:

```
<?php  
// Coor ახალი კლასის შექმნა :  
class Coor {  
    // მონაცემები (თვისებები):  
    var $name;  
    // მეთოდები:  
    function Getname() {  
        echo $this->name;  
    }  
    function Setname($name) {  
        $this->name = $name;  
    }  
}  
  
// Coor კლასის ობიექტის შექმნა:  
$object = new Coor;  
// სახელის შესაცვლელად Setname() მეთოდს ვიყენებთ:  
$object->Setname("ნიკოლოზი");
```

```
// წვდომისათვის კი Getname() მეთოდს:  
$object->Getname();  
// ეს სცენარი გამოიტანს სახელს 'ნიკოლოზი'  
?>
```

\$this მაჩვენებლის გამოყენება შეიძლება არა მარტო მონაცემებზე, არამედ მეთოდებზე წვდომისათვის.

```
function Setname($name) {  
    $this->name = $name;  
    $this->Getname();  
}
```

## ობიექტების ინიციალიზაცია

ზოგჯერ ისეთი მდგომარეობა წარმოიშობა, რომ ობიექტის ინიციალიზაცია აუცილებელია ობიექტის თვისებებს საწყისი მნიშვნელობები მიენიჭოს. დავუშვათ, რომ კლასის სახელია Coor და იგი ადამიანის ორ მახასიათებელს, სახელსა და იმ ქალაქის დასახელებას შეიცავს, სადაც იგი ცხოვრობს. შეიძლება ჩაიწეროს მეთოდი (ფუნქცია), რომელიც ობიექტის ინიციალიზაციას მოახდენს, მაგალითად Init():

```
<?php  
// Coor ახალი კლასის შექმნა :  
class Coor {  
    // მონაცემები (თვისებები):  
    var $name;  
    var $city;
```

```

// ინიციალიზაციის მეთოდი:
function Init($name) {
    $this->name = $name;
    $this->city = "თბილისი";
}

}

// Coor კლასის ობიექტის შექმნა:
$object = new Coor;
// ობიექტის ინიციალიზაციისათვის მაშინვე ვიძახებთ მეთოდს:
$object->Init();
?>

```

მთავარია, არ დაგვავიწყდეს ობიექტის შექმნისთანავე ფუნქციის ან რაიმე მეთოდის გამოძახება ახალი ობიექტის შექმნასა (ოპერატორი new) და მის ინიციალიზაციას (Init გამოძახებას) შორის.

იმისათვის, რომ გარკვეული მეთოდი ობიექტის შექმნის დროს ავტომატურად იქნეს გამოძახებული, მას უნდა მივცეთ იგივე სახელი, რაც კლასს აქვს (Coor):

```

function Coor ($name)
    $this->name = $name;
    $this->city = "თბილისი";
}

```

ობიექტის ინიციალიზაციის მეთოდს კონსტრუქტორი ეწოდება. მაგრამ, PHP-ს არ გააჩნია დესტრუქტორები, ვინაიდან სკრიპტის მუშაობის დამთავრებისთანავე რესურსები ავტომატურად თავისუფლდებიან.

## კლასების მემკვიდრეობითობა PHP-ში

მემკვიდრეობითობა უბრალოდ კლასის ზუსტი ასლის შექმნას კი არ ნიშნავს, არამედ უკვე არსებული კლასის გაფართოებას, რათა შთამომავალმა რაიმე ახალი, მხოლოდ ამ ფუნქციისათვის დამახასიათებელი შეასრულოს. განვიხილოთ კონკრეტული მაგალითი:

```
<?php
class Paren {
function parent_func() { echo "<h1>ეს საბაზო ფუნქცია</h1>"; }
function test () { echo "<h1>ეს საბაზო კლასია</h1>"; }
}

class Child extends Paren {
function child_func() { echo "<h2>ეს მემკვიდრე ფუნქცია</h2>"; }
function test () { echo "<h2>ეს მემკვიდრე (ქვეკლასი)
კლასია</h2>"; }
}

$object = new Paren;
$object = new Child;

$object->parent_func(); // გამოიტანს "ეს საბაზო ფუნქცია"
$object->child_func(); // გამოიტანს "ეს მემკვიდრე ფუნქცია"
$object->test(); // გამოიტანს "ეს მემკვიდრე (ქვეკლასი) კლასია"
?>
```

ამ სკრიპტის მუშაობის შედეგი იქნება:



# ეს საბაზო ფუნქციაა

## ეს მემკვიდრე ფუნქციაა

### ეს მემკვიდრე (ქვეკლასი) კლასია

extends საგასაღებო სიტყვა გვეუბნება, რომ child შვილობილ კლასს parent კლასის ყველა მეთოდი და თვისება მემკვიდრეობით ერგო. Parent კლასს, ჩვეულებრივ, საბაზო კლასს ან სუპერკლასს, ხოლო child მემკვიდრე კლასს ან ქვეკლასს უწოდებენ.

## კლასების პოლიმორფიზმი PHP-ში

პოლიმორფიზმი საბაზო კლასის თვისებაა, ქვეკლასის ფუნქციები გამოიყენოს. ქვემოთაა მოყვანილი კლასის თვისების - პოლიმორფიზმის მაჩვენებელი პრაქტიკული მაგალითი.

```
<?php
class Base {
    function funct() {
        echo "<h2>საბაზო კლასის ფუნქცია</h2>";
    }
    function base_func() {
        $this->funct();
    }
}

class Derivative extends Base {
```

```
function funct() {  
  echo "<h3>ქვეკლასის ფუნქცია</h3>";  
}  
}  
  
$b = new Base();  
$d = new Derivative();  
  
$b->base_func();  
$d->funct();  
$d->base_func();  
?>
```

სკრიპტი მუშაქობის შედეგად გამოიტანს:

```
საბაზო კლასის ფუნქცია  
ქვეკლასის ფუნქცია  
ქვეკლასის ფუნქცია
```

განხილულ მაგალითში Base კლასის base\_func() ფუნქცია იმავე სახელწოდების ფუნქციად Derivative კლასში იყო გადაწერილი. ამ სახით განსაზღვრულ ფუნქციას ვირტუალური ფუნქცია ეწოდება.

**ლამაზი სკრიპტი ანუ დაპროგრამების  
სტილი**

სკრიპტი ლამაზი უნდა იყოს, ეს უბრალოდ სილამაზისათვის არ არის საჭირო.

საქმე ისაა, რომ ძალზე მცირე რაოდენობის პროგრამა იწერება სრულყოფილად და მუდმივად, ისე, რომ მას კორექტირება არ დასჭირდეს და მაშინვე, როგორც საჭიროა ისე მუშაობდეს. ლამაზად დაწერილი სკრიპტის გამართვა და მოდიფიცირება გაცილებით მარტივია. ამის გარდა, სხვადასხვა მიზეზთა გამო ერთი პიროვნების მიერ დაწერილ პროგრამასთან მუშაობა ხშირად სხვა პიროვნებას უწევს. ლამაზად და სწორად დაწერილ კოდთან მუშაობა კი გაცილებით მოსახერხებელი და მარტივია.

განვიხილოთ, როგორ უნდა გამოიყურებოდეს ლამაზად დაწერილი კოდი. ქვემოთ მოყვანილია დეფექტებით დაწერილი კოდების მაგალითები.

```
№1
<?PHP for($i=0;$i<5;$i++){if($i%2)echo $i;echo 'O<hr>';} ?>

№2
<?PHP
                                for(
                                $i
                                =
                                0
                                ;
                                $i
<5
                                ;
                                $i++)
                                {
                                if
                                ($i
                                %
                                2)echo
                                $i;
```

```

                                echo    'O<hr>'
; }                               ?>
№3
<?PHP $a = $b+$c * $d ?>
№4
<?PHP
for ($i = 0; $i < 5; $i++) {
    if ($i % 2)
        echo $i;
    echo '<hr>';
}
?>

```

ყველა ზემოთ მოყვანილი სკრიპტი სინტაქსურად სწორადაა დაწერილი და მუშაობს.

განვიხილოთ ისინი მიმდევრობით:

კოდი №1 - აქ ყველაფერი ერთ სტრიქონადაა დაწერილი და ძალზე რთული გასარჩევია თუ როდის გამოიტანს ეს კოდი ასომთავრულით O სიმბოლოს;

კოდი №2 - აქ გარჩევაც არ უნდა, ყველაფერი ისედაც ნათელია. იმავე კითხვაზე პასუხის გაცემა, ალბათ, აქაც რთული იქნება;

კოდი №3 - ამ კოდის ნაკლი გახლავთ ის, რომ ერთი თვალის გადავლებით ისეთი შთაბეჭდილება იქმნება, თითქოს პირველად შეკრების ოპერაცია უნდა შესრულდეს. ასეთ აღქმას ჰარების მოცემული განლაგება იწვევს;

კოდი №4 - ტაბულაციის დილაკის გამოყენების გამო შეიძლება მოგვეჩვენოს, რომ ტეგი <hr> მხოლოდ პირობის შესრულების დროს გამოიტანს ხაზს, მაგრამ თუ ფიგურული ფრჩხილების განლაგებას კარგად დავაკვირდებით, მაშინ დავინახავთ, რომ ეს ასე არ არის.

განსაკუთრებით, დიდ კოდებში მსგავსმა სტრუქტურირების ელემენტებმა შეიძლება პრობლემები შეგვიქმნას.

## ძირითადი წესები

ზემოთ განხილული პირველი და მეორე მაგალითის შესახებ სალაპარაკო არაფერია.

რაც შეეხება მესამე მაგალითს, აქედან შეიძლება ასეთი დასკვნა გავაკეთოთ, რომ კოდის ჩაწერის დროს უნდა ვეცადოთ, რომ ჰარები განვათავსოთ ერთნაირად. არ უნდა იყო ზოგან ორი, ზოგან ოთხი და ზოგან არცერთი. აქვე უნდა შევნიშნოთ, რომ დიდი გამოსახულების დაწერის დროს გამრავლების ოპერაცია, მართალია მაინც პირველი შესრულდება, მაგრამ სასურველია იგი ფრჩხილებში მოვათავსოთ. რაც შეეხება ჰარებს: მძიმეები და სხვა ოპერატორები ორივე მხრიდან ყოველთვის ჰარებით გამოჰყავით.

მეოთხე მაგალითში გამოყენებული დაცილებებიდან ჩანს მათი როლი კოდის დაწერაში. წანაცვლებებმა უნდა დაგვანახოს პროგრამის შიგნით ურთიერთგანლაგების სტრუქტურა. ეს ნიშნავს, რომ ერთმანეთში ჩალაგებული ყველა ოპერატორი, მაგალითად, ციკლში შემავალი ყველა ოპერატორი, მარცხენა კიდიდან უნდა იყოს უფრო მეტად დაცილებული, ვიდრე თვით ციკლის ოპერატორი. ერთი დონის ოპერატორები მარცხენა კიდიდან მიმართ ერთი და იმავე მანძილით უნდა იყოს დაცილებული.

## ფიგურული ფრჩხილების განლაგება

ფიგურული ფრჩხილების განთავსების მრავალნაირი სტილი არსებობს:

<?PHP				
if	(\$a	>	\$b)	{

```

}
?>
<?PHP
if ($a > $b)
{
    // ოპერატორები
}
?>
<?PHP
if ($a > $b)
{
    // ოპერატორები
}
?>
<?PHP
if ($a > $b)
{
    // ოპერატორები
}
?>

```

პროგრამის დაწერის პირველი მეთოდი უფრო მოსახერხებელია if, for და სხვა მსგავსი ოპერატორების გამოყენების შემთხვევაში. ფუნქციისათვის უფრო მოსახერხებელი იქნება მეორე შემთხვევა.

პროგრამის დაწერის მესამე და მეოთხე შემთხვევებზე მართალია არ გამოიწვევს შეცდომას, მაგრამ ტაბულაციისა და

ჰარების დიდი რაოდენობით გამოყენების გამო პროგრამის დაწერა მოუხერხებელი იქნება.

## ცვლადების, ფუნქციებისა და კლასების სახელები

სახელები პროგრამისტს ცვლადის, ფუნქციის ან სხვათა დანიშნულებაზე უნდა მიუთითებდეს. მაგალითად, თუ პროგრამაში ჩაწერილია კოდი \$a = 'Table', ეს მომხმარებელს არაფერს არ ეუბნება, მაგრამ ჩანაწერი \$product = 'Table' უკვე იძლევა რაღაც შეტყობინებას ცვლადის შესახებ. ასევე ხდება, მაგალითად, \$r = g(\$a, \$b) და \$sum = add(\$serv\_1, \$serv\_2) ფუნქციების შემთხვევაშიც მათ შორის სხვაობა აშკარაა. ყოველთვის უნდა ვეცადოთ ცვლადებს, ფუნქციას და კლასებს ისეთი სახელები შევურჩიოთ, რომლებიც მათ ტიპსა და მნიშვნელობაზე მიგვითითებს, მაგრამ არავითარ შემთხვევაში სახელი:

```
add_first_service_and_one_more_service_returning_their_summ_
as_i nteger_value.
```

თუ რატომ, ამას ალბათ ახსნის გარეშეც მიხვდება მომხმარებელი.

## კომენტარები

კომენტარების შესახებ PHP-ის შესწავლის დასაწყისში გვექონდა საუბარი. ახლა განვსაზღვროთ, რა შემთხვევაში დაგვჭირდება კოდის კომენტირება:

- ამოცანის არაჩვეულებრივი გადაწყვეტა. იმიტომ, რომ შეიძლება შემდეგ ვერ გაიხსენოთ რა იგულისხმეთ მოცემულ შემთხვევაში და მითუმეტეს, სხვა პროგრამისტმა შეიძლება ვერ გაარჩიოს ამოცანის მოცემული გადაწყვეტა;
- ამოცანის ამოხსნა, რაზედაც დიდ ხანს ფიქრი დაგვჭირდა;
- დიდი მოცულობის ბლოკების დანიშნულება;

- დიდი მოცულობის ლოგიკური პირობები;
- პროგრამის რთული ნაწილები.

საჭირო არ არის, მაგალითად, ასეთი კოდის კომენტირება:

```
ibase_connect($host, $user, $passwd); // ინტერფეისზე მონაცემთა
ბაზასთან დაკავშირება
```

როგორი უნდა იყოს კომენტარი. მაგალითად, შეადარეთ:

```
<?PHP
$a = 1; // A ცვლადს მივანიჭოთ ერთიანი
$b = 6; // b ცვლადს მივანიჭოთ 6
// ციკლი $i-ს მიხედვით 1-დან 6-მდე
for ($i = 1; $i <= $b; $i++ )
    $a = $a * $i; // A გავამრავლოთ $i-ზე
?>
<?PHP
// $b რიცხვის ფაქტორიალის პოვნა
$a = 1;
$b = 6;
for ($i = 1; $i <= $b; $i++ )
    $a = $a * $i;
?>
```

რა თქმა უნდა, ამ ორ კოდს შორის არის სხვაობა. პირველ შემთხვევაში თითქმის არც ერთი კომენტარი დამატებით არავითარ ინფორმაციას არ გვაწვდის, აქ ისედაც ყველაფერი ნათელი იყო. მეორე შემთხვევაში კი კომენტარში ცალსახადაა განმარტებული მოცემული სკრიპტის დანიშნულება.

პირველ რიგში, კომენტარი უნდა გავუკეთოთ იმას რატომ ვაკეთებთ ამას და არა იმას, რას ვაკეთებთ. ქვემოთ მოცემულია



პროგრამა, რომელშიც საერთოდ არ არის გამოყენებული კომენტარი, მაგრამ ფუნქციის სახელიდან გამომდინარე ცალსახადაა განსაზღვრული მისი დანიშნულება.

```

<?PHP
function factorial($b)
$res = 1;
for ($i = 1; $i <= $b; $i++)
    $res = $res * $i;
?>

```

ჩვეულებრივ, ერთი ბრძანების კომენტარი წერტილ-მძიმის შემდეგ უნდა განვათავსოთ, ხოლო ბლოკის კომენტარი - მისი პირველი ბრძანების წინ, სტრიქონში.

მართალია, ბევრ პროგრამისტს კომენტარების წერა არ უყვარს, მაგრამ, როდესაც დიდი მოცულობის სკრიპტებს ვწერთ სასურველია მასში კომენტარები განვათავსოთ.

## ოპერატორი GOTO

ოპერატორი GOTO დაპროგრამების მრავალ ენაში გამოიყენება და PHP 5.3.0-შიც გამოჩნდა. იგი პროგრამის სხვა ნაწილში გადასასვლელად გამოიყენება.

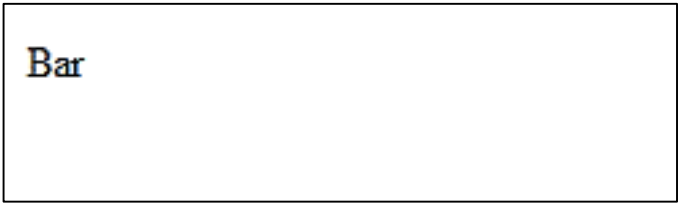
ადგილს, სადაც უნდა გადავიდეთ, ვუთითებთ ჭდის საშუალებით, რომელსაც ორი წერტილი მოსდევს. GOTO ოპერატორს გადასასვლელად სასურველი ჭდე მოსდევს. ოპერატორი GOTO არ არის განუსაზღვრელი უფლებების მქონე, რაც იმას ნიშნავს, რომ გადასასვლელი ჭდე იმავე ფაილში უნდა მდებარეობდეს, რომელშიც ეს ოპერატორია. იგულისხმება, რომ არ შეიძლება ფუნქციის ან მეთოდის საზღვრებს გარეთ ან გარედან ერთ-ერთი მათგანის შიგნით გადასვლა. ასევე, შეუძლებელია,

გარედან ნებისმიერი ციკლური სტრუქტურის ან switch ოპერატორის შიგნით გადასვლა. მაგრამ იქედან გამოსვლა შეუძლებელია. ჩვეულებრივ, GOTO ოპერატორი შეიძლება მრავალდონიანი break ოპერატორის ნაცვლად გამოვიყენოთ. მაგალითად:

```
<?php
goto a;
echo 'Foo';

a:
echo 'Bar';
?>
```

მოცემული მაგალითის შესრულების შედეგი იქნება:



ქვემოთ მოყვანილია GOTO ოპერატორის ციკლში გამოყენების მაგალითი:

```
<?php
for($i=0,$j=50; $i<100; $i++) {
    while($j--) {
        if($j==17) goto end;
    }
}
echo "i = $i";
```

```
end:
echo          'j'                hit                17';
?>
```

შედეგი ასეთი იქნება:

```
j = 17
```

შემდეგი მაგალითი კი არ იმუშავებს, ვინაიდან ჭდე ციკლის შიგნითაა განთავსებული:

```
<?php
goto          loop;
for($i=0,$j=50;          $i<100;          $i++)          {
                    while($j--)          {
loop:          }
}
echo          "$i          =          $j";
?>
```

ამ მაგალითის შესრულების შედეგად მივიღებთ:

```
Fatal error: 'goto' into loop or switch statement is disallowed in C:\xampp\htdocs\mag22.php on line 2
```

## ჩართვის კონსტრუქცია PHP-ში

ჩართვის კონსტრუქცია საშუალებას გვაძლევს რამდენიმე ცალკეული პროგრამისაგან PHP პროგრამა (სკრიპტი) ავაწყოთ.

PHP-ში ჩართვის ორი ძირითადი კონსტრუქცია არსებობს: require და include.

### ჩართვის კონსტრუქცია require

ჩართვის კონსტრუქცია require საშუალებას გვაძლევს კოდი სცენარის შესრულებამდე ჩავრთოთ. მისი ზოგადი სინტაქსი ასეთია:

```
require ფაილის_სახელი;
```

გაშვების (და არა შესრულების) დროს პროგრამა ინტერპრეტატორი უბრალოდ ინსტრუქციას მითითებულ ფაილში ჩაწერილი ინფორმაციით (ეს ფაილი შეიძლება PHP-ზე ჩაწერილ სცენარსაც შეიცავდეს, რომელიც ჩვეულებრივ <? და ?> ტეგებით იქნება შემოსაზღვრული) შეცვლის. ამას, (include კონსტრუქციისაგან განსხვავებით) პროგრამის უშუალოდ გაშვების წინ გააკეთებს. ეს ძალზე მოსახერხებელია სცენარში HTML-კოდებში ჩაწერილი სხვადასხვა შაბლონის გამოყენების დროს. მოვიყვანოთ მაგალითი:

```
ფაილი header.html:
```

```
<html>  
<head><title>It is a title</title></head>  
<body bgcolor=green>
```

```
ფაილი footer.html:  
&copy; Home Company, 2005.  
</body></html>
```

```
ფაილი script.php  
<?php  
require "header.htm";  
// სცენარი თვით დოკუმენტის ტანს გამოიტანს  
require "footer.htm";  
?>
```

ამგვარად, კონსტრუქცია `require` საშუალებას იძლევა PHP სცენარი რამდენიმე ცალკეული ფაილისაგან ავაწყოთ, რომლებიც შეიძლება როგორც HTML-გვერდები, ასევე PHP-სკრიპტები იყოს.

## ჩართვის კონსტრუქცია `include`

კონსტრუქცია `include` ასევე განკუთვნილია PHP სცენარის კოდში ფაილების ჩასართავად.

`require` კონსტრუქციისაგან განსხვავებით კონსტრუქცია `include` საშუალებას იძლევა PHP სკრიპტის კოდში ფაილები სცენარის შესრულების დროს ჩართოს. `include` კონსტრუქციის სინტაქსი შემდეგნაირად გამოიყურება:

```
include ფაილის_სახელი;
```

`require` და `include` კონსტრუქციებს შორის არსებული პრინციპული განსხვავება განვიხილოთ კონკრეტულ პრაქტიკულ მაგალითზე. შევქმნათ ათი სხვადასხვა ფაილი სახელებით `1.txt`, `2.txt` და ა. შ. `10.txt`, რომლებშიც ჩაწერილი იქნება ჩვეულებრივი

ათობითი რიცხვები 1, 2 ..... 10 (თითო რიცხვი თითოეულ ფაილში). ახლა კი შევქმნათ შემდეგი PHP სცენარი:

```
<?php
// იქმნება ციკლი, რომლის ტანში include კონსტრუქციაა
გამოყენებული
for($i=1; $i<=10; $i++) {
include "$i.txt";
echo "<br>";
}
// ჩავრთეთ ათი ფაილი: 1.txt, 2.txt, 3.txt ... 10.txt
?>
```

შედეგად მივიღებთ ათი რიცხვისაგან შემდგარ ჩანაწერს.

```
1
2
3
4
5
6
7
8
9
10
```

აქედან, შეიძლება გამოვიტანოთ დასკვნა, რომ თითოეული ფაილი ციკლის შესრულების მომენტში მხოლოდ ერთხელ იყო

ჩართული. ახლა, თუ include კონსტრუქციას require კონსტრუქციით შევცვლით, მაშინ სცენარი კრიტიკული შეცდომის (fatal error) გენერირებას მოახდენს.

PHP სცენარს კომპიუტერისათვის გასაგებ კოდებში გარდაქმნის, რისთვისაც სცენარის სტრიქონებს რიგ-რიგობით გაანალიზებს, ვიდრე include კონსტრუქციამდე არ მივა. include-მდე მისვლის შემდეგ PHP სცენარის ტრანსლირებას წყვეტს და include კონსტრუქციით მითითებულ ფაილზე გადაერთვება. ამგვარად, ტრანსლატორის ასეთი ქცევის გამო სცენარის სწრაფქმედება მცირდება, განსაკუთრებით კი მაშინ, როდესაც include კონსტრუქციით საკმაოდ დიდი რაოდენობის ფაილის ჩართვა ხდება. require კონსტრუქციასთან ასეთი პრობლემები არ გვაქვს, რადგანაც require-ის მეშვეობით ფაილების ჩართვა სცენარის შესრულებამდე ხორციელდება ანუ ტრანსლაციის მომენტში ფაილი სცენარში უკვე ჩართულია.

ამგვარად, require კონსტრუქციის გამოყენება მიზანშეწონილია იქ, სადაც სცენარში ფაილების დინამიკური ჩართვა არ არის საჭირო, ხოლო include კონსტრუქცია PHP-ის სკრიპტის კოდში მხოლოდ დინამიკური ცვლილებების განხორციელების მიზნით უნდა გამოვიყენოთ.

include კონსტრუქცია შორეული ფაილების ჩართვის საშუალებას იძლევა. მაგალითად:

```
<?php
// შემდეგი მაგალითი არ იმუშავებს, რადგან ლოკალური ფაილის
ჩართვას ცდილობს
include 'file.php?foo=1&bar=2';
// შემდეგი მაგალითი იმუშავებს
include 'http://www.example.com/file.php?foo=1&bar=2';
?>
```

იმისათვის, რომ მისაწვდომი იყოს შორეული ფაილების ჩართვა, აუცილებელია კონფიგურაციულ ფაილში (php.ini) დავაყენოთ allow\_url\_fopen=1.

## ერთჯერადი ჩართვის კონსტრუქციები **require\_once** და **include\_once**

დიდ PHP სცენარებში კონსტრუქციები include და require ძალზე ხშირად გამოიყენება. ამიტომ, საკმაოდ ძნელია ერთი და იმავე ფაილის რამდენიმეჯერ ჩართვის კონტროლი, რასაც შეცდომამდე მივყავართ და მისი აღმოჩენა კი რთულია.

PHP-ში ამ პრობლემის გადაწყვეტა გათვალისწინებულია. ერთჯერადი ჩართვის კონსტრუქციის require\_once და include\_once გამოყენებით დარწმუნებული ვიქნებით რომ ერთი და იგივე ფაილი ორჯერ არ ჩაირთვება. ერთჯერადი ჩართვის კონსტრუქცია require\_once და include\_once ისევე მუშაობს, როგორც კონსტრუქცია require და include შესაბამისად. მათ შორის სხვაობა მხოლოდ ის არის, რომ ფაილის ჩართვის წინ ინტერპრეტატორი ამოწმებს, მოცემული ფაილი მანამდე იყო ჩართული თუ არა. თუ ჩართული იყო, მაშინ ამ ფაილის ხელახალი ჩართვა აღარ მოხდება.



# PHP – MySQL

PHP-ის ერთ-ერთი მთავარი დანიშნულებაა მონაცემთა ბაზებთან მუშაობა - მონაცემთა ბაზის შექმნა, მონაცემთა ბაზაში მონაცემების ჩაწერა, რედაქტირება, წაშლა, მონაცემთა ბაზებიდან ინფორმაციის ამოღება (წაკითხვა). PHP-ს მონაცემთა ბაზებთან სამუშაოდ მრავალფეროვანი და საინტერესო ფუნქციები აქვს.

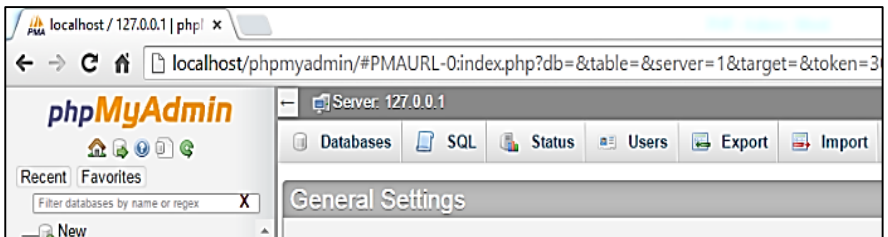
განვიხილოთ PHP-ის მხოლოდ MySQL მონაცემთა ბაზასთან სამუშაო ძირითადი ფუნქციები.

## მონაცემთა ბაზის აგება

მართვის სისტემის საშუალებით MySQL მონაცემთა ბაზის ასაგებად მონაცემთა ბაზაში ერთი ცხრილი დაემატება, რომელშიც მონაცემებს ჩავწერთ.

ბრაუზერის სამისამართო ველში შემდეგი მისამართი ავკრიბოთ: localhost/phpmyadmin და <Enter> კლავიშს დავაჭიროთ თითი (შეამოწმეთ ლოკალურ სერვერზე გააქტიურებულია თუ არა MySQL-ის შესაბამისი ველი).

მონიტორის ეკრანზე შემდეგი ფანჯარა გაიხსნება:



მონაცემთა ბაზის ასაგებად ზედა ჰორიზონტალური მენიუს Databases ჩანართზე დავაჭიროთ მაუსი, შედეგად მივიღებთ:

### Databases

Create database ⓘ

Database name  Collation  Create

Create database ველში ჩავწერთ მონაცემთა ბაზის სახელი მაგ: phpbase, Collation ველში ავირჩიოთ utf8-general-ci კოლექცია (აღნიშნული კოლექციის არჩევა იმ შემთხვევაში ხდება, როდესაც მონაცემთა ბაზაში ტექსტი ქართული უნიკოდით გვინდა ჩავწერთ):

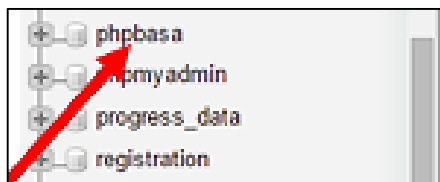
### Databases

Create database ⓘ

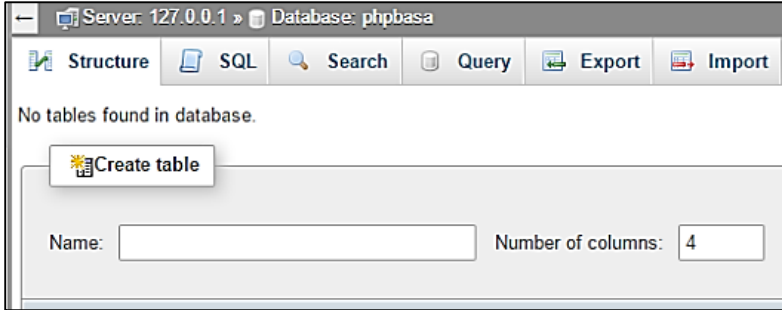
phpbase utf8\_general\_ci Create

მაუსი Create ღილაკზე დავაწკაპუნოთ.

ბრაუზერის ფანჯრის მარცხენა მხარეს მონაცემთა ბაზების ჩამონათვალში მოვძებნოთ ჩვენ მიერ შექმნილი მონაცემთა ბაზა.



მაუსი ჩვენ მიერ შექმნილ ბაზაზე დავაწკაპუნოთ, შემდეგად მივიღებთ:



აქ არ განვიხილავთ, რას წარმოადგენს მონაცემთა ბაზა, როგორ ხდება მისი აგება, როგორი სტრუქტურა აქვს მას. ჩვენ MySQL მონაცემთა ბაზის მართვის სისტემაში ბაზის აგებას განვიხილავთ და მარტივ მონაცემთა ბაზას ავაგებთ.

Name ველში ცხრილის სახელი, მაგალითად, menu, ხოლო Number of columns ველში ცხრილში სვეტების ანუ ველების რაოდენობა ჩავწეროთ. ჩვენ შემთხვევაში 3.



მაუსი Go ღილაკზე დავაწკაპუნოთ, რის შემდეგაც გაიხსნება ფანჯარა:

Name	Type	Length/Values
<input type="text"/>	INT	<input type="text"/>
<input type="text"/>	INT	<input type="text"/>
<input type="text"/>	INT	<input type="text"/>

Name ველში ველების სახელები იწერება, Type ველში ხდება მონაცემთა ტიპის არჩევა, იმის მიხედვით, თუ რა ტიპის მონაცემების ჩაწერა ხდება ველში. Length/Values ველში შესაბამისი მონაცემის მაქსიმალური სიგრძის ან შესაბამისი მნიშვნელობის მითითება ხდება.

Name ველში ჩაწეროთ ველების სახელები. id - იდენტიფიკატორი, name - სახელი, date - დაბადების თარიღი.

Type ველში მოვნიშნოთ შესაბამისი ველის ტიპები: int - მთელი ტიპი id ველისათვის, varchar - სიმბოლური ტიპი name ველისათვის, date - დროითი ტიპი date ველისათვის.

Length/Values ველში ჩაწეროთ შესაბამისად 8 - id ველისათვის, 50 - name ველისათვის, რაც ნიშნავს, რომ სახელის ჩასაწერად მაქსიმუმ 50 სიმბოლოს გამოყენება შეგვიძლია, დროითი ტიპის მონაცემის სიგრძე არ ეთითება.

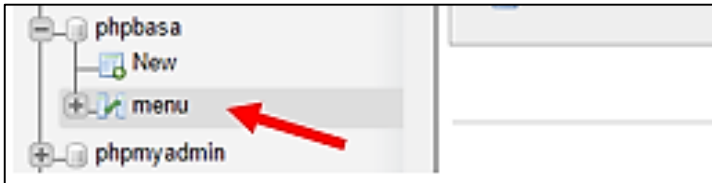
id ველს index ველში მივუთითოთ Primary, რაც ცხრილის შიგა გასაღებს აღნიშნავს. A\_I (Auto Increment) ველში გავაქტივოთ მოსანიშნი ალამი, რაც ველში მნიშვნელობების დამატების შემთხვევაში მის ავტომატურ გაზრდას გამოიწვევს. დანარჩენი ველები უცვლელად დავტოვოთ.

Structure				
Attributes	Null	Index	A_I	Comments
<input type="text"/>	<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>	<input type="text"/>
<input type="text"/>	<input type="checkbox"/>	---	<input type="checkbox"/>	<input type="text"/>
<input type="text"/>	<input type="checkbox"/>	---	<input type="checkbox"/>	<input type="text"/>

ამის შემდეგ მაუსი save (აღნიშნული ღილაკი მარჯვენა ქვედა კუთხეში ჩანს) ღილაკზე დავაწკაპუნოთ.

Structure					
Collation	Attributes	Null	Index	A_I	Com
<input type="text"/>	<input type="text"/>	<input type="checkbox"/>	---	<input type="checkbox"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="checkbox"/>	---	<input type="checkbox"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="checkbox"/>	---	<input type="checkbox"/>	<input type="text"/>

რის შედეგადაც phpbase ბაზაში ცხრილი menu ჩამატება. იმისათვის, რომ menu ცხრილში ჩავწეროთ მონაცემები საჭიროა მასზე მალსი დავაწკაპუნოთ.



ხოლო შემდეგ გახსნილი ფანჯრის ჰორიზონტალურ მენიუში Insert ღილაკს დავაჭიროთ თითი.

Column	Type	Function	Null	Value
id	int(2)			
name	varchar(50)			
date	date			

Go

Ignore

Column	Type	Function	Null	Value
id	int(2)			
name	varchar(50)			
date	date			

Go

Insert as new row and then Go back to previous page

Go Reset

Continue insertion with 2 rows

ჩუმათობის პრინციპით ბაზაში ორი ჩანაწერის დამატება შეგვიძლია, რომელიც ფანჯრის ქვედა ზოლში Continue insertion with ველშია მითითებული. შეგვიძლია აგრეთვე ჩანაწერების

რაოდენობის შეცვლაც. ჩვენს შემთხვევაში დავამატოთ ორი ჩანაწერი, id ველს არ ვავსებთ, რადგანაც Auto Increment რეჟიმი გვაქვს არჩეული, დანარჩენ ველებში შესაბამისი მნიშვნელობები შევიტანოთ:

The screenshot shows the phpMyAdmin interface for a table named 'menu' in the 'phpbasa' database. The table structure is as follows:

Column	Type	Function	Null	Value
id	int(2)			
name	varchar(50)			ქობა
date	date			2013-04-05

Below the structure, there are two rows of data being inserted:

Column	Type	Function	Null	Value
id	int(2)			
name	varchar(50)			გიორგი
date	date			2012-12-28

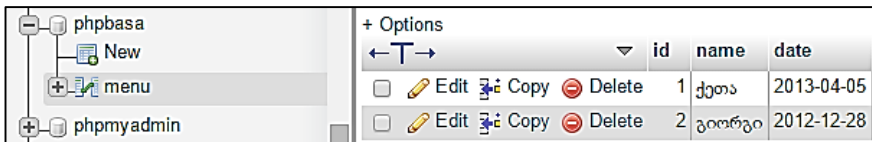
At the bottom, there are controls for inserting the data: "Insert as new row" (selected), "and then", "Go back to previous page", and "Go" and "Reset" buttons.

და Go ღილაკზე დავაწკაპუნოთ.

The screenshot shows the phpMyAdmin interface after the insertion. A green message box indicates: "2 rows inserted. Inserted row id: 2". Below the message, the SQL query used for the insertion is displayed:

```
INSERT INTO `phpbasa`.`menu` (`id`, `name`, `date`) VALUES (NULL, 'ქობა', '2013-04-05'), (NULL, 'გიორგი', '2012-12-28');
```

ჩაწერილი მონაცემების სანახავად საჭიროა მაუსი menu ცხრილზე დავაწკაპუნოთ..



როგორც ხედავთ, menu ცხრილში ორი მონაცემი ჩაიწერა, (თითოეულ სტრიქონს ჩანაწერი ეწოდება). შემდეგში შეგიძლიათ კიდევ ჩაწეროთ სხვა მონაცემები, განახორციელოთ არსებული მონაცემების რედაქტირება ან წაშლა.

menu ცხრილში დავამატოთ კიდევ 4 ჩანაწერი.



PHP-ის საშუალებით წავიკითხოთ MySQL მონაცემთა ბაზაში ჩაწერილი ინფორმაცია, ჩაწეროთ ახალი ინფორმაცია, წავშალოთ არსებული ინფორმაცია ან რედაქტირება გავუკეთოთ.

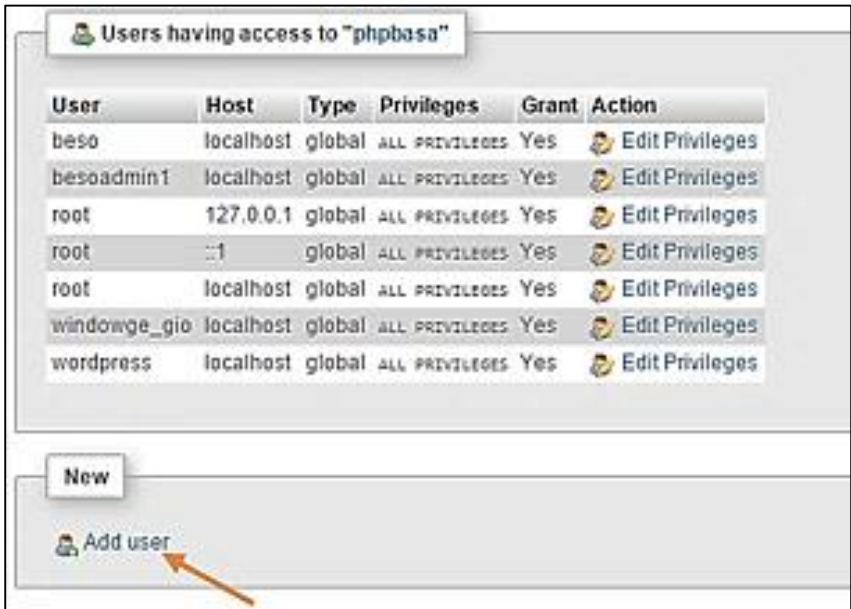
სერვერთან და შესაბამისად სერვერზე ბაზასთან კავშირის დასამყარებლად საჭიროა phpMyAdmin-ში განისაზღვროს მომხმარებლის უფლებები, თუმცა შესაძლებელია მომხმარებლის უფლებების განსაზღვრის გარეშეც დამყარდეს ლოკალურ სერვერზე ბაზასთან კავშირი.



განვსაზღვროთ მომხმარებლის უფლებები.  
მოვნიშნოთ ჩვენი ბაზა, ამ შემთხვევაში phpbase.



მაუსი დავაჭიროთ ზედა ჰორიზონტალურ მენიუში მდებარე Privileges ჩანართზე:



და შემდეგ Add user დილაკზე.

ეკრაზე გახსნილი ფანჯრის User name ველში ჩავწერთ მომხმარებლის სახელი, Host ველში ჩავწერთ ჰოსტის სახელი (ლოკალურ სერვერზე localhost), Password ველში ჩავწერთ პაროლი, Re-type ველში დავადასტუროთ პაროლი. ჩვენს

შემთხვევაში User name: phpuser, Password: phppassword, აღმით მოვნიშნოთ ველი Check All (რითაც ჩვენს მომხმარებელს ვაძლევთ ყველა უფლებას).

**Add user**

**Login Information**

User name: Use text field: ▼ phpuser

Host: Use text field: ▼ localhost

Password: Use text field: ▼ .....

Re-type: .....

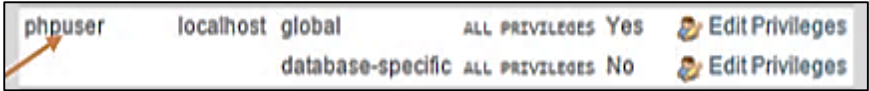
Generate password: Generate

**Database for user**

- Create database with same name and grant all privileges.
- Grant all privileges on wildcard name (username\_%).
- Grant all privileges on database "phpbasa".

**Global privileges**  Check All

დავაჭიროთ Go ღილაკს თითი, რის შედეგადაც ახალი მომხმარებელი შეიქმნება, რომლის ნახვაც Privileges ჩანართზე გადასვლითაა შესაძლებელი.



ჩატარებული ქმედებების შედეგად გვაქვს მონაცემთა ბაზა php baza, რომელიც შეიცავს menu ცხრილს, რომელიც თავის მხრივ შეიცავს id, name, date ველებს და შესაბამისად menu ცხრილში ექვსი ჩანაწერია.

+ Options				id	name	date
<input type="checkbox"/>		Edit		Copy		Delete
			1	ქეთა	2013-04-05	
			2	გიორგი	2012-12-28	
			3	ზაზა	1995-05-01	
			4	საბა	1996-06-12	
			5	ნათია	1984-01-11	
			6	მარიამი	1990-08-02	

მონაცემთა ბაზასთან სამუშაოდ საჭიროა შესაბამის ჰოსტთან და მონაცემთა ბაზასთან კავშირის დამყარება. ამისათვის, შემდეგი ფუნქცია გამოიყენება:

**mysqli\_connect** ("ჰოსტის სახელი", "მომხმარებლის სახელი", "პაროლი", "ბაზის სახელი")

ეს ფუნქცია შედეგად აბრუნებს ობიექტს, რომლის შენახვაც აუცილებელია ცვლადში.

ჩვენს შემთხვევაში გვექნება

```
$conn = mysqli_connect("localhost", "phpuser", "phppassword", "phpbaza");
```

თუ მონაცემთა ბაზაში ქართული უნიკოდით ჩაწერილი მონაცემები გვაქვს, საჭიროა ჩავწეროთ მოთხოვნა, რომელიც ქართულ უნიკოდთან მუშაობის საშუალებას მოგვცემს:

```
mysqli_query($conn, "set names utf8");
```

`$conn` ობიექტში ბაზასთან კავშირია შენახული.

მონაცემთა ბაზასთან მუშაობა ოთხი ძირითადი მოთხოვნის დამუშავებისაგან შედგება, ესენია: ბაზიდან მონაცემების ამოღება (წაკითხვა), მონაცემების ჩაწერა ბაზაში, მონაცემების რედაქტირება, მონაცემების წაშლა.

## მონაცემთა ბაზიდან მონაცემების ამოღება

მონაცემთა ბაზიდან მონაცემების ამოსაღებად `select` მოთხოვნა გამოიყენება (აქვე უნდა აღვნიშნოთ, რომ ჩვენ არ შევხებით SQL ენის დეტალურ განხილვას, ეს არ წარმოადგენს აღნიშნული თავის ამოცანას).

ამისათვის ლოკალური სერვერის გამშვებ საქალაქში შევქმნათ `phptodb` ფოლდერი, მასში კი `.php` გაფართოების `select` ფაილი.

ფაილში ავკრიბოთ შემდეგი კოდი:

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset='utf8'>
</head>
<body>
<?php
```

```

        $conn = mysqli_connect("localhost", "phpuser",
"phppassword", "phpbaza");
        mysqli_query($conn, "set names utf8");
        $result = mysqli_query($conn, "SELECT * FROM menu");
        $row = mysqli_fetch_array($result);
        echo $row['name'];
?>
</body>
<html>

```

ვნახოთ ჩვენი ჩაწერილი კოდის შედეგი.



როგორც ვხედავთ, მონაცემთა ბაზიდან გამოვიდა name ველის პირველი ჩანაწერი. mysqli\_fetch\_array() ფუნქცია, მოთხოვნის შესაბამის ასოციურ (ასევე ჩვეულებრივ) მასივს წარმოადგენს, რომელიც პირველ გამოყენებაზე მოთხოვნის შესაბამისად ჩანაწერებიდან პირველ სტრიქონს ამუშავებს და მას მასივად გარდაქმნის.

ჩვენს შემთხვევაში \$row სამელებმენტთან მასივს წარმოადგენს, რომელიც პირველი ჩანაწერის ელემენტებს \$row['id']=1, \$row['name']=ქეთა, \$row['date']=2013-04-05 შეიცავს.

თუ გვინდა, რომ ყველა ჩანაწერი გამოვიტანოთ, მაშინ შესაბამისად აღნიშნული პროცესი ციკლში მოვაქციოთ და კოდი შემდეგნაირად ჩავწეროთ:

```

<!DOCTYPE HTML>
<html>
<head>
    <meta charset='utf8'>
</head>
<body>
<?php
    $conn = mysqli_connect("localhost", "phpuser",
"phppassword", "phpbaza");
    mysqli_query($conn, "set names utf8");
    $result = mysqli_query($conn, "SELECT * FROM menu");
    while($row = mysqli_fetch_array($result)){
        echo "იდენტიფიკატორი=".$row['id']."<br>";
        echo "სახელი=".$row['name']."<br>";
        echo "დაბადების თარიღი=".$row['date']."<br><hr
width=200 align=left>";
    }
?>
</body>
</html>

```

კოდის შესრულების შედეგად მიიღება:

იდენტიფიკატორი=1  
სახელი=ქეთა  
დაბადების თარიღი=2013-04-05

---

იდენტიფიკატორი=2  
სახელი=გიორგი  
დაბადების თარიღი=2012-12-28

---

იდენტიფიკატორი=3  
სახელი=ზაზა  
დაბადების თარიღი=1995-05-01

---

იდენტიფიკატორი=4  
სახელი=საბა  
დაბადების თარიღი=1996-06-12

---

იდენტიფიკატორი=5  
სახელი=ნათია  
დაბადების თარიღი=1984-01-11

---

იდენტიფიკატორი=6  
სახელი=მარიამი  
დაბადების თარიღი=1990-08-02

---

## მონაცემების ჩაწერა მონაცემთა ბაზაში

მონაცემთა ბაზაში მონაცემთა ჩასაწერად insert მოთხოვნა გამოიყენება. insert მოთხოვნის დასამუშავებლად, phptodb ფოლდერში შევქმნათ insert.php ფაილი. დავწეროთ შემდეგი კოდი:

```
<!DOCTYPE HTML>  
<html>  
<head>  
    <meta charset='utf8'>  
</head>
```

```

<body>
<?php
    $conn = mysqli_connect("localhost", "phpuser",
"phppassword", "phpbaza");
    mysqli_query($conn, "set names utf8");
    $result = mysqli_query($conn, "INSERT INTO menu
(name, date) VALUES ('ლევანი','1985-02-14')");
?>
</body>
</html>

```

აღნიშნული კოდის გაშვების შემდეგ ჩვენი მონაცემთა ბაზის menu ცხრილის ბოლოში ჩაემატება ახალი ჩანაწერი id=7; name=ლევანი; date=1985-02-14.

ახლა დავწეროთ კოდი, რომელიც ვიზუალური ინტერფეისის საშუალებით ბაზაში ახალ მონაცემს ჩაამატებს.

```

<!DOCTYPE HTML>
<html>
<head>
    <meta charset='utf8'>
    <style>
        .table { border-collapse: collapse;}
        .table td { padding: 5px; border: 1px solid silver;}
    </style>
</head>
<body>
<form method="post">
    <table cellpadding="0" cellspacing="0" class="table">
        <tr>

```

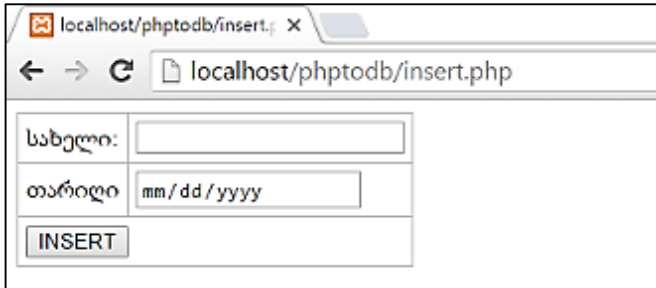


```

                <td>სახელი:</td> <td><input type="text"
name="name"/> </td>
            </tr>
            <tr>
                <td>თარიღი:</td> <td> <input type="date"
name="date"> </td>
            </tr>
            <tr>
                <td colspan="2"> <input type="submit"
name="insert" value="INSERT"></td>
            </tr>
        </table>
</form>
<?php
    if(isset($_POST['insert'])){
        $conn = mysqli_connect("localhost", "phpuser",
"phppassword","phpbaza");
        mysqli_query($conn, "set names utf8");
        $name = $_POST['name'];
        $date = $_POST['date'];
        $result = mysqli_query($conn, "INSERT INTO menu
(name, date) VALUES ('$name', '$date')");
        header("location: insert.php");
    }
?>
</body>
</html>

```

კოდის გაშვების შედეგად ბრაუზერში მივიღებთ გრაფიკულ ინტერფეისს:



შევავსოთ ველები და INSERT ლილაკზე დავაწკაპუნოთ მაუსი, რის შედეგადაც მონაცემები ჩვენს მონაცემთა ბაზაში ჩაიწერება.

header("location: insert.php"); - კოდის ფრაგმენტი კოდის შესრულების შემდეგ insert.php გვერდს ჩატვირთავს, რაც უზრუნველყოფს POST მასივის გათავისუფლებას, ეს კი თავის მხრივ პროგრამის განახლების დროს მონაცემთა ბაზაში ერთი და იმავე მონაცემების ჩაწერისაგან დაგვიცავს.

## მონაცემების რედაქტირება მონაცემთა ბაზაში

მონაცემთა ბაზაში მონაცემების რედაქტირება update მოთხოვნის საშუალებით ხორციელდება. update მოთხოვნის დასამუშავებლად, phptodb ფოლდერში update.php ფაილი შევქმნათ.

დავწეროთ შემდეგი კოდი:

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset='utf8'>
</head>
<body>
<?php
```

```

        $conn = mysqli_connect("localhost", "phpuser",
"phppassword", "phpbaza");
        mysqli_query($conn, "set names utf8");
        $result = mysqli_query($conn, "UPDATE menu SET
name='დაჩი' WHERE id=3");
?>
</body>
<html>

```

კოდის გამვების შედეგად id=3 მომხმარებლის სახელის ველში არსებული ინფორმაცია წაიშლება და ჩაიწერება „დაჩი“.

დავწეროთ კოდი, რომელიც ვიზუალური ინტერფეისის საშუალებით ბაზაში არსებული მონაცემების რედაქტირებას მოახდენს.

```

<!DOCTYPE HTML>
<html>
<head>
    <meta charset='utf8'>
    <style>
        .edit { text-decoration: none; color:blue; font-weight:
bold;}
        .table { border-collapse: collapse;}
        .table td { padding: 5px; border: 1px solid silver;}
    </style>
</head>
<body>
    <table cellpadding="0" cellspacing="0" class="table">
        <tr>
            <td>

```

```

        <?php
            $conn = mysqli_connect("localhost",
"phppuser", "phppassword", "phpbaza");
            mysqli_query($conn, "set names utf8");
            $result = mysqli_query($conn, "SELECT
* FROM menu");

            while($row =
mysqli_fetch_array($result)){
                echo "სახელი =
".$row['name']. "<a class='edit' href=?id=".$row['id']. ">
რედაქტირება</a><br>";
                echo "დაბადების თარიღი =
".$row['date']. "<br><hr width=200 align=left>";
            }
        ?>
    </td>
    <td width="300" valign="top" align="center">
        <?php
            $name = "";
            $date = "";
            if(isset($_GET['id'])){
                $id = $_GET['id'];
                $result = mysqli_query($conn,
"SELECT name, date FROM menu WHERE id=$id");
                $row =
mysqli_fetch_array($result);

                $name = $row['name'];
                $date = $row['date'];
            }
        </td>
    </tr>
</table>
</div>

```

```

                                if(isset($_POST['update']) &&
isset($_GET['id']))){
                                $name = $_POST['name'];
                                $date = $_POST['date'];
                                $result = mysqli_query($conn,
"UPDATE menu SET name='$name', date='$date' WHERE id=$id");
                                header("location: update.php");
                                }
                                ?>
                                <form method="post">
                                <table cellpadding="0"
cellspacing="0" class="table">
                                <tr>
                                <td>სახელი:</td> <td><input type="text" name="name"
value="<?php echo $name?>" /> </td>
                                </tr>
                                <tr>
                                <td>თარიღი:</td> <td> <input type="date" name="date"
value="<?php echo $date?>"> </td>
                                </tr>
                                <tr>
                                <td colspan="2" style="text-align: right;">
                                <input type="submit" name="update"
value="UPDATE"></td>
                                </tr>
                                </table>
                                </form>

```

```

                </td>
            </tr>
        </table>
</body>
</html>

```

კოდის გაშვების შედეგად ბრაუზერში მივიღებთ გრაფიკულ ინტერფეისს

<p>სახელი = ქეთა <b>რედაქტირება</b>  დაბადების თარიღი = 2013-04-05</p> <hr/> <p>სახელი = გიორგი <b>რედაქტირება</b>  დაბადების თარიღი = 2012-12-28</p> <hr/> <p>სახელი = დანი <b>რედაქტირება</b>  დაბადების თარიღი = 1995-05-01</p> <hr/> <p>სახელი = საბა <b>რედაქტირება</b>  დაბადების თარიღი = 1996-06-12</p> <hr/> <p>სახელი = ნათია <b>რედაქტირება</b>  დაბადების თარიღი = 1984-01-11</p> <hr/> <p>სახელი = მარიამი <b>რედაქტირება</b>  დაბადების თარიღი = 1990-08-02</p> <hr/> <p>სახელი = თორნიკე <b>რედაქტირება</b>  დაბადების თარიღი = 1980-04-09</p> <hr/> <p>სახელი = მათე <b>რედაქტირება</b>  დაბადების თარიღი = 2016-02-09</p>	<table border="1"> <tr> <td>სახელი:</td> <td><input type="text"/></td> </tr> <tr> <td>თარიღი</td> <td><input type="text" value="mm / dd / yyyy"/></td> </tr> <tr> <td colspan="2" style="text-align: center;"><input type="button" value="UPDATE"/></td> </tr> </table>	სახელი:	<input type="text"/>	თარიღი	<input type="text" value="mm / dd / yyyy"/>	<input type="button" value="UPDATE"/>	
სახელი:	<input type="text"/>						
თარიღი	<input type="text" value="mm / dd / yyyy"/>						
<input type="button" value="UPDATE"/>							

ფანჯრის მარცხენა ნაწილში მონაცემთა ბაზაში ჩაწერილი ყველა მონაცემი იქნება გამოტანილი, შესაბამისად საშუალება გვექნება რომელიმე მათგანი ავირჩიოთ და რედაქტირება გავუკეთოთ მონაცემებს.

## მონაცემების წაშლა მონაცემთა ბაზაში

მონაცემთა ბაზაში მონაცემების წასაშლელად გამოიყენება delete მოთხოვნა. ამ მოთხოვნის დასამუშავებლად, phptodb ფოლდერში შევქმნათ delete.php ფაილი.

დავწეროთ შემდეგი კოდი:

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset='utf8'>
</head>
<body>
<?php
    $conn = mysqli_connect("localhost", "phpuser",
"phppassword", "phpbaza");
    mysqli_query($conn, "set names utf8");
    $result = mysqli_query($conn, "DELETE FROM menu
WHERE id=2");
?>
</body>
</html>
```

კოდის გაშვების შედეგად წაიშლება menu ცხრილში id=2 ჩანაწერი.

დავწეროთ კოდი, რომელიც ვიზუალური ინტერფეისის საშუალებით წაშლის ბაზაში არსებულ მონაცემებს.

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset='utf8'>
    <style>
        .edit { text-decoration: none; color:blue; font-weight:
bold;}
        .table { border-collapse: collapse;}
        .table td { padding: 5px; border: 1px solid silver;}
    </style>
</head>
<body>
    <table cellpadding="0" cellspacing="0" class="table">
        <tr>
            <td>
                <form method="post">
<?php
    $conn = mysqli_connect("localhost", "phpuser",
"phppassword", "phpbaza");
    mysqli_query($conn, "set names utf8");
    $result = mysqli_query($conn, "SELECT * FROM menu");
    while($row = mysqli_fetch_array($result)){
        $name = $row['name'];
        $date = $row['date'];
        $id=$row['id'];
?>
    <table cellpadding="0" cellspacing="0" class="table">
```



```

        <tr>
            <td width="200">სახელი: - <?php echo $name?></td>
            <td width="200">თარიღი: - <?php echo $date?></td>
            <td> <input type="checkbox" name="id[<?php echo
$id?>]"> </td>
        </tr>
    </table>
<?php
    }
?>
<br>
    <input type="submit" name="delete" value="DELETE">
        </form>
        </td>
    </tr>
</table>
<?php
    if(isset($_POST['delete'])){
        $id = $_POST['id'];
        $id = implode(",", array_keys($id));
        $result = mysqli_query($conn, "DELETE FROM menu
WHERE id IN ({$id})");
        header("location: delete.php");
    }
?>
</body>
<html>

```

კოდის გაშვების შედეგად მივიღებთ ინტერფეისს:

სახელი: - ქეთა	თარიღი: - 2013-04-05	<input type="checkbox"/>
სახელი: - დაჩი	თარიღი: - 1995-05-01	<input type="checkbox"/>
სახელი: - საბა	თარიღი: - 1996-06-12	<input type="checkbox"/>
სახელი: - ნათია	თარიღი: - 1984-01-11	<input type="checkbox"/>
სახელი: - თორნიკე	თარიღი: - 1980-04-09	<input type="checkbox"/>
სახელი: - მათე	თარიღი: - 2016-02-09	<input type="checkbox"/>

მოვნიშნოთ სასურველი ჩანაწერი და მაუსით DELETE ღილაკზე დავაწკაპუნოთ, რის შედეგადაც მონიშნული მონაცემი ბაზიდან წაიშლება.

`<input type="checkbox" name="id[<?php echo $id?>]>` კოდი POST გლობალური მასივისთვის მასივის მიწოდებას უზრუნველყოფს, მოცემულ შემთხვევაში `$_POST["id"]` - მასივი გვექნება, რომელსაც იმდენი ელემენტი ექნება რამდენ ველსაც მოვნიშნავთ, ამ ელემენტების შესაბამისი იდენტიფიკატორები - ამ მასივის id ინდექსები იქნება, ხოლო მნიშვნელობები checkbox-ის აქტიური მნიშვნელობები.

`$id = implode(",", array_keys($id));` კოდი \$id მასივის სტრიქონად გარდაქმნას უზრუნველყოფს, იგი ამ მასივის ინდექსებისაგან შედგება, რომლებიც ერთმანეთისაგან სიმბოლო ,, , “ მძიმითაა გამოყოფილი.

`$result = mysqli_query($conn, "DELETE FROM menu WHERE id IN ({$id})");` კოდი \$id სტრიქონში მძიმით გამოყოფილი მნიშვნელობების menu ცხრილიდან წაშლას უზრუნველყოფს.

## ფაილის ატვირთვა სერვერზე

ფაილებთან მუშაობა პროგრამირებაში ერთ-ერთი მნიშვნელოვანი საკითხია, PHP-ს ფაილებთან და საქალაქდებთან მუშაობის მძლავრი შესაძლებლობები აქვს.

ამ თავში ფაილის ატვირთვის შესაძლებლობას განვიხილავთ, რისთვისაც შესაბამის ფორმას ავაგებთ.

ლოკალური სერვერის გამშვებ საქალაქდეში შევქმნათ phptofile საქალაქდე, ამავე საქალაქდეში შევქმნათ upload.php ფაილი და ქვესაქალაქდე files, რომელშიც ჩვენ ატვირთულ ფაილებს შევინახავთ.

ფაილის ასატვირთად ფორმას აუცილებლად უნდა დავამატოთ ატრიბუტი enctype="multipart/form-data" შესაბამისი მნიშვნელით. ფაილის ატვირთვის მარტივ ფორმას შემდეგი სახე აქვს:

```
<form method="post" enctype="multipart/form-data">
  <table cellpadding="0" cellspacing="0" class="table">
    <tr>
      <td>ფაილის არჩევა:</td> <td><input type="file"
accept="image/*" name="avatar"> </td>
    </tr>
    <tr>
      <td colspan="2"> <input type="submit" name="insert"
value="UPLOAD"></td>
    </tr>
  </table>
</form>
```

ფორმაში <input type="file" accept="image/\*" name="avatar"> ტეგის საშუალებით იქმნება ფაილების ატვირთვის ფორმა,

accept="image/\*" - ატრიბუტი კი განსაზღვრავს, რა ტიპის ფაილი შეიძლება მოიძიოს მომხმარებელმა, ამ შემთხვევაში შესაძლებელია მხოლოდ გრაფიკული ფაილების მოძიება, თუმცა რა თქმა უნდა, აღნიშნული ატრიბუტი უსაფრთხოებას სრულად ვერ უზრუნველყოფს იმ კუთხით, რომ მომხმარებელი სხვა ტიპის ფაილს არ ატვირთავს.

ფორმის შედეგად ბრაუზერში მივიღებთ:

ფაილის ატვირთვისას გამოიყენება \$\_FILES გლობალური მასივი, რომელიც სერვერზე ატვირთული ფაილის მონაცემების მიღებას უზრუნველყოფს.

შესაბამის PHP კოდს შემდეგი სახე აქვს (PHP კოდი შესაძლებელია ჩავრთოთ იგივე ფაილში, სადაც ფორმის კოდია განთავსებული, ჩვენს შემთხვევაში upload.php ფაილში):

```
<?php
if(isset($_POST['up'])){
    $fileName = $_FILES['avatar']['name'];
    $fileSize = $_FILES['avatar']['size'];
    $fileType = $_FILES['avatar']['type'];
    $fileTempSrc = $_FILES['avatar']['tmp_name'];
    echo "File Name - ".$fileName."<br/>";
    echo "File Size - ".$fileSize."<br/>";
    echo "File Type - ".$fileType."<br/>";
    $src ="files/".$fileName;
    move_uploaded_file($fileTempSrc, $src);
```

```
}  
?>
```

კოდში, როგორც ვხედავთ, `$_FILES['avatar']` წარმოადგენს გლობალურ მასივს, რომელიც თავის მხრივ მოიცავს შემდეგს:

`$_FILES['avatar']['name']` - ატვირთული ფაილის სახელი;

`$_FILES['avatar']['size']` - ატვირთული ფაილის ზომა;

`$_FILES['avatar']['type']` - ატვირთული ფაილის ტიპი;

`$_FILES['avatar']['tmp_name']` - ატვირთული ფაილის დროებითი მისამართი;

`move_uploaded_file(tmp_src, src)` - ფუნქციას `tmp_src` მისამართიდან ფაილი გადააქვს `src` მისამართზე. ჩვენს შემთხვევაში დროებით მისამართზე განთავსებულ ფაილს `files` საქალაქოდეში გადაიტანს იმავე სახელით, რა სახელიც ფაილს ატვირთვის მომენტში ერქვა.

საბოლოოდ `upload.php` ფაილში ჩაწერილი პროგრამული კოდი შემდეგნაირად გამოიყურება:

```
<!DOCTYPE html>  
<html>  
<head>  
  <meta charset="UTF-8">  
  <title>Title</title>  
  <style>  
    .table { border-collapse: collapse;}  
    .table td { padding: 5px; border: 1px solid silver;}  
  </style>  
</head>  
<body>  
<form method="post" enctype="multipart/form-data">
```

```

<table cellpadding="0" cellspacing="0" class="table">
  <tr>
    <td>ვაილის არჩევა:</td> <td><input type="file"
accept="image/*" name="avatar"> </td>
  </tr>
  <tr>
    <td colspan="2"> <input type="submit" name="up"
value="UPLOAD"></td>
  </tr>
</table>
</form>
<?php
  if(isset($_POST['up'])){
    $fileName = $_FILES['avatar']['name'];
    $fileSize = $_FILES['avatar']['size'];
    $fileType = $_FILES['avatar']['type'];
    $fileTempSrc = $_FILES['avatar']['tmp_name'];
    echo "File Name - ".$fileName."<br/>";
    echo "File Size - ".$fileSize."<br/>";
    echo "File Type - ".$fileType."<br/>";
    echo "File Temp_Name - ".$fileTempSrc."<br/>";
    $src="files/".$fileName;
    move_uploaded_file($fileTempSrc, $src);
  }
?>
</body>
</html>

```

განხილულ მაგალითში აუცილებელია phptofile საქაღალდეში გვქონდეს files ქვესაქაღალდე, რომელშიც შესაბამისი ფაილები აიტვირთება.

## სესიებთან მუშაობა

ვებსაიტის აგების პროცესში ერთი გვერდიდან (ლინკიდან) მეორე გვერდზე (ნიკლზე) გადასვლისას ინფორმაციის შენახვა ერთ-ერთ მნიშვნელოვან საკითხს წარმოადგენს. ვებდაპროგრამებაში აღნიშნული საკითხის გადაწყვეტას სესიები ემსახურება.

სესიების გამოსაყენებლად php-ში \$\_SESSION გლობალური მასივი გვაქვს, რომლის საშუალებითაც შესაძლებელია ვებსაიტზე ნავიგაციის პროცესში მონაცემების შენახვა. სესიების გამოსაყენებლად საჭიროა ფაილის დასაწყისში session\_start() ფუნქცია ჩავთვოთ. სესიებთან მუშაობის უკეთესად გასაცნობად განვიხილოთ პრაქტიკული მაგალითი.

ლოკალური სერვერის გამშვებ საქალაქში შევქმნათ phptosession საქალაქი, ამ საქალაქშივე შევქმნათ page1.php, page2.php, page3.php ფაილები.

page1.php ფაილში ჩავწეროთ შემდეგი კოდი:

```
<?php
    session_start();
?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Page 1</title>
</head>
<body>
<?php
    $_SESSION["name"] = "Teimuraz";
    $_SESSION["lastName"] = "Sturua";
```



```

$name = "Beso";
$lastName = "Tabatadze";
echo "Session Name - <b>".$_SESSION["name"]."</b><br/>";
echo "Session LastName - <b>".$_SESSION["lastName"]."</b><br/>";
echo "Name - <b>".$name."</b><br/>";
echo "LastName - <b>".$lastName."</b><br/>";
echo "<a href='page2.php'>PAGE 2</a><br/>";
echo "<a href='page3.php'>PAGE 3</a>";
?>
</body>
</html>

```

ფაილის თავში session\_start() ფუნქციაა ჩართული აგრეთვე გვაქვს ორი ტიპის მონაცემი:

\$\_SESSION["name"] - \$\_SESSION მასივში და \$name - ცვლადში შენახული მონაცემები, შესაბამისად ბრაუზერში მივიღეთ:

```

Session Name - Teimuraz
Session LastName - Sturua
Name - Beso
LastName - Tabatadze
PAGE 2
PAGE 3

```

ახლა ვნახოთ, როგორ გამოჩნდება page2.php ფაილის ჩატვირთვისას აღნიშნული ცვლადების მნიშვნელობები.

page2.php ფაილში ჩაწეროთ შემდეგი კოდი:

```

<?php
    session_start();
?>

```

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Page 2</title>
</head>
<body>
<?php
  echo "Session Name - <b>".$_SESSION["name"]."</b><br/>";
  echo "Session LastName - <b>".$_SESSION["lastName"]."</b><br/>";
  echo "Name - <b>".$name."</b><br/>";
  echo "LastName - <b>".$lastName."</b><br/>";
  session_destroy();
  echo "<a href='page1.php'>PAGE 1</a><br/>";
  echo "<a href='page3.php'>PAGE 3</a>";
?>
</body>
</html>

```

როგორც ვხედავთ, page2.php ფაილში იმ ცვლადების მნიშვნელობების გამოტანა ხდება, რომლებიც page1.php ფაილში განვსაზღვრეთ, მივიღებთ შედეგს:

```

Session Name - Teimuraz
Session LastName - Sturua

Notice: Undefined variable: name in D:\xampp\htdocs\phptosession\page2.php on line 14
Name -

Notice: Undefined variable: lastName in D:\xampp\htdocs\phptosession\page2.php on line 15
LastName -
PAGE 1
PAGE 3

```

შედეგიდან ჩანს, რომ \$\_SESSION მასივში შენახული მონაცემები page2.php ფაილში აღქმადია, ჩვეულებრივ ცვლადებში შენახული მონაცემები კი - არა, შესაბამისად მივიღეთ შეცდომა.

სესიის გასაუქმებლად session\_destroy() ფუნქცია გამოიყენება, აღნიშნული ფუნქციის გამოყენების შემდეგ \$\_SESSION მასივიდან ინფორმაცია წაიშლება. სადემოსტრაციოდ განვიხილოთ page3.php ფაილი.

page3.php-ში ჩავწეროთ შემდეგი კოდი:

```
<?php
    session_start();
?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Page 3</title>
</head>
<body>
<?php
    echo "Session Name - <b>".$_SESSION["name"]."</b><br/>";
    echo "Session LastName - <b>".$_SESSION["lastName"]."</b><br/>";
    echo "Name - <b>".$name."</b><br/>";
    echo "LastName - <b>".$lastName."</b><br/>";
    echo "<a href='page1.php'>PAGE 1</a><br/>";
    echo "<a href='page2.php'>PAGE 2</a>";
?>
</body>
</html>
```

თუ page3.php ფაილზე page2.php ფაილიდან გადავალთ მივიღებთ შედეგს:

```
Notice: Undefined index: name in D:\xampp\htdocs\phptosession\page3.php on line 12
Session Name -

Notice: Undefined index: lastName in D:\xampp\htdocs\phptosession\page3.php on line 13
Session LastName -

Notice: Undefined variable: name in D:\xampp\htdocs\phptosession\page3.php on line 14
Name -

Notice: Undefined variable: lastName in D:\xampp\htdocs\phptosession\page3.php on line 15
LastName -
PAGE 1
PAGE 2
```

როგორც ვხედავთ, page3.php ფაილისათვის \$\_SESSION მასივში შენახული მონაცემები უკვე აღარ არის აღქმადი, რაც გამომწვეულია page2.php ფაილში არსებული session\_destroy() ფუნქციით.

კონკრეტული სესიის გათიშვა შესაძლებელია session\_unset() ფუნქციის საშუალებით. სესიაში ჩაწერილი მონაცემი ჩუმათობის პრინციპით 20 წუთს ინახება.

## შინაარსი

<b>რა არის PHP?</b> .....	<b>3</b>
PHP-ის შესაძლებლობანი.....	4
PHP-ის უპირატესობა .....	6
ტრადიციულობა.....	6
სიმარტივე.....	7
ეფექტურობა .....	8
უსაფრთხოება.....	8
მოქნილობა.....	9
PHP-ის განვითარების ისტორია .....	10
PHP-ის ბირთვი .....	12
<b>Web-სერვერის ჩატვირთვა და ინსტალაცია</b> .....	<b>17</b>
<b>PHP 5-ის ჩატვირთვა და ინსტალაცია</b> .....	<b>24</b>
<b>პირველი პროგრამა PHP-ზე</b> .....	<b>31</b>
კომენტარი PHP სკრიპტებში .....	41
PHP-info.....	43
<b>ცვლადები PHP-ში</b> .....	<b>45</b>
მონაცემთა ტიპები PHP-ში .....	47
ტიპი Boolean (ორობითი მონაცემები) .....	48
ტიპი integer (მთელი რიცხვი) .....	50

ტიპი float (მცოცავმძიმისანი რიცხვი) .....	51
ტიპი string (სტრიქონი).....	52
მონაცემთა გაერთიანება (კონკატენაცია).....	57
ცვლადის არსებობის შემოწმება.....	60
ტიპი array (მასივი) .....	61
მარტივი მასივები და სიები PHP-ში .....	62
ასოციური მასივები PHP-ში .....	65
ტიპი object (ობიექტი).....	68
ტიპი resource (რესურსები) .....	69
ტიპი NULL (ცარიელი ტიპი) .....	69
ფსევდოტიპი mixed (შერეული ტიპი).....	70
ფსევდოტიპი number (რიცხვი) .....	70
ფსევდოტიპი callback (უკუ გამოძახების) .....	70
<b>კონსტანტა PHP-ში.....</b>	<b>73</b>
კონსტანტის არსებობის შემოწმება.....	74
PHP-ის სტანდარტული კონსტანტები.....	74
<b>გამოსახულება PHP-ში .....</b>	<b>76</b>
<b>PHP-ის ოპერატორები .....</b>	<b>79</b>
არითმეტიკული ოპერატორები .....	79
ინკრემენტისა და დეკრემენტის ოპერატორები.....	80
მინიჭების ოპერატორები .....	83

<b>ბმულები PHP-ში .....</b>	<b>85</b>
მყარი ბმულები PHP-ში.....	85
დინამიკური ცვლადები .....	87
მყარი ბმულები და მომხმარებლის ფუნქციები.....	88
ბმულის წაშლა .....	91
ბიტური ოპერატორები.....	92
შედარების ოპერატორები .....	93
ლოგიკური ოპერატორები .....	94
PHP-ის ოპერატორების პრიორიტეტები.....	95
სტრიქონული ოპერატორები.....	96
PHP-ს მასივებთან სამუშაო ოპერატორები .....	96
სტრიქონის დამუშავება.....	98
მარტივი სინტაქსი .....	99
რთული (ფიგურული) სინტაქსი .....	101
სტრიქონში სიმბოლოზე წვდომა და მისი შეცვლა.....	103
<b>PHP ენის კონსტრუქციები .....</b>	<b>105</b>
პირობითი ოპერატორები.....	106
კონსტრუქცია if.....	106
კონსტრუქცია else .....	108
კონსტრუქცია elseif.....	109
ციკლები PHP-ში.....	110
ციკლი წინაპირობით while.....	110

ციკლი შემდგომი პირობით do while .....	113
ციკლი მთვლელით for .....	114
მასივების გადავსების ციკლი foreach .....	116
კონსტრუქცია break .....	118
კონსტრუქცია continue.....	120
ამორჩევის კონსტრუქციები.....	121
მნიშვნელობის დაბრუნების კონსტრუქცია.....	127
კონსტრუქცია return.....	127
<b>ფუნქციები .....</b>	<b>130</b>
PHP-ის მათემატიკური ფუნქციები.....	131
სტრიქონული ფუნქციები და ოპერატორები .....	134
სტრიქონებთან სამუშაო ფუნქციები .....	136
ბაზური სტრიქონული ფუნქცია.....	136
ტექსტის ბლოკებთან სამუშაო ფუნქციები .....	138
ცალკეულ სიმბოლოებთან სამუშაო ფუნქციები.....	140
ჰარების წასაშლელი ფუნქცია .....	141
სიმბოლოთა გარდაქმნის ფუნქციები .....	142
რეგისტრის შესაცვლელი ფუნქცია.....	144
ლოკალის მომართვა (ლოკალური მონაცემების მომართვა) .....	145
გამოტანის ბუფერის გასუფთავების ფუნქცია .....	147
სტრიქონულ ტიპად გარდაქმნა .....	147



მონაცემთა ტიპის მინიჭება და ინდიკაცია .....	149
სტრიქონის რიცხვად გარდაქმნა.....	150
მასივებთან სამუშაო ფუნქციები.....	152
ოპერაციები მასივებზე .....	154
მასივის ელემენტების მოწესრიგება (დახარისხება).....	154
ოპერაციები მასივის გასაღებებსა და მნიშვნელობებზე.	159
მასივების შერწყმა.....	161
მასივის ნაწილის მიღება .....	162
მასივის ელემენტების ჩამატება და წაშლა .....	163
რიცხვთა დიაპაზონის - სიის შექმნა .....	166
მასივის ელემენტების მთვლელი .....	168
მასივისა და მისი ელემენტების წაშლა.....	170
მასივებთან მუშაობის ზოგიერთი თავისებურება.....	170
მასივად გარდაქმნა (ტიპი array) .....	171
ფაილებთან სამუშაო ფუნქციები .....	176
ფაილის გახსნა.....	176
ფაილის დამუშავება.....	182
საქალაქდესთან სამუშაო ფუნქციები.....	185
თარიღისა და დროის ფუნქციები.....	193
ელექტრონული ფოსტის ფუნქციები .....	201
<b>GD - გამოსახულებასთან მუშაობა .....</b>	<b>205</b>
გამოსახულების დასამუშავებელი ფუნქციები .....	206

<b>მომხმარებლის ფუნქცია PHP-ში .....</b>	<b>216</b>
PHP-ის მომხმარებლის ფუნქციის თავისებურებანი .....	216
მომხმარებლის ფუნქციის შექმნა .....	222
კონსტრუქცია return.....	223
მომხმარებლის ფუნქციისათვის არგუმენტის გადაცემა	225
არგუმენტების გადაცემა ბმულით .....	226
პარამეტრები ჩუმათობით.....	232
რეკურსიული ფუნქციები .....	233
პირობით განმსაზღვრელი ფუნქციები.....	235
ცვლადი რაოდენობის არგუმენტები ფუნქციაში .....	236
<b>ობიექტზე ორიენტირებული დაპროგრამების</b>	
<b>საფუძვლები .....</b>	<b>240</b>
ინკაპსულაცია .....	242
პოლიმორფიზმი .....	243
მემკვიდრეობითობა .....	243
ძირითადი ცნებები.....	244
განმარტების სირთულე.....	245
<b>კლასები და ობიექტები PHP-ში.....</b>	<b>248</b>
PHP-ში კლასებთან და ობიექტებთან წვდომა.....	249
ობიექტების ინიციალიზაცია.....	252
კლასების მემკვიდრეობითობა PHP-ში .....	254
კლასების პოლიმორფიზმი PHP-ში.....	255

<b>ლამაზი სკრიპტი ანუ დაპროგრამების სტილი .....</b>	<b>256</b>
ძირითადი წესები .....	259
ფიგურული ფრჩხილების განლაგება .....	259
ცვლადების, ფუნქციებისა და კლასების სახელები.....	261
კომენტარები.....	261
ოპერატორი GOTO.....	263
<b>ჩართვის კონსტრუქცია PHP-ში.....</b>	<b>266</b>
ჩართვის კონსტრუქცია require.....	266
ჩართვის კონსტრუქცია include .....	267
ერთჯერადი ჩართვის კონსტრუქციები require_once და include_once.....	270
<b>PHP – MySQL.....</b>	<b>271</b>
მონაცემთა ბაზის აგება .....	271
მონაცემთა ბაზიდან მონაცემების ამოღება.....	282
მონაცემების ჩაწერა მონაცემთა ბაზაში.....	285
მონაცემების რედაქტირება მონაცემთა ბაზაში.....	288
მონაცემების წაშლა მონაცემთა ბაზაში.....	293
<b>ფაილის ატვირთვა სერვერზე.....</b>	<b>297</b>
<b>სესიებთან მუშაობა .....</b>	<b>302</b>

რედაქტორი ი. მეგრელიშვილი

გადაეცა წარმოებას 13.01.2017. ხელმოწერილია დასაბეჭდად  
20.02.2017. ქალაქის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 19,5.

საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, თბილისი,  
კოსტავას 77



ი.მ. „გოჩა დალაქიშვილი“,  
თბილისი, ვარკეთილი 3, კორპ. 333, ბინა 38