თეიმურაზ სტურუა
გიორგი კუჭავა

საგამომცემლო სახლი
„ტექნიკური უნივერსიტეტი“

საქართველოს ტექნიკური უნივერსიტეტი

თეიმურაზ სტურუა, გიორგი კუჭავა

ვებპროგრამება – PHP

(თეორია და პრაქტიკა)

დამტკიცებულია სახელმძღვანელოდ
საქართველოს ტექნიკური უნივერსიტეტის
სარედაქციო-საგამომცემლო საბჭოს
მიერ. 05.07.2019, ოქმი №2

თბილისი
2019

უაკ 681.3

სახელმძღვანელოში განხილულია PHP-ის თანამედროვე ვერსია. წიგნში აღწერილია PHP-ის ჩამოტვირთვისა და ინსტალაციის ეტაპები, დაპროგრამების ძირითადი ცნებები, როგორიცა ცვლადები, ციკლი, პირობითი ოპერატორები და მასივები, აგრეთვე ობიექტზე ორიენტირებული დაპროგრამების საფუძვლები და PHP 5-ში მათი გამოყენების შესაძლებლობები. სახელმძღვანელოში მრავლადაა მოყვანილი შესაბამისი მაგალითები და მოცემულია დამოუკიდებლად შესასრულებელი მრავალი დავალება.

განკუთვნილია ინფორმატიკისა და მართვის სისტემების ფაკულტეტის პროფესიული სწავლების, ბაკალავრიატისა და მაგისტრატურის სტუდენტებისათვის. სახელმძღვანელო საინტერესო იქნება აგრეთვე იმ სპეციალისტებისათვისაც, ვისაც დინამიკური ვებსაიტების შექმნის შესწავლა სურს.

რეცენზენტები: ევროპის უნივერსიტეტის ასოცირებული
პროფესორი ბესიკ ტაბატაძე

საქართველოს ტექნიკური უნივერსიტეტის
ინფორმატიკისა და მართვის სისტემების
ფაკულტეტის ასოცირებული პროფესორი
ირაკლი ბოჭორიშვილი,

ISBN 978-9941-28-530-1

http://www.gtu.ge

Verba volant,
scripta manent

შესავალი

PHP არის ღია საწყისი კოდით საერთო დანიშნულების სცენარის ფართოდ გამოყენებული ენა. ეს არის დაპროგრამების ენა, რომელიც სპეციალურად ვებსერვერზე გამოსაყენებელი ვებდანართების (სცენარების) დასაწერად შეიქმნა.

PHP 5 – ეს არის PHP (PHP: Hypertext Preprocessor) დაპროგრამების ენის ბოლო ვერსია. PHP ენა თავდაპირველად 1994 წელს შეიქმნა რასმუს ლერდორფის (Rasmus Lerdorf) მიერ დინამიკური, ინტერაქტიული ვებსაიტების შესაქმნელად. მას შემდეგ მრავალი სპეციალისტის ძალისხმევით PHP თანდათან სრულფასოვანი დაპროგრამების ენა გახდა. ამის საფუძველს გვაძლევს ის, რომ PHP სულ უფრო პოპულარული და სრულფასოვანი ობიექტზე ორიენტირებული დაპროგრამების ენის პრინციპების მატარებელი ენა გახდა.

რა არის PHP?

ცნობილია, რომ PHP წარმოადგენს დაპროგრამების ენას, რომლის საშუალებითაც შეიძლება დაიწეროს პროგრამები, ისინი შეიძლება გამოყენებულ იქნეს კროს-პლატფორმის HTML-თავსებადი, ვებსცენარების ჩასაწერი და სერვერული ენის სახით. განვიხილოთ თითოეული მათგანი ცალკ-ცალკე:

- კროს-პლატფორმა – ეს ნიშნავს, რომ PHP-კოდი ყოველ-გვარი ცვლილების გარეშე შეიძლება გამოყენებულ იქნეს სხვადასხვა ოპერაციულ სისტემაზე მომუშავე კომპიუტერებზე. მაგალითად, PHP-სცენარი, რომელიც მუშაობს Windows-ზე მომუშავე კომპიუტერზე, ასევე წარმატებით იმუშავებს Linux-ზე მომუშავე კომპიუტერებზეც;

- HTML-თავსებადი – ნიშნავს, რომ PHP-კოდი შეიძლება ჩაიწეროს ისეთ ფაილებში, რომლებიც PHP-ინსტრუქციებისა და HTML-კოდის „ნარევს“ წარმოადგენს;
- სერვერული ენა – ნიშნავს, რომ PHP-პროგრამა მუშაობს ვებსერვერზე;
- ვებსცენარების ჩასაწერი ენა – ნიშნავს, რომ PHP-პროგრამების შესრულებაზე გაშვება ვებბრაუზერის საშუალებითაა შესაძლებელი.

ყოველივე ეს ნიშნავს, რომ PHP-ზე ჩაწერილი პროგრამები მუშაობენ ვებსერვერზე, რომლებშიც შეიძლება ერთმანეთში შერეული იყოს როგორც PHP-კოდი, ასევე HTML-კოდი. ასეთ პროგრამებთან წვდომა ხორციელდება ვებბრაუზერის საშუალებით, რომელზეც PHP დამუშავების შედეგები აისახება და ვებსერვერი პროგრამას აბრუნებს HTML-კოდის სახით. სხვა სიტყვებით რომ ვთქვათ, ვებსერვერზე ვების საშუალებით იქმნება ყველასათვის ხელმისაწვდომი პროგრამა.

ვებსაიტების უმეტესობა არ არის სტატიკური, არამედ დინამიკურია, უმეტეს შემთხვევაში კი – ინტერაქტიული. მათ შეუძლიათ აჩვენონ სტატიების სია, რომლებიც მომხმარებლისათვის საინტერესო გარკვეულ სიტყვას შეიცავს, ან უახლესი ამბები, ან მიესალმოს მომხმარებელს სახელით, როდესაც იგი სისტემაში დარეგისტრირდება. ასეთი საიტები მომხმარებელს საშუალებას აძლევს, რომ ურთიერთობა დაამყაროს მათთან და მომხმარებლის არჩევანის შესაბამისად შესთავაზოს მას სხვადასხვა სახის ინფორმაცია.

მსგავსი საიტების შექმნა მხოლოდ HTML-კოდის გამოყენებით შეუძლებელია. აქ დასახმარებლად PHP ენა მოდის,

რომლის მეშვეობითაც ისეთი საიტების დაპროგრამებაა შესაძლებელი, რომლებიც:

- წარმოადგენს მონაცემებს სხვადასხვა წყაროებიდან, როგორიცაა მონაცემთა ბაზა, ფაილები ან სხვა ვებგვერდები;
- შეიცავს ინტერაქტიულ ელემენტებს, როგორიცაა საძიებო საშუალებები, შეტყობინებების მიმოცვლა და საზოგადოებრივი აზრის გამოკითხვა;
- მომხმარებელს საშუალებას აძლევს განახორციელოს გარკვეული ქმედებები, როგორიცაა ელ-ფოსტის გაგზავნა ან შესყიდვების განხორციელება.

სხვა სიტყვებით რომ ვთქვათ, PHP შეიძლება გამოყენებულ იქნეს ისეთი საიტების დასაწერად, რომლებსაც რეგულარული მომხმარებელი ყოველდღიურად ხვდება. საიტების უმეტესობა თავისთავში მოიცავს საძიებო სისტემებს ან საინფორმაციო პორტალებს, ან ელექტრონულ კომერციას, ან ზოგიერთ მათგანს, ან ყველა პროგრამას ერთად.

PHP 5 შეიძლება გამოყენებულ იქნეს ფართო სპექტრის პროგრამების შესაქმნელად: მარტივი უტილიტებით დაწყებული, როგორიცაა ტექსტური რედაქტორი, ძლიერი ვებაპლიკაციებით, როგორიცაა პროტოკოლირების დისპეტჩერი.

კლიენტის მხარეს შესრულებადი სხვა ნებისმიერი კოდისაგან 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 კლიენტი შეიძლება ვერც კი მიხვდეს ჩვეულებრივი HTML-ფაილის თუ სკრიპტის შესრულების შედეგს იღებს.

PHP-მ ბოლო წლების განმავლობაში განვითარების დიდი გზა გაიარა და ვებდაპროგრამების 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 1994 წელს რასმუს ლერდორფმა შექმნა. 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 ვებდანართების 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“.

ცხრათვიანი ტესტირების შემდეგ PHP 3.0 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 მუშაობდა.

ახალი „ამძრავი ძალა“, ანუ ბირთვი, რომელსაც „Zend Engine“ უწოდეს, ამ პრობლემას წარმატებით ართმევს თავს და პირველად 1999 წელს იქნა წარმოდგენილი. PHP 4.0, რომელიც ამ „ამძრავ ძალას“ დაეფუძნა და ხმარებაში შემოიტანა დამატებითი ფუნქციების ნაკრები, ოფიციალურად 2000 წლის მაისში გამოვიდა. ზემოთ ჩამოთვლილი სიახლეების გარდა PHP 4.0-ში კიდევ მრავალი სიახლე იყო, როგორიცაა 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 გამოუშვეს. 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 მოდელებს 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 მიიჩნიეს Unicode-ის მხარდაჭერის სირთულის გამო და მასზე მუშაობა შეწყდა. PHP 6-ის საწყისმა კოდმა 5.4-ის ვერსიის დამუშავებაში გადაინაცვლა.

HTTP და ინტერნეტი

თუ თქვენ ადრე იყავით დაკავებული ვებდაპროგრამებით, მაშინ სასარგებლოა მოკლედ მიმოვიხილოთ ინტერნეტის ფუნქციონირების ძირითადი პრინციპები.

ვებსერვერი ეს არის პროგრამა, რომელიც იყენებს HTTP (HyperText TransferProtocol) ჰიპერტექსტური ინფორმაციის გადაცემის პროტოკოლს[1], რომელიც ემსახურება ფაილების გადაცემას ვებგვერდებიდან მომხმარებლამდე, მათი მოთხოვნის შესაბამისად. ამ პროცესის კონკრეტული მაგალითია კლიენტ-სერვერული მოდელი, კომპიუტერები, რომლებზეც განთავსებულია ვებსაიტები და გააჩნიათ შესაბამისი ვებსერვერ პროგრამები. ვებსერვერებს შორის ყველაზე გავრცელებულია

[1] ქსელური პროტოკოლი – ეს არის წესებისა და მოქმედებების ერთობლიობა (ან მოქმედებათა მიმდევრობა), რომელიც საშუალებას გვაძლევს განვახორციელოთ კავშირი ან მონაცემთა მიმოცვლა ორ ან მეტ ქსელში ჩართულ მოწყობილობებს შორის.

Apache, Microsoft-ის Internet Information Server (IIS), კომპანია NGNIX-ის ngnix (გამოითქმის engine x), Google-ის Google Web Server (GWS), IBM-ის Domino Servers და Novell-ის NetWare Server.

ვებსერვერები გამოიყენება როგორც ინტერნეტში და ინტრანეტში[2], აგრეთვე ელექტრონული წერილების გადამგზავნ პროგრამებად და ვებგვერდების შექმნის პროცედურების მხარდამჭერ საშუალებებად.

ჰიპერტექსტური ინფორმაციის გადაცემის პროტოკოლი (HyperText TransferProtocol, HTTP) აღწერს, თუ როგორ უნდა გადაეცეს ვებგვერდები ინტერნეტის საშუალებით. HTTP არის მეთოდი, რომელიც გამოიყენება ინფორმაციის ქსელში (World Wide Web) გადაცემისთვის. მისი ძირითადი ამოცანაა HTML-გვერდების გამოქვეყნების შესაძლებლობა და მათზე წვდომის უნარი.

ტექნოლოგიური სტანდარტების დამუშავებისა და დანერგვის ორგანიზაცია მსოფლიო ქსელის კონსორციუმისათვის (World Wide Web Consortium, W3C) ინტერნეტის საინჟინრო პრობლემების სამუშაო ჯგუფი (Internet Engineering Task Force, IETF) ერთობლივად კოორდინაციას უწევს „მოთხოვნა-პასუხის“ ტიპის HTTP პროტოკოლის განვითარებას, რომელიც აღწერს კლიენტებსა და სერვერებს შორის ურთიერთქმედებას.

მოთხოვნის წყაროს წარმოადგენს კლიენტი. როგორც წესი, ეს არის ვებბრაუზერი, რომელსაც მომხმარებელი აგენტი (user

[2] ინტრანეტი (ინგლ. Intranet) – ინტერნეტისაგან განსხვავებით ეს არის ლოკალური ქსელი, შექმნილი ორგანიზაციის საჭიროებისათვის და მასზე წვდომა მხოლოდ ორგანიზაციაში მომუშავე პირებს აქვთ.

agent) ეწოდება. მოთხოვნის მიმღები სერვერი, სადაც ინახება ან იქმნება HTML ფაილი, სურათები და სხვა რესურსები, გამავალ სერვერს (origin server) ეწოდება. მომხმარებელ აგენტსა და გამავალ სერვერს შორის შეიძლება იყოს განთავსებული ისეთი შუალედური სერვერები, როგორიცაა პროქსი-სერვერები (proxies).

HTTP-კლიენტი ახდენს მოთხოვნის ინიცირებას და მართვის გადაცემის პროტოკოლით (Transmission Control Protocol, TCP) დაშორებული სერვერის განსაზღვრული პორტით (გაჩუმების პრინციპით ეს არის 80 პორტი) ამყარებს კავშირს. HTTP-სერვერი ელოდება მოთხოვნას კლიენტისაგან და ამიტომ იგი მუდმივად აკონტროლებს ამ პორტს. მოთხოვნის მიღებისთანავე სერვერი უკან აგზავნის მდგომარეობის სტრიქონს, მაგალითად, „HTTP/1.1 200 OK“, და პასუხს. მდგომარეობის სტრიქონის ტიპის მიხედვით პასუხი შეიძლება იყოს მოთხოვნილი ფაილი, შეტყობინება შეცდომის შესახებ ან რაიმე სხვა ინფორმაცია. მოთხოვნის დამუშავების დამთავრებითანავე სერვერი წყვეტს არსებულ კავშირს, ანუ კლიენტის ყოველ მომდევნო HTTP-მოთხოვნაზე ახალი TCP-კავშირი იხსნება.

HTTP პროტოკოლი ეფუძნება TCP პროტოკოლს, რომელიც, თავის მხრივ, ეფუძნება ინტერნეტპროტოკოლს (Internet Protocol, IP). ბოლო ორი პროტოკოლის აღწერისას, როგორც წესი, ისინი ერთად მოიხსენიება TCP/IP.

ქსელის კვანძებში განთავსებულ დანართებს ერთმანეთთან კავშირის დასამყარებლად და მონაცემთა გაცვლისათვის TCP პროტოკოლის გამოყენება შეუძლიათ. ეს პროტოკოლი გამგზავნიდან მომღებამდე მონაცემთა საიმედო

მიწოდების გარანტიას იძლევა. მრავალი გამოყენებითი პროტოკოლი ეფუძნება TCP პროტოკოლს, მათ შორისაა Web, e-mail და Secure Shell (SSH).

PHP-ის ბირთვი

როგორც ცნობილია, დაპროგრამების ენა ორი სახის არსებობს: ინტერპრეტირებადი და კომპილირებადი.

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 გავუშვათ.

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.

ინფორმაციის ბაზა
Web-დასწრები
PHP
MySQL
Oracle
ODBC
XML
MAP
LDAP
Java
...
მიდგომა
PHP ZEND ბირთვი
ფუნქციები მიდგომები
ინტერპრეტატორი
ტრანსლატორი
ვებ-სერვერის ინტერფეისი
მომხმარებელი
ვებსერვერის დისკი

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.

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;

- ვებდანართების 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.

ვებსერვერის ჩატვირთვა და ინსტალაცია

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

ვებსერვერის მუშაობისათვის არ არის აუცილებელი კომპიუტერის, რომელზედაც ის არის დაყენებული, ინტერნეტში ან ლოკალურ ქსელში ჩართვა. ვებსერვერთან, რომელიც მოცემულ კომპიუტერშია დაყენებული, წვდომა ამავე კომპიუტერში არსებული ვებბრაუზერის საშუალებით შეიძლება განხორციელდეს იმ შემთხვევაშიც კი, როცა ამ კომპიუტერს არც მოდემი და არც ქსელური პლატა არ გააჩნია, თუმცა თვით პროგრამული უზრუნველყოფის ჩატვირთვისა და მისი დაყენებისათვის ინტერნეტი აუცილებელია.

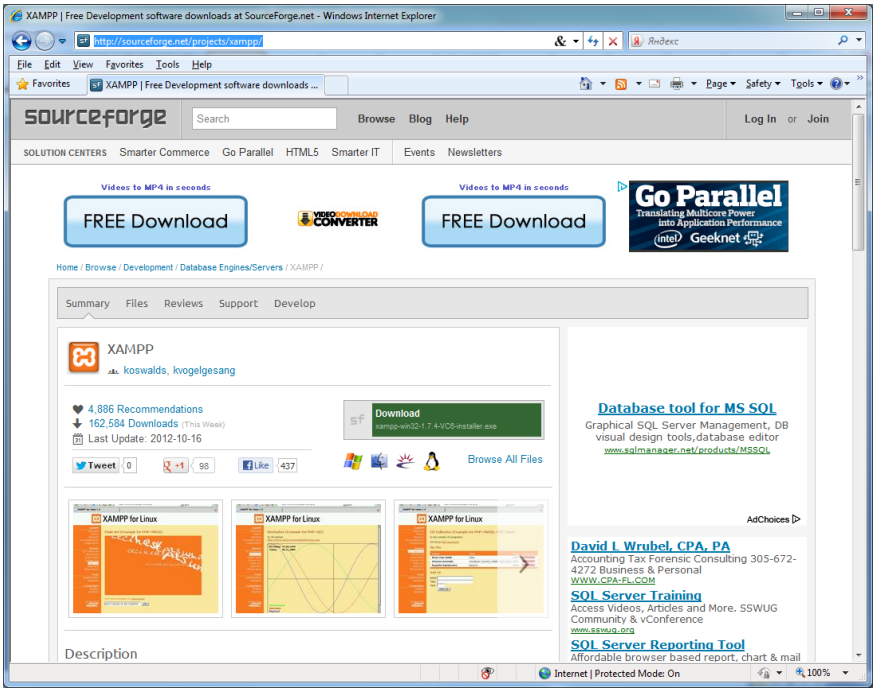
PHP-ის მუშაობისათვის განკუთვინილი ვებსერვერია Apache, მაგრამ უფრო მოხერხებული სერვერია XAMPP 1.7.4 Win32, XAMPP Windows 1.8.1.

ვებსერვერის გადმოწერა შემდეგი საიტებიდანაა შესაძლებელი:

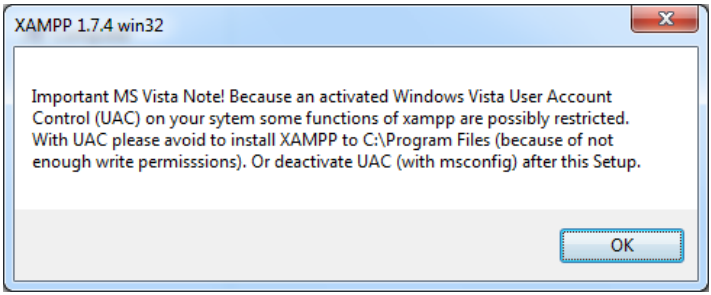
http://sourceforge.net/projects/xampp/

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

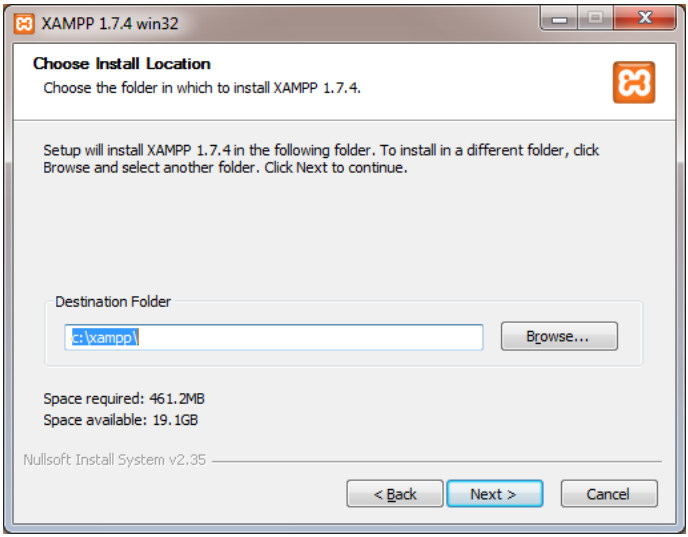
პირველ შემთხვევაში ეკრანზე გაიხსნება ფანჯარა, საიდანაც შესაძლებელია XAMPP 1.7.4 Win32 სერვერის გადმოწერა.



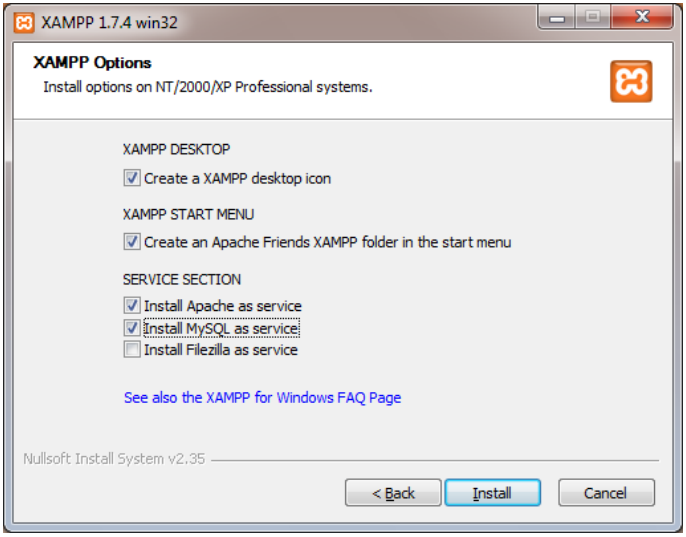
Download ღილაკზე თითის დაჭერით გაიხსნება ახალი ფანჯარა და რამდენიმე წამში დაიწყება ვებსერვერის გადმოწერა. გადმოწერის შემდეგ უნდა მოვახდინოთ მისი ინსტალაცია, რისთვისაც xampp-win32-1.7.4-VC6-installer ფაილი გამოიყენება. ამ ფაილის ნიშნაკზე ორჯერ დაწკაპუნებით გაიხსნება შემდეგი დიალოგური ფანჯარა:



OK ღილაკზე თითის დაჭერით საინსტალაციო დიალოგი დაიწყება:

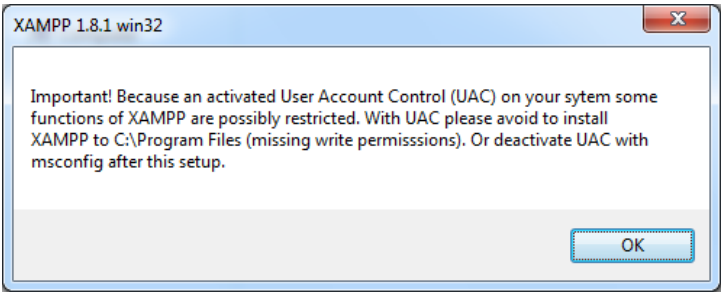


ინსტალაციის შემდეგ ეტაპზე გადასვლა Next ღილაკით ხორციელდება:

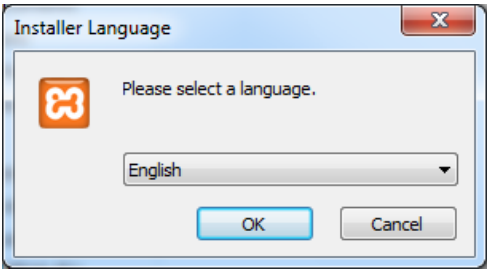


აქ კი SERVICE SECTION ველში სასურველი სერვისები უნდა მოვნიშნოთ და Install ღილაკის საშუალებით ინსტალაციის ბოლო ეტაპი ჩავრთოთ.

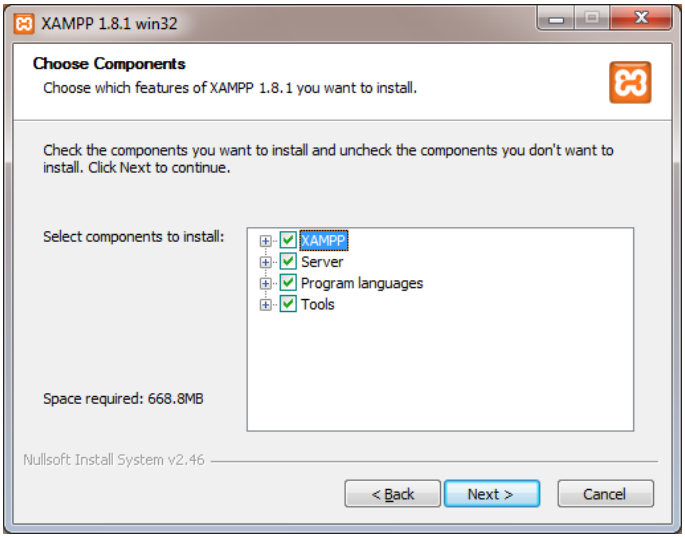
თითქმის ანალოგიურად მიმდინარეობს XAMPP Windows 1.8.1 სერვერის ინსტალაციაც, ოღონდ აქ უნდა გავხსნათ შესაბამისი ფაილი – xampp-win32-1.8.1-VC9-installer, სადაც პირველი დიალოგური ფანჯრის შემდეგ

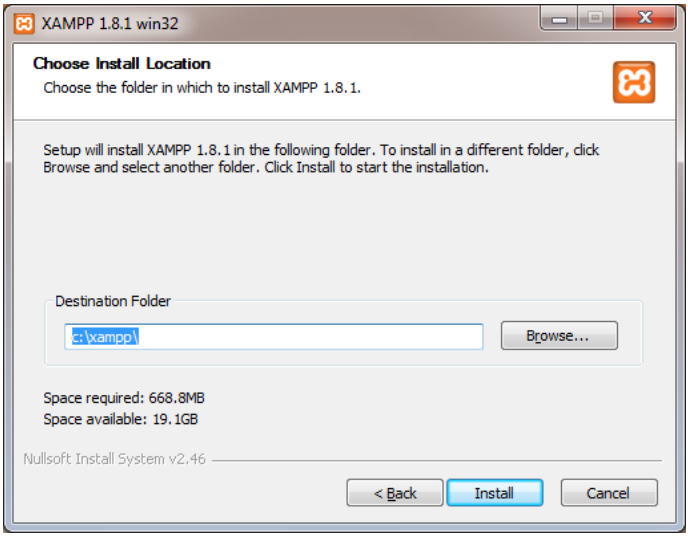


უნდა განისაზღვროს ინსტალაციის ენა:

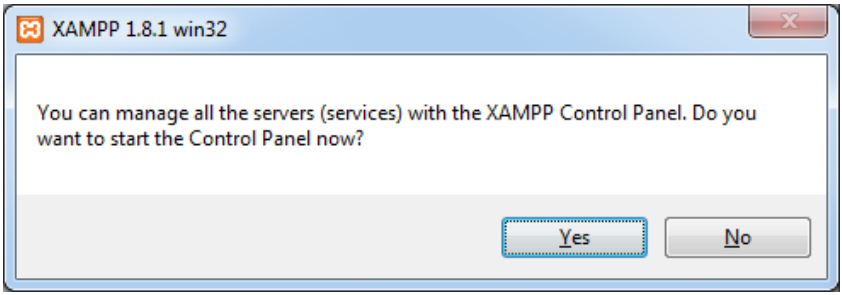


საინსტალაციო ენის შერჩევის შემდეგ დაიწყება სერვერის ინსტალაცია, გაიხსნება პირველი დიალოგური ფანჯარა და გამოვა განსხვავებული ფანჯრები:

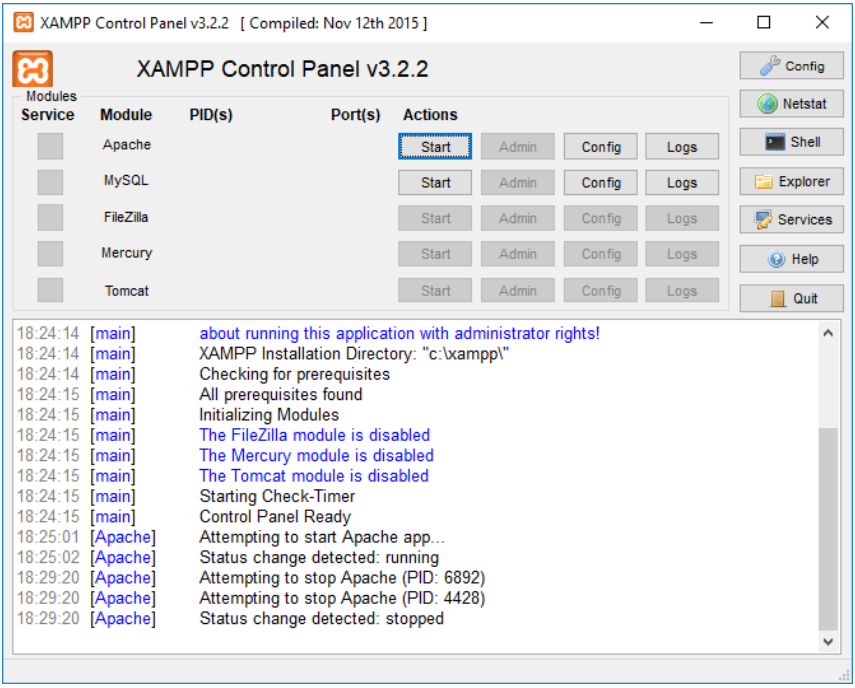




ინსტალაციის დასრულების შემდეგ ეკრანზე გამოჩნდება დიალოგური ფანჯარა, რომლის მიხედვითაც შესაძლებელია სერვერის Control Panel-ის დაყენებაც:



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



შესაბამისი რეჟიმების არჩევის შემდეგ სერვერის ინსტალაცია დამთავრდება.

XAMPP-ს სამუშაოდ მე-80 პორტი სჭირდება. TCP/IP[3] და UDP[4] ქსელებში პორტი არის ლოგიკური კავშირის დასრულების წერტილი, გზა, რომლითაც კლიენტი-პროგრამა განსაზღვრავს სერვერულ პროგრამას კომპიუტერის ქსელში. პორტის ნომერი განსაზღვრავს პორტის სახეობას. მაგალითად, „პორტი 80“ გამოიყენება HTTP[5] ტრანზიტისთვის. პორტის ნომრები შექმნილია IANA[6]-ის მიერ.

მე-80 პორტს ასევე იყენებს Skype, SQL Server და სხვა. თუ კომპიუტერში რომელიმე ნახსენები პროგრამაა დაინსტალირებული, XAMPP მე-80 პორტს ვეღარ გამოიყენებს და Apache სერვერი ვეღარ იმუშავებს, რის გამოსასწორებლადაც XAMPP პორტის ნომერი უნდა შევცვალოთ.

ამისათვის XAMPP-ის კონტროლ პანელში, Apache-ის გასწვრივ Config ღილაკზე მაუსის მარჯვენა კლავიშით გავხსნათ კონტექსტური მენიუ და ამ მენიუში ავირჩიოთ Apache (httpd.conf), როგორც ეს სურათზეა ნაჩვენები:

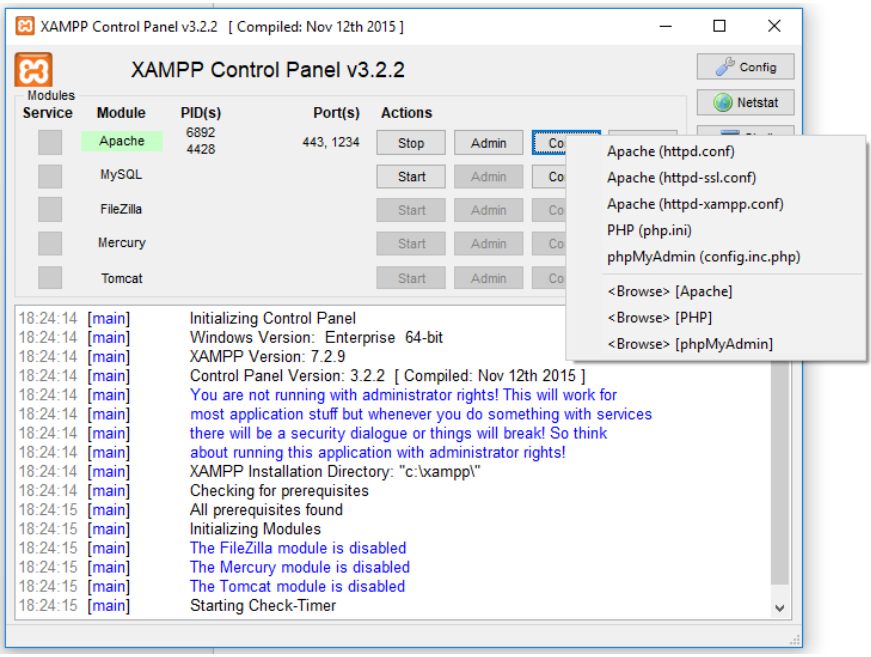
---

[3] TCP/IP – **T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol – კომუნიკაციური პროტოკოლების ნაკრები, რომლებიც ინტერნეტში ჰოსტების დაკავშირებისთვის გამოიყენება.

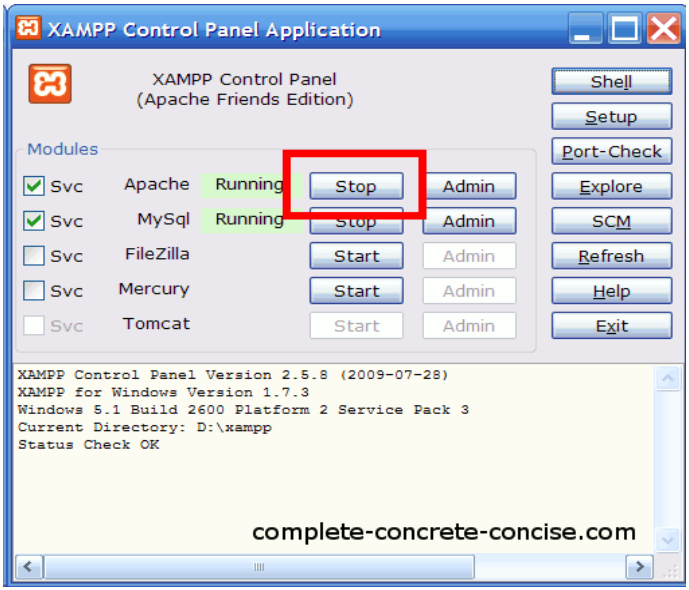
[4] UDP – **U**ser **D**atagram **P**rotocol – IP ქსელის მეშვეობით ინფორმაციის გასაგზავნად და მისაღებად იყენებს პირდაპირ გზას.

[5] HTTP – **H**yper **T**ext **T**ransfer **P**rotocol – განსაზღვრავს თუ, როგორი სახითაა შეტყობინებები დაფორმატებული და გადაგზავნილი.

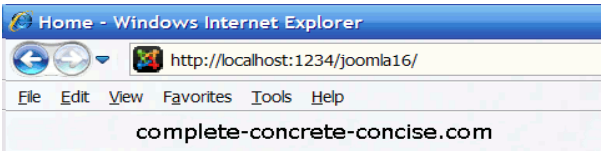
[6] IANA – **I**nternet **A**ssigned **N**umbers **A**uthority – ორგანიზაცია, რომელიც ინტერნეტ არქიტექტურაში IP მისამართების კავშირს განსაზღვრავს.



გამოვა ტექსტური დოკუმენტის ფანჯარა და მასში <Ctrl>+<F> კლავიშთა კომბინაციით მოვძებნოთ, სიტყვა Listen 80, რიცხვი 80 ჩავანაცვლოთ რიცხვით, თუნდაც შეგვიძლია ავიღოთ რიცხვი 1234, შევინახოთ ფაილი და მოვახდინოთ გადატვირთვა (რესტარტი) Apache-ში, რათა ცვლილებამ მოახდინოს ეფექტი. ეს მოხდება XAMPP-ის Control Panel-ის Stop ღილაკზე დაჭერით და შემდეგ ისევ Start ღილაკზე დაჭერით.



როდესაც ხელახალ დაკავშირებას მოვახდენთ Appache სერვერთან, ჩვენ უნდა ავკრიფოთ პორტის ნომერი URL ველში, მაგალითად, ვკრეფთ შემდეგნაირად – localhost:1234 (ან იმ რიცხვს, რომელსაც შევარჩევთ პორტის ნომრად):



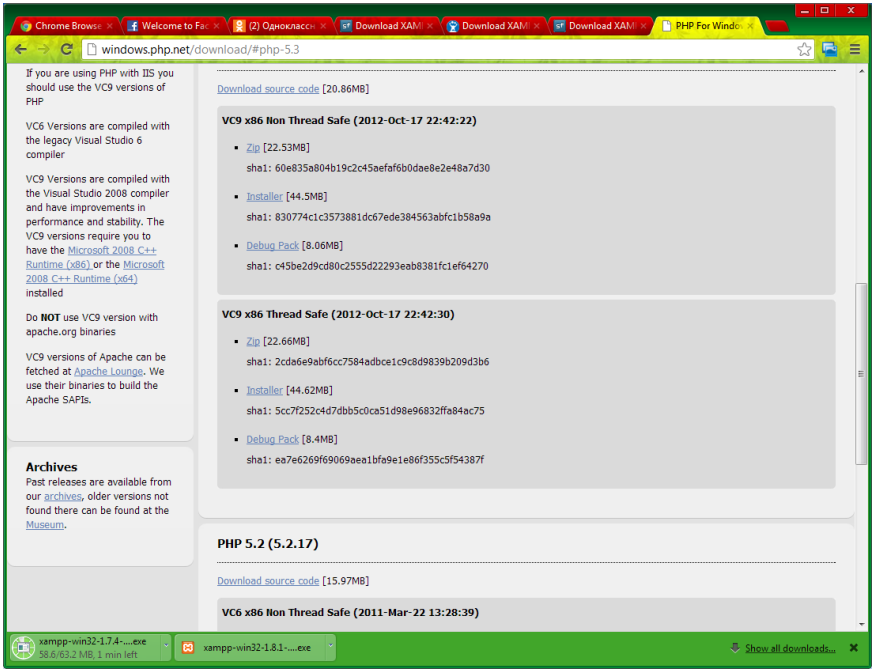
PHP 5-ის ჩატვირთვა და ინსტალაცია

ვებსერვერის დაყენებისა და გაშვების შემდეგ PHP 5-ის დაყენებაც შეიძლება. PHP 5-ის ყველაზე ბოლო ვერსიის მიღება შესაძლებელია www.php.net ვებსაიტის „Downloads“ გვერდზე გადასვლით, რომელიც ქვემოთ სურათზეა მოცემული (მისი სახე

დროის მიხედვით შეიძლება შეიცვალოს). მასში ავარჩიოთ ჩვენთვის მისაღები ვერსია და გადმოვწეროთ კომპიუტერში.

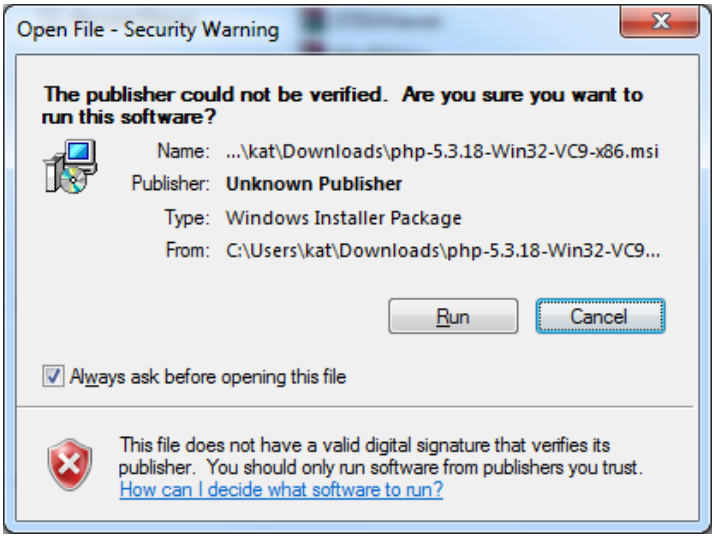
თუ გადმოვწერთ დაარქივებულ (zip-კოდი) ფაილს ანუ Zip რეჟიმს ავირჩევთ, მაშინ ახალი ფოლდერი (საქაღალდე) უნდა შევქმნათ, სადაც ამ ფაილს შევინახავთ და დავაარქივებთ. ამის შემდეგ საინსტალაციო პროგრამას – php-5.3.17-nts-Win32-VC9-x86 გავუშვებთ შესასრულებლად.

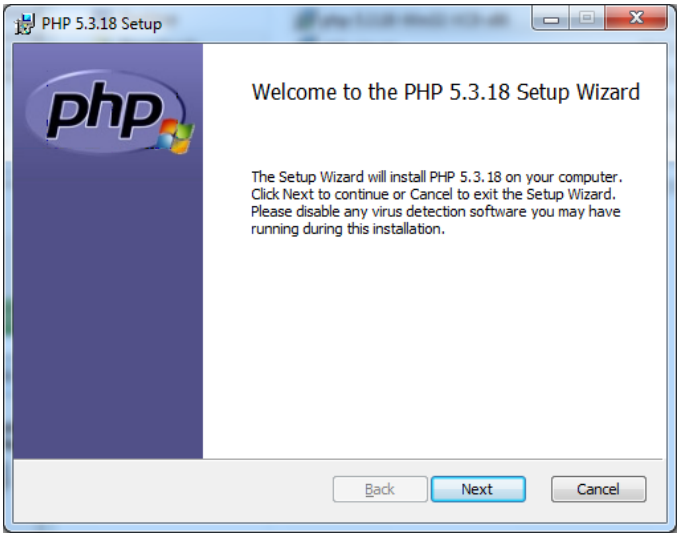
თუ Installer რეჟიმს ავირჩევთ, მაშინ კომპიუტერში საინსტალაციო პროგრამა გადმოიწერება და შესაძლებელი იქნება მისი პირდაპირ შესასრულებლად გაშვება.

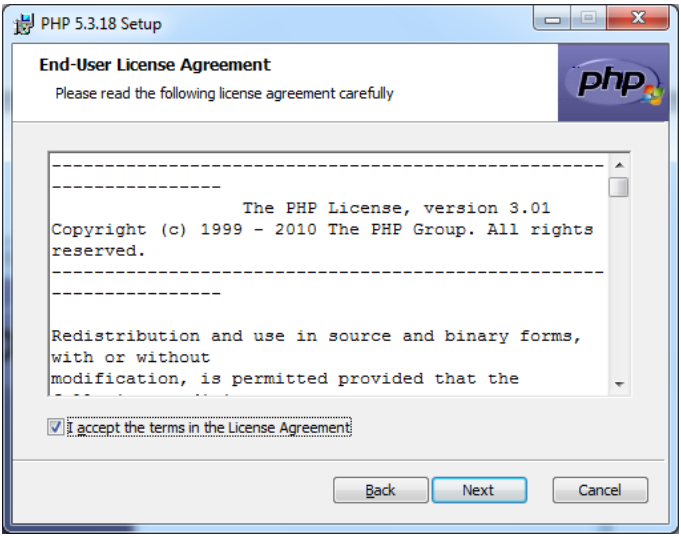


პროგრამის გაშვების შემდეგ ეკრანზე გამოვა Open File – Security Warning დიალოგური ფანჯარა, სადაც Run ღილაკზე

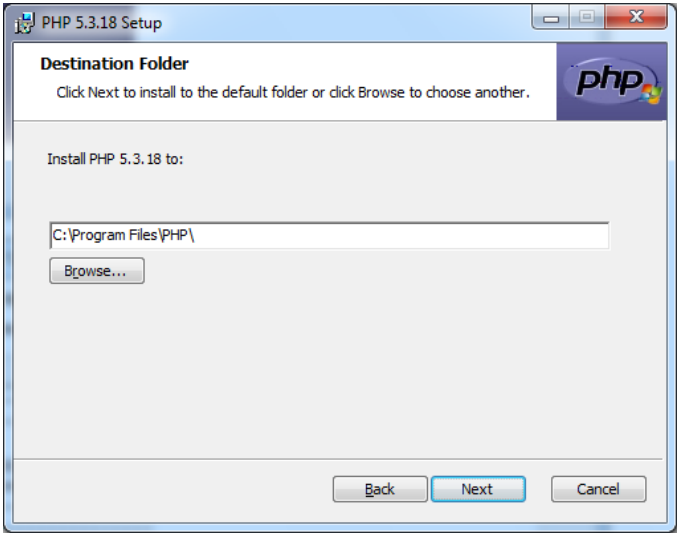
თითის დაჭერით პროგრამის ინსტალაცია დაიწყება, ხოლო ყოველ შემდეგ ნაბიჯზე გადასასვლელად Next ღილაკი უნდა გამოვიყენოთ:

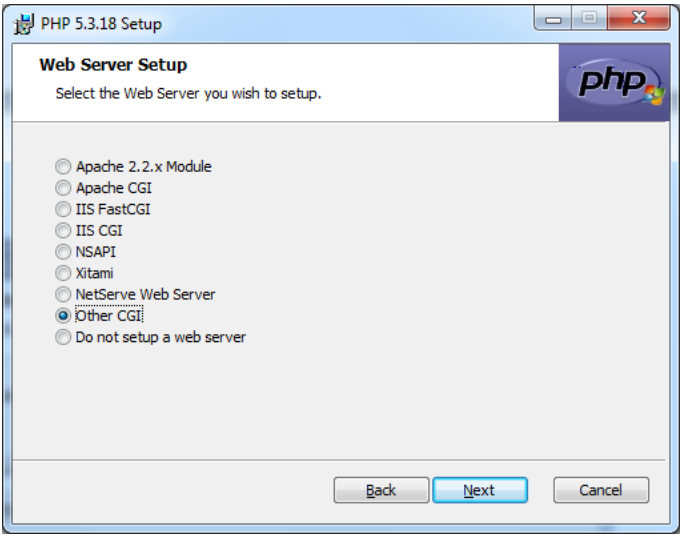






ინსტალაციის შემდეგ ეტაპზე გადასასვლელად I accept the terms in the License Agreement ველი ალმით უნდა მოინიშნოს.

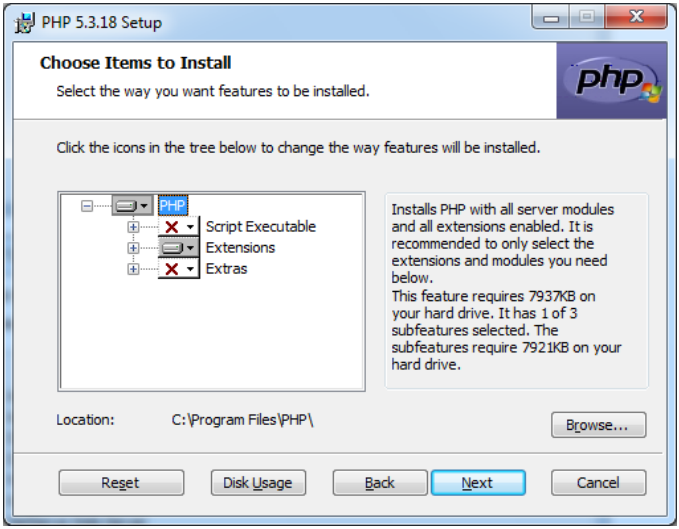


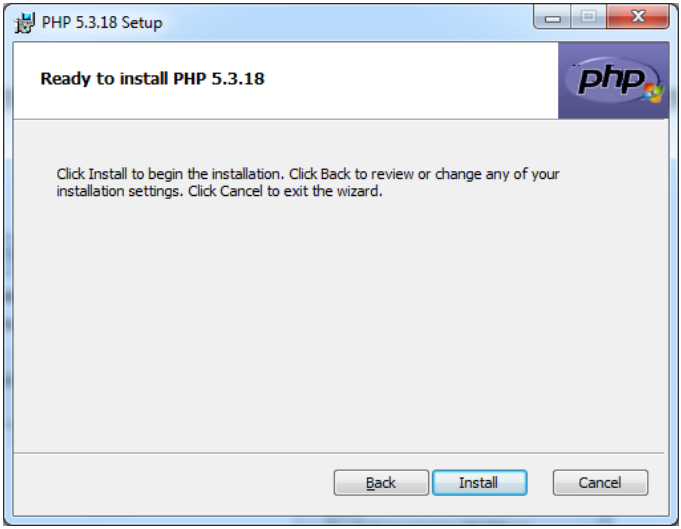


PHP 5-ის ინსტალაციის დროს აუცილებელია ვებსერვერზე გადაწყდეს, სახელდობრ, როგორ იმუშავებს PHP: ან როგორც CGI-პროგრამა, ან როგორც ცალკეული სტატიკური ან დინამიკური მოდული. CGI (Common Gateway Interface – „შლუზის (კარიბჭე)“ გარე პროგრამის საერთო ინტერფეისი) ინტერფეისის სტანდარტი ვებსერვერთან კავშირისათვის.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

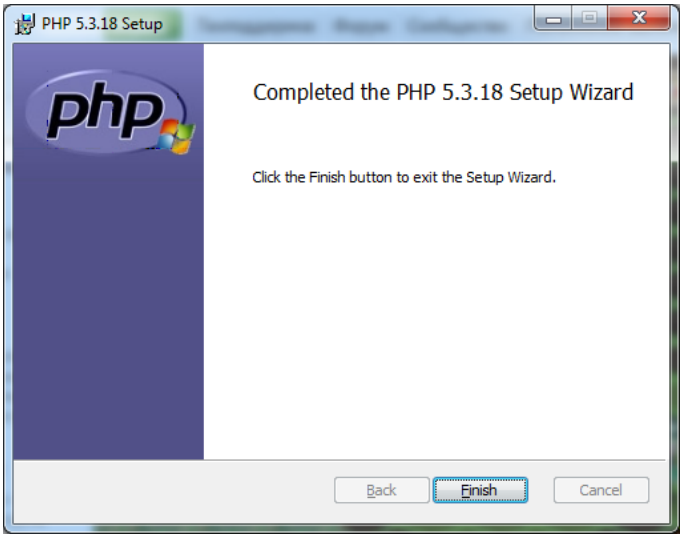
წარმოდგენილი ვებგვერდი, ბრაუზერისათვის გადაგზავნის წინ PHP-ინტერპრეტატორის მიერ დამუშავდეს.

ამ თემის შესახებ დიდი მოცულობის დოკუმენტაცია შეიძლება PHP-ის საიტზე (www.php.net) მოვიძიოთ.





Install ღილაკის 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 ვებსერვერი 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 იგი ჯერ სერვერზე უნდა დავაყენოთ. ამისათვის ჩაწერილი PHP-სკრიპტი start.php სახელით შევინახოთ. შემდეგ კი დავაკოპიროთ 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 ვებსერვერზე და მის მომართვაზეა დამოკიდებული. ამის შემდეგ თქვენი ბრაუზერის სამისამართო სტრიქონში აკრიფეთ მისამართი http://localhost/start.php და თუ ყველაფერი სწორადაა დაყენებული და მომართული, დაინახავთ ტექსტს Hello, World!

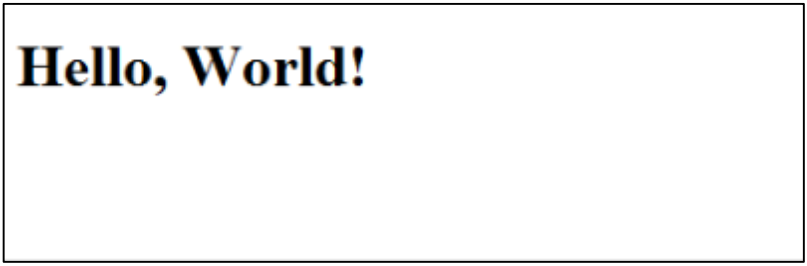
გავარჩიოთ ჩვენი მარტივი სკრიპტის კოდი:

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!

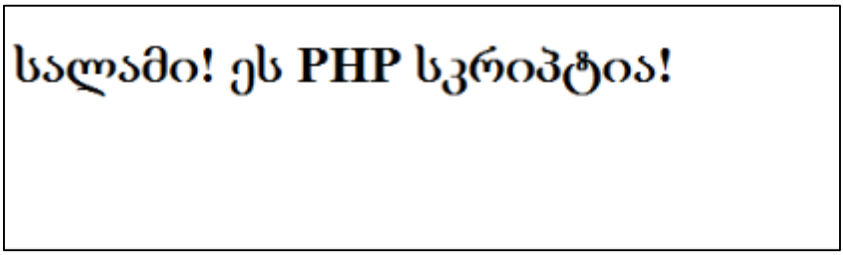


ახლა დავწეროთ სკრიპტი ისე, რომ მან შეძლოს ტექსტის ქართულად გამოტანა. ამისათვის, საჭიროა სკრიპტის აკრეფის შემდეგ მისი შენახვის დროს Notepad ტექსტურ რედაქტორს მივცეთ Save ან Save As ბრძანება ან გამოვიყენოთ <Ctrl>+<S> კლავიშთა კომბინაცია. გახსნილ დიალოგური ფანჯრის File name ველში ჩავწეროთ მოცემული კოდის 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>.

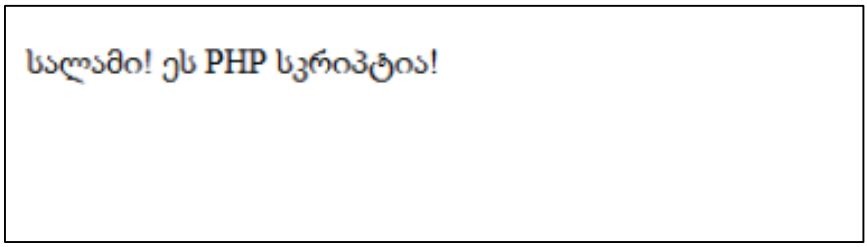


როგორც ზემოთ განხილულ მაგალითში იყო, აქაც მთელი ინფორმაცია, html-კოდების ჩათვლით, echo ოპერატორის საშუალებით გამოგვყავდა. ზოგადად, PHP-სკრიპტები შეიძლება გამოტანილი იყოს არა echo ოპერატორის საშუალებით, არამედ PHP-კოდები შეიძლება უშუალოდ იყოს ჩასმული html-დანართში. განვიხილოთ მაგალითი:

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

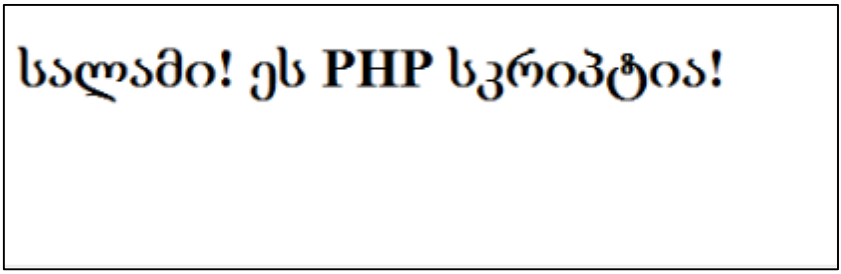
ახლა ჩვენ შეგვიძლია მოცემული PHP-სკრიპტი .php ფაილის სახით სერვერზე შევინახოთ, მაგალითად, mag3.php სახელით და შედეგები შევამოწმოთ.

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



თუ იგივე სკრიპტში გამოსატანი ტექსტი გვინდა გამოვიტანოთ როგორც სათაური, მაშინ მას უნდა ჩავუმატოთ შესაბამისი ტეგები, მაგალითად:

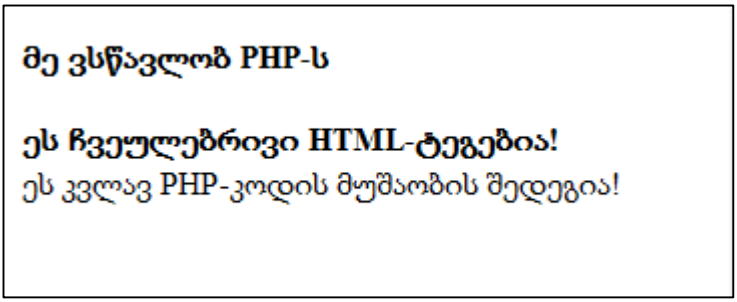
```
<html>
   <head>
      <title>ჩემი პირველი PHP სკრიპტი</title>
   </head>
   <body>
      <h1><?php
      echo "სალამი! ეს PHP სკრიპტია!";
      ?></h1>
   </body>
</html>
```



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>
```



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

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

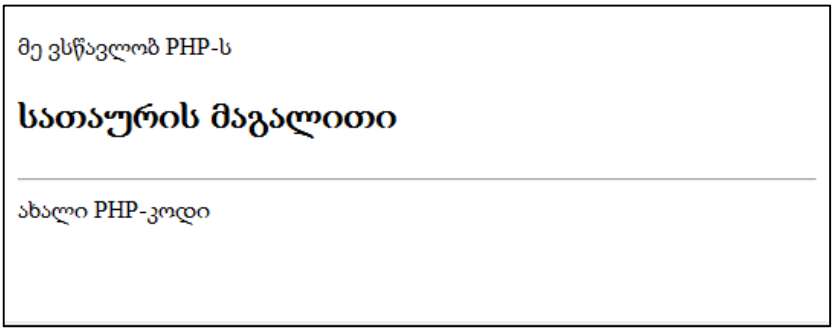
1. პირველი სტრატეგიით PHP არის HTML-ის დანამატი და, შესაბამისად, PHP-სკრიპტი HTML-დოკუმენტში ჩადგმულ ფრაგმენტს წარმოადგენს;
2. მეორე სტრატეგიით კი HTML არის PHP-ის დანამატი და HTML-ტეგი PHP-სკრიპტის მიერ გამოტანილი შეტყობინების დაფორმატების საშუალებას წარმოადგენს.

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



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.

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.

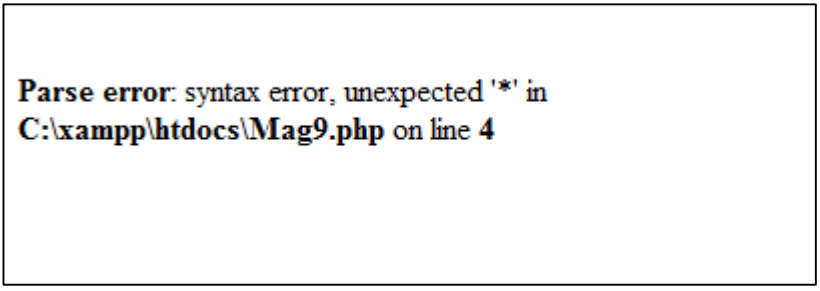
ერთსტრიქონიანი კომენტარი სტრიქონის ან PHP-კოდის მომდევნო ბლოკის ბოლომდე გრძელდება.

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

ყურადღებით უნდა იყოთ, რათა 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.



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:

PHP Version 5.3.8

System	Windows NT IRAKLI2-PC 6.1 build 7601 (Windows 7 Ultimate Edition Service Pack 1) i586
Build Date	Aug 23 2011 11:47:20
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /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"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626,TS,VC9
PHP Extension Build	API20090626,TS,VC9
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
Registered PHP	php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, phar

ცვლადები PHP-ში

დაპროგრამების თითქმის 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 წავშალოთ, 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). 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 გვჭირდება რიცხვის ტექსტურ ინფორმაციად დამუშავება, 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.

```
<?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.

ცვლადის არსებობის შემოწმება

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

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. 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 " ".

გამოსახულება PHP-ში

გამოსახულება PHP-ის 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 = funct() ჩანაწერი $x = 5 ჩანაწერის იდენტურია. ფუნქცია არის გამოსახულება, რომელსაც გააჩნია ის მნიშვნელობა, რომელსაც ფუნქცია აბრუნებს. ვინაიდან funct() აბრუნებს 5-ს, ამიტომ გამოსახულება 'funct()' წარმოადგენს 5-ს. როგორც წესი, ფუნქცია აბრუნებს არა სტატიკურ მნიშვნელობას, არამედ გამოთვლით მნიშვნელობას.

მინიჭების ოპერატორები

მინიჭების საბაზო ოპერატორი აღინიშნება (=)-ით. 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 ერთსა და იმავე მონაცემზე მიუთითებს და არავითარი კოპირება არ ხდება.

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, მცოცავმძიმიანი და სტრიქონული ტიპის) შეიძლება შეიცავდეს.

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

define ($name, $value, $case_sen), სადაც:
$name – კონსტანტის სახელი;
$value – კონსტანტის მნიშვნელობა;
$case_sen – 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_sen 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 დამოკიდებული. ქვემოთ მოყვანილია მათი სია.

სახელი	აღწერა
__LINE__	ფაილში მიმდინარე სტრიქონი
__FILE__	მიმდინარე ფაილის სრული გზა და სახელი
__FUNCTION__	ფუნქციის სახელი
__CLASS__	კლასის სახელი
__METHOD__	კლასის მეთოდის სახელი

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). მათგან ოთხი სკალარული ტიპისაა:

- boolean (ორობითი მონაცემები);
- integer (მთელი რიცხვები);
- float (მცოცავმძიმიანი 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 ტიპი);
- number (რიცხვითი);
- callback (უკუგამოძახების).

ოპერატორი ეწოდება კონსტრუქციას, რომელიც ერთი ან რამდენიმე მნიშვნელობისაგან შედგება და რომლის გამოთვლაც შეიძლება როგორც ახალი მნიშვნელობა (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.

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 – მივმართოთ, როგორც ბლოკს.

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

ფუნქციის სახელი (პარამეტრი1, . . .)

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

PHP-ის ფუნქციების შესახებ ინფორმაცია შეიძლება მოიპოვოთ 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 მონაცემთა ბაზების განხილვის შემდეგ გავეცნობით 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.

რიცხვითი მონაცემები

ტიპი Boolean (ორობითი მონაცემები)

ეს არის უმარტივესი ტიპი. იგი გამოსახულების ჭეშმარიტებას გამოხატავს და შეიძლება იყოს ან TRUE ან FALSE.

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

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

ჩვეულებრივ, ვიყენებთ რაიმე ოპერატორს, 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) სახით გამოიყენება. მაგრამ უმეტეს შემთხვევაში ტიპის მოცემის გამოყენება არ არის აუცილებელი, ვინაიდან თუ ოპერატორს, ფუნქციას ან მმართველ კონსტრუქციას ბულის არგუმენტი ესაჭიროება, მნიშვნელობა ავტომატურად გარდაიქმნება.

ლოგიკურ ტიპად გარდაქმნის დროს შემდეგი მნიშვნელობები განიხილება როგორც FALSE:

- თვით ბულის FALSE;
- მთელი 0 (ნული);
- მცოცავმძიმიანი რიცხვი 0.0 (ნული);
- ცარიელი სტრიქონი და სტრიქონი "0";
- ნულელემენტიანი მასივი;
- ნულცვლადიანწევრებიანი 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 = {..., -2, -1, 0, 1, 2, ...}, ჩვეულებრივ, იგი 32 ბიტი სიგრძის (–2 147 483 648-დან 2 147 483 647-მდე) რიცხვია.

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 (მცოცავმძიმიანი რიცხვი)

Double – ძალიან დიდი სიზუსტის ნამდვილი რიცხვია (ამ რიცხვების გამოყენება შეიძლება მათემატიკური გამოთვლების უმეტესობაში).

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

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

არითმეტიკული ოპერატორები

სკოლის არითმეტიკის საფუძვლები ალბათ, ყველას ახსოვს. 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, თავდაპირველად $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>";
?>
```

პოსტფიქსული იწკრემენტი

a++ უნდა იყოს 5: 5
a უნდა იყოს 6: 6

პრეფიქსული იწკრემენტი

++a უნდა იყოს 6: 6
a უნდა იყოს 6: 6

პოსტფიქსული 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.

ბიტური ოპერატორები

მოცემული ოპერატორები განკუთვნილია მთელი ცვლადებისათვის ბიტური ჯგუფების მისანიჭებლად ან მოსახსნელად, რადგანაც ნებისმიერი რიცხვი არის ბიტების მიმდევრობა. მთელი რიცხვები PHP-ში 32-ბიტიანია. 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 | $b	ბიტური 'ან'	იმ ბიტების მინიჭება მოხდება, რომელიც მინიჭებულია ან $a-ში ან $b-ში.
$a ^ $b	'ან'-ის უარყოფა	იმ ბიტების მინიჭება მოხდება, რომელიც მინიჭებულია ან მხოლოდ $a-ში ან მხოლოდ $b-ში.
~ $a	უარყოფა	იმ ბიტების მინიჭება მოხდება, რომელიც $a-ში მინიჭებული არ არის და პირიქით.
$a << $b	მარცხნივ წანაცვლება	$a ცვლადის ყველა ბიტი წაინაცვლებს $b პოზიციით მარცხნივ (ყოველი პოზიცია 2-ზე გამრავლებას გულისხმობს)
$a >> $b	მარჯვნივ წანაცვლება	$a ცვლადის ყველა ბიტი წაინაცვლებს $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	იგივურად ტოლია	**TRUE** თუ $a ტოლია $b-სი და იმავე ტიპისაა
$a != $b	არ უდრის	**TRUE** თუ $a არ უდრის $b-ს.
$a <> $b	არ უდრის	**TRUE** თუ $a არ უდრის $b-ს.
$a !== $b	იგივურად არ არის ტოლი	**TRUE** თუ $a არ უდრის $b-ს ან თუ ისინი სხვადასხვა ტიპისაა.
$a < $b	ნაკლებია	**TRUE** თუ $a მკაცრად ნაკლებია $b-ზე.
$a > $b	მეტია	**TRUE** თუ $a მკაცრად მეტია $b-ზე.
$a <= $b	ნაკლებია ან ტოლია	**TRUE** თუ $a ნაკლებია ან ტოლი $b-ზე.
$a >= $b	მეტია ან ტოლია	**TRUE** თუ $a მეტია ან ტოლი $b-ზე.

ლოგიკური ოპერატორები

ქვემოთ მოყვანილია PHP-ს ლოგიკური ოპერატორების ცხრილი:

მაგალითი	დასახელება	შედეგი
$a and $b	ლოგიკური 'და'	**TRUE** თუ $a-ც და $b-ც არის **TRUE**.
$a or $b	ლოგიკური 'ან'	**TRUE** თუ ან $a, ან $b არის**TRUE**.

$a xor $b	'ან'-ის უარყოფა	**TRUE** თუ $a, ან $b არის **TRUE**, მაგრამ ორივე ერთად არა.
! $a	უარყოფა	**TRUE** თუ $a არ არის **TRUE**.
$a && $b	ლოგიკური 'და'	**TRUE** თუ $a-ც და $b-ც **TRUE**.
$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:

პრიორი-ტეტი	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 გეეჭვებათ ან გეშინიათ რომ შეცდეთ, უმჯობესია გამოიყენოთ ფრჩხილები.

PHP-ის მათემატიკური ფუნქციები

მათემატიკური კონსტანტების ცხრილი

კონსტანტა	მნიშვნელობა	აღწერა
M_PI	3.14159265358979323846	რიცხვი **π**
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	π/2
M_PI_4	0.78539816339744830962	π/4
M_1_PI	0.31830988618379067154	1/π
M_2_PI	0.63661977236758134308	2/π
M_SQRTPI	1.77245385090551602729	sqrt(π)
M_2_SQRTPI	1.12837916709551257390	2/sqrt(π)
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 π
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 არკტანგესს რადიანებით;
atan – არკტანგესი;
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 (ნატუალური ლოგარითმის ფუძე);
expm1 – აბრუნებს exp(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 + 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 π;
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 მონაცემები

ტიპი string (სტრიქონი)

სტრიქონი PHP-ში არის სიმბოლოების ნებისმიერი სიგრძის ნაკრები. 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 მეხსიერებითაა 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 ფორმაა მისი მოთავსება აპოსტროფებში (სიმბოლო ').

თუ სტრიქონის შიგნით საჭიროა აპოსტროფების გამოყენება, მაშინ 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.

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.

მონაცემთა გაერთიანება (კონკატენაცია)

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 გადაებას ერთმანეთს. მაგალითად,

```php
<?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.

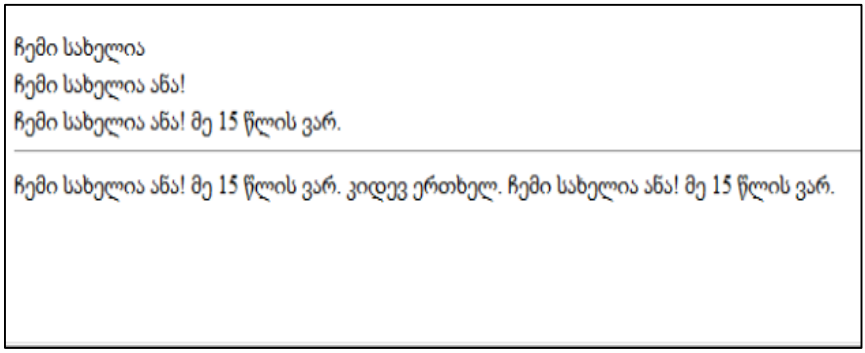
ზემოთ განხილული შესაძლებლობის გარდა, გაერთიანების ოპერაცია შეიძლება ორი სტრიქონის გასაერთიანებლად გამოვიყენოთ ისე, რომ შედეგი სხვა ცვლადს მივანიჭოთ. მაგალითად,

```php
<?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";
?>
```



სტრიქონის დამუშავება

თუ სტრიქონი ბრჭყალების 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 განსაზღვროთ, 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;
```

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:

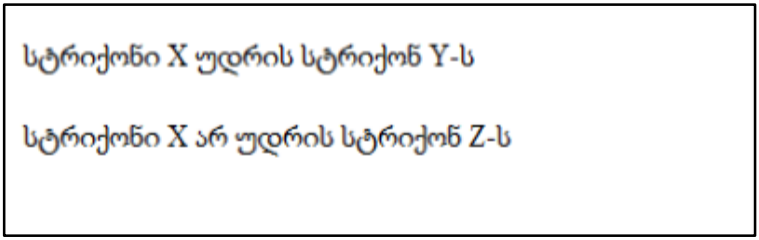
```
<?php
$x="სტრიქონი";
```

```
$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:



სტრიქონებთან სამუშაო ფუნქციები

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

ბაზური სტრიქონული ფუნქცია

ფუნქცია strlen(string $st)

მოცემული ფუნქცია 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 სტრიქონში
არის 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
```

ტექსტის ბლოკებთან სამუშაო ფუნქციები

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

ფუნქცია substr_replace(string $str1, string $str2, int $start [,int $length])

$str1 სტრიქონში $start პოზიციიდან $length რაოდენობის სიმბოლოს $str2 ქვესტრიქონით შეცვლის და შესაბამის შედეგს დააბრუნებს.

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
 ტეგით:

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

როგორც ვხედავთ, ის, რომ
\n სტრიქონში თუ კვლავ არის სტრიქონის გადაყვანის სიმბოლო, ფუნქციის მუშაობაზე

არავითარ გავლენას არ ახდენს, ანუ ფუნქცია სტრიქონში მხოლოდ ერთჯერად გაივლის. აღწერილი ამოცანის გადასაწყვეტადაც nl2br() ფუნქცია გამოიყენება, რომელიც ცოტათი უფრო სწრაფად მუშაობს.

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 კიდევ 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
```

ფუნქცია htmlentities(string, flags, character-set, double_encode)

მოცემული ფუნქცია სტრიქონებთან მომუშავე კიდევ ერთი გამოსადეგი ფუნქციაა. string – აუცილებელი პარამეტრია და გარდასაქმნელ სტრიქონს განსაზღვრავს. flags – არააუცილებელი პარამეტრია, რომელიც ფუნქციას უთითებს, თუ როგორ უნდა დაამუშაოს ბრჭყალები, არასწორი კოდირება და გამოყენებული დოკუმენტის ტიპი. მისი მნიშვნელობებია:

ENT_COMPAT – ჩუმათობით მნიშვნელობა. მხოლოდ ორმაგი ბრჭყალების კოდირებას ახდენს;

ENT_QUOTES – ორმაგი და ერთმაგი ბრჭყალების კოდირებას ახდენს;

ENT_NOQUOTES – არ ახდენს ბრჭყალების კოდირებას.

character-set – არააუცილებელი პარამეტრი, სტრიქონი, რომელიც მიუთითებს თუ სიმბოლოების რომელი ნაკრები უნდა იქნეს გამოყენებული. double_encode – არააუცილებელი პარამეტრი, რომელიც მიუთითებს უნდა მოხდეს თუ არა არსებული HTML-ობიექტის კოდირება.

TRUE – ჩუმათობით მნიშვნელობა. გარდაქმნის ყველაფერს;

FALSE – არ მოახდენს არსებული HTML-ობიექტის კოდირებას.

ეს ფუნქცია გარდაქმნილ სტრიქონს აბრუნებს. თუ სტრიქონი დაუშვებელ კოდირებას შეიცავს, მაშინ ცარიელ სტრიქონს დააბრუნებს.

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, რომელიც 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. ვინაიდან ეს ფუნქცია ძალიან სწრაფად მუშაობს, მისი გამოყენება შეიძლება ყველგან, სადაც შეცდომითი ჰარის არსებობის სულ მცირე ეჭვი მაინც გვაქვს.

ფუნქცია ltrim(string $st)

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

ფუნქცია chop(string $st)

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

რეგისტრის შესაცვლელი ფუნქცია

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

ფუნქცია strtolower(string $str)

მოცემული ფუნქცია 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 strtr() 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 გარდავქმნათ, რომლითაც ინახება დოკუმენტები სერვერზე.

ფუნქცია strtoupper(string $str)

ამ ფუნქციას სტრიქონი ზედა რეგისტრში გადაჰყავს.

გამოტანის ბუფერის გასუფთავების ფუნქცია

ფუნქცია flush()

ამ ფუნქციის მუშაობას სტრიქონთან პირდაპირი კავშირი არა აქვს, მაგრამ სხვა ფუნქციებიდანაც შორს დგას.

echo-ს გამოყენების დროს მონაცემები კლიენტს პირდაპირ არასდროს 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, settype() ფუნქციის გამოყენება. მისი სინტაქსია:

ფუნქცია settype (mixed var, string type)

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

- "boolean" (ან "bool");
- "integer" (ან "int");
- "float" (ან "double");
- "string";
- "array";
- "object";
- "null".

წარმატების შემთხვევაში შედეგად აბრუნებს TRUE-ს; წინააღმდეგ შემთხვევაში – FALSE-ს. 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 მცოცავმძიმიანი რიცხვი (float) სტრიქონად გარდაიქმნება ისე, რომ სტრიქონი მოცემულ რიცხვში გამოყენებული ციფრებისაგან (მცოცავმძიმიანი 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 გამოიყენოთ.

NULL ყოველთვის ცარიელ სტრიქონად გარდაიქმნება.

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

PHP-ის მნიშვნელობების გარდაქმნა შესაძლებელია მისი მუდმივად შენახვის მიზნითაც. ამ მეთოდს სერიალიზაცია ეწოდება და შეიძლება serialize() ფუნქციის მეშვეობით შესრულდეს.

მონაცემთა ტიპის მინიჭება და ინდიკაცია

ფუნქცია settype (mixed var, string type)

ზემოთ უკვე აღვნიშნეთ, რომ PHP-ში ცვლადის ტიპის წინასწარ აღწერა არ არის აუცილებელი. ცვლადზე მინიჭებული მნიშვნელობის მიხედვით ინტერპრეტატორი მის ტიპს თავად განსაზღვრავს, მაგრამ ზოგჯერ საჭიროა ცვლადის ტიპის

ცხადად განსაზღვრა. 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 გვაინტერესებს. 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:

$foo=11.5; ტიპი: double
$foo=-1299; ტიპი: double
$foo=1; ტიპი: integer
$foo=1; ტიპი: integer
$foo=11; ტიპი: integer
$foo=14.2; ტიპი: double
$foo=11; ტიპი: double
$foo=11; ტიპი: double
$foo=Text; ტიპი: string
$foo=1; ტიპი: boolean

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. ასოცირებული მასივი – ასეთ მასივს ასოცირებული ბუნება აქვს და ელემენტზე მიმართვისათვის გამოიყენება ე. წ. გასაღები, რომელიც მის მნიშვნელობასთან ლოგიკურადაა დაკავშირებული.

PHP-ის მნიშვნელოვანი თავისებურება არის ის, რომ PHP, სხვა ენებისაგან განსხვავებით, უშუალოდ 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 ნულით 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 მაგივრად 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:

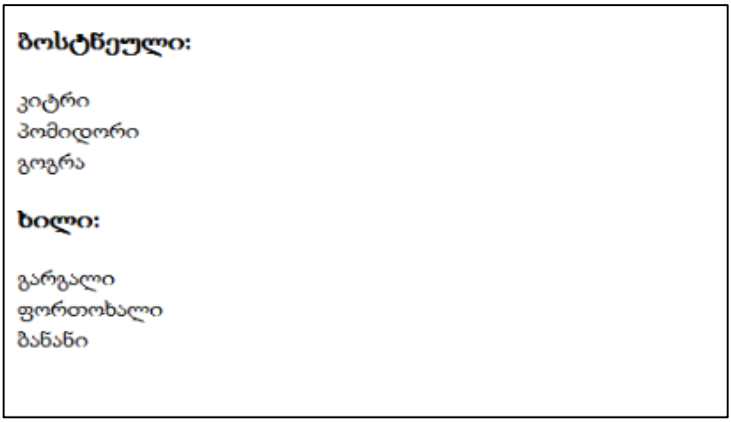
$სახელი[ინდექსი1][ინდექსი2]...[ინდექსიN];

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:



ასოცირებული 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, ასოცირებული მასივი ეწოდება. ასოცირებული მასივის ინდექსს გასაღები ეწოდება.

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

ერთგანზომილებიანი ასოცირებული მასივი

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

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

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

```
echo $names["ბერიძე"];
```

მრავალგანზომილებიანი ასოცირებული მასივი

მრავალგანზომილებიანი ასოცირებული მასივი შეიძლება შეიცავდეს რამდენიმე გასაღებს (ელემენტს), რომელთაგან თითოეული ასოცირებული მასივის კონკრეტულ ინდექსს

შეესაბამება. მოვიყვანოთ მრავალგანზომილებიანი ასოცირებული მასივის მაგალითი:

```
<?php
// მრავალგანზომილებიანი ასოცირებული 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 ასოცირებული მასივის ელემენტზე მიმართვა შემდეგნაირად ხდება:

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

მრავალგანზომილებიანი ასოცირებული მასივის შესაქმნელად სპეციალური array ფუნქცია გამოვიყენეთ.

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

```
<?php
// მრავალგანზომილებიანი ასოცირებული 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 ასოცირებული 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?>
```

PHP-ს მასივებთან სამუშაო ოპერატორები

მოვიყვანოთ PHP-ს მასივებთან სამუშაო ოპერატორების სია:

მაგალითი	დასახელება	შედეგი
$a + $b	გაერთიანება	$a მასივისა და $b მასივის გაერთიანება.
$a == $b	ტოლია	**TRUE** იმ შემთხვევაში, თუ $a და $b მასივი ერთსა და იმავე ელემენტებს შეიცავს.

$a === $b	იგივურად ტოლია	**TRUE** იმ შემთხვევაში, თუ $a და $b მასივი ერთსა და იმავე ელემენტებს ერთი და იმავე მიმდევრობით შეიცავს.
$a != $b	არ უდრის	**TRUE** თუ $a მასივი არ უდრის $b მასივს.
$a <> $b	არ უდრის	**TRUE** თუ $a მასივი არ უდრის $b მასივს.
$a !== $b	იგივურად არ უდრის	**TRUE** თუ $a მასივი იგივურად არ უდრის $b 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)
?>
```

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();

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

// სამელემენტიან ასოციურ მასივს ქმნის:
$arr3 = array("ბერიძე"=>"ზაზა", "კახაძე"=>"ნიკა", "მელაძე"=>"ლევანი");

// 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");
?>
```

ოპერაციები მასივებზე

მასივის ელემენტების მოწესრიგება (დახარისხება)

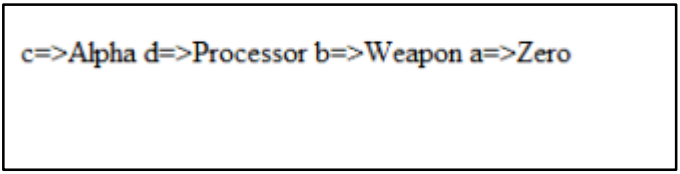
დავიწყოთ ყველაზე მარტივით – მასივის მოწესრიგებით. ამისათვის PHP-ში მრავალი ფუნქცია არსებობს. მათი საშუალებით ასოცირებული მასივის და სიების მოწესრიგება შეიძლება ზრდადობით ან კლებადობით, ან ისეთი მიმდევრობით, როგორიც მომხმარებელს ესაჭიროება.

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.

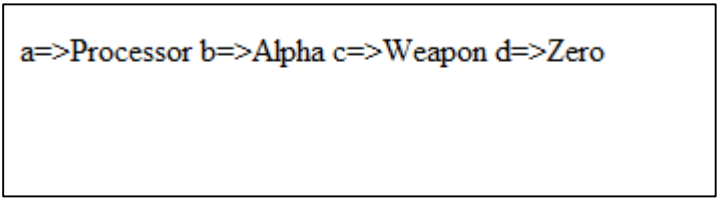


ფუნქცია arsort() იგივე მოქმედებას ასრულებს, ოღონდ მოწესრიგება კლებადობის მიხედვით ხდება.

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

ფუნქცია ksort() პრაქტიკულად asort() 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 ";
```



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 საქაღალდეში არსებული ფაილებისა და ქვესაქაღალდეების სია. შეიძლება დაგვჭირდეს ამ სიის არა მარტო ლექსიკოგრაფიული მიმდევრობით გამოტანა, არამედ ისე, რომ საქაღალდეების 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)
{ // საქაღალდე უსწრებს ფაილს
if(is_dir($f1) && !is_dir($f2)) return -1;
// ფაილი ყოველთვის მოსდევს საქაღალდეს
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 $Files შეიცავს მასივს გასაღებით – ფაილის
სახელებით
// მიმდინარე საქაღალდეში დავალაგოთ.
uksort($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,

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

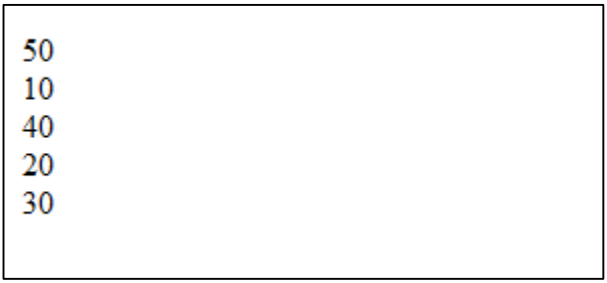
```
10
20
30
40
```

სიის გადანაცვლება shuffle() ფუნქციების მეშვეობით

ფუნქცია ”გადაანაცვლებს” 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.



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 კომუტატიური 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 – ასეთ მასივს:

```
"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) შედეგად აბრუნებს ასოცირებული მასივის ნაწილს გასაღები => მნიშვნელობა სახით. $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, რომელთა 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
```

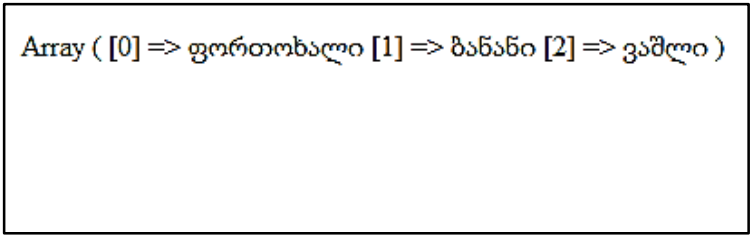
ყურადღება მიაქციეთ იმას, რომ array_push() ფუნქცია მასივს აღიქვამს, როგორც სტეკს და ელემენტს ყოველთვის ბოლოში უმატებს. იგი მასივში ელემენტების ახალ რიცხვს აბრუნებს.

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 ავაგოთ 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);
?>
```



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 მოყვანილია 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 ხდება ყველა დარჩენილი ელემენტის რიცხვითი ინდექსების კორექტირება.

ფუნქცია array_unique(array $Arr)

array_unique() ფუნქცია აბრუნებს მასივს, რომელიც შედგება $Arr მასივის ყველა უნიკალური ელემენტისა და მათი გასაღებისაგან. საშედეგო 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 მოყვანილია ზოგიერთი 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",
"შინდისფერი")

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

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

ფუნქცია implode()

ეს ფუნქცია შესრულების შემდეგ აბრუნებს მასივის ელემენტებისაგან შემდგარ სტრიქონს. მის პირველ არგუმენტს წარმოადგენს გამყოფი სიმბოლო, რომლის მითითებაც არ არის აუცილებელი, ხოლო მეორე არგუმენტს – მასივის სახელი.

მისი სინტაქსია:

implode (გამყოფი სიმბოლო, მასივი)

გამყოფი სიმბოლო მიუთითებს თუ რომელი სიმბოლო უნდა განთავსდეს მასივის ელემენტებს შორის. მისი მითითება არ არის აუცილებელი. ჩუმათობის პრინციპით გამოიყენება მნიშვნელობა "" (ცარიელი სტრიქონი). მაგალითი:

```
<?php
$arr = array('Hello','World!','Beautiful','Day!');
echo implode(" ",$arr);
?>
```

შედეგად მივიღებთ:

ფუნქცია explode ()

მოცემული ფუნქცია სტრიქონს მასივის ელემენტებად დაშლის.

მისი სინტაქსია:

explode (გამყოფი სიმბოლო, სტრიქონი, ელემენტების რაოდენობა)

გამყოფი სტრიქონი აუცილებელია, იგი მიუთითებს თუ სად უნდა გაიყოს სტრიქონი. მეორე არგუმენტიც აუცილებელია. მესამე არგუმენტი მიუთითებს მასივის დასაბრუნებელი ელემენტების რაოდენობას. მისი შესაძლო მნიშვნელობებია:

0-ზე მეტი – შედეგად აბრუნებს მასივს მითითებული რაოდენობის ელემენტებით;

0-ზე ნაკლები – შედეგად აბრუნებს მასივს ბოლო ელემენტის გარეშე;

0-ის ტოლი – აბრუნებს ერთელემენტიან მასივს.

მაგალითი:

```php
<?php
$str = 'one,two,three,four';
// ელემენტების რაოდენობა ნულის ტოლია
print_r(explode(',',$str,0));
print "<br>";
// ელემენტების რაოდენობა დადებითია
print_r(explode(',',$str,2));
print "<br>";
// ელემენტების რაოდენობა უარყოფითია
print_r(explode(',',$str,-1));
?>
```

```
Array ( [0] => one,two,three,four )
Array ( [0] => one [1] => two,three,four )
Array ( [0] => one [1] => two [2] => three )
```

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 მთვლელი

ფუნქცია count (mixed var [, int mode])

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

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

ქვემოთ count() ფუნქციის რეკურსიული გამოყენების მაგალითია მოყვანილი:

```php
<?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 ერთელემენტიან 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?\n";
}
?>
```

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",
                                   "b" => "ბანანი",
                                   "c" => "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.

PHP-ში გამოყენებული სხვა ტიპის მონაცემები

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();
?>
```

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, როგორიცაა 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'));
?>
```

PHP ენის კონსტრუქციები

PHP-ის ნებისმიერი სცენარი რამდენიმე კონსტრუქციითაა ფორმირებული. კონსტრუქცია შეიძლება იყოს ოპერატორი, ფუნქცია, ციკლი, პირობითი კონსტრუქცია, ასევე კონსტრუქცია, რომლებიც არაფერს აკეთებს (ცარიელი კონსტრუქცია). ჩვეულებრივ, კონსტრუქცია წერტილ-მძიმით მთავრდება. ამის გარდა, კონსტრუქციები შეიძლება დაჯგუფებული იყოს, რაც ფიგურულ ფრჩხილებში {...} მოთავსებულ კონსტრუქციათა 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 კიდევ ერთი ალტერნატული სინტაქსი აქვს, რომელსაც ქვემოთ განვიხილავთ.

კონსრტუქცია 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 შეიძლებოდა 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 ალტერნატულ სინტაქსს გამოვიყენებთ:

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

ალტერნატული 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 ალტერნატული სინტაქსი გათვალისწინებული არ არის.

მაგალითი:

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

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

განხილული სცენარი გამოიტანს: 12345678910

ციკლი მთვლელით for

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

for ციკლის სინტაქსი შემდეგია:

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

for ციკლი მუშაობას მაინიციალიზებელი_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 უზრუნველვყავით. ამ შემთხვევაში ჩვენ იტერაციის შემდგომი ბრძანება აღარ გვჭირდებოდა.

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 ალტერნატული სინტაქსი:

```
for (მაინიციალიზებელი_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:

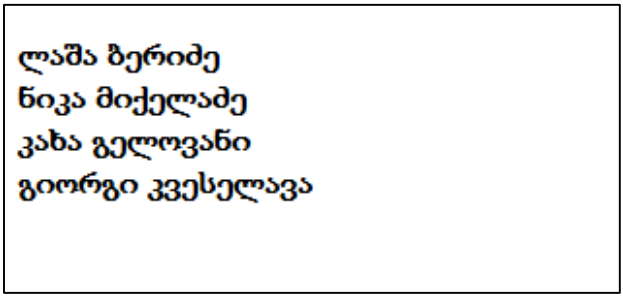
```
foreach (მასივი as $გასაღები=>$მნიშვნელობა)
ბრძანებები;
```

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"]="ლაშა";
$names["მიქელაძე"]="ნიკა";
$names["გელოვანი"]="კახა";
$names["კვესელავა"] = "გიორგი";
```

```
foreach ($names as $key => $value) {
echo "<b>$value $key</b><br>";
}
?>
```

განხილული სცენარი გამოიტანს:



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

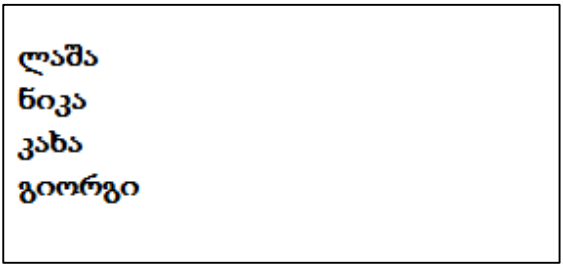
```
foreach (მასივი as $მნიშვნელობა)
ბრძანებები;
```

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:



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

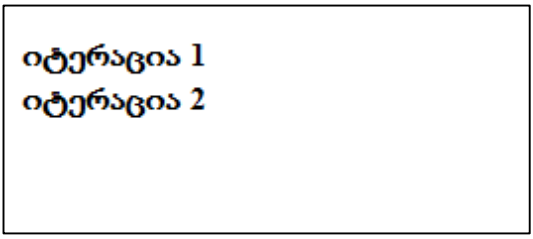
ციკლიდან გამოსვლას მყისვე ასრულებს. იგი შეიძლება ერთ არააუცილებელ პარამეტრთან ერთად იყოს მოცემული – ეს არის რიცხვი, რომელიც მიუთითებს, თუ ერთმანეთში ჩალაგებული რომელი ციკლიდან უნდა განხორციელდეს გამოსვლა. ჩუმათობის პრინციპით გამოიყენება რიცხვი 1 ანუ მიმდინარე ციკლიდან გამოსვლა. break კონსტრუქციის სინტაქსია:

```
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:



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
    {
    }
  }
}
```

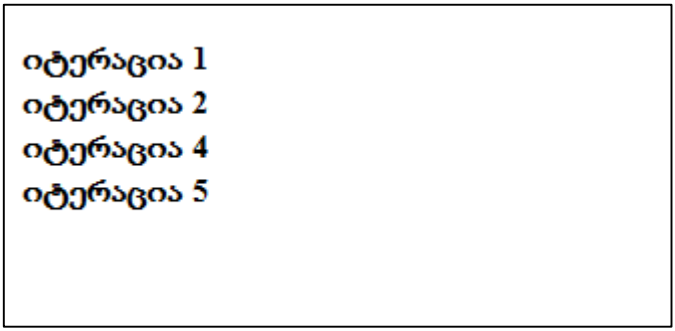
კონსტრუქცია continue

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 ციკლ-ფილტრებში, 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:



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

ამორჩევის კონსტრუქციები

ხშირად მიმდევრობით განლაგებული რამდენიმე if-else კონსტრუქციის მაგივრად მიზანშეწონილია ამორჩევის სპეციალური switch-case კონსტრუქციის გამოყენება. 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 აღემატება 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 მნიშვნელობაi. 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:

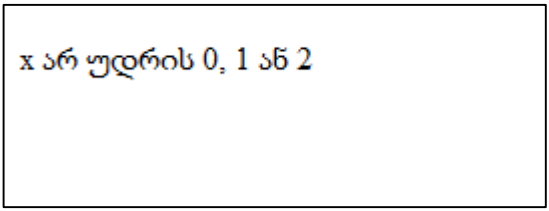
```
<?php
switch ($x) {
case 0:
case 1:
case 2:
    echo "x ნაკლებია 3-ზე, მაგრამ არ არის უარყოფითი";
    break;
case 3:
    echo "x=3";
}
?>
```

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:



switch-case კონსტრუქციას ასევე აქვს ალტერნატული 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.

PHP-ში გამოყენებული სხვა ფუნქციები

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 გამოიყენებს.

თუ მითითებული ფაილი მიმდინარე საქაღალდეშია, მაშინ მისი სახელის (მასზე მიმართვის გზის მითითების გარეშე) მითითებაა საკმარისი. თუ ფაილი სხვა ადგილზე მდებარეობს, მაშინ სრული მიმართვის გზის მითითება აუცილებელია. მიმდინარე საქაღალდის შესაცვლელად გამოიყენება ფუნქცია chdir(). მას აუცილებლად უნდა მივაწოდოთ იმ საქაღალდის 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 თუ მომხმარებელმა ჩართო

უსაფრთხო რეჟიმი[7], open_basedir-ს[8] 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.

---

[7] 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 პროვაიდერები, კონკრეტულად ამ რეჟიმს იყენებენ.

[8] საქაღალდეთა 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 საქაღალდეთა იერარქიული ხის სიაში არ არის, მაშინ PHP ფაილის გახსნაზე უარს ამბობს. ყველა სიმბოლური ბმული ამოიცნობა და გარდაიქმნება, რის გამოც სიმბოლური ბმულის მეშვეობით ამ შეზღუდვისთვის გვერდის ავლა შეუძლებელია.
სპეციალური მნიშვნელობა მიუთითებს, რომ მიმდინარედ ითვლება საქაღალდე, რომელშიც მოცემული სკრიპტი არის განთავსებული. ამ შემთხვევაში ფრთხილად უნდა ვიყოთ, რადგან სკრიპტის სამუშაო საქაღალდე chdir() ფუნქციის საშუალებით მარტივად შეიძლება შევცვალოთ.

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'	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.
'x+'	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.

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 წარმოიშვას 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 საქაღალდეებთან 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 კითხულობს.

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 შეხვდება) 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 გავეცნოთ 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.

საქაღალდესთან სამუშაო ფუნქციები

PHP-ში როგორც ფაილს, ისე საქაღალდესაც სჭირდება გახსნა, ხოლო მუშაობის დამთავრების შემდეგ დახურვა. ამას opendir($path) და closedir($res) ფუნქციები აკეთებს.

ფუნქცია opendir

opendir (string path)

შესრულების შედეგად აბრუნებს კატალოგის (საქაღალდის) 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 საქაღალდის 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] => .
)
*/
?>
```

საქაღალდეებთან 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)

ეს ფუნქცია 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 იგივურად 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)

თუ ეს ფაილი არსებობს და ის საქაღალდეს წარმოადგენს, შედეგად აბრუნებს TRUE-ს. 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)

თუ ეს ფაილი არსებობს და ის ჩვეულებრივი ფაილია, შედეგად აბრუნებს TRUE-ს.

ფუნქცია is_link

is_link (string filename)

თუ ეს ფაილი არსებობს და ის სიმბოლური ბმულია, შედეგად აბრუნებს TRUE-ს.

ფუნქცია filetype

filetype (string filename)

ეს ფუნქცია 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 კეშ-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 საქაღალდე /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 საქაღალდე.

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 კვირაა: '. date('Y-m-d', $nextWeek) ."\n";
// ან strtotime() ფუნქციის მეშვეობით:
echo 'შემდეგი კვირაა: '. 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, მიმდინარე სისტემურ დროს გამოიყენებს.

format არის სტრიქონი, რომელიც დაფორმატების სიმბოლოებს ან ჩვეულებრივ სიმბოლოებს შეიცავს. ჩვეულებრივი სიმბოლოები გამოაქვს ისე, როგორც არის, 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
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

სიმბოლო format სტრიქონში	აღწერა	დაბრუნებული მნიშვნელობის მაგალითი
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 შეესაბამება, წინააღმდეგ შემთხვევაში 0 otherwise.
j	თვის რიცხვი, ნიშნადი ნულების გარეშე	1-დან 31-მდე
l (პატარა 'L')	კვირის დღეების სრული სახელწოდება	Sunday-დან Saturday-მდე
L	ნაკიანი წლის მაჩვენებელი	1, თუ წელი ნაკიანია, თუ არა – 0.

სიმბოლო format სტრიქონში	აღწერა	დაბრუნებული მნიშვნელობის მაგალითი
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
t	თვეში დღეების რაოდენობა	28-დან 31-მდე
T	დროითი ზონა სერვერზე	მაგალითად, EST, MDT ...

სიმბოლო format სტრიქონში	აღწერა	დაბრუნებული მნიშვნელობის მაგალითი
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 ნიმერი 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 სტრიქონში	აღწერა	დაბრუნებული მნიშვნელობის მაგალითი
Z	დროითი ზონის წანაცვლება წამებით. UTC-დან დასავლეთის დროითი ზონისათვის უარყოფითია, ხოლო აღმოსავლეთით – დადებითი	-43200-დან 43200-მდე

ელექტრონული ფოსტის ფუნქციები

ფუნქცია mail ()

აღნიშნული ფუნქციის ფორმატია:

mail (string $to **, string** $subject **, string** $message **[, st**
ring $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 ალმების გადასაცემად იქნეს გამოყენებული ბრძანების სტრიქონის არგუმენტების სახით, წერილის გაგზავნის კონფიგურირების პროგრამისათვის, sendmail_path დირექტივის მითითებით.

თუ წერილი გასაგზავნადაა მიღებული, ფუნქცია შესრულების შედეგად აბრუნებს TRUE-ს, წინააღმდეგ შემთხვევაში – FALSE-ს.

აქვე უნდა აღინიშნოს, რომ თუ წერილი გადასაცემადაა 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 არგუმენ-ტებით 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');
?>
```

ფუნქცია filter_var ()

აღნიშნული ფუნქციის ფორმატია:

filter_var(var, filtername, options)

მითითებული ფილტრის საშუალებით ახდენს მონაცემთა ფილტრაციას. მისი არგუმენტებია:

var – აუცილებელი არგუმენტი. ფილტრაციისათვის განკუთვნილი ცვლადი;

filtername – არააუცილებელი არგუმენტი. მიუთითებს იდენტიფიკატორს ან ფილტრის სახელს გამოყენებისათვის;

options – არააუცილებელი არგუმენტი. გამოყენებისათვის აძლევს ერთ ან რამდენიმე ალამს/ოპციას.

ფუნქციის მუშაობის შედეგად ბრუნდება გაფილტრული მონაცემები ან FALSE – შეცდომის შემთხვევაში.

მაგალითი: შეამოწმეთ, არის თუ არა $email ცვლადი ელექტრონული ფოსტის სწორი მისამართი.

```
<?php
$email = "t.sturua@gmail.com";

if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
  echo("$email ელექტრონული ფოსტის სწორი მისამართია");
} else {
  echo("$email ელექტრონული ფოსტის არა სწორი მისამართია");
}
?>
```

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 შესაძლებელია გენერირებული მეტა-მონაცემების წაკითხვა. 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 მრავალი ფერის, ნახევარტონისა და სხვადასხვა შეფერილობის გამოყენების საშუალებას მოგვცემს. როგორც ცნობილია RGB ფორმატი ეს არის სამი საბაზო ფერი წითელი (Red), მწვანე

(Green) და ლურჯი (Blue). მათი გამოყენებით შესაძლებელია ნებისმიერი ფერის მიღება. 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 $width×$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 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.

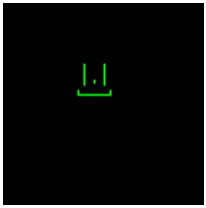
შავი ფერის მისაღებად სამი 0-იანი გვჭირდება, ხოლო თეთრი ფერისათვის სამჯერ 255.

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($img);
// მეხსიერების გათავისუფლება
imagedestroy($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 მეხსიერებაში 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 წვეროო 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 მითითებულია, რომ 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-ის 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 ინფორ-მაცია სტრიქონის 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. ქვემოთ 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.

დაბოლოს, 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 გამოსახულებისათვის 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 მნიშვნელობა იქნება გამოტანილი.*

ფუნქციის შეცდომის შემთხვევაში გამოიტანს FALSE მნიშვნელობას.

ბმულები PHP-ში

მართალია PHP-ში მაჩვენებლის ცნება არ არის, მაგრამ შესაძლებელია სხვა ცვლადზე ბმულის შექმნა. 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 იძლევა იმის საშუალებას, 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 ასეთი ქმედება განხორციელდა, 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 კიდევ 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 კიდევ 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.

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.

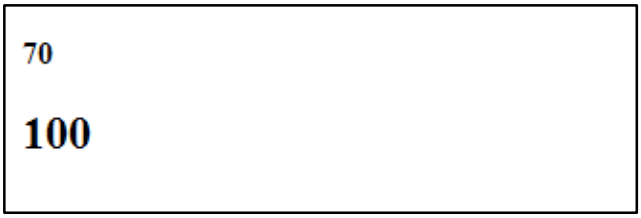
PHP-სათვის ფუნქციაში ყველა გამოცხადებული და გამოყენებული ცვლადები ჩუმათობის პრინციპით ლოკალურია, ანუ ჩუმათობის პრინციპით ფუნქციის ტანში გლობალური ცვლადის მნიშვნელობის შეცვლა შეუძლებელია.

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.

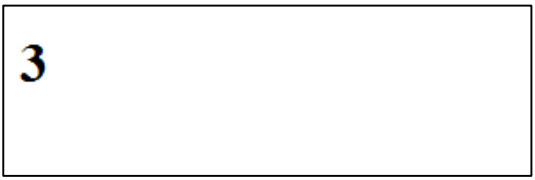


ზემოთ მოყვანილი ნაკლოვანების თავიდან ასაცილებლად PHP-ში არსებობს სპეციალური ინსტრუქცია global, რომელიც მომხმარებლის ფუნქციას გლობალურ ცვლადებთან მუშაობის საშუალებას აძლევს. მოცემული პრინციპი განვიხილოთ კონკრეტულ მაგალითზე:

```
<?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.



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.

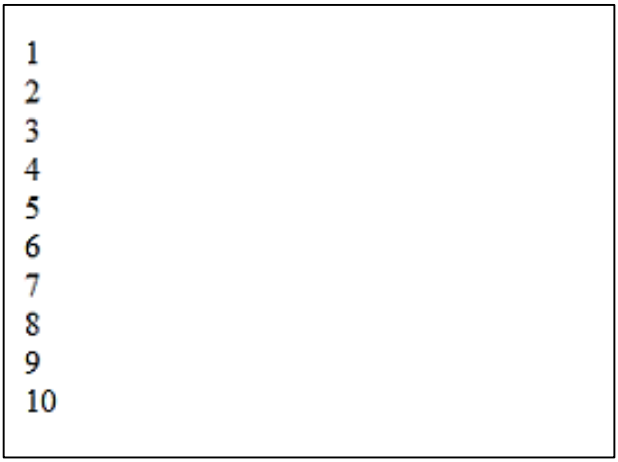
$GLOBALS – ეს არის ასოცირებული მასივი, რომლის გასაღებს გლობალური ცვლადის სახელი, ხოლო მნიშვნელობას ამ ცვლადის მნიშვნელობა წარმოადგენს. ყურადღება მიაქციეთ იმას, რომ $GLOBALS ხედვის ნებისმიერ არეში არსებობს, ეს იმით აიხსნება, რომ ეს მასივი სუპერგლობალურია.

ლოკალური და გლობალური ცვლადების გარდა PHP-ში არსებობს კიდევ ერთი ტიპის ცვლადი – სტატიკური ცვლადი.

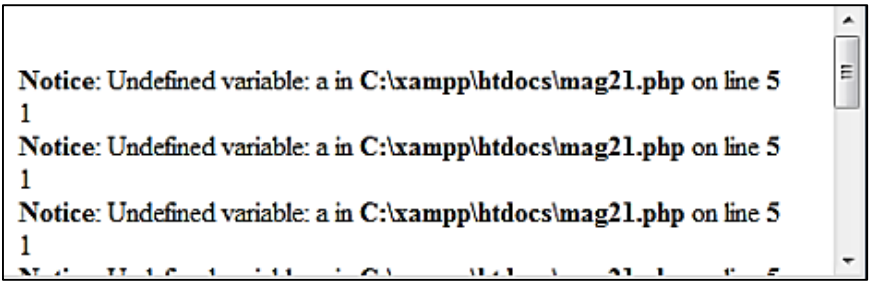
თუ მომხმარებლის ფუნქციის ტანში გამოცხადებულია სტატიკური ცვლადი, მაშინ კომპილატორი ფუნქციის მუშაობის დამთავრების შემდეგ მის მნიშვნელობას არ წაშლის. ქვემოთ მოყვანილია სტატიკური ცვლადის შემცველი მომხმარებლის ფუნქციის მუშაობის მაგალითი:

```
<?php
  function funct()
  {
    static $a;
    $a++;
    echo "$a"."<br>";
  }
  for ($i = 0; $i++<10;) funct();
?>
```

მოცემული სკრიპტის მუშაობის შედეგად მივიღებთ:



თუ ამ სკრიპტში static-ს წავშლით, მაშინ მივიღებთ შედეგს:



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.

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) { ... };
?>
```

funct () 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:

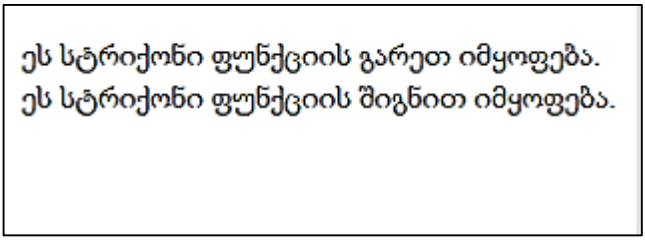
პარამეტრი = 777

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;
?>
```



განვიხილოთ კიდევ 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 სპეციფიკურ 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);
```

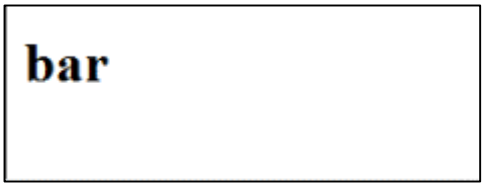
ფუნქციის შიგნით unset () ოპერატორის ქცევა წასაშლელი ცვლადის ტიპზეა დამოკიდებული.

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>";
?>
```

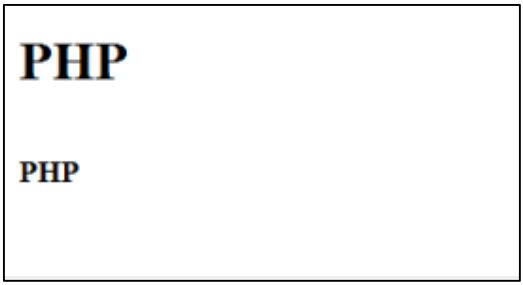
ეს მაგალითი შედეგად გამოიტანს:



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>";
?>
```

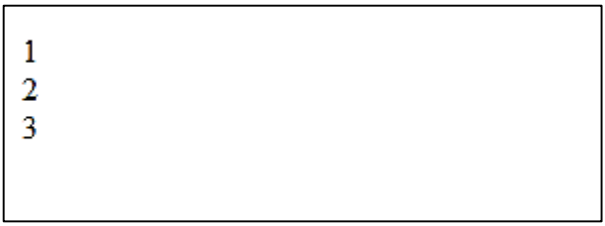
ამ მაგალითის შედეგია:



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:



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.

something

Notice: Undefined variable: bar in
C:\xampp\htdocs\mag21.php on line **9**

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);
?>
```

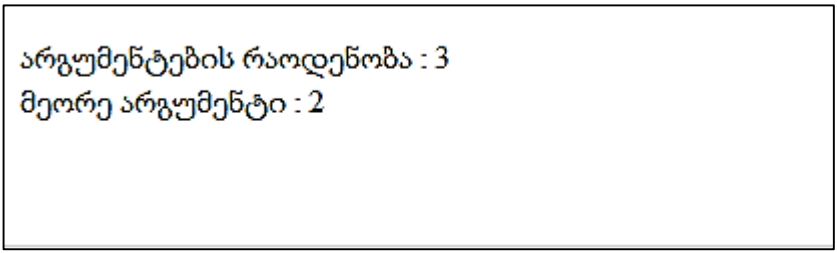
```
არგუმენტების რაოდენობა : 3
```

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);
?>
```



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.

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 აუცილებელია ობიექტის ინიციალიზაცია, ანუ ობიექტის თვისებებს საწყისი მნიშვნელობები უნდა 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 თავისუფლდება.

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_funct() { echo "<h1>ეს საბაზო ფუნქციაა</h1>"; }
function test () { echo "<h1>ეს საბაზო კლასია</h1>"; }
}

class Child extends Paren {
function child_funct() { echo "<h2>ეს მემკვიდრე
ფუნქციაა</h2>"; }
function test () { echo "<h2>ეს მემკვიდრე (ქვეკლასი)
კლასია</h2>"; }
```

```
}

$object = new Paren;
$object = new Child;

$object->parent_funct(); // გამოიტანს "ეს საბაზო ფუნქციაა"
$object->child_funct(); // გამოიტანს "ეს მემკვიდრე ფუნქციაა"
$object->test(); // გამოიტანს "ეს მემკვიდრე (ქვეკლასი) კლასია"
?>
```

ამ სკრიპტის მუშაობის შედეგი იქნება:

```
ეს საბაზო ფუნქციაა
ეს მემკვიდრე ფუნქციაა
ეს მემკვიდრე (ქვეკლასი) კლასია
```

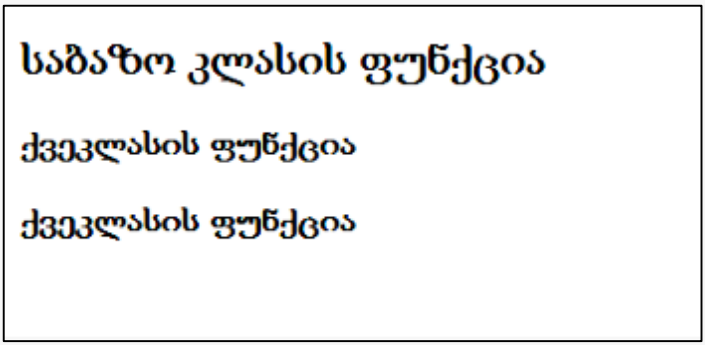
extends საგასაღებო სიტყვა გვეუბნება, რომ child შვილობილ კლასს parent კლასის ყველა მეთოდი და თვისება მემკვიდრეობით ერგო. Parent კლასს, ჩვეულებრივ, საბაზო კლასს ან სუპერკლასს, ხოლო child მემკვიდრე კლასს ან ქვეკლასს უწოდებენ.

კლასების პოლიმორფიზმი PHP-ში

ქვეკლასის ფუნქციების გამოყენება პოლიმორფიზმის საბაზო კლასის თვისებაა. ქვემოთ მოყვანილია კლასის თვისების – პოლიმორფიზმის მაჩვენებელი პრაქტიკული მაგალითი.

```
<?php
class Base {
 function funct() {
 echo "<h2>საბაზო კლასის ფუნქცია</h2>";
 }
 function base_funct() {
 $this->funct();
 }
}

class Derivative extends Base {
 function funct() {
 echo "<h3>ქვეკლასის ფუნქცია</h3>";
 }
}

$b = new Base();
$d = new Derivative();

$b->base_funct();
$d->funct();
$d->base_funct();
?>
```

სკრიპტი მუშაობის შედეგად გამოიტანს:



განხილულ მაგალითში Base კლასის base_funct() ფუნქცია იმავე სახელწოდების ფუნქციად Derivative კლასში იყო გადაწერილი. ამ სახით განსაზღვრულ ფუნქციას ვირტუალური ფუნქცია ეწოდება.

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 შეიძლება. ჩვეულებრივ, GOTO ოპერატორი შეიძლება მრავალდონიანი break ოპერატორის ნაცვლად გამოვიყენოთ. მაგალითად:

```
<?php
goto a;
echo 'Foo';

a:
echo 'Bar';
?>
```

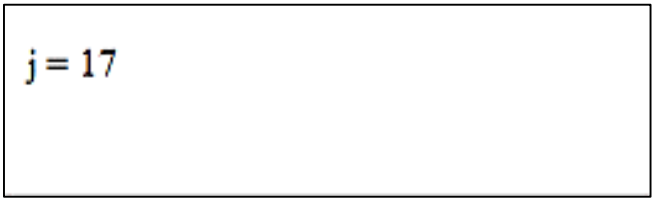
მოცემული მაგალითის შესრულების შედეგი იქნება:

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';
?>
```

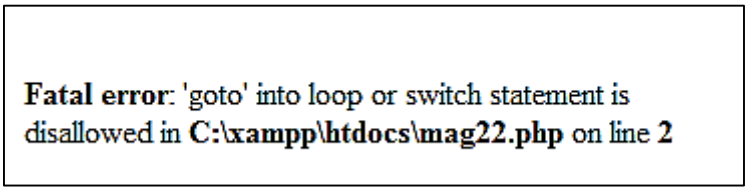
შედეგი ასეთი იქნება:



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 = $i";
?>
```

ამ მაგალითის შესრულების შედეგად მივიღებთ:



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.

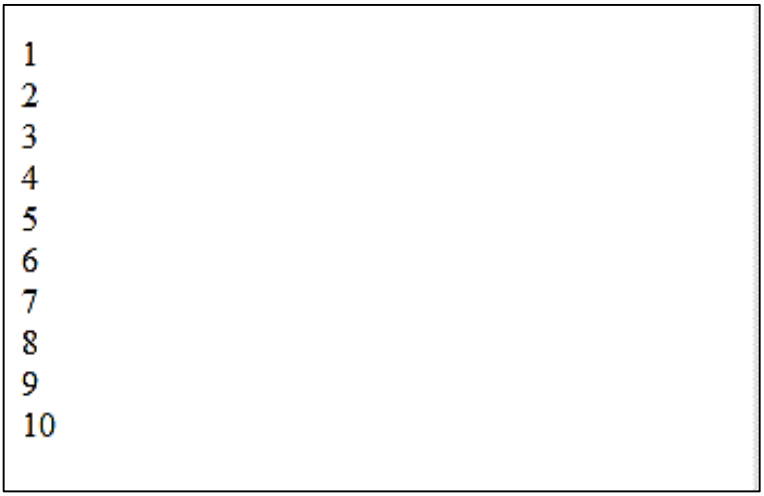
require კონსტრუქციისაგან განსხვავებით კონსტრუქცია include საშუალებას იძლევა PHP სკრიპტის კოდში ფაილები სცენარის შესრულების დროს ჩაირთოს. include კონსტრუქციის სინტაქსი შემდეგნაირად გამოიყურება:

```
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.



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 გააანალიზებს, 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.

ლამაზი სკრიპტი ანუ დაპროგრამების სტილი

სკრიპტი ლამაზი უნდა იყოს, ეს უბრალოდ სილამაზი-სათვის 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 ღილაკის 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 იყოს ზოგან ორი, ზოგან ოთხი და ზოგან არც ერთი. 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 კიდის მიმართ ერთი და იმავე მანძილით უნდა იყოს დაცილებული.

ფიგურული ფრჩხილების განლაგება

ფიგურული ფრჩხილების განთავსების მრავალნაირი სტილი არსებობს:

```
<?PHP
if ($a > $b) {
   // ოპერატორები
}
?>

<?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 შემთხვევებც 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_sum m_as_i nteger_value.

თუ რატომ, ამას ალბათ ახსნის გარეშეც მიხვდება მომხმარებელი.

კომენტარები

კომენტარების შესახებ PHP-ის შესწავლის დასაწყისში გვქონდა საუბარი. ახლა განვსაზღვროთ, რა შემთხვევაში დაგვჭირდება კოდის კომენტირება:

- ამოცანის არაჩვეულებრივი გადაწყვეტა. იმიტომ, რომ შეიძლება შემდეგ ვერ გაიხსენოთ რა იგულისხმეთ მოცემულ შემთხვევაში და მით უმეტეს, 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;
?>
```

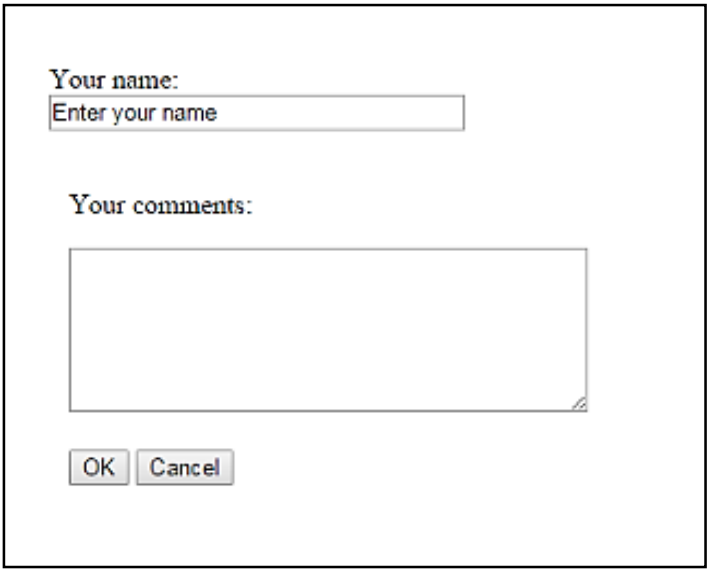
ჩვეულებრივ, ერთი ბრძანების კომენტარი წერტილ-მძიმის შემდეგ უნდა განვათავსოთ, ხოლო ბლოკის კომენტარი – მისი პირველი ბრძანების წინ, სტრიქონში.

მართალია, ბევრ პროგრამისტს კომენტარების წერა არ უყვარს, მაგრამ, როდესაც დიდი მოცულობის სკრიპტებს ვწერთ, სასურველია მასში კომენტარები განვათავსოთ.

HTML5-ის ფორმის ტეგებთან მუშაობა PHP-ის სკრიპტებში

ფორმები მსოფლიო აბლაბუდაში ინტერაქტიული ურთიერთქმედებისთვის ერთ-ერთ ყველაზე პოპულარული საშუალებაა. ფორმების საშუალებით მომხმარებლისგან შესაძლებელია ინფორმაციის მიღება. ფორმები რამდენიმე ველისგან შედგება, სადაც მომხმარებელს შეუძლია გარკვეული ინფორმაცია შეიტანოს ან რომელიმე პარამეტრი აირჩიოს. მას შემდეგ რაც მომხმარებელი გაგზავნის ინფორმაციას, ის სერვერზე მოთავსებული სკრიპტის მიერ დამუშავდება. სკრიპტი მოკლე პროგრამაა, რომელიც თითოეული ფორმის დასამუშავებლად სპეციალურადაა შექმნილი.

HTML-ის დახმარებით მხოლოდ ფორმის ვიზუალურ მხარეს განვსაზღვრავთ. იმისათვის, რომ ფორმამ რეალურად იმუშაოს და შევსებული ფორმის მონაცემები დამუშავდეს, HTML-ის ცოდნა საკმარისი არ არის, აქ უკვე სხვა ტექნოლოგიაა საჭირო. HTML-ს არ შეუძლია მონაცემების დამუშავება, ამისათვის სერვერული დაპროგრამების სპეციალური ენა PHP არსებობს, ამიტომაც ფორმა ყოველთვის იმ სერვერულ დამმუშავებელთან უნდა იყოს დაკავშირებული, რომელიც ფორმის მონაცემებს დაამუშავებს. განვიხილოთ მუშაობის სქემა. მაგალითად, როდესაც ქვემოთ მოცემული ტიპის ფორმის შევსება ხდება და მომხმარებელი OK ღილაკს აჭერს ხელს, მონაცემები გადაეგზავნება მის დამმუშავებელს. ეს ფაილი, მაგალითად, იყოს formdata.php.



ეს ფაილი ფორმიდან მიღებულ მონაცემებს საიტის მფლობელის სურვილის მიხედვით დაამუშავებს. შეიძლება მას სურდეს ფორმაში არსებული ინფორმაცია ელექტრონული წერილის სახით მიუვიდეს ან მოხდეს მისი ჩაწერა ბაზაში. ამისათვის formdata.php-ში შესაბამისი კოდი უნდა ჩაიწეროს, რომელიც ან ერთი, ან მეორე გზით უზრუნველყოფს ინფორმაციის მიწოდებას.

ფორმები ინფორმაციის შეტანისათვისაა განკუთვნილი. ყველა ფორმა <form> (ინგლ. form – ფორმა) ტეგით იწყება და </form> ტეგით მთავრდება. ფორმას შეიძლება შემდეგი ატრიბუტები ჰქონდეს: name, action, method, target.

- ატრიბუტი name ფორმის სახელს განსაზღვრავს. თუ დოკუმენტში რამდენიმე ფორმა არ არის გამოყენებული, მაშინ მისი მითითება აუცილებელი არ არის;
- ატრიბუტი action უზრუნველყოფს ფორმის მონაცემების

შესაბამისი ფაილისთვის დასამუშავებლად გადაგზავნას.

- ატრიბუტი method ფორმის პარამეტრების გადაცემის მეთოდს განსაზღვრავს. იგი get, post ან request მნიშვნელობებს ღებულობს.
- ატრიბუტი target განსაზღვრავს ფანჯარას, რომელშიც გაგზავნილი ფორმის დამუშავების შედეგები დაბრუნდება. მასში ფანჯრის სახელი ცხადად უნდა ჩაიწეროს ან ერთ-ერთი შემდეგი პარამეტრი უნდა მიეთითოს:
    - blank – დოკუმენტი ახალ ფანჯარაში გაიხსნება;
    - self – დოკუმენტი იმავე ფანჯარაში გაიხსნება, რომელშიც მიმდინარე დოკუმენტია გახსნილი;
    - parent – დოკუმენტი იმ ფრეიმში გაიხსნება, რომელიც მიმდინარე დოკუმენტის შემცველი ფრეიმის მიმართ მშობლიურ ფრეიმს წარმოადგენს (თუ მშობლიური ფრეიმი არ არსებობს, მაშინ დოკუმენტი მიმდინარე ფრეიმში გაიხსნება);
    - top – დოკუმენტი მიმდინარე ფანჯარაში გაიხსნება და ყველა ფანჯარას დაიჭერს.

ჩვენ form1.php სკრიპტს შემდეგი სახე ექნება:

```
<?php

?>
<!DOCTYPE html>
<html>
<head>
<title>forms1</title>
</head>
<body>
```

```
<form action ="formdata.php" method="GET">
<p>
Your name: <input type="text" size="30">
</p>
<p>Your comments:</p>
<textarea rows="7" cols="50" > </textarea> <br>
<br>
<input type="submit" value="OK">
<input type="reset" value="Cancel">
</form>
</body>
</html>
```

მართვის ელემენტების ფორმაზე განთავსება

მონაცემთა შესატანი ველის, რომელსაც ზემოთ გავეცანით, და მართვის სხვა ელემენტების შესაქმნელად <input> (ინგლ. input – შეტანა) ტეგი გამოიყენება. ამ ტეგს name, size, checked, maxlength, src, type ატრიბუტები აქვს.

- ატრიბუტი name – მონაცემთა შესატანი ველის სახელს განსაზღვრავს. მოცემული სახელი განიხილება, როგორც ველის უნიკალური იდენტიფიკატორი, რომლის მიხედვითაც შემდგომში შეიძლება მომხმარებლის მიერ ამ ველში შეტანილი მონაცემები მივიღოთ;
- ატრიბუტი size – მონაცემთა შესატანი ველის ზომას განსაზღვრავს. ეს ატრიბუტი განსაზღვრავს ველის სიგრძეს და არა მასში შესატანი სიმბოლოების რაოდენობას. თუ მონაცემთა შესატან ველში შეტანილი სიმბოლოების რაოდენობა ველის სიგრძეს აღემატება, მაშინ იქ გადაფურცვლის ზოლი გაჩნდება;

• ატრიბუტი checked – ალმებით მოსანიშნად და გადამრთველებისათვის გამოიყენება. იგი უჩვენებს, რომ ვებგვერდის გახსნის დროს ამ ატრიბუტის მქონე ველი მონიშნული იქნება;

• ატრიბუტი maxlength – მომხმარებლის მიერ მონაცემთა შესატან ველში შესატანი სიმბოლოების დასაშვები რაოდენობაა. ამ რაოდენობაზე მეტი სიმბოლოს შეტანის მცდელობის დროს ბრაუზერი ხმოვან სიგნალს გამოსცემს და ზედმეტი სიმბოლოს შეტანის საშუალებას არ მოგვცემს. თუ მოცემული ატრიბუტი არ იქნება მითითებული, მაშინ მისი მნიშვნელობა ზოგადად უსასრულობის ტოლია;

• ატრიბუტი src – გამოსახულების URL-მისამართს მიუთითებს (image ატრიბუტთან ერთად გამოიყენება);

• ატრიბუტი type – მართვის ელემენტის ტიპს განსაზღვრავს. თუ იგი არ იქნება მითითებული, მაშინ გამოდის ერთსტრიქონიან მონაცემთა შესატანი ველი. ყველა დანარჩენი ტიპი აუცილებლად უნდა მიეთითოს; type ატრიბუტმა შეიძლება შემდეგი მნიშვნელობები მიიღოს:

- Checkbox, რომელიც მონაცემთა შესატან ველს ალმით მოსანიშნ ველად გარდაქმნის;
- Hidden, რომელიც მართვის ელემენტს მალავს – იგი ბრაუზერის მიერ არ აისახება და მომხმარებელს ჩუმათობის პრინციპით მინიჭებული მნიშვნელობის შეცვლის უფლებას არ აძლევს. მართვის ასეთი ელემენტი პროგრამისათვის უცვლელი ინფორმაციის გადასაცემად გამოიყენება, მაგალითად, მომხმარებლის იდენტიფიკატორი ან პაროლი;
- image საშუალებას იძლევა მართვის ელემენტის სახელს გამოსახულება დაუკავშირდეს.

- password მონაცემთა შესატან ველს გარეგნულად არ ცვლის, მაგრამ ბრაუზერი მომხმარებლის მიერ შეტანილ მნიშვნელობებს ეკრანზე არ ასახავს;
- radio მონაცემთა შესატან ველს ისეთ გადამრთველად გადააქცევს, რომელიც საშუალებას იძლევა name ატრიბუტის ერთი და იმავე მნიშვნელობის და value ატრიბუტით მოცემული გადამრთველის სხვადასხვა მნიშვნელობას შორის მხოლოდ ერთი მნიშვნელობა აირჩეს.
- text ერთსტრიქონიან მონაცემთა შესატან ველს აღწერს. მაქსიმალური სიგრძის მნიშვნელობისა და მონაცემთა შესატანი ველის განსაზღვრისათვის maxlength და size ატრიბუტები გამოიყენება.
- reset მონაცემთა შესატან ველს ღილაკად გარდაქმნის, რომელზეც მაუსის დაწკაპუნებით ფორმის მართვის ყველა ელემენტი მათთვის ჩუმათობის პრინციპით მინიჭებულ მნიშვნელობას მიიღებს;
- submit მონაცემთა შესატან ველს ღილაკად გარდაქმნის, რომელზეც მაუსის დაწკაპუნებით ხდება ინფორმაციის გაგზავნა სერვერზე.

- *ატრიბუტი value* – საშუალებას იძლევა მართვის ელემენტს მიენიჭოს მნიშვნელობა ჩუმათობის პრინციპით ან მნიშვნელობა, რომელიც name ატრიბუტს შესაბამისი გადამრთველის დაყენების საშუალებით მიენიჭება (გადამრთველის ტიპისათვის მოცემული ატრიბუტი აუცილებელია).

ფორმაში მრავალსტრიქონიანი ტექსტური ველების შესაქმნელად, რომელიც მომხმარებელს დიდი მოცულობის ინფორმაციის შეტანის საშუალებას აძლევს, <textarea> ტეგი გამოიყენება. ამ ტეგს name, rows და cols ატრიბუტები აქვს.

- ატრიბუტი name – მონაცემთა შეტანის ველის სახელს განსაზღვრავს;
- ატრიბუტი rows – მონაცემთა შეტანის ველის სიმაღლეს განსაზღვრავს სიმბოლოებში;
- ატრიბუტი cols – მონაცემთა შეტანის ველის სიგანეს განსაზღვრავს სიმბოლოებში;

იმისათვის, რომ მონაცემთა შეტანის ველში ფორმის გახსნის დროს ჩუმათობის პრინციპით რაიმე ტექსტი გამოვიდეს, საჭიროა იგი <textarea> და </textarea> ტეგებს შორის ჩავსვათ.

ღილაკის შექმნა

მიუხედავად იმისა, რომ ფორმებში ღილაკების შესაქმნელად <input> ტეგი გამოიყენება, HTML 5-ში ამ მიზნით კიდევ <button> ტეგის გამოყენებაა შესაძლებელი. <button> აქვს შემდეგი ატრიბუტები: type, name, value, autofocus, disabled, form, formaction, formenctype, formmethod და formnovalidate.

- ატრიბუტი type ღილაკის ტიპს განსაზღვრავს. მას შემდეგი მნიშვნელობების მიღება შეუძლია: button – ჩვეულებრივი, submit – ფორმის გაგზავნისა და reset – გაგზავნის შეწყვეტის ღილაკები;
- ატრიბუტი name ღილაკის სახელს განსაზღვრავს;
- ატრიბუტი value ღილაკის წარწერას განსაზღვრავს;
- ატრიბუტი autofocus მიუთითებს, რომ ღილაკი გვერდის ჩატვირთვისთანავე გახდეს აქტიური;
- ატრიბუტი disabled მიუთითებს, რომ ღილაკი უნდა იყოს უმოქმედო (გამორთული);

- form ატრიბუტის მნიშვნელობა იმ ფორმის სახელია, რომელსაც აღნიშნული ღილაკი ეკუთვნის;
- formaction ატრიბუტის მნიშვნელობა ის URL მისამართია, რომელზეც ფორმის მონაცემები უნდა გადაიგზავნოს. გამოიყენება მაშინ, როდესაც type=submit;
- ატრიბუტი formenctype მიუთითებს, თუ როგორ უნდა მოხდეს ფორმის მონაცემების კოდირება მისი გადაგ-ზავნის წინ (გამოიყენება მხოლოდ type=submit შემ-თხვევაში). მისი შესაძლო მნიშვნელობებია: application, x-www-form-urlencoded, multipart/ form-data text და plain;
- ატრიბუტი formmethod ფორმის მონაცემების გაგზავნის მეთოდს მიუთითებს. გამოიყენება მხოლოდ მაშინ, როცა type=submit. მისი შესაძლო მნიშვნელობებია: get და post;
- ატრიბუტი formnovalidate მიუთითებს იმაზე, რომ ფორმის მონაცემები გაგზავნის წინ არ უნდა შემოწმდეს. გამოიყენება მხოლოდ მაშინ, როცა type=submit.

form1.php სკრიპტის შესრულებაზე გაშვების შემთხვევაში ბრაუზერის URL ველში ჩაიწერება:

localhost:1234/form1.php

როგორც უკვე ზემოთ აღვნიშნეთ, ამ ფორმის შევსების და OK ღილაკზე დაჭერის შემთხვევაში შევსებული ინფორმაცია გადაეცემა formdata.php ფაილს, რომელიც ფორმაში შეტანილ მონაცემებს მომხმარებლის სურვილის მიხედვით დაამუშავებს, ხოლო URL ველში ძველი მისამართი შეიცვლება და გამოჩნდება შემდეგი:

localhost:1234/formdata.php

ახლა ტექსტური ველის სტრიქონს მივცეთ შემდეგ სახე:

Your name: <input type="text" size="30" name=”Saxeli”>

და ამ ველში შევიტანოთ სახელი – Nikolozi. OK ღილაკზე დაჭერის შემთხვევაში URL ველში გამოჩნდება შემდეგი ბმული:

localhost:1234/formdata.php?Saxeli=Nikolozi

თუ name=”Your name”, ამ შემთხვევაში URL ბმულს შემდეგი სახე ექნებოდა:

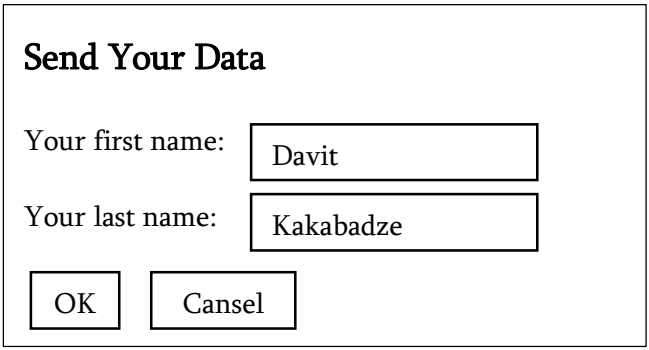
localhost:1234/formdata.php?Your+name=Nikolozi

ეხლა დავწეროთ სკრიპტი, რომელშიც ორი ტექსტური ველი გვექნება, მაგალითად

```
<?php

?>
<!DOCTYPE html>
<html>
<head>
<title>forms1</title>
</head>
<body>
<h2>Send Your  Data</h2>
<form action ="formdata.php" method="GET">
<p>
Your first name: <input type="text" size="30" name=”Saxeli”>
</p>
<p>
Your last name: <input type="text" size="30" name=”Gvari”>
```

```
</p>
<br>
<input type="submit" value="OK">
<input type="reset" value="Cancel">
</form>
</body>
</html>
```

შედეგს ექნება შემდეგი სახე, რომელშიც შევიტანოთ მონაცემები:



OK ღილაკზე ხელის დაჭერის შემთხვევაში URL ველში გამოჩნდება შემდეგი ბმული:

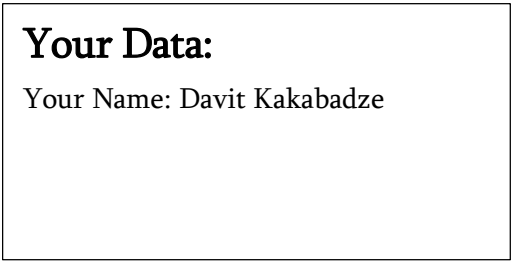
localhost:1234/formdata.php?Saxeli=Nikolozi&Gvari=Kakabadze

ფორმის მონაცემებთან წვდომის კიდევ ერთი საშულებაა **$_GET**. შევქმნათ ისეთი კოდის მქონე ფორმა, რომელშიც მონაცემების შეტანის შემდეგ, ამ მონაცემებს რაიმე წინადადებაში ჩასვამს. პირველი პროგრამა იგივე იყოს, ხოლო მეორე formdata.php სკრიპტს ჰქონდეს შემდეგი სახე:

```
<?php
```

```
  $A=$_GET(Saxeli);
  $B=$_GET(Gvari);
?>
<!DOCTYPE html>
<html>
<head>
<title>form values</title>
</head>
<body>
        <h1>Your Data:</h1>
        <?php echo 'Your Name: ', $A,' ',$B; ?>
</body>
</html>
```

ფორმაში მონაცემების შეტანისა და OK ღილაკზე ხელის დაჭერის შემდეგ ფანჯარაში მივიღებთ:



თუ პირველ პროგრამაში ფორმის ტეგში GET პარამეტრს შევცვლით POST-ით, მაშინ პროგრამა გამოგვიტანს შემდეგ შეტყობინებას:

Notice: Underfined index: Saxeli in **C:\xampp\htdocs\formdata.php** on line 2

Notice: Underfined index: Gvari in **C:\xampp\htdocs\formdata.php** on line 3

Your Data:

Your Name:

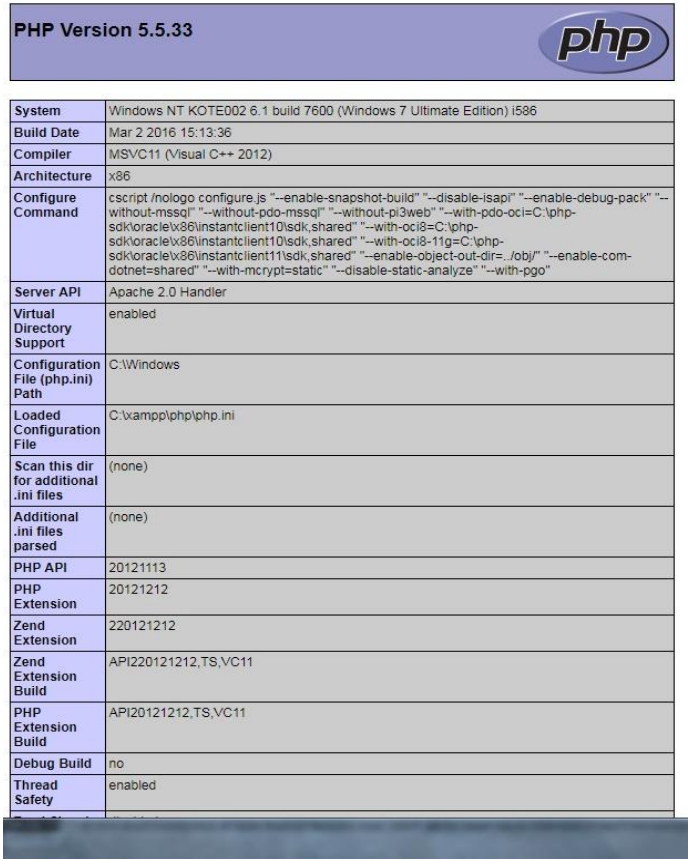
თუ მეორე პროგრამაში $_GET ფუნქციას $_POST-ით შევცვლით, მაშინ პროგრამა იმუშავებს უშეცდომოდ, ხოლო თუ მეორე პროგრამაში $_GET ფუნქციას $_REQUEST ფუნქციით ჩავანაცვლებთ, მაშინ ფორმის ტეგში რომელ პარამეტრს გამოვიყენებთ, GET-ს თუ POST-ს, არა აქვს მნიშვნელობა. შედეგი ყოველთვის სწორი იქნება.

დავალებები და ამოცანები

ინფორმაციის გამოტანა

1. XAMPP-ის გამართულად მუშაობის შესამოწმებლად Notepad++-ში აკრიფეთ phpinfo(), რომელიც უნდა მოთავსდეს php ტეგებს შორის;

შედეგი, რომელიც უნდა აისახოს ეკრანზე:



PHP Version 5.5.33

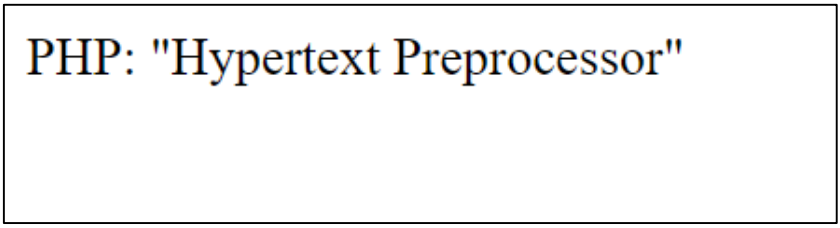
System	Windows NT KOTE002 6.1 build 7600 (Windows 7 Ultimate Edition) i586
Build Date	Mar 2 2016 15:13:36
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\x86\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\x86\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\x86\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20121113
PHP Extension	20121212
Zend Extension	220121212
Zend Extension Build	API220121212,TS,VC11
PHP Extension Build	API20121212,TS,VC11
Debug Build	no
Thread Safety	enabled

2. შევქმნათ ყველაზე მარტივი პროგრამა PHP-ზე, ბრჭყალებს შორის მოათავსეთ სიტყვები: Hello, PHP! და echo-ს საშუალებით გამოიტანეთ შედეგი ეკრანზე.

შედეგი, რომელიც უნდა აისახოს ეკრანზე:

3. ერთმაგ ბრჭყალებს შორის მოვაქციოთ ორმაგი ბრჭყალებიც, შემდეგი სახით 'PHP: "HypertextPreprocessor"' და echo-ს საშუალებით გამოვიტანოთ ეკრანზე.

შედეგი, რომელიც უნდა აისახოს ეკრანზე:

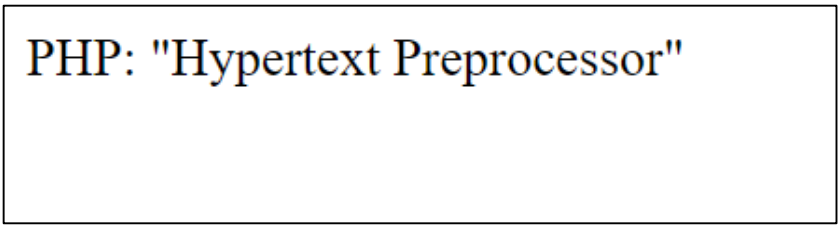


4. 3 ცალი ერთმაგი ბრჭყალი და „\“ (საწინააღმდეგოდ დახრილი ხაზი) სიმბოლო შემდეგნაირად განავალაგოთ ეკრანზე გამოსატან ტექსტში: 'PHP\'s Web-site' და echo საშუალებით გავუშვათ პროგრამა.

შედეგად მივიღებთ:

5. 4 ცალი ორმაგი ბრჭყალი და 2 ცალი „\“ (საწინააღმდეგოდ დახრილი ხაზი) სიმბოლო შემდეგი თანმიმდევრობით განვათავსოთ ეკრანზე გამოსატან ტექსტში. echo-ს შემდეგ ვწერთ: "PHP: \"Hypertext Preprocessor\"";

შედეგი იქნება:



6. გამოვიყენოთ HTML ტეგები PHP კოდში, გავუზარდოთ ტექსტს შრიფტს ზომა <h1>, </h1> ტეგების საშუალებით, მაგალითად: "<h1>PHP</h1>".

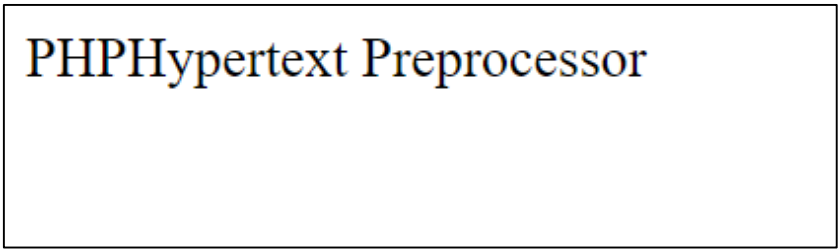
შედეგი, რომელიც უნდა აისახოს ეკრანზე:

7. გამოვიყენოთ ბრჭყალები და წერტილი სტრიქონში, შემდეგი სახით:

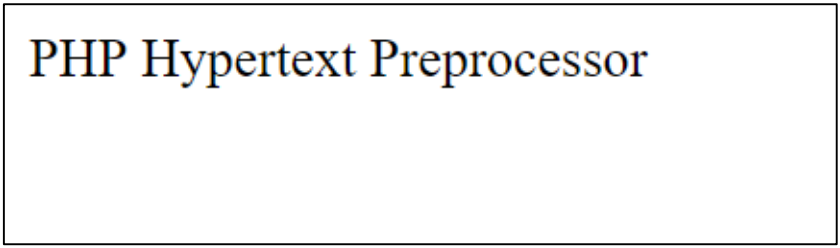
"PHP"."Hypertext Preprocessor"

და echo-ს საშუალებით გამოვიტანოთ ეკრანზე.

შედეგი, რომელიც უნდა აისახოს ეკრანზე:



8. ბრჭყალების, მძიმის და ინტერვალის გამოყენებით შემდეგნაირად წარმოვადგინოთ ტექსტი PHP-ში: "PHP"," Hypertext Preprocessor" და echo-ს საშუალებით გამოვიტანოთ ეკრანზე.



9. echo-ს 2-ჯერ გამოყენებით, შემდეგნაირად წარმოვადგნოთ ეკრანზე გამოსატანი სკრიპტი:

```
echo “PHP”;
echo “Hypertext Preprocessor”;
```

და შევასრულოთ ბრძანება.

შედეგი, რომელიც უნდა აისახოს ეკრანზე:

10. ერთდროულად echo-ს და print-ის გამოყენებით, გამოვიტანოთ ეკრანზე სიტყვა “PHP”, echo-ს საშუალებით და სიტყვა “ Hypertext Preprocessor” print-ის საშუალებით.
შედეგი, რომელიც უნდა აისახოს ეკრანზე:

11. შექმენით მრავალხაზოვანი კომენტარი PHP-ში, რისთვისაც ტექსტური რედაქტორის ფანჯარაში ავკრიფოთ შემდეგი კოდი:

```
<?php
/*
PHP შექმნა Rasmus Lerdorf-მა
1995 წელს
*/
print "PHP";
echo "Hypertext Preprocessor";

?>
```

12. შექმენით ერთხაზიანი კომენტარი, რისთვისაც ტექსტური რედაქტორის ფანჯარაში ავკრიფოთ შემდეგი კოდი:

```
<?php
// Zeev Suraski-მ განავითარა კოდი და შექმნა PHP 3
print "PHP";
echo "Hypertext Preprocessor";

?>
```

13. შექმენით ერთხაზიანი კომენტარის ალტერნატიული ვარიანტი. შედეგი, რომელიც უნდა აისახოს ეკრანზე:

```
<?php
#Andi Gutmans-ც მუშაობდა PHP-ს თავდაპირველი ვერსიის შექმნაზე
print "PHP";
echo "Hypertext Preprocessor";

?>
```

რიცხვით ცვლადებთან მუშაობა

1. შევქნათ ცვლადი და გავუტოლოთ ორი რიცხვის ჯამს, მაგალითად $example=7+2; და echo-ს საშუალებით შედეგი გამოვიტანოთ ეკრანზე.

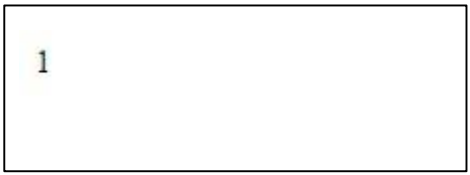
```
9
```

2. შევქმნათ ცვლადი და გაუტოლოთ ორი რიცხვის სხვაობას, მაგალითად $example=7-2; და შედეგი echo-ს საშუალებით გამოვიტანოთ ეკრანზე.

```
5
```

3. შექმენით ცვლადი, რომელიც შეასრულებს გაყოფას, მაგალითად $example=7/2, echo-ს საშუალებით გამოვიტანოთ შედეგი.

4. შექმენით ისეთი ცვლადი, რომელიც ნაშთს გამოიტანს ეკრანზე, მაგალითად $example=7%2.



5. შევქმნათ ცვლადი და შეკრება მოვახდინოთ ცვლადების გამოყენებით: $var=3; $var=$var+6; შედეგი გამოვიტანოთ ეკრანზე:

6. შევასრულოთ შეკრება ცვლადების საშუალებით, შემდეგნაირად: $var=3; $var+=6; შედეგი იქნება იგივე.

7. შევასრულოთ გამოკლება ცვლადების საშუალებით, შემდეგნაირად: $var=3; $var-=10; შედეგი გამოვიტანოთ ეკრანზე:

8. შევარულოთ გამრავლების ოპერაცია ცვლადების საშუალებით, შემდეგნაირად: $var=3; $var*=10; შედეგს ექნება შემდეგი სახე:

9. ცვლადების მეშვეობით შევასრულოთ გაყოფის ოპერაცია, $var=3; $var/=10; შედეგს ექნება შემდეგი სახე:

10. შევასრულოთ ნაშთის გამოტანის მოქმედება, ცვლადების მეშვეობით შემდეგი სახით $var=13; $var%=5; შედეგს ექნება შემდეგი სახე::

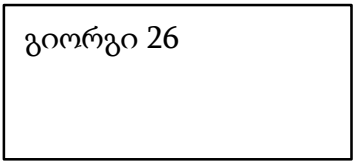
11. შექმენით ინკრიმენტიანი ცვლადი შემდეგნაირად: $var=3; $var++; მივიღებთ შედეგს:

12. შექმენით დეკრიმენტიანი ცვლადი შემდეგნაირად: $var=3; $var--; მივიღებთ შედეგს:

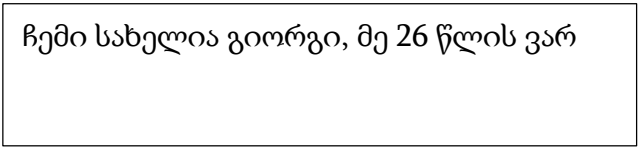
13. გამოვთვალოთ მრავალმოქმედებიანი მათემატიკური გამოსახულება: $var=5+3*2; შედეგს ასეთი სახე ექნება:

სტრიქონულ ცვლადებთან მუშაობა

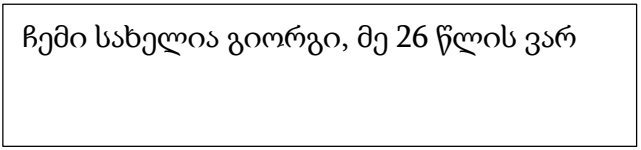
1. შევქმნათ ცვლადები $name და $age, ცვლად $name-ში შევიტანოთ პიროვნების სახელი, ხოლო $age-ში – ციფრი ან რიცხვი, პიროვნების ასაკის განსასაზღვრად, შემდეგ echo ოპერატორის საშუალებით შედეგი გამოვიტანოთ ეკრანზე. მაგალითად,



2. პირველი დავალების მსგავსად შევქმნათ ცვლადები $name და $age და echo ოპერატორისა და აპოსტროფის დახმარებით ეკრანზე პასუხი შემდეგი სახით გამოვიტანოთ.



3. შევქმნათ ცვლადები $name და $age. ორმაგ ბრჭყალებში მოვათვასოთ ტექსტური ობიექტებიც და ცვლადებიც ისე, რომ ეკრანზე გამოვიდეს შემდეგი შეტყობინება:



4. განსაზღვრეთ ორი სტრიქონული ცვლადი, რომელთა მნიშვნელობა იქნება თქვენი სახელი და გვარი. მონაცემები გამოიტანეთ სამი სხვადასხვა ვარიანტით. მონაცემთა გამოტანის დროს გამოიყენეთ აპოსტროფები, ბრჭყალები და heredoc-სინტაქსი.

5. განსაზღვრეთ ორი სტრიქონული ცვლადი, რომელთა მნიშვნელობა იქნება თქვენი სახელი და გვარი. ეკრანზე

გამოიტანეთ ინფორმაცია თქვენს შესახებ შემდეგი ნიმუშის მიხედვით:

ჩემი სახელია - გიორგი.
ჩემი გვარია - ბერძენიშვილი.
მე 21 წლის ვარ!

6. განსაზღვრეთ ორი სტრიქონული ცვლადი, რომელთა მნიშვნელობა იქნება თქვენი სახელი და გვარი. მოახდინეთ მონაცემთა გაერთიანება (კონკატენაცია) და ეკრანზე გამოიტანეთ ინფორმაცია თქვენ შესახებ შემდეგი ნიმუშის მიხედვით:

ჩემი სახელი და გვარია - გიორგი ბერძენიშვილი. მე 21 წლის ვარ!

პირობითი ოპერატორები

1. If ფუნციის და ცვლადის გამოყენებით (ცვლადის მნიშვნელობა იყოს 1-ზე მეტი) შევქმნათ ისეთი მათემატიკური გამოსახულება, სადაც თუ ცვლადის მნიშვნელობა მეტია 1-ზე, მაშინ ეკრანზე გამოიტანოს შემდეგი ტექსტი: „PHP დაპროგრამების საუკეთესო ენაა“.

2. იმავე პროგრამაში ცვლადის მნიშვნელობა შევცვალოთ 1-ის ტოლი ან 1-ზე ნაკლები მნიშვნელობით.

ამ შემთხვევაში შედეგად უნდა მივიღოთ ცარიელი ფანჯარა.

3. If ფუნციის და ცვლადის გამოყენებით (ცვლადის მნიშვნელობა იყოს 1-ზე მეტი) შევქმნათ ისეთი მათემატიკური გამოსახულება, სადაც თუ ცვლადის მნიშვნელობა მეტია 1-ზე, მაშინ ეკრანზე პირველ სტრიქონში გამოიტანოს შემდეგი ტექსტი: „PHP დაპროგრამების საუკეთესო ენაა“, ხოლო მეორე სტრიქონში კი – „PHP ცვლადის მნიშვნელობა ტოლია – “ და ცვლადის მნიშვნელობა. მეორე სტრიქონზე გადასასვლელად გამოვიყენოთ HTML ტეგი <br/>.

მაგალითად, შედეგად შეიძლება მივიღოთ:

PHP დაპროგრამების საუკეთესო ენაა
PHP ცვლადის მნიშვნელობა ტოლია – 5

4. If ფუნციის და ცვლადის გამოყენებით (ცვლადის მნიშვნელობა იყოს 1-ის ტოლი) შევქმნათ ისეთი მათემატიკური გამოსახულება, სადაც თუ ცვლადის მნიშვნელობა მეტია 1-ზე ან ტოლია 1-ის, მაშინ ეკრანზე პირველ სტრიქონში გამოიტანოს შემდეგი ტექსტი: „PHP დაპროგრამების საუკეთესო ენაა“, ხოლო მეორე სტრიქონში კი „PHP ცვლადის მნიშვნელობა ტოლია – “ და ცვლადის მნიშვნელობა. მეორე სტრიქონზე გადასასვლელად გამოვიყენოთ HTML ტეგი <br/>.

შედეგად მივიღებთ:

PHP დაპროგრამების საუკეთესო ენაა
PHP ცვლადის მნიშვნელობა ტოლია – 1

5. If ფუნციისა და ცვლადის გამოყენებით (ცვლადის მნიშვნელობა იყოს 1-ის ტოლი) შევქმნათ ისეთი მათემატიკური გამოსახულება, სადაც თუ ცვლადის მნიშვნელობა ტოლია 1-ის, მაშინ ეკრანზე პირველ სტრიქონში გამოიტანოს შემდეგი ტექსტი: „PHP დაპროგრამების საუკეთესო ენაა“, ხოლო მეორე სტრიქონში კი „PHP ცვლადის მნიშვნელობა ტოლია – “ და ცვლადის მნიშვნელობა. მეორე სტრიქონზე გადასასვლელად გამოვიყენოთ HTML ტეგი <br/>. PHP-ში ტოლობის შესამოწმებლად გამოიყენეთ ტოლობის ოპერატორი (==).

შედეგად მივიღებთ:

PHP დაპროგრამების საუკეთესო ენაა
PHP ცვლადის მნიშვნელობა ტოლია – 1

6. იმავე პროგრამაში ცვლადის მნიშვნელობა შევცვალოთ 1-ისაგან განსხვავებული მნიშვნელობით. ამ შემთხვევაში შედეგად უნდა მივიღოთ ცარიელი ფანჯარა.

7. If ფუნციისა და ცვლადის გამოყენებით (ცვლადის მნიშვნელობა იყოს 1-სგან განსხვავებული) შევქმნათ ისეთი მათემატიკური გამოსახულება, სადაც თუ ცვლადის მნიშვნელობა არ უდრის 1-ს, მაშინ ეკრანზე პირველ სტრიქონში გამოიტანოს შემდეგი ტექსტი: „PHP დაპროგრამების საუკეთესო ენაა“, ხოლო მეორე სტრიქონში კი თავდაპირველად – ცვლადის მნიშვნელობა, ხოლო შემდეგ ტექსტი – „ არ უდრის 1-ს“. PHP-ში ტოლობის შესამოწმებლად გამოიყენეთ ოპერატორი „არ უდრის“ (!=).

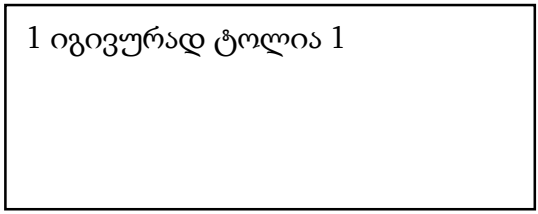
მაგალითად, შედეგად შეიძლება მივიღოთ:

```
PHP დაპროგრამების საუკეთესო ენაა
5 არ უდრის 1-ს
```

8. იმავე პროგრამაში ცვლადის მნიშვნელობა შევცვალოთ 1-ის ტოლი მნიშვნელობით. ამ შემთხვევაში შედეგად უნდა მივიღოთ ცარიელი ფანჯარა.

9. If ფუნციისა და ორი ცვლადის გამოყენებით, რომელთა მნიშვნელობაც ერთმანეთის ტოლია, იგივურად ტოლობის ოპერატორის (===) საშუალებით, შევადაროთ ეს ორი ცვლადი ერთმანეთს და თუ ისინი იგივურად ტოლია, შედეგად გამოვიტანოთ პირველი ცვლადის მნიშვნელობა, შემდეგ ტექსტი „ იგივურად ტოლია “ და მეორე ცვლადის მნიშვნელობა.

მაგალითად, შედეგად შეიძლება მივიღოთ:

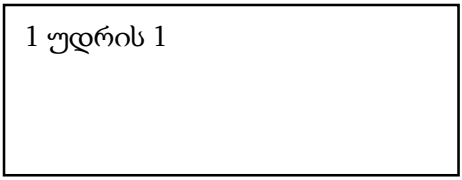


10. If ფუნციისა და ორი ცვლადის გამოყენებით და იგივურად ტოლობის ოპერატორის (===) საშუალებით, შევადაროთ ორი ცვლადი ერთმანეთს. ორივე ცვლადის რიცხვითი მნიშვნელობა იყოს ერთი და იგივე, მაგრამ ერთ-ერთს ეს მნიშვნელობა მივანიჭოთ როგორც ტექსტური, ხოლო მეორეს როგორც რიცხვითი მნიშვნელობა (მაგალითად, $a1='1' და $a2=1). თუ ისინი იგივურად ტოლი იქნება, შედეგად გამოვიტანოთ პირველი ცვლადის მნიშვნელობა, შემდეგ ტექსტი – „ იგივურად ტოლია “ და მეორე ცვლადის მნიშვნელობა.

ამ შემთხვევაში შედეგად უნდა მივიღოთ ცარიელი ფანჯარა.

11. იმავე პროგრამაში იგივურად ტოლობის ოპერატორი შევცვალოთ ტოლობის (==) ოპერატორით და გამოსატანი ტექსტი შევცვალოთ შემდეგნაირად „ უდრის “.

მაგალითად, შედეგად შეიძლება მივიღოთ:



12. If ფუნციის და ცვლადის გამოყენებით (ცვლადის მნიშვნელობა მოთავსებული იყოს 1-სა და 10-ს შორის) შევქმნათ ისეთი მათემატიკური გამოსახულება, რომ როცა ცვლადის მნიშვნელობა მეტია 1-ზე და ნაკლებია 10-ზე, მაშინ ეკრანზე გამოიტანოს შემდეგი ტექსტი: „PHP ცვლადის მნიშვნელობა მეტია 1-ზე და ნაკლებია 10-ზე“. PHP-ში ამ პირობის შესამოწმებლად გამოიყენეთ ოპერატორი „ლოგიკური და“ (&&).

შედეგად მივიღებთ:

PHP ცვლადის მნიშვნელობა მეტია 1-ზე და ნაკლებია 10-ზე

13. იგივე პროგრამაში ცვლადის მნიშვნელობა ავიღოთ 1-ზე ნაკლები ან 10-ზე მეტი. ამ შემთხვევაში შედეგად უნდა მივიღოთ ცარიელი ფანჯარა.

14. If ფუნციის და ცვლადის გამოყენებით (ცვლადის მნიშვნელობა ნაკლები იყოს 1-ზე ან მეტი 10-ზე) შევქმნათ ისეთი მათემატიკური გამოსახულება, რომ როცა ცვლადის მნიშვნელობა ნაკლებია 1-ზე და მეტია 10-ზე, მაშინ ეკრანზე გამოიტანოს შემდეგი ტექსტი: „PHP ცვლადის მნიშვნელობა ნაკლებია 1-ზე ან მეტია 10-ზე“. PHP-ში ამ პირობის შესამოწმებლად გამოიყენეთ ოპერატორი „ლოგიკური ან“ (||).

შედეგად მივიღებთ:

PHP ცვლადის მნიშვნელობა ნაკლებია 1-ზე ან მეტია 10-ზე

15. if-else კონსტრუქციის და ცვლადის გამოყენებით შევქმნათ ისეთი მათემატიკური გამოსახულება, რომ როცა ცვლადის მნიშვნელობა მეტია 0-ზე, მაშინ ეკრანზე გამოიტანოს ტექსტი: „მოცემული ცვლადი დადებითია“, ხოლო თუ ნაკლებია ან ტოლი 0-ის, მაშინ: „მოცემული ცვლადი უარყოფითია“.

თუ ცვლადის მნიშვნელობა დადებითია, მაშინ შედეგად მივიღებთ:

მოცემული ცვლადი დადებითია

ხოლო თუ ცვლადის მნიშვნელობა უარყოფითია, მაშინ შედეგად მივიღებთ:

მოცემული ცვლადი უარყოფითია

16. elseif კონსტრუქციის და ცვლადის გამოყენებით შევქმნათ ისეთი მათემატიკური გამოსახულება, რომ როცა ცვლადის მნიშვნელობა მეტია 0-ზე, მაშინ ეკრანზე გამოიტანოს ტექსტი: „მოცემული ცვლადის მნიშვნელობა “ <ცვლადის

მნიშვნელობა> „ დადებითია“, თუ ნაკლებია 0-ზე, მაშინ: „მოცემული ცვლადის მნიშვნელობა “ <ცვლადის მნიშვნელობა> „ უარყოფითია“ და თუ ტოლია 0-ის, მაშინ: „მოცემული ცვლადის მნიშვნელობა “ <ცვლადის მნიშვნელობა> „ არც დადებითია და არც უარყოფითი“.

თუ ცვლადის მნიშვნელობა დადებითია, მაგალითად, ტოლია 5-ის, მაშინ შედეგად მივიღებთ:

```
მოცემული ცვლადის მნიშვნელობა 5 დადებითია
```

თუ ცვლადის მნიშვნელობა უარყოფითია, მაგალითად, ტოლია -2-ის, მაშინ შედეგად მივიღებთ:

```
მოცემული ცვლადის მნიშვნელობა -2 უარყოფითია
```

ხოლო თუ ცვლადის მნიშვნელობა 0-ის ტოლია, მაშინ შედეგად მივიღებთ:

```
მოცემული ცვლადის მნიშვნელობა 0 არც დადებითია და
არც უარყოფითი
```

ციკლის ოპერატორები

1. გამოვიყენოთ ციკლი წინაპირობით while და მის სინტაქსში უტოლობის (<), ინკრიმენტისა და ცვლადის გამოყენებით, შევადგინოთ სკრიპტი, რომელიც 10-ჯერ გამოიტანს შემდეგ ტექსტს: „ცვლადის მნიშვნელობა გაუტოლდა – <რიცხვი>;“. ცვლადის მნიშვნელობა შეიცვალოს 0-დან 9-მდე. შედეგად უნდა მივიღოთ:

```
ცვლადის მნიშვნელობა გაუტოლდა – 0
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 2
ცვლადის მნიშვნელობა გაუტოლდა – 3
ცვლადის მნიშვნელობა გაუტოლდა – 4
ცვლადის მნიშვნელობა გაუტოლდა – 5
ცვლადის მნიშვნელობა გაუტოლდა – 6
ცვლადის მნიშვნელობა გაუტოლდა – 7
ცვლადის მნიშვნელობა გაუტოლდა – 8
ცვლადის მნიშვნელობა გაუტოლდა – 9
```

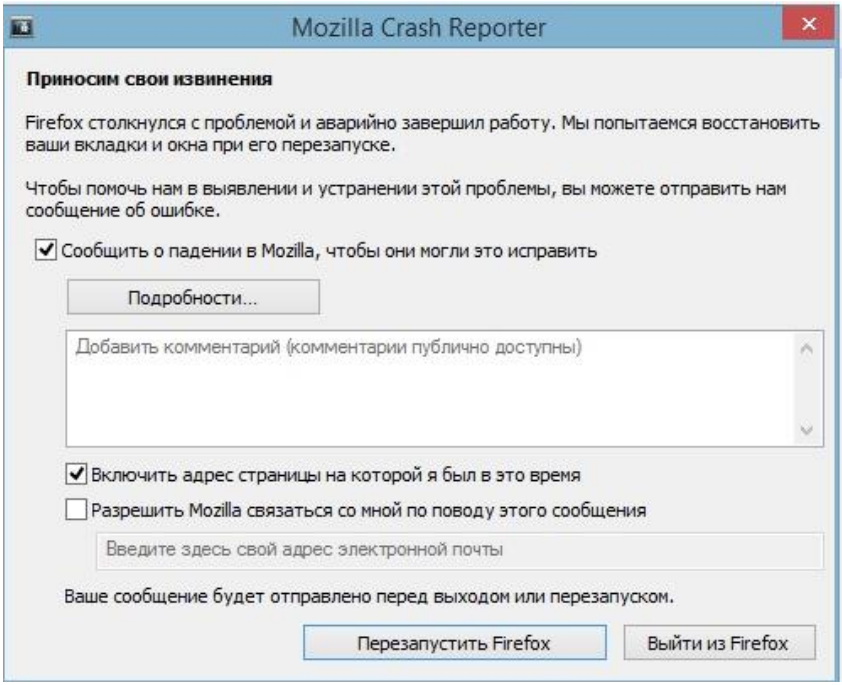
2. იმავე პროგრამაში უტოლობის „<“ ნიშანი შევცვალოთ „<=“ ნიშნით, ხოლო ცვლადის მნიშვნელობა – 1-დან 10-მდე. შედეგად მივიღებთ:

```
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 2
ცვლადის მნიშვნელობა გაუტოლდა – 3
ცვლადის მნიშვნელობა გაუტოლდა – 4
ცვლადის მნიშვნელობა გაუტოლდა – 5
ცვლადის მნიშვნელობა გაუტოლდა – 6
ცვლადის მნიშვნელობა გაუტოლდა – 7
ცვლადის მნიშვნელობა გაუტოლდა – 8
ცვლადის მნიშვნელობა გაუტოლდა – 9
ცვლადის მნიშვნელობა გაუტოლდა – 10
```

3. იგივე პროგრამა დავწეროთ ინკრიმენტის გარეშე. იკრიმენტის გარეშე ცვლადის მნიშვნელობა არ შეიცვლება და იგი ყოველთვის 1-ის ტოლი იქნება, რაც გამოიწვევს პროგრამის ჩაციკვლას. Google Chrome ბრაუზერში შედეგს ექნება შემდეგი სახე:

```
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 1
ცვლადის მნიშვნელობა გაუტოლდა – 1
```

ხოლო Mozilla Firefox ბრაუზერში გამოიტანს შემდეგ შეტყობინებას:



4. გამოვიყენოთ ციკლი შემდგომი პირობით do while და მის სინტაქსში უტოლობის (<), ინკრიმენტისა და ცვლადის გამოყენებით შევადგინოთ სკრიპტი, რომელიც 10-ჯერ გამოიტანს შემდეგ ტექსტს: „შედეგის ხაზი განმეორდა <რიცხვი>-ჯერ;“. ცვლადის მნიშვნელობა შეიცვალოს 0-დან 9-მდე. შედეგად უნდა მივიღოთ:

```
შედეგის ხაზი განმეორდა 0-ჯერ;
შედეგის ხაზი განმეორდა 1-ჯერ;
შედეგის ხაზი განმეორდა 2-ჯერ;
შედეგის ხაზი განმეორდა 3-ჯერ;
შედეგის ხაზი განმეორდა 4-ჯერ;
შედეგის ხაზი განმეორდა 5-ჯერ;
შედეგის ხაზი განმეორდა 6-ჯერ;
შედეგის ხაზი განმეორდა 7-ჯერ;
შედეგის ხაზი განმეორდა 8-ჯერ;
შედეგის ხაზი განმეორდა 9-ჯერ;
```

5. იმავე პროგრამაში ცვლადის მნიშვნელობა ავიღოთ 10-ზე მეტი, მაგალითად 11, შედეგი ასეთი იქნება:

```
შედეგის ხაზი განმეორდა 11-ჯერ;
```

6. გამოვიყენოთ ციკლი მთვლელით for და ცვლადის, უტოლობისა და ინკრიმენტის საშუალებით შევადგინოთ სკრიპტი, რომელიც გამოიტანს მთვლელის მნიშვნელობას შემდეგი სახით: „მთვლელის მნიშვნელობა ტოლია – <რიცხვი>-ის;“. მთვლელის მნიშვნელობა იცვლებოდეს 0-დან 9-მდე.

```
მთვლელის მნიშვნელობა ტოლია – 0-ის;
მთვლელის მნიშვნელობა ტოლია – 1-ის;
მთვლელის მნიშვნელობა ტოლია – 2-ის;
მთვლელის მნიშვნელობა ტოლია – 3-ის;
მთვლელის მნიშვნელობა ტოლია – 4-ის;
მთვლელის მნიშვნელობა ტოლია – 5-ის;
მთვლელის მნიშვნელობა ტოლია – 6-ის;
მთვლელის მნიშვნელობა ტოლია – 7-ის;
მთვლელის მნიშვნელობა ტოლია – 8-ის;
მთვლელის მნიშვნელობა ტოლია – 9-ის;
```

7. გამოვიყენოთ ციკლი მთვლელით for და ცვლადის, უტოლობისა და დეკრიმენტის საშუალებით შევადგინოთ სკრიპტი, რომელიც გამოიტანს მთვლელის მნიშვნელობას შემდეგი სახით: „მთვლელის მნიშვნელობა ტოლია – <რიცხვი>;“. მთვლელის მნიშვნელობა იცვლებოდეს 10-დან 1-მდე. შედეგს ასეთი სახე ექნება:

მთვლელის მნიშვნელობა ტოლია – 10-ის;
მთვლელის მნიშვნელობა ტოლია – 9-ის;
მთვლელის მნიშვნელობა ტოლია – 8-ის;
მთვლელის მნიშვნელობა ტოლია – 7-ის;
მთვლელის მნიშვნელობა ტოლია – 6-ის;
მთვლელის მნიშვნელობა ტოლია – 5-ის;
მთვლელის მნიშვნელობა ტოლია – 4-ის;
მთვლელის მნიშვნელობა ტოლია – 3-ის;
მთვლელის მნიშვნელობა ტოლია – 2-ის;
მთვლელის მნიშვნელობა ტოლია – 1-ის;

8. გამოვიყენოთ ციკლი მთვლელით for და ცვლადის, უტოლობისა და ინკრიმენტის საშუალებით შევადგინოთ სკრიპტი, რომელიც 1000-ჯერ გამეორდება და თან გამოიტანს შემდეგ ტექსტს
 ტეგის გამოყენების გარეშე: „შედეგი გამეორდება 1000-ჯერ!!!“. ამ სკრიპტის შედეგი იქნება:

შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!! შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!! შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!! შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!! შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!! შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი გამეორდება 1000-ჯერ!!!შედეგი

9. თუ იმავე პროგრამაში
 ტეგს გამოვიყენებთ, მაშინ ასეთ შედეგს მივიღებთ:

შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!

10.იგივე პროგრამაში if ფუნქციის, ინკრიმენტისა და break კონსტრუქციის ჩამატებით, შევადგინოთ ისეთი პროგრამა, რომლიც შედეგის 10-ჯერ გამოტანის შემდეგ გაჩერდება.
შედეგს ექნება ასეთი სახე:

შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!
შედეგი გამეორდება 1000-ჯერ!!!

11. გამოვიყენოთ ციკლი მთვლელით for, ცვლადი, if ფუნქცია, ინკრიმენტი, მოდულით გაყოფა (%) და continue კონსტრუქცია და შევადგინოთ სკრიპტი, რომელიც, როცა ციკლის მთვლელი იცვლება 0-დან 9-მდე, გამოიტანს მხოლოდ კენტ რიცხვებს.

```
1
3
5
7
9
```

ამორჩევის კონსტრუქციები

1. ამორჩევის სპეციალური switch-case კონსტრუქციის და ცვლადის გამოყენებით შევადგინოთ სკრიპტი, რომელიც იმუშავებს შემდეგნაირად: თუ ცვლადის მნიშვნელობა ტოლია 1-ის, მაშინ შედეგად ფანჯარაში გამოიტანს შეტყობინებას „Variable is equal to 1”; თუ ცვლადის მნიშვნელობა ტოლია 5-ის, მაშინ – „Variable is equal to 5”, ხოლო თუ ცვლადი 10-ის ტოლია, მაშინ – „Variable is equal to 10”. თუ ცვლადის მნიშვნელობა არც ერთ პირობას არ აკმაყოფილებს, მაშინ შედეგად ცარიელ ფანჯარას მივიღებთ.

2. ამორჩევის სპეციალური switch-case ალტერნატიული კონსტრუქციის (default-endswitch) და ცვლადის გამოყენებით შევადგინოთ სკრიპტი, რომელიც იმუშავებს შემდეგნაირად: თუ ცვლადის მნიშვნელობა ტოლია 1-ის, მაშინ შედეგად ფანჯარაში

გამოიტანს შეტყობინებას „Variable is equal to 1”; თუ ცვლადის მნიშვნელობა ტოლია 5-ის, მაშინ – „Variable is equal to 5”, ხოლო თუ ცვლადი 10-ის ტოლია, მაშინ – „Variable is equal to 10”. თუ ცვლადის მნიშვნელობა არც ერთ პირობას არ აკმაყოფილებს, მაშინ შედეგად გამოიტანოს შეტყობინება „Variable is not equal to 1, 5, 10.”.

სტრიქონული ფუნქციები

1. დავწეროთ ისეთი სკრიპტი, რომ print ოპერატორში განთავსებული Strlen[9] ფუნქციის გამოყენებით მოცემულ სტრიქონებში დავთვალოთ სიმბოლოების რაოდენობა და შედეგები გამოვიტანოთ:

ა. PHP;

ბ. PHP!;

გ. PHP Darwin!;

დ. PHP_Darwin_net;

შედეგი იქნება ასეთი:

```
3
4
11
14
```

[9] ფუნქცია strlen(string $st) ერთ-ერთი ყველაზე გამოსადეგი ფუნქციაა. შედეგად სტრიქონის სიგრძეს ანუ მასში შემავალი სიმბოლოების რაოდენობას აბრუნებს. სტრიქონი შეიძლება ნებისმიერ სიმბოლოს შეიცავდეს, მათ შორის ნულოვანი კოდით.

2. Strlen ფუნქციისა და ცვლადის გამოყენებით დავთვალოთ ტექსტურ მონაცემში ($St=’Hello’) შემავალ სიმბოლოთა რაოდენობა და შედეგი გამოვიტანოთ.
შედეგს ექნება ასეთი სახე:

```
5
```

3. Strlen ფუნქციისა და ცვლადის გამოყენებით, დავთვალოთ ტექსტურ მონაცემში ($St=’Hello’) შემავალ სიმბოლოთა რაოდენობა და შედეგი გამოვიყენოთ for ციკლის მთვლელად. ცვლადის, უტოლობისა და ინკრიმენტის საშუალებით შევადგინოთ სკრიპტი, რომელიც გამოიტანს მთვლელის მნიშვნელობას შემდეგი სახით:

```
0
1
2
3
4
```

4. Strpos ფუნქციის გამოყენებით გავიგოთ, რომელი პოზიციიდან იწყება პირველი სიტყვა „PHP“ წინადადებაში „1997 წელს PHP-2.0 ვერსია შეიქმნა, 1998 წელს – PHP-3.0, 1999 წელს –

PHP-4.0, 2004 წელს – PHP-5.0, 2006 წლიდან კი დაიწყო მუშაობა მეექვსე ვერსიაზე.“

შედეგს ექნება ასეთი სახე:

```
10
```

5. მეოთხე დავალებაში შერჩეულ წინადადებაში სიტყვა „PHP“ გვხვდება რამდენიმეჯერ. იმისათვის რომ გამოვთვალოთ მეორე „PHP“-ის პოზიცია წინადადებაში, საჭიროა strpos ფუნქციის მესამე არგუმენტად მივუთითოთ პირველი „PHP“-ის დასრულების პოზიცია, ანუ 13. ამ მითითებით პროგრამას ვეუბნებით, რომ გამოტოვოს პირველი 13 სიმბოლო და სიტყვა „PHP“-ის ძებნა მომდევნო სიმბოლოებში გააგრძელოს. შედეგად მივიღებთ:

```
46
```

6. წინადადებაში „1997 წელს PHP-2.0 ვერსია შეიქმნა, 1998 წელს – PHP-3.0, 1999 წელს – PHP-4.0, 2004 წელს – PHP-5.0, 2006 წლიდან კი დაიწყო მუშაობა მეექვსე ვერსიაზე.“ რამოდენიმეჯერ განმეორებული სიტყვის საწყისი პოზიციების ნომრების ერთდროულად გამოსათვლელად საჭიროა ჯერ Strlen ფუნქციის საშუალებით გამოვთვალოთ მთელი წინადადების სიგრძე და შემდეგ ციკლი while ოპერატორის დახმარებით

გამოვთვალოთ ყველა „PHP“-ის პოზიციის ნომერი. შედეგი შემდეგი სახით გამოვიტანოთ:

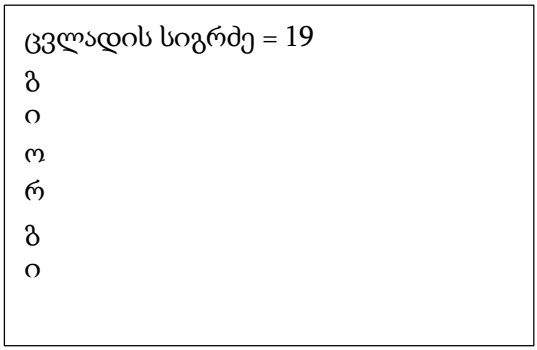
```
PHP-ის პოზიციაა – 10
PHP-ის პოზიციაა – 46
PHP-ის პოზიციაა – 67
PHP-ის პოზიციაა – 88
```

7. წინადადებაში „1997 წელს PHP-2.0 ვერსია შეიქმნა, 1998 წელს – PHP-3.0, 1999 წელს – PHP-4.0, 2004 წელს – PHP-5.0, 2006 წლიდან კი დაიწყო მუშაობა მეექვსე ვერსიაზე.“ substr ფუნქციის გამოყენებით ამოვჭრათ სიმბოლოების მიმდევრობები და შედეგი გამოვიტანოთ ეკრანზე. substr ფუნქციის მეორე და მესამე არგუმენტად რიგ-რიგობით მივუთითოთ შემდეგი რიცხვები: 1. 10 და 7; 2. 46 და 7; 3. 67 და 7; 4. 88 და 7: 5. -17 და 14. შედეგად მივიღებთ:

```
PHP-2.0
PHP-3.0
PHP-4.0
PHP-5.0
მეექვსე ვერსია
```

8. განსაზღვრეთ სტრიქონული ცვლადი, რომლის მნიშვნელობა იქნება თქვენი სახელი და გვარი. განსაზღვრეთ ამ ცვლადის სიგრძე და შესაბამისი მნიშვნელობა გამოიტანეთ ეკრანზე. ამ მნიშვნელობის ქვემოთ კი გამოვიტანოთ მხოლოდ

სახელი ვერტიკალურად. მაგალითად, თუ ჩვენი სახელი და გვარია გიორგი ბერძენიშვილი, მაშინ ამ სკრიპტის შესრულების შედეგად უნდა მივიღოთ:



9. განსაზღვრეთ სტრიქონული ცვლადი, რომლის მნიშვნელობა იქნება თქვენი სახელი და გვარი. გამოიყენეთ substr_replace ფუნქცია და სახელი ჩაანაცვლეთ შემდეგი სიმბოლოებით „**** “. მაგალითად, თუ ჩვენი სახელი და გვარია გიორგი ბერძენიშვილი, მაშინ ამ პროგრამის შესრულების შედეგად უნდა მივიღოთ:

10.განსაზღვრეთ სტრიქონული ორი ცვლადი, რომელთა მნიშვნელობა იქნება: პირველში – სიტყვა შვილზე დამთავრებული, ხოლო მეორე – ძე-ზე დამთავრებული გვარების მიმდევრობა. მაგალითად, პირველი იყოს „ბერძენიშვილი, გაფრინდაშვილი, ჩოლოყაშვილი“, ხოლო მეორე –

„ბარდაველიძე, მაღლაკელიძე, მანჯავიძე“. Str_replace ფუნქციის გამოყენებით, პირველში სიტყვა „შვილი“ და მეორეში – „ძე“ ჩავანაცვლოთ **** სიმბოლოებით. შედეგს ექნება ასეთი სახე:

ბერძენი****, გაფრინდა****, ჩოლოყა****
ბარდაველი****, მაღლაკელი****, მანჯავი****

11.განსაზღვრეთ სტრიქონული ცვლადი, რომელსაც მივანიჭებთ მნიშვნელობას „cisco, idark, darwin, darknet”. Strtoupper ფუნქციის დახმარებით ამ ცვლადის მნიშვნელობას შევუცვალოთ რეგისტრი. შედეგი ასეთი იქნება:

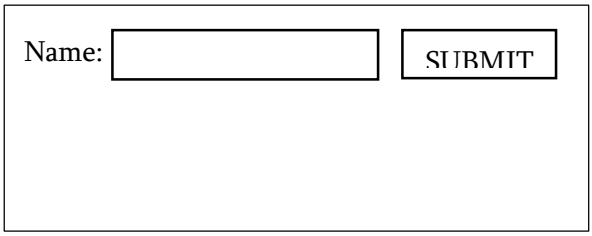
CISCO, IDARK, DARWIN, DARKNET

12.განსაზღვრეთ სტრიქონული ცვლადი, რომელსაც მივანიჭებთ მნიშვნელობას „CISCO, IDARK, DARWIN, DARKNET”. Strtoupper ფუნქციის დახმარებით ამ ცვლადის მნიშვნელობას შევუცვალოთ რეგისტრი. შედეგი ასეთი იქნება:

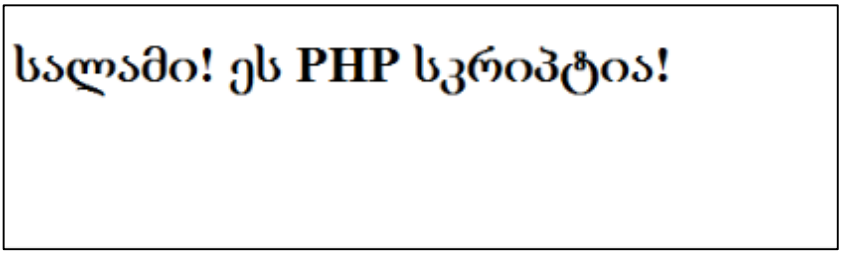
cisco, idark, darwin, darknet

ფორმები

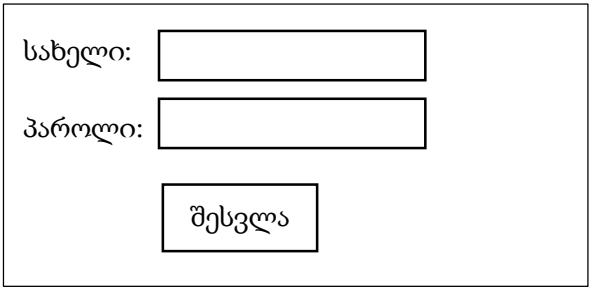
1. დაწერეთ სკრიპტი, რომელშიც პირველი დონის სათაურის ტეგებს შორის მოთავსებული იქნება სიტყვები: „სალამი! ეს PHP სკრიპტია!“. შექმენით მეორე სკრიპტიც, სადაც ფორმის შესაქმნელად გამოყენებული იქნება HTML-კოდი, რომელიც გამოიტანს ტექსტურ ველს და ღილაკს წარწერით „SUBMIT“.



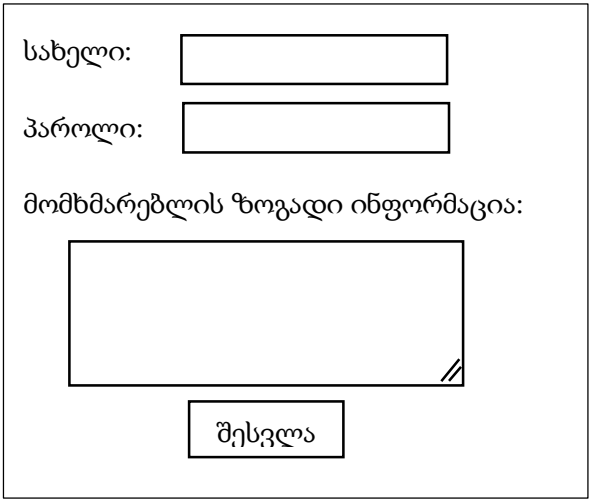
თუ Name ველში შეიტანთ ტექსტს და შემდეგ SUBMIT ღილაკს დააჭერთ ხელს, მაშინ უნდა მოხდეს შეტანილი ტექსტის გადაგზავნა პირველ სკრიპტში და ეკრანზე უნდა მივიღოთ შემდეგი:



2. დაწერეთ სკრიპტი, რომელიც გამოიტანს სახელისა და პაროლის ველს. ინფორმაციის გადაცემის მეთოდად გამოიყენეთ request მეთოდი. შედეგს უნდა ჰქონდეს შემდეგი სახე:

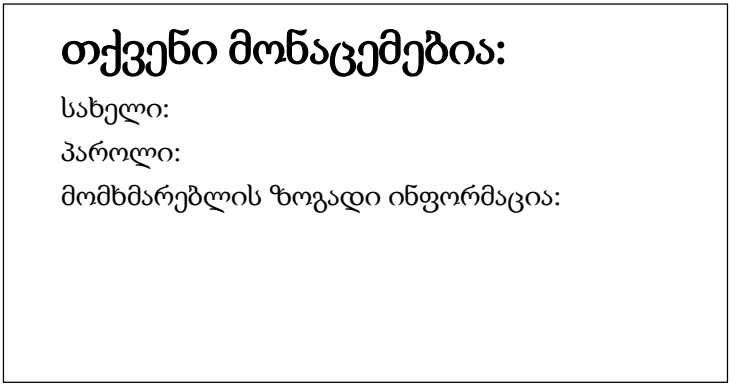


3. დაწერეთ სკრიპტი, რომელიც სახელისა და პაროლის ველთან ერთად ტექსტურ არესაც გამოიტანს. ინფორმაციის გადაცემის მეთოდად გამოიყენეთ request მეთოდი. შედეგს უნდა ჰქონდეს შემდეგი სახე:

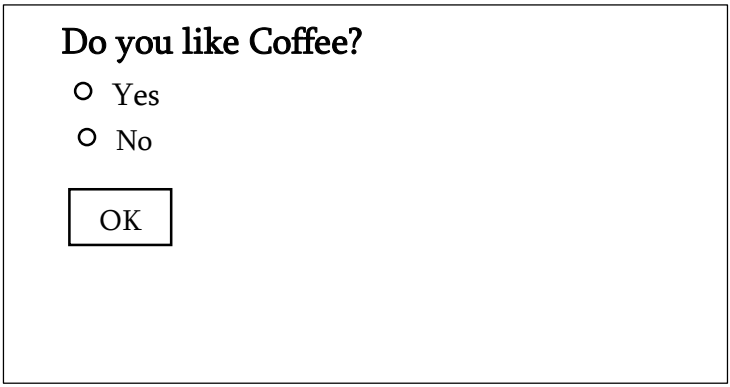


4. დაწერეთ სკრიპტი, რომელიც სახელისა და პაროლის ველთან ერთად ტექსტურ არესაც გამოიტანს. ინფორმაციის გადაცემის მეთოდად გამოიყენეთ post მეთოდი. დაწერეთ მეორე სკრიპტი, რომელიც შეტანილი მონაცემების გადაცემის შემდეგ

გამოიტანს შეტანილ მონაცემებს. მაგალითად, შედეგს უნდა ჰქონდეს შემდეგი სახე:

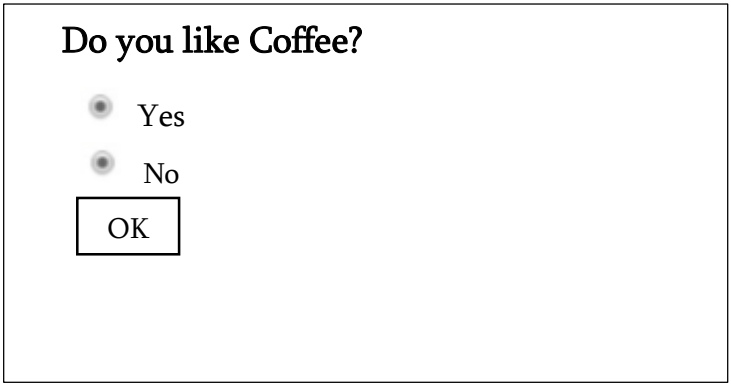


5. დაწერეთ სკრიპტი, რომელიც ფანჯარაში გამოიტანს შეკითხვას „Do you like Coffee?“, ხოლო პასუხები „Yes“ და „No“ იყოს გადამრთველი. ბოლოს ღილაკი „OK“. ინფორმაციის გადაცემის მეთოდად გამოიყენეთ post მეთოდი. ფორმას ჰქონდეს შემდეგი სახე:



6. იგივე სკრიპტში გადამრთველის თითოეულ ტეგში name არგუმენტს მივანიჭოთ სხვადასხვა სახელი. პროგრამის

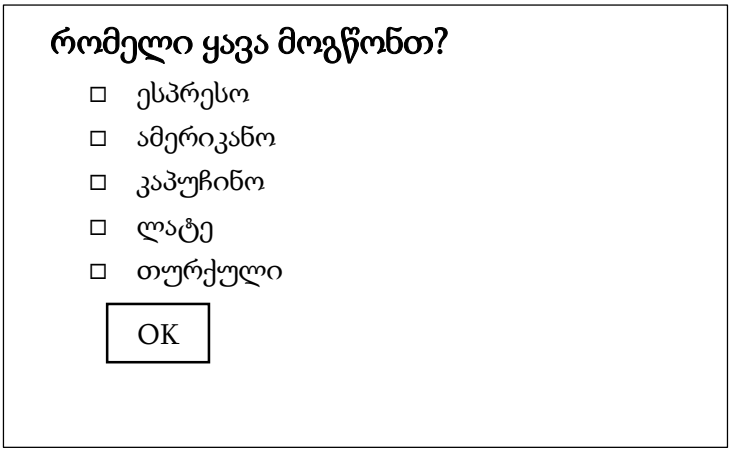
შესრულების შემდეგ ვნახავთ, რომ შესაძლებელი გახდება ორივე პასუხის ერთდროულად მონიშვნა.



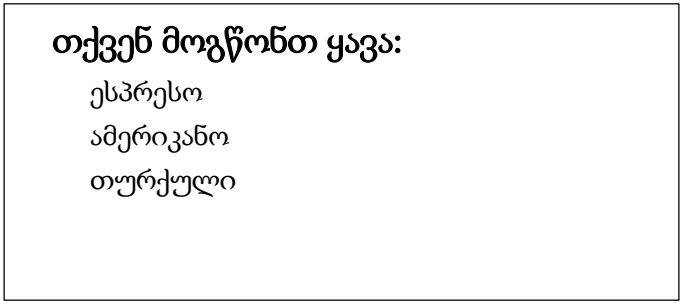
7. იგივე სკრიპტში PHP კოდი ჩასვით HTML კოდში და გავუშვათ შესრულებაზე.

8. დაწერეთ მეორე სკრიპტი, რომელიც წინა ამოცანაში პასუხის გაცემის შემთხვევაში გამოიტანს შედეგს. თუ პასუხი იყო „Yes“, მაშინ გამოიტანოს შემდეგი ტექსტი „You like coffee.“, ხოლო თუ პასუხი იყო „No“, მაშინ – „You do not like coffee“.

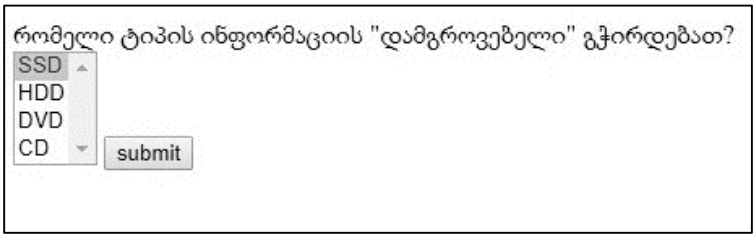
9. დაწერეთ სკრიპტი, რომელიც ფანჯარაში გამოიტანს შეკითხვას „რომელი ყავა მოგწონთ?“, ხოლო პასუხი იქნება ალმით მოსანიშნი ჩამონათვალი და ბოლოს ღილაკი „OK“. ინფორმაციის გადაცემის მეთოდად გამოიყენეთ post მეთოდი. ფორმას ჰქონდეს შემდეგი სახე:



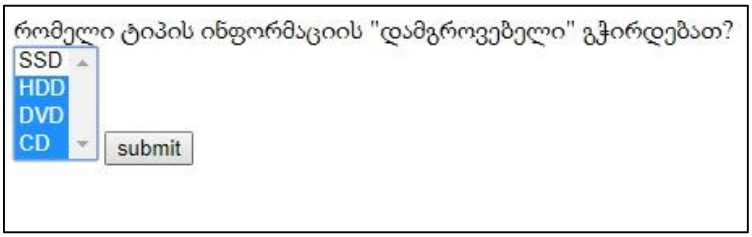
10. დაწერეთ მეორე სკრიპტი, რომელიც წინა ამოცანაში პასუხის გაცემის შემთხვევაში გამოიტანს შედეგს. მაგალითად:



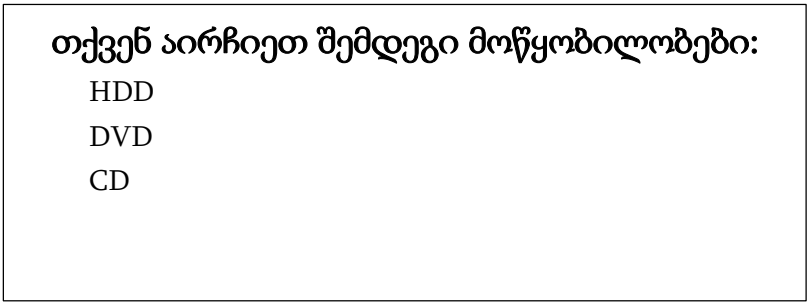
11. დაწერეთ სკრიპტი, რომელშიც მრავალი პარამეტრიდან მომხმარებელი სურვილის მიხედვით ამოირჩევს ერთ ან რამდენიმე დასახელების საგანს, რისთვისაც გამოიყენეთ <select> ტეგი. მიღებულ ფორმას ჰქონდეს შემდეგი სახე:



12. დაწერეთ მეორე სკრიპტი, რომელიც წინა ამოცანაში პასუხის გაცემის შემთხვევაში გამოიტანს შედეგს. მაგალითად: თუ მოვნიშნავთ შემდეგ მონაცემებს:



შედეგად გამოვიტანოთ შემდეგი პასუხი:



13. დაწერეთ სკრიპტი, სამომხმარებლო ფორმის შესაქმნელად, რომელიც მომხმარებლის სახელის, პაროლის და მასზე დამატებითი ინფორმაციის შესავსებ ველებს შეიცავს. შედეგს ჰქონდეს შემდეგი სახე:

14. იმავე სკრიპტში შევიტანოთ შესწორება და შეზღუდვა დავუწესოთ მომხმარებლის სახელს, თუ ეს ველი არ არის შევსებული, მაშინ მომხმარებელი არ დაარეგისტრიროს. ამასთან დავუმატოთ პაროლის გამეორების ველი. შედეგს ჰქონდეს შემდეგი სახე:

პასუხის გაგზავნის შემთხვევაში ეკრანზე გამოიტანოს შემდეგი ინფორმაცია:

15. შევქმნათ ისეთი სკრიპტი, რომლის დროსაც მომხმარებელი ვერ დარეგისტრირდება სისტემაში, თუ შეუვსებელად დატოვებს რომელიმე ველს. თუ რომელიმე ველი დარჩა შესავსები, პროგრამამ მოგვცეს შეტყობინება: „აუცილებლად შეავსეთ ყველა ველი“ და დარეგისტრირების ფორმა გამოიტანოს ხელახლა.

16. შევქმნათ ისეთი სკრიპტი, რომლის დროსაც მომხმარებელი ვერ დარეგისტრირდება სისტემაში, თუ შეუვსებელად დატოვებს რომელიმე ველს და ამასთან მომხმარებლის სახელის სიგრძე აღემატება 10 სიმბოლოს. თუ რომელიმე ველი დარჩა შესავსები, პროგრამამ მოგვცეს შეტყობინება: „აუცილებლად შეავსეთ ყველა ველი“, ხოლო თუ

მომხმარებლის სახელის სიგრძე აღემატება 10 სიმბოლოს, მაშინ მოგვცეს შეტყობინება: „მომხმარებლის სახელის სიგრძე აღემატება დასაშვებ ზღვარს“ და დარეგისტრირების ფორმა გამოიტანოს ხელახლა.

17. ზემოთ დაწერილ სკრიპტში შევიტანოთ ცვლილებები და ჩამოთვლილ შეზღუდვებს დავუმატოთ პაროლისა და მომხმარებლის შესახებ ინფორმაციის სიგრძის შეზღუდვა, აგრეთვე მოხდეს პაროლისა და პაროლის გამეორების ველების შედარება. შესაბამისად მოხდეს შეტყობინებების გამოტანა: „პაროლის სიგრძე აღემატება დასაშვებ ნორმას“, „პაროლის სიმბოლოები ერთმანეთს არ ემთხვევა“, „ინფორმაცია მომხმარებლის შესახებ ძალიან ვრცელია“, ხოლო თუ ინფორმაციის შეტანა სწორად განხორციელდა, მაშინ გამოიტანოს ინფორმაცია შემდეგი სახით:

თქვენი მონაცემები გაიგზავნა:

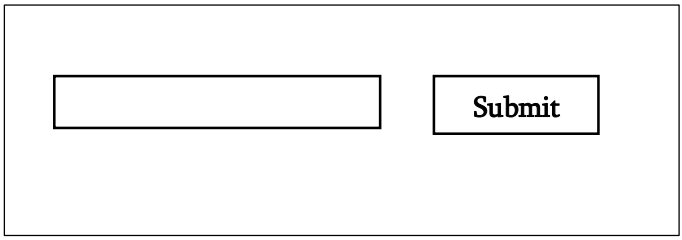
Username:

Password:

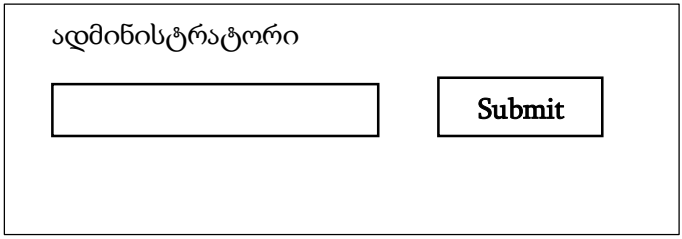
Gender:

Info:

18. დაწერეთ სკრიპტი, რომელიც გამოიტანს ტექსტურ ველს და ღილაკს შემდეგი ნიმუშის მიხედვით:



ტექსტურ ველში შევიტანოთ რაიმე სიტყვა, მაგალითად, ადმინისტრატორი და Submit ღილაკზე ხელის დაჭერით ფორმაში შეტანილი სიტყვა დაიბეჭდოს ფორმის ზემოთ:

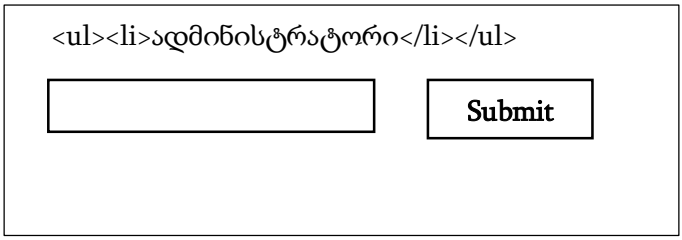


როგორც ვიცით, საიტის გატეხვის ერთ-ერთი ფორმაა, სამომხმარებლო ფორმაში შეტანილ ინფორმაციაზე html ტეგების დართვა, მაგალითად: … და მათ შორის გარკვეული ინფორმაციის შეტანა. Submit ღილაკზე დაჭერის შემდეგ, ტექსტურ ველში შეტანილი სიტყვა ტეგის შესაბამისი ფორმით დაიბეჭდება. სისტემის HTML ტეგებისგან დასაცავად გამოვიყენოთ htmlentities ფუნქცია.

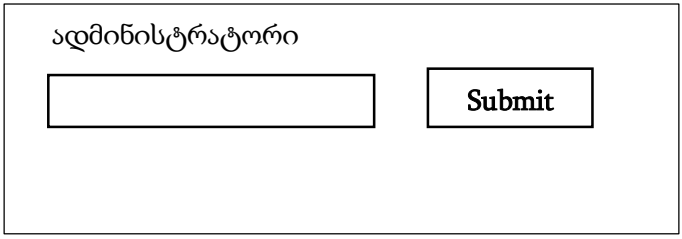
მაგალითად, თუ ტექსტურ ველში შევიტანთ სიტყვას შემდეგი სახით:

ადმინისტრატორი

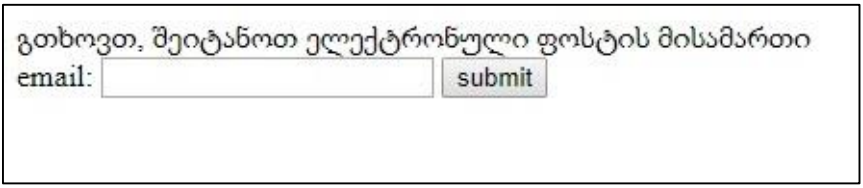
შედეგად მივიღებთ:



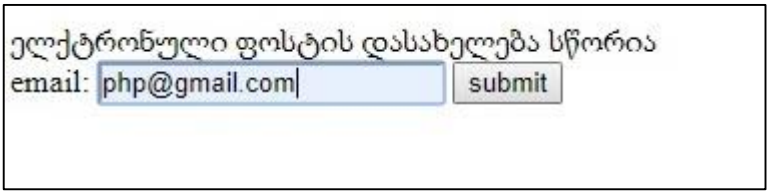
19. იმავე სკრიპტში ინფორმაციის დაცვისათვის გამოვიყენოთ strip_tags ფუნქცია. შედეგი უნდა მივიღოთ შემდეგი:



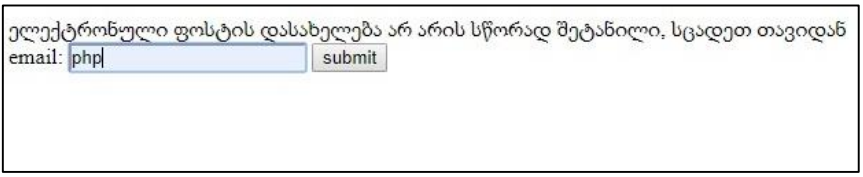
20. დაწერეთ სკრიპტი, რომელიც გამოიტანს ტექსტურ ველს და ღილაკს შემდეგი ნიმუშის მიხედვით:



ელექტრონული ფოსტის დასახელების სისწორის შესამოწმებლად გამოიყენეთ filter_var ფუნქცია. ელექტრონული ფოსტის მისამართის შეტანის შემდეგ, თუ დასახელება სწორია, მაშინ გამოიტანოს შემდეგი შეტყობინება, მაგალითად:



ხოლო, თუ მისამართის დასახელება არასწორია, მაშინ შეტყობინება: ელექტრონული ფოსტის დასახელება არ არის სწორად შეტანილი, სცადეთ თავიდან.



მასივები

1. შევქმნათ 4-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება საქართველოს ქალაქები: თბილისი; ქუთაისი; ბათუმი; სოხუმი.

დავწეროთ სკრიპტი და რიგ-რიგობით ეკრანზე გამოვიტანოთ მასივის ელემენტები (გავითვალისწინოთ რომ მასივის ელემენტების გადანომვრა 0-ით იწყება).

თუ მასივის გამოტანის დროს მისი ელემენტის ნომერს არ მივუთითებთ, მაშინ შედეგად მივიღებთ შემდეგ შეტყობინებას:

Notice: Array to conversioon in **C:\xampp\htdocs\EXEC\mag.php** on line **3** Array

2. შევქმნათ 4-ელემენტიანი ერთგანზომილებიანი მასივი ისე, რომ მასივის თითოეულ ელემეტს ცალკ-ცალკე მივანიჭოთ მნიშვნელობა. მასივის ელემენტები იყოს საქართველოს ქალაქები: თბილისი; ქუთაისი; ბათუმი; სოხუმი.

დავწეროთ სკრიპტი და რიგ-რიგობით ეკრანზე გამოვიტანოთ მასივის ელემენტები (გავითვალისწინოთ რომ მასივის ელემენტების გადანომვრა 0-ით იწყება).

3. შევქმნათ 4-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება საქართველოს ქალაქები: თბილისი; ქუთაისი; ბათუმი; სოხუმი.

დავწეროთ სკრიპტი და რიგ-რიგობით ეკრანზე გამოვიტანოთ მასივის ოთხივე ელემენტი.

შემდეგ მასივის ნომერ 3 ელემენტს მივანიჭოთ მნიშვნელობა „თელავი“ და ისევ გამოვიტანოთ ოთხივე ელემენტის მნიშვნელობა.

შედეგს ექნება ასეთი სახე:

```
თბილისი
ქუთაისი
ბათუმი
სოხუმი
თბილისი
ქუთაისი
ბათუმი
თელავი
```

4. შევქმნათ 4-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება საქართველოს ქალაქები: თბილისი; ქუთაისი; ბათუმი; სოხუმი.

დავწეროთ სკრიპტი და რიგ-რიგობით ეკრანზე გამოვიტანოთ მასივის ოთხივე ელემენტი.

შემდეგ მასივის ნომერ 3 ელემენტი unset ფუნქციის საშუალებით წავშალოთ და ისევ გამოვიტანოთ ოთხივე ელემენტის მნიშვნელობა. ვინაიდან მესამე ელემენტი წაშლილია, პროგრამა შეცდომას გამოიტანს.

შედეგს ექნება ასეთი სახე:

```
თბილისი
ქუთაისი
ბათუმი
სოხუმი
თბილისი
ქუთაისი
ბათუმი
Notice: Underfined offset: 3 in C:\xampp\htdocs\EXEC\mag.php on line 11
```

5. შევქმნათ 4-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება საქართველოს ქალაქები: თბილისი; ქუთაისი; ბათუმი; სოხუმი.

დავწეროთ სკრიპტი და რიგ-რიგობით ეკრანზე გამოვიტანოთ მასივის ოთხივე ელემენტი ციკლის for ფუნქციის საშუალებით.

შედეგს ექნება ასეთი სახე:

```
თბილისი
ქუთაისი
ბათუმი
სოხუმი
```

6. იმავე პროგრამაში შევიტანოთ მხოლოდ ერთადერთი ცვლილება, მასივის მეხუთე ელემენტად დავამატოთ „თელავი“ და გავუშვათ შესრულებაზე. შედეგი არ შეიცვლება, ვინაიდან ციკლში არ შევცვალეთ მთვლელის ზედა მნიშვნელობა.

7. შევქმნათ 5-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება საქართველოს ქალაქები: თბილისი; ქუთაისი; ბათუმი; სოხუმი; თელავი.

დავწეროთ სკრიპტი და რიგ-რიგობით ეკრანზე გამოვიტანოთ მასივის ყველა ელემენტი ციკლის for ფუნქციის საშუალებით, ხოლო მთვლელის ზედა მნიშვნელობის გამოსათვლელად გამოვიყენოთ count ფუნქცია.

შედეგს ექნება ასეთი სახე:

```
თბილისი
ქუთაისი
ბათუმი
სოხუმი
თელავი
```

8. იმავე პროგრამაში შევიტანოთ ცვლილებები, მასივის ელემენტებად დავამატოთ „ცხინვალი“, „გორი“, „ზუგდიდი“, „რუსთავი“, „ოზურგეთი“ და გავუშვათ შესრულებაზე. შედეგად მივიღებთ:

```
თბილისი
ქუთაისი
ბათუმი
სოხუმი
თელავი
ცხინვალი
გორი
ზუგდიდი
რუსთავი
ოზურგეთი
```

9. შევქმნათ 5-ელემენტიანი ერთგანზომილებიანი ასოცირებული მასივი array ფუნქციის საშუალებით, რომლის გასაღები იქნება ოჯახის წევრთა სახელები და მნიშვნელობა კი – ასაკი: შვილი => 19; მამა => 42; დედა => 39; ბაბუა => 65; ბებია => 61.

დავწეროთ სკრიპტი და ეკრანზე გამოვიტანოთ ოჯახის წევრთა ასაკი. მივიღებთ შედეგს:

```
19
42
39
65
61
```

10. იგივე პროგრამაში შევიტანოთ ცვლილებები ისე, რომ ეკრანზე მივიღოთ შემდეგი შედეგი:

```
შვილი 19 წლისაა.
მამა 42 წლისაა.
დედა 39 წლისაა.
ბაბუა 65 წლისაა.
ბებია 61 წლისაა.
```

11. შევქმნათ 5-ელემენტიანი ერთგანზომილებიანი ასოცირებული მასივი array ფუნქციის საშუალებით, რომლის გასაღები იქნება ოჯახის წევრთა სახელები და მნიშვნელობა კი – ასაკი: შვილი => 19; მამა => 42; დედა => 39; ბაბუა => 65; ბებია => 61.

დავწეროთ სკრიპტი, რომელშიც გამოყენებული იქნება foreach ოპერატორი და ეკრანზე გამოვიტანოთ ოჯახის წევრთა ასაკი. მივიღებთ შედეგს:

```
19
42
39
65
61
```

12. იმავე პროგრამაში შევიტანოთ ცვლილებები ისე, რომ ეკრანზე მივიღოთ შემდეგი შედეგი:

შვილი 19 წლისაა.
მამა 42 წლისაა.
დედა 39 წლისაა.
ბაბუა 65 წლისაა.
ბებია 61 წლისაა.

13. შევქმნათ 4-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება: ერთი; ორი; სამი; ოთხი.

დავწეროთ სკრიპტი და ეკრანზე გამოვიტანოთ მასივის ელემენტები implode ფუნქციის დახმარებით ისე, რომ სიტყვებს შორის იყოს თითო ინტერვალი.

შედეგს ექნება ასეთი სახე:

ერთი ორი სამი ოთხი

14. შევქმნათ 4-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება: ერთი; ორი; სამი; ოთხი.

დავწეროთ სკრიპტი და ეკრანზე გამოვიტანოთ მასივის ელემენტები implode ფუნქციის დახმარებით ისე, რომ სიტყვებს შორის ჩამატებული იყოს სხვადასხვა სიმბოლოები, მაგალითად, „+“, „-„, „X“, „ ,“.

შედეგს ექნება ასეთი სახე:

```
ერთი+ორი+სამი+ოთხი
ერთი-ორი-სამი-ოთხი
ერთიXორიXსამიXოთხი
ერთი,ორი,სამი,ოთხი
```

15. შევქმნათ 4-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება: ერთი; ორი; სამი; ოთხი.

დავწეროთ სკრიპტი და ეკრანზე გამოვიტანოთ მასივის ელემენტები explode ფუნქციის დახმარებით ისე, რომ ამ ფუნქციის პირველ არგუმენტად მივუთითოთ სიმბოლო „ ,“ (მძიმე), ხოლო მესამე არგუმენტი არ მივუთითოთ. მასივის გამოსატანად გამოვიყენოთ ოპერატორი print_r.

შედეგს ექნება ასეთი სახე:

```
Array ( [0] =>ერთი, [1] =>ორი, [2] =>სამი, [3] =>ოთხი )
```

16. შევქმნათ 4-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება: ერთი; ორი; სამი; ოთხი.

დავწეროთ სკრიპტი და ეკრანზე გამოვიტანოთ მასივის ელემენტები explode ფუნქციის დახმარებით ისე, რომ ამ ფუნქციის პირველ არგუმენტად მივუთითოთ სიმბოლო „ ,“

(მძიმე), ხოლო მესამე არგუმენტად რიგ-რიგობით მივუთითოთ: 0, 2, -1. მასივის გამოსატანად გამოვიყენოთ ოპერატორი print_r.

შედეგს ექნება ასეთი სახე:

```
Array ( [0] =>ერთი,ორი,სამი,ოთხი )
Array ( [0] =>ერთი, [1] =>ორი,სამი,ოთხი )
Array ( [0] =>ერთი, [1] =>ორი, [2] =>სამი )
```

17. შევქმნათ 5-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება საქართველოს ქალაქები: თბილისი; ქუთაისი; ბათუმი; სოხუმი; თელავი.

დავწეროთ სკრიპტი და sort ფუნქციისა და foreach ოპერატორის საშუალებით მასივის ელემენტები დავალაგოთ ალფავიტის მიხედვით და შედეგი გამოვიტანოთ ეკრანზე.

შედეგს ექნება ასეთი სახე:

```
ბათუმი
თბილისი
თელავი
სოხუმი
ქუთაისი
```

18. იმავე პროგრამაში sort ფუნქცია შევცვალოთ rsort ფუნქციით, შედეგად მივიღებთ:

```
ქუთაისი
სოხუმი
თელავი
თბილისი
ბათუმი
```

19. შევქმნათ 5-ელემენტიანი ერთგანზომილებიანი ასოცირებული მასივი array ფუნქციის საშუალებით, რომლის გასაღები იქნება ოჯახის წევრთა სახელები და მნიშვნელობა კი ასაკი: შვილი => 19; მამა => 42; დედა => 39; ბაბუა => 65; ბებია => 61.

დავწეროთ სკრიპტი და ეკრანზე გამოვიტანოთ ოჯახის წევრთა ასაკი დალაგებული ზრდადობით (sort ფუნქცია) და კლებადობით (rsort ფუნქცია). მივიღებთ შედეგს:

```
19
39
42
61
65
```

და

```
65
61
42
39
19
```

20. იმავე მასივისა და ksort ფუნქციის გამოყენებით მასივის ელემენტები დავალაგოთ მისი გასაღებების მიხედვით და გამოვიტანოთ როგორც გასაღების დასახელება, ასევე შესაბამისი მნიშვნელობაც. შედეგს ექნება შემდეგი სახე:

```
ბაბუა 65
ბებია 61
დედა 39
მამა 42
შვილი 19
```

21. შევქმნათ 5-ელემენტიანი ერთგანზომილებიანი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება საქართველოს ქალაქები: თბილისი; ქუთაისი; ბათუმი; სოხუმი; თელავი.

დავწეროთ სკრიპტი და array_slice ფუნქციის დახმარებით ამოვჭრათ მასივის სამი ელემენტი დაწყებული მეორედან და ეკრანზე გამოვიტანოთ მიღებული მასივი.

შედეგს ექნება ასეთი სახე:

```
ქუთაისი
ბათუმი
სოხუმი
```

22. შევქმნათ 5-ელემენტიანი ერთგანზომილებიანი ორი მასივი array ფუნქციის საშუალებით, რომლის ელემენტებიც იქნება საქართველოს ქალაქები: ერთი – თბილისი; ქუთაისი; ბათუმი; სოხუმი; თელავი და მეორე – ცხინვალი; გორი; ზუგდიდი; რუსთავი; ოზურგეთი.

დავწეროთ სკრიპტი და array_merge() ფუნქციის დახმარებით გავაერთიანოთ ეს ორი მასივი. შედეგს ექნება ასეთი სახე:

```
თბილისი
ქუთაისი
ბათუმი
სოხუმი
თელავი
ცხინვალი
გორი
ზუგდიდი
რუსთავი
ოზურგეთი
```

შინაარსი

რედაქტორი ნ. ქაფიანიძე

გადაეცა წარმოებას 09.07.2019. ხელმოწერილია დასაბეჭდად 30.09.2019. ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 21,5. №3111.

საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, თბილისი, კოსტავას 77

Verba volant, scripta manent

ი.მ. „გოჩა დალაქიშვილი“,
თბილისი, ვარკეთილი 3, კორპ. 333, ბინა 38