გელა ღვინეფაძე

თანამედროვე ტექნოლოგიები ვებდანართების დასაპროექტებლად

„ტექნიკური უნივერსიტეტი“

საქართველოს ტექნიკური უნივერსიტეტი

გელა ღვინეფაძე

თანამედროვე ტექნოლოგიები ვებდანართების დასაპროექტებლად

რეკომენდებულია საქართველოს
ტექნიკური უნივერსიტეტის
სარედაქციო-საგამომცემლო საბჭოს
მიერ, 29.03.2017, ოქმი №1

თბილისი
2017

უაკ 681.3.06

სახელმძღვანელოში განხილულია ვებდანართების შესაქმნელად განკუთვნილი თანამედროვე სერვერული საშუალებები: დაპროგრამების ენა PHP, ასევე - Javascript ენის შესაძლებლობების გაფართოებისათვის დამუშავებული ფრეიმვორკული ტექნოლოგია AngularJS და სხვ.

სახელმძღვანელო განკუთვნილია ინფორმატიკის სპეციალობათა შემსწავლელი სტუდენტების, მაგისტრანტებისა და ამ საკითხებით დაინტერესებული პირებისთვის.

რეცენზენტები: საქართველოს ტექნიკური უნივერსიტეტის
ინფორმატიკის და მართვის სისტემების
ფაკულტეტის მართვის ავტომატიზებული სისტემების
დეპარტამენტის პროფესორი გია სურგულაძე,

საქართველოს ტექნიკური უნივერსიტეტის
ინფორმატიკის და მართვის სისტემების
ფაკულტეტის მართვის სისტემების დეპარტამენტის
პროფესორი ვლადიმერ კეკენაძე

ISBN 978-9941-20-876-8

http://www.gtu.ge

შესავალი

არცთუ დიდი ხნის წინ Web კვანძებიდან მომხმარებელს ადგილზე, ძირითადად, სტატიკური HTML ფაილები გადმოეცემოდა, რომელთაც ის ბროუზერის მეშვეობით მხოლოდ კითხულობდა. დღეს კი WEB სივრცეში უმეტესწილად დინამიკური WEB ფურცლები, ანუ DHTML ფაილები გადაიგზავნება. მათი აწყობა კი ხდება WEB კვანძების სერვერზე განთავსებული WEB პროგრამების გაშვებით, მომხმარებლის მიერ საკუთარი კომპიუტერიდან სისტემასთან დიალოგის შედეგად.

კლიენტური ენებისაგან განსხვავებით, როგორიცაა, მაგალითად, ჩვენთვის უკვე კარგად ცნობილი HTML და Javascript ენები, სერვერულ ენაზე დაწერილი პროგრამა სრულდება სწორედ სერვერზე და ის კლიენტს უგზავნის მის მიერ მოთხოვნილ მონაცემებს, მეტწილად მონაცემთა ბაზის ცხრილებიდან ამოკრეფილს და ამა თუ იმ ჭრილში ფორმირებულს. ასე რომ, როდესაც ადგილზე გადმოგზავნილი ინფორმაციის შიგთავსს ვათვალიერებთ, მასში აღმოვაჩენთ მხოლოდ შედეგს (HTML ფაილის სახით) და არა – სერვერული ენის ოპერატორებს, რომელთაც მათზე დაკისრებული ფუნქცია უკვე შეასრულეს.

აღვნიშნავთ, რომ ინტერნეტში ფუნქციონირებისათვის განკუთვნილი სერვერული პროგრამების შესაქმნელად სადღეისოდ WEB დაპროგრამების ყველაზე პოპულარული ენა გახლავთ PHP. მისი დახმარებით განსაკუთრებით მარტივდება სერვერებზე განლაგებულ მონაცემთა ბაზებთან ურთიერთობა (სერვერული ტექნოლოგიებიდან სახელმძღვანელოში ჯერ განვიხილავთ ამ ენას).

თავდაპირველად PHP (personal home page) ენა შეიქმნა, როგორც მაკროსების კრებული მარტივი სახის პირადი WEB ფურცლების ფორმირებისთვის. შემდეგ კი ის იქცა დაპროგრამების სრულფასოვან ენად, რომელზე დაწერილ კონსტრუქციებს განათავსებენ სერვერზე ატვირთული ფაილების HTML ტექსტებში. WEB ფურცლის გამოძახებისას PHP-ის პრეპროცესორის მიერ *(იხ. დანართი)* ადგილზე ხდება ამ კონსტრუქციების *(პროგრამული ფრაგმენტების)* დამუშავება და HTML-ის არსებულ ფრაგმენტებს შორის ჩაემატება შესაბამისი ინფორმაცია, მაგალითად: ბაზიდან ამოღებული მონაცემები, სერვერზე ფორმირებული გრაფიკული გამოსახულებები და სხვ. ამის გამო PHP-ს ჰიპერტექსტის პრეპროცესორადაც მოიხსენიებენ.

PHP-ის ესა თუ ის ვერსია მისაერთებელი მოდულის სახით დღეს კომპილირდება ყველაზე პოპულარული WEB სერვერ-პროგრამა Apache-სთვის. სერვერზე PHP პროგრამები, უმეტესწილად, ურთიერთობენ MySQL მონაცემთა ბაზასთან *(რომელიც უფასოდ ვრცელდება)*. ასე რომ, Apache, MySQL და PHP ტრიადას დღევანდელ დღეს, შეიძლება ითქვას,

კონკურენტი არ ჰყავს. თუმცა, ცხადია, PHP-ს მუშაობა შეუძლია სხვა WEB გარემოში და სხვა ბაზებთანაც.

უმეტეს წილად, PHP პროგრამას საბოლოო, შესრულებად სახეს ინტერპრეტატორი აძლევს, რომელსაც, პროცესის დაჩქარების მიზნით, მიეწოდება ტრანსლატორის მიერ ამ პროგრამის ბაიტ-კოდად გადა-მუშავების შედეგი.

შენიშვნა: *გავიხსენოთ, რომ ტრანსლატორი არის სისტემური პროგრამა, რომელიც ამა თუ იმ ენაზე დაწერილ კომპიუტერულ პროგრამას თარგმნის სხვა ენაზე, ხოლო კომპილატორი იმ ტიპის ტრანსლატორია, რომელსაც მიწოდებული პროგრამა გადაჰყავს ორობით, შესრულებად კოდში. ამრიგად, კომპილატორი თავიდან ბოლომდე ამუშავებს გადმოცემულ კოდს და მხოლოდ ამის შემდეგ ასრულებს მას. რაც შეეხება ინტერპრეტატორს, ესეც მთარგმნელი სისტემური პროგრამაა, ოღონდ ის სათითაოდ, თანმიმდევრულად ამუშავებს ოპერატორებს და ასრულებს მათ შეცდომის არ არსებობის შემთხვევაში.*

აღსანიშნავია, რომ CGI მიდგომასთან შედარებით *(იხ. დანართი)* PHP ფლობს მნიშვნელოვან უპირატესობას - მისი ინტერპრეტატორი ფუნქციო-ნირებს სერვერთან უფრო მჭიდრო ინტეგრაციის პირობებში, კერძოდ, საჭირო აღარ არის თითოეული პროგრამის შესასრულებლად სისტემის ხელახლა გაშვება, რაც მნიშვნელოვნად ზრდის მუშაობის სისწრაფეს. საერთოდ კი, შესაძლებელი არის PHP პროგრამის კომპილირებაც, რაც კიდევ უფრო ამაღლებს სისტემის მწარმოებლურობას.

ამასთან ერთად, PHP-ს სხვა მნიშვნელოვანი ღირსებებიც გააჩნია:

- პროდუქტის ღიაობა (გამოცდილი პროგრამისტებისათვის იძლევა ენაში ახალი შესაძლებლობების დამატების საშუალებას);
- პროგრამების დაწერის გაადვილება HTML ტექსტისა და თვით პროგრამული ნაწილის განცალკევების გამო;
- პროგრამების ერთი ოპერაციული სისტემიდან სხვაში გადატანის სიმარტივე;
- ენაში ჩადებული საშუალებების დახმარებით განსაკუთრებით ადვილდება სერვერზე განლაგებულ მონაცემთა ბაზასთან ურთიერთობა.
- პროგრამების შესრულების საკმარისი სისწრაფე;
- უსაფრთხოების დაცვის თვალსაზრისით, ხაზგასასმელია ის გარემოება, რომ კლიენტის მხარეს მხოლოდ HTML კოდი მიეწოდება – პროგრამის კოდის ნახვა მას არ შეუძლია. ამასთან ერთად, ინფორმაციის დასაშიფრავად ენაში გათვალისწინებულია სპეციალური ფუნქციები;
- დაბოლოს, ეს პროგრამული პროდუქტი დამპროექტებელთა ნებართვით უფასოდ ვრცელდება.

P H P-ის დაყენება

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

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

შევისწავლოთ, თუ როგორ ხდება PHP-ის დაყენება Apache Web სერვერთან სამუშაოდ კომპიუტერის მართვისას Linux ოპერაციული სისტემის მიერ.

შენიშვნა: *საერთოდ, შესაძლებელია Apache-ისთან უკვე მიერთებული PHP-ის ვარიანტის მოძიებაც; მეტიც, დღეს უპირატესობას ანიჭებენ ამ ენასთან სამუშაოდ განკუთვნილ, უკვე მთლიანად აწყობილი პაკეტების გამოყენების ხერხს (იხ. ქვემოთ), მაგრამ, როცა ამ პროცესს თვითონ განვაგებთ, არჩევანი ფართოვდება – ამ შემთხვევაში სისტემურ პარამეტრებს ჩვენვე ვანიჭებთ უფრო სასურველ მნიშვნელობებს.*

კომპიუტერზე PHP-ის დაყენების უფლება აქვს მხოლოდ ადმინისტრატორს *(სუპერმომხმარებელს).* PHP-ის, როგორც Apache-ისათვის მოდულის, კომპილირება ორი გზითაა შესაძლებელი:

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

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

/www/bin/httpd -1

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

PHP-ის ჩამოტვირთვის შედეგად გადმოგვეცემა gzip უტილიტით შეკუმშული Php-5.0.4pll.tar.gz დისტრიბუტიული ფაილი.

მისი გაშლა შემდეგი ბრძანებით ხდება:

tar -xvzf php-5.0.4pll.tar.gz

გადავდივართ PHP დისტრიბუტივის კატალოგში:

cd../php-5.0

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

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

ბოლო მნიშვნელობა მიუთითებს იმ კატალოგზე, რომელშიც Apache Web სერვერია დაყენებული, mysql ბაზისთვის კი იგულისხმება, რომ ის დუმილით გათვალისწინებულ კატალოგშია დაყენებული. მაგრამ, თუკი ეს ასე არ არის, საჭიროა ინფორმაციის ფიქსირება ამ განსხვავებული კატალოგის შესახებაც:

```
--with-mysqli=/path/to/dir
```

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

Apache სერვერის კონფიგურირება

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

```
AddType application / x-httpd-php.php

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

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

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

შესაძლებელია ამ ფაილის განთავსება მიმდინარე კატალოგშიც. ასეთ შემთხვევაში მას ექნება ლოკალური მოქმედების არეალი. ჯერჯერობით კი ჩვენ ვკმაყოფილდებით ამ ფაილის პარამეტრებისთვის მინიჭებული სტანდარტული მნიშვნელობებით.

php.ini ფაილში შეიძლება მნიშვნელობის განსაზღვრა შემდეგი დირექტივისთვისაც: short_open_tag=On (True,Yes), რაც უზრუნველყოფს php-ბლოკების გაფორმებას <? *ბლოკი* ?> სახით.

ამ შესაძლებლობის გამორთვა ხდება დირექტივისთვის Off (False,No) მნიშვნელობის მინიჭებით.

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

```
?php
?>
```

დასასრულს, როგორც ზემოთ უკვე აღვნიშნეთ, PHP ენით სარგებლობისათვის, ბოლო ხანებში, დიდი პოპულარობა მოიპოვეს სპეციალიზებულმა პაკეტებმა, რომლებშიც ზემოთ აღნიშნული და სხვა სახის სერვისული სამუშაოები უკვე ჩატარებულია და ტრიადა (Apache, MySQL, PHP) გაწყობილი სახით ეწოდება მომხმარებელს.

ასეთი პაკეტებიდან გავეცნოთ რამდენიმე მათგანს, სადღეისოდ ყველაზე პოპულარულებს.

ესენია:

XAMP (XAMPP), WAMP, DENVER.

პაკეტების დაყენება

DENVER

დავიწყოთ დაყენების თვალსაზრისით ყველაზე მარტივი და რესურსების მიმართ ნაკლებად მომთხოვნი პაკეტიდან.

DENVER-ის სახელწოდება მომდინარეობს შემდეგი რუსული სახელებიდან - Джентльменский набор Web-разработчика («Д.н.w.р»).

რუსული ჯგუფის მიერ დამუშავებული ეს პროექტი მოიცავს ლოკალურ სერვერს და პროგრამულ გარსს.

Web-დამპროექტებლებს მისი მეშვეობით შეუძლიათ ადგილზე, ინტერნეტში გაუსვლელად იმუშაონ პროექტზე.

პაკეტის გადმოსაწერად (DENVER 3 ვერსიისათვის ეს უფასოა) უნდა მივაკითხოთ საიტს http://www.denwer.ru/.

დიალოგში ვირჩევთ სასურველ ოფციას.

როგორც უკვე აღვნიშნეთ, კომპიუტერზე პაკეტის ინსტალირება ძალიან მარტივად ხდება.

WAMP

Wamp პაკეტის სახელწოდება შეირჩა, როგორც მისი კომპონენტების აბრევიატურა:

Windows - Microsoft კომპანიის საყოველთაოდ ცნობილი ოპერაციული სისტემაა; **A**pache – პოპულარული ლოკალური ვებსერვერი; **M**ySQL –

ინტერნეტში ფართო გამოყენების მქონე მონაცემთა ბაზა; **P**HP – დაპროგრამების ენა.

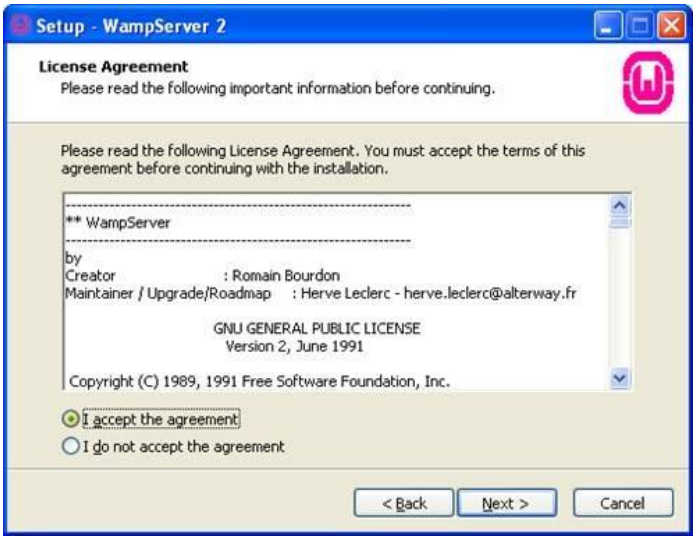
WAMP სისტემის დასაყენებლად მივაკითხოთ შემდეგ მისამართს - http://www.wampserver.com/en/ და ვირჩევთ რეჟიმს:

Download the latest release of Wampserver 2 (Wampserver 2-ის ბოლო ვერსიის ჩამოტვირთვა).

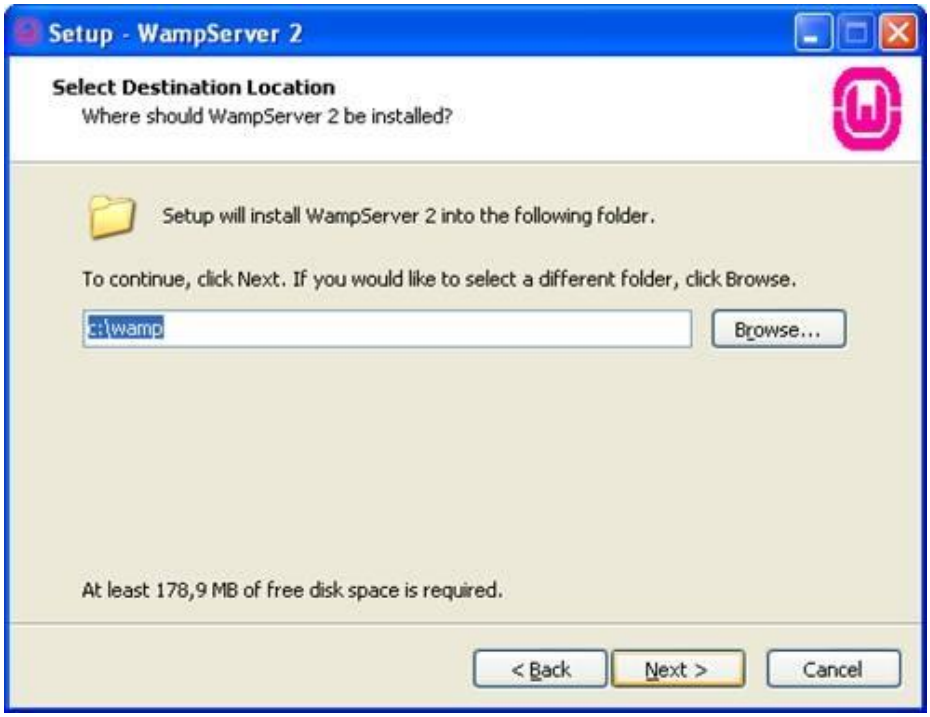
დაიწყება კომპიუტერზე WampServer2.1e-x32.exe ფაილის გადმოწერა.

გავუშვათ ეს ფაილი შესრულებაზე. ოპერაციულმა სისტემამ შესაძლებელია მოითხოვოს ჩვენი თანხმობა პროგრამის დაყენებაზე, ხოლო ანტივირუსულმა პროგრამამ გვკითხოს – ვენდობით მას? ხელს ვაჭერთ შესრულებაზე თანხმობის ღილაკს.

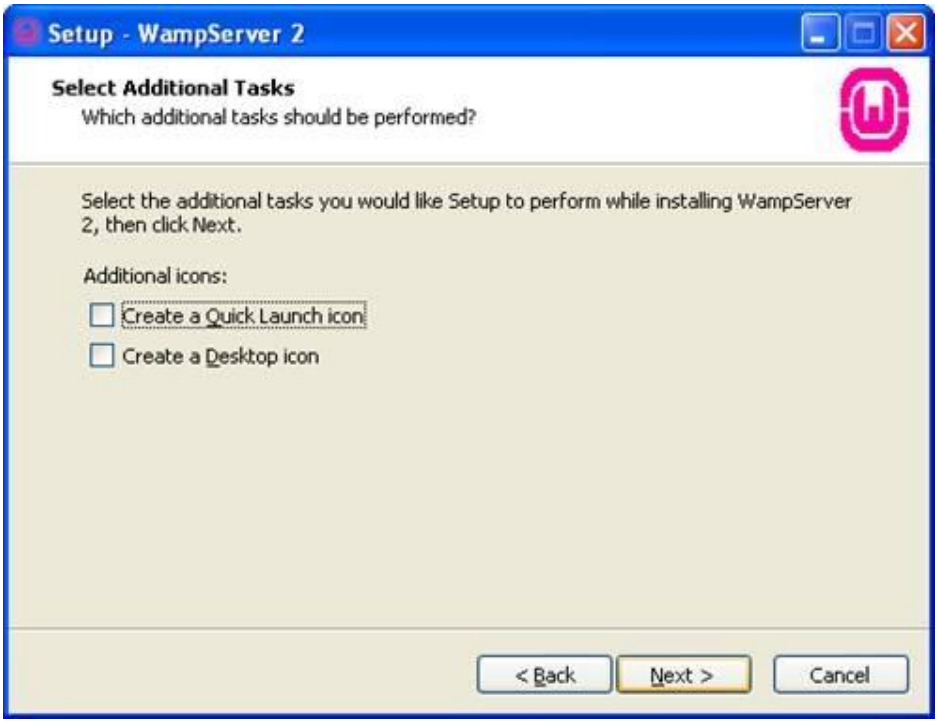
ზემოთ, ფანჯარაში ნაჩვენებია კომპიუტერზე დასაყენებელი პროგრამების და ქვესისტემების სია, მათი ვერსიების მითითებით. ხელს ვაჭერთ Next ღილაკს.



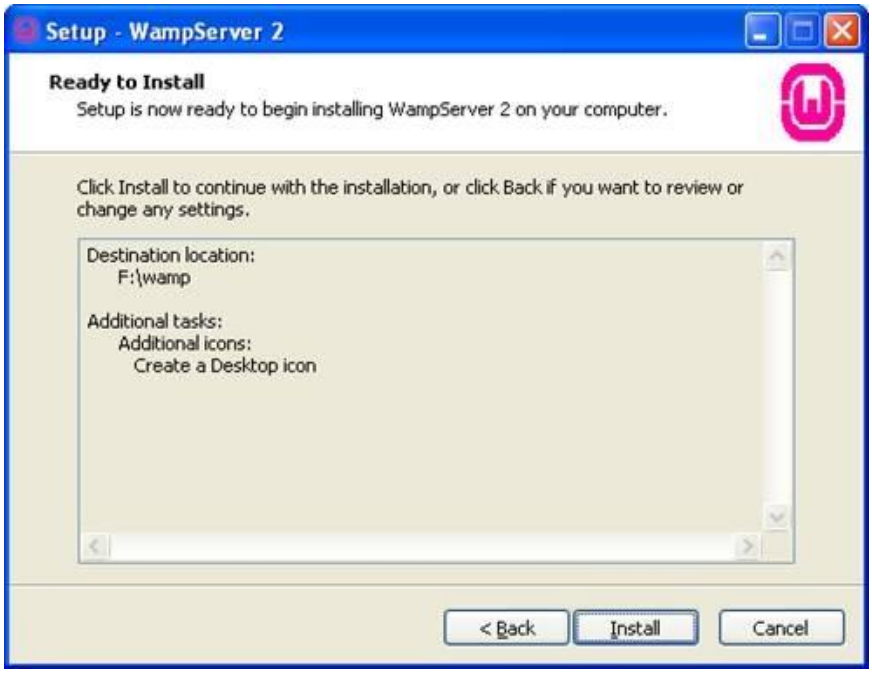
ამ ფანჯარაში მონიშნულად ვტოვებთ I accept the agreement ოფციას. Next.



აქაც ჯობია, თანხმობა გავცეთ ინსტალატორის მიერ შემოთავაზებულ ვარიანტზე. Next.

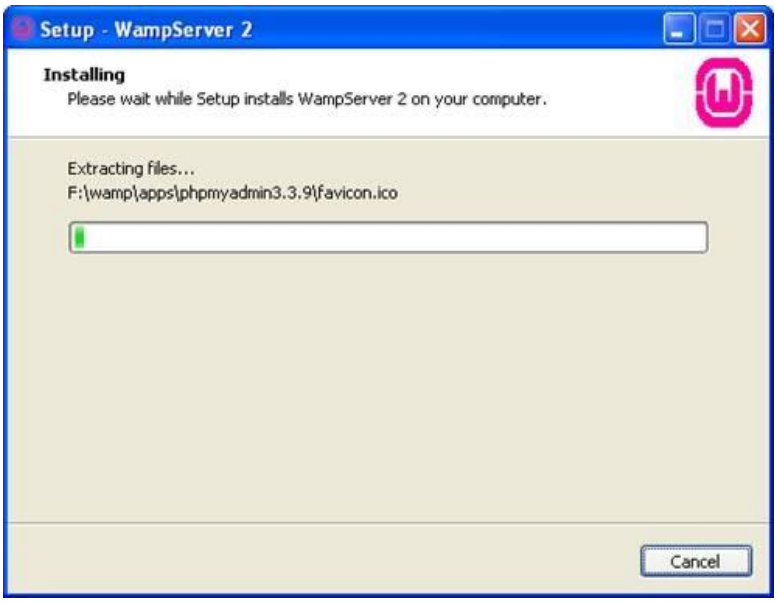


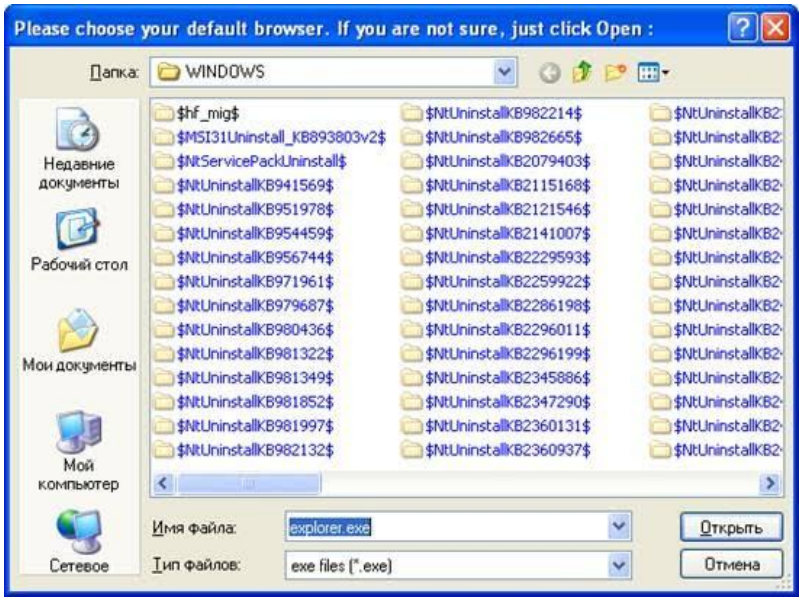
Create a Quick Launch icon და Create a Desktop icon ოფციების მონიშვნის შედეგად პროგრამის გამშვები ხატულა განთავსდება როგორც სწრაფი გაშვების პანელზე, ასევე - სამუშაო მაგიდაზე. Next.



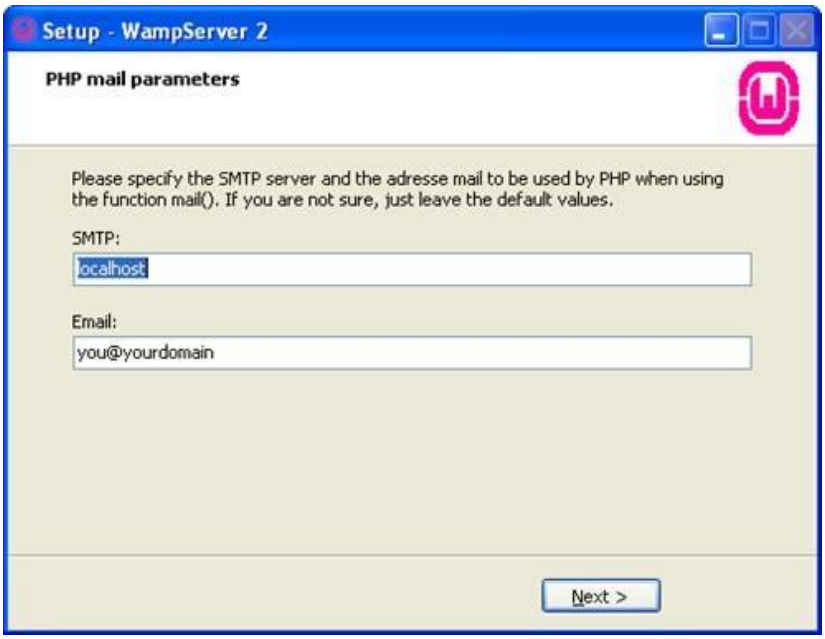
ზემო ფანჯარაში პროგრამის დაყენების დასაწყებად დავაწკაპუნოთ Install ღილაკზე.

იწყება ინსტალირების პროცესი:





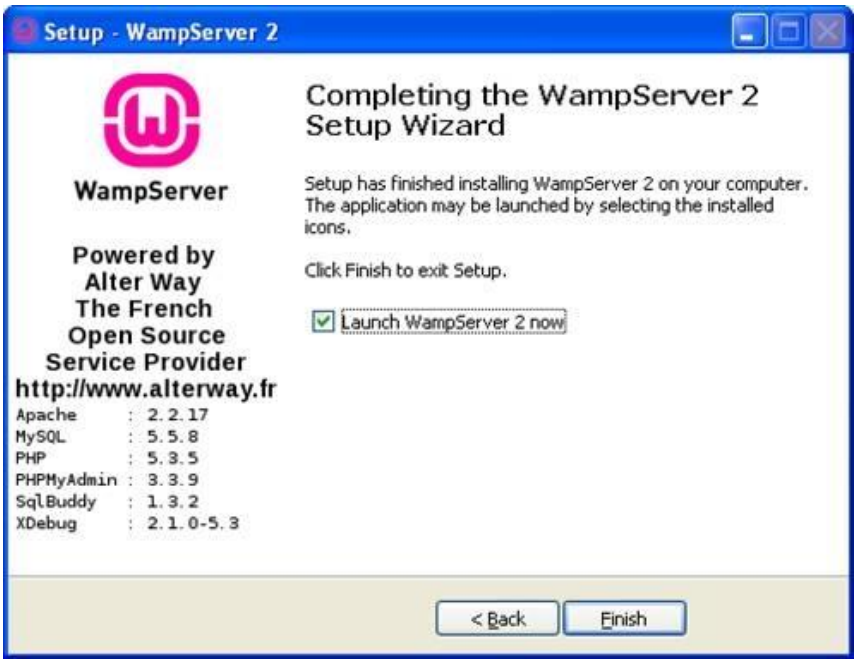
მომდევნო ეტაპზე (ზემოთ) ვირჩევთ Wamp-თან მომუშავე ბროუზერს.



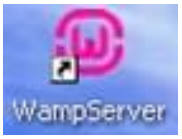
აქ კი ვუჩვენებთ საიტის სახელს (შემოთავაზებულია localhost) და ელექტრონული ფოსტის მისამართს (დუმილით you@youdomain).

Next.

გამოდის ინსტალირების დამამთავრებელი ეტაპის ფანჯარა:



მასში Launch WampServer 2 now ოფციის მონიშნულად დატოვებისა და Finish-ზე დაწკაპუნების შემდეგ, გაიშვება Wamp პროგრამა. ამასთანავე, სამუშაო მაგიდაზე დაიდება ამ პროგრამის გამშვები ხატულაც:



პროგრამის გაშვების მომენტისათვის ქვედა მარჯვენა პანელში გამოჩნდება ამ მომენტის აღმნიშვნელი წითელი ფერის ხატულა  , ხოლო როდესაც პროგრამა გაიშვება, ხატულა ნარინჯისფრად შეიღებება  .

დაბოლოს, როდესაც გააქტიურდება პროგრამაში შემავალი დანართები, ხატულა გახდება მწვანე ფერის  .

შევნიშნოთ, რომ მწვანე ფერის ხატულის კონტექსტური მენიუს Language პუნქტთან მიდგომით, შესაძლებელია სისტემასთან საურთიერთობოდ დუმილით გათვალისწინებული ინგლისურის ნაცვლად ავირჩიოთ სხვა ენაც.

პროგრამიდან გამოსვლაც ამავე კონტექსტურ მენიუში შესაბამისი პუნქტის არჩევითაა შესაძლებელი.

შემდეგ, თუ Wamp პროგრამის შესრულებაზე ბოლომდე გაშვება ვერ ხერხდება, ეკრანზე ხატულა მწვანე ფერს ვერ მიიღებს. ეს ნიშნავს, რომ კომპიუტერში უკვე მუშაობს ისეთი პროგრამა, რომელიც იმავე რესურსებს იყენებს, რასაც Wamp. შესაძლებელია, შეფერხების მიზეზი იყოს

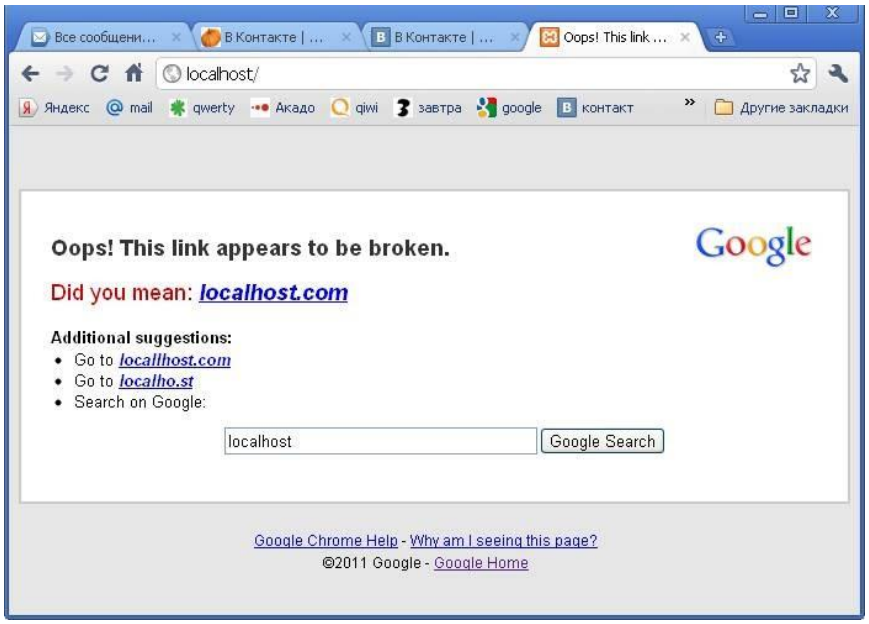
ინტერნეტთან მომუშავე რომელიმე პროგრამა, მაგალითად, ფაილების ჩატვირთვისათვის განკუთვნილი Download Master ან Skype პროგრამა. ასეთ შემთხვევაში საჭიროა გამოვრთოთ კონკურენტი პროგრამები და ხელახლა გავუშვათ შესრულებაზე Wamp.

შესამოწმებლად, ყველაფერი ნორმალურად დამთავრდა თუ არა, ბროუზერში ვკრეფთ მისამართს:

http://localhost

ეკრანზე უნდა აისახოს ლოკალური სერვერის ქვემოთ ნაჩვენები საშინაო ფურცელი:

წარუმატებლობის შემთხვევაში ეკრანზე გამოდის ასეთი ინფორმაცია:



ზემოთ აღნიშნული პროგრამის ღილაკისათვის გამოვიძახებთ კონტექსტურ მენიუს, მასში ავირჩევთ Apache რეჟიმს, შემდეგ Service პუნქტს და ვახდენთ პორტ 80-ის ტესტირებას.

ქვემოთ, სურათზე ასახულია ის შემთხვევა, როდესაც შეფერხების მიზეზი შეიძლება გამხდარიყო Skype პროგრამა.

ვხურავთ Skype პროგრამას და ხელახლა გავუშვებთ შესრულებაზე Wamp-ს.

XAMPP

სადღეისოდ XAMPP (Apache + MySQL + PHP + Perl) გახლავთ Web-სივრცეში ყველაზე პოპულარული, უფასო პაკეტი სერვერული Web-პროგრამების შესაქმნელად. მისი დაწერისას მიმართეს ღია საწყის კოდს, რათა მომხმარებელს გაადვილებოდა პაკეტის ინსტალირება და მისით სარგებლობა.

XAMPP პაკეტის გადმოწერა შესაძლებელია საიტიდან:

https://www.apachefriends.org/index.html#

მისი ინსტალირება საკმაოდ მარტივი წესების დაცვით არის შესაძლებელი.

ვაჩვენოთ, თუ როგორ ხდება Windows ოპერაციული სისტემისათვის PHP-პროგრამების დაწერისა და გამართვისთვის განკუთვნილი, დღეს ყველაზე უფრო პოპულარული XAMPP (Apache, MySql и PHP) პაკეტის ჩამოტვირთვა-ინსტალაცია.

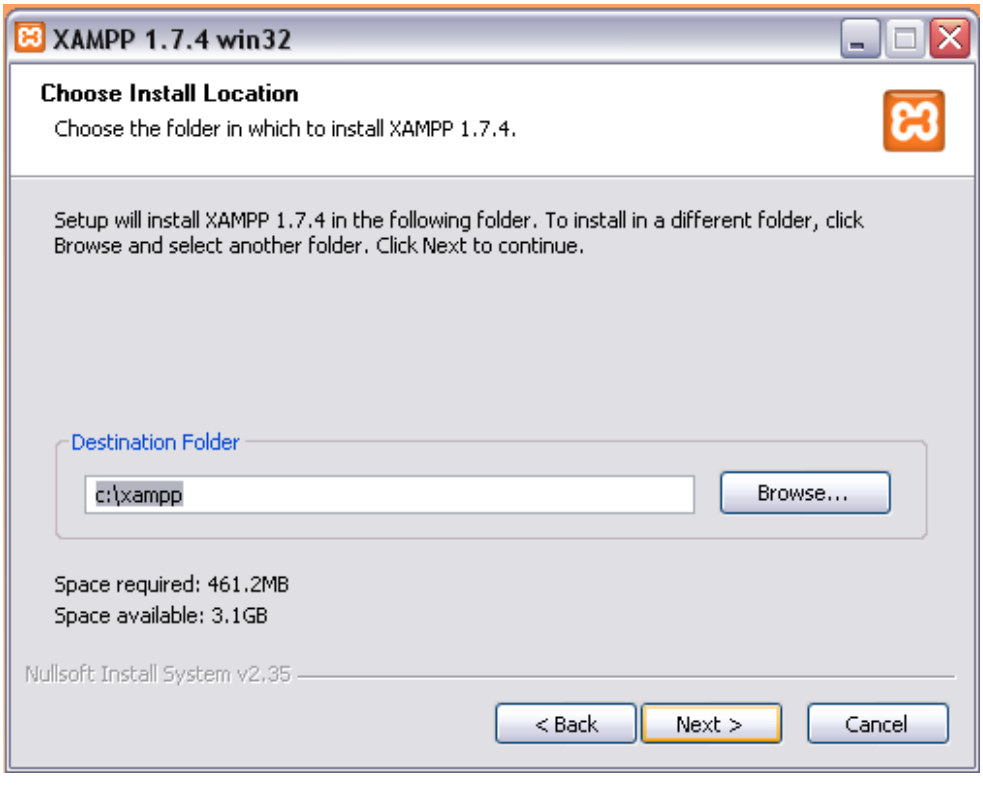
Windows-ზე დასაყენებლად XAMPP-ის ბოლო ვერსიის ჩამოსატვირთად უნდა მივმართოთ შემდეგ ოფიციალურ საიტს:

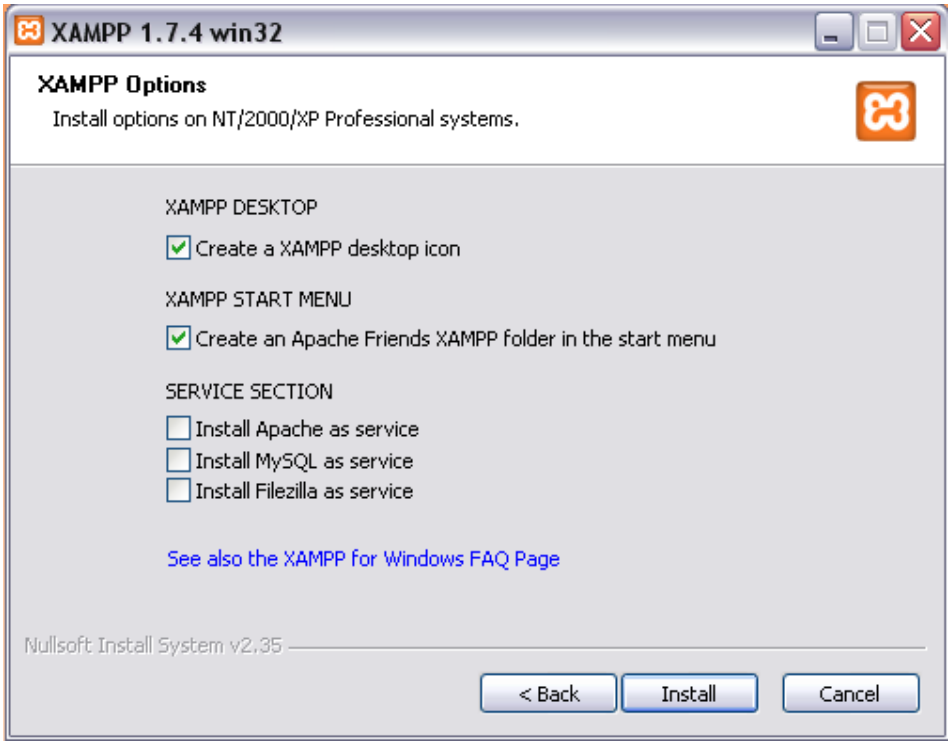
https://www.apachefriends.org/download.html

მოვძებნით ჩვენთვის საინტერესო პაკეტს, მაგალითად, xampp-win32-1.7.4-VC6-installer.exe-ს და ვიწყებთ მის ჩამოტვირთვას ჩვენი კომპიუტერის რომელიმე საქაღალდეში. გავუშვებთ ამ პროგრამას შესრულებაზე და მალე ეკრანზე გამოვა შემდეგი შემცველობის ფანჯარა:

ვაჭერთ Next-ზე.

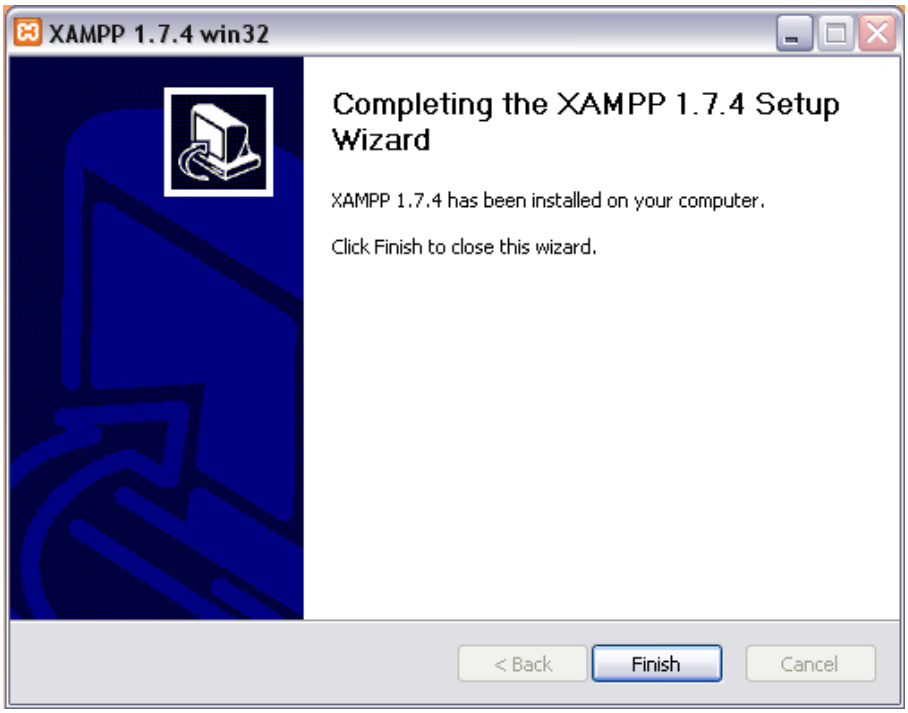
პაკეტის განთავსებისათვის, მომდევნო ფანჯარაში, როგორც წესი, ამჯობინებენ მასში გამოსული, დუმილის წესით შერჩეული c:\xampp საქაღალდის დატოვებას.





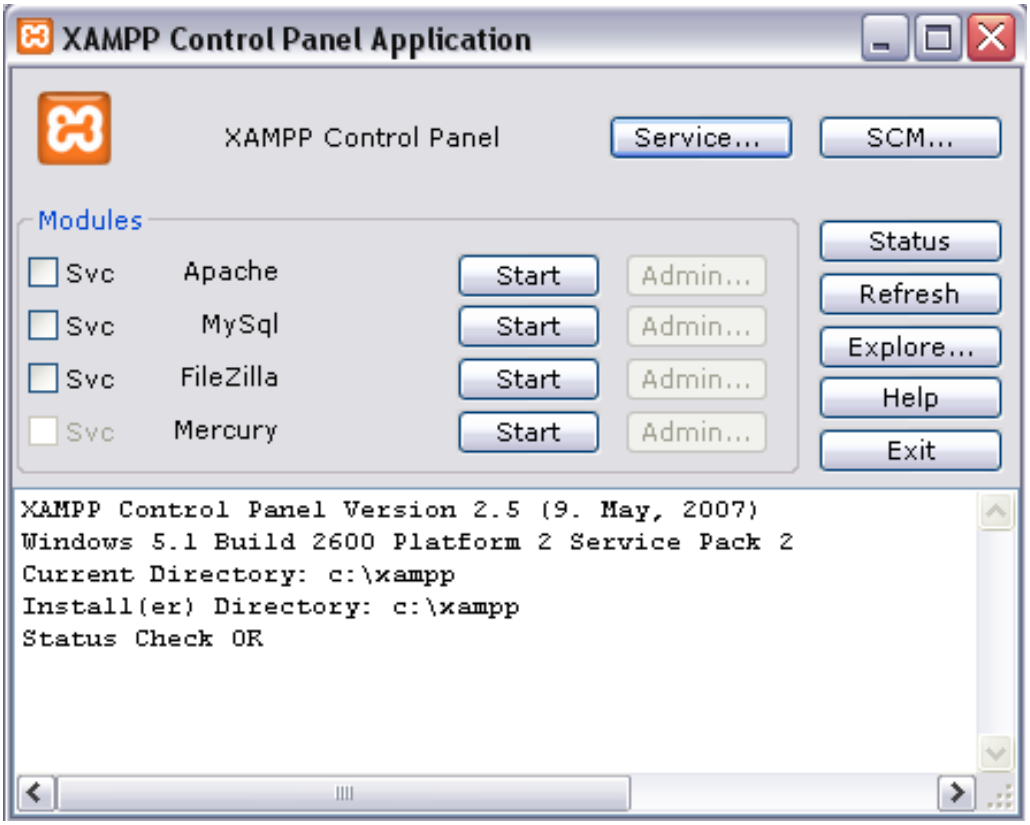
ზემო ფანჯარაში დამატებით შესაძლებელია მოვნიშნოთ მესამე და მეოთხე ოფციებიც, რათა Apache და MySQL სერვისები ავტომატურად იქნეს გაშვებული Windows ოპერაციული სისტემის სტარტისთანავე.

"Install" და პროცესის დასრულების შემდეგ გამოსულ ფანჯარაში გავდივართ ფინიშზე.



მომდევნო დიალოგურ ფანჯარაში გვილოცავენ პაკეტის წარმატებით დაყენებას და გვთავაზობენ XAMPP–ის მართვის პანელის გაშვებას.

თანხმობის გაცემის შემთხვევაში ეკრანზე გამოვა შესაბამისი ფანჯარა, რომლიდანაც შესაძლებელი ხდება სერვისების გაშვება-გამორთვის და სხვ. პროცესების მართვა:



ბროუზერის სამისამართო სტრიქონში ვკრეფთ http://localhost/-ს.

ქვემოთ გამოსული ფანჯრიდან სისტემაში არსებული ბაზების დასათვალიერებლად შეიძლება გამოვიძახოთ phpMyAdmin, ასევე - გავეცნოთ ინფორმაციას php-ის მიმდინარე პარამეტრების შესახებ და სხვ.

დაბოლოს, აღვნიშნავთ, რომ წინა ორი, ადრე განხილული პაკეტისაგან განსხვავებით, php გაფართოების მქონე პროგამული ფაილების შენახვა-გაშვებისათვის xampp-ში დუმილით დანიშნულია c:\xampp\htdocs საქაღალდე.

ჩვენი პირველი პროგრამა PHP-ზე

Notepad-ში ან რომელიმე სპეციალიზებულ ტექსტურ რედაქტორში ავკრიფოთ შემდეგი ტექსტი:

```
<?php
  print “Hello Web!”;
?>
```

ფაილი დავიმახსოვროთ first.php სახელით.

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

```
Hello Web!
```

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

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

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

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

print()-ის გარდა, ინფორმაციის გამოსატანად PHP ენა იყენებს echo() ფუნქციასაც. მისთვისაც დასაშვებია ფრჩხილების გამოტოვება.

განსხვავება ამ ორ ფუნქციას შორის დიდი არაა:

- print()-თან შედარებით, echo() რამდენადმე უფრო სწრაფად მოქმედებს;
- გამოყენების შემდეგ print() ფუნქცია აბრუნებს 1-ის ტოლ მნიშვნელობას, რომელიც შესაძლებელია, გამოყენებული იქნეს გამოსახულებებში, echo()-სთვის კი დასაბრუნებელი მნიშვნელობა გათვალისწინებული არაა.

მოვიყვანოთ ამ ფუნქციების გამოყენების მაგალითი:

ლისტინგი

```
<!DOCTYPE html>
<html>
<body>
<?php
  echo "<h2>PHP is Fun!</h2>";
  echo "Hello world!<br>";
  echo "I'm about to learn PHP!<br>";
  echo "This ", "string ", "was ", "made ", "with
multiple parameters.";
?>
</body>
</html>
```

ეკრანზე აისახება ასეთი ინფორმაცია:

PHP is Fun!

Hello world!
I'm about to learn PHP!
This string was made with multiple parameters.

მივაქციოთ ყურადღება, რომ როგორც print(), ასევე echo() ფუნქციის გამოყენების შემთხვევაში გამოსატანი ტექსტისათვის დასაშვებია მისი დაფორმატება html ენის წესების დაცვით!

PHP ინსტრუქციები წერტილ-მძიმით უნდა დაბოლოვდეს. გამონაკლისს შეიძლება წარმოადგენდეს ბლოკში ბოლო ინსტრუქცია, თუმცა, ეს

რეკომენდებული არ გახლავთ *(საეჭვო შემთხვევებში შეიძლება ასეთმა გადაწყვეტილებამ ინტერპრეტატორი შეცდომაში შეიყვანოს).*

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

ლისტინგი 1

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

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

PHP პროგრამებში ხშირად გამოიყენება **კომენტარები**. აი, მათი სახეები:

```
// ეს კომენტარია.

# თქვენ წარმოიდგინეთ, ესეც კომენტარია.

/* ეს კი
  რამდენიმე სტრიქონზე
  გაწელილი კომენტარია.
*/
```

კომენტარებში ჩაწერილი ინფორმაცია ინტერპრეტატორის მიერ იგნორირდება.

უნდა აღინიშნოს, რომ კომენტარების გამოყენება რეკომენდებულია არა მხოლოდ მაშინ, როდესაც ამის საჭიროება აშკარა გახლავთ, არამედ მაშინაც, როდესაც ისინი, ერთი შეხედვით, ზედმეტია, მაგრამ კოდის დამყოფი ხაზების როლში მყოფნი, აკეთებენ მნიშვნელოვან საქმეს - აუმჯობესებენ პროგრამის აღქმადობას (თუმცა, როგორც აღვნიშნეთ, პროგრამის შესრულების მიმდინარეობაზე ისინი არავითარ გავლენას ახდენენ).

შევნიშნავთ, რომ სასურველია, დიდი მოცულობის პროგრამებისათვის, კომენტარების გამოყენების ეფექტიანობის გაზრდის მიზნით, გავითვალისწინოთ შემდეგი რეკომენდაციები:

- კომენტარები იყოს ერთდროულად მოკლე და, რაც შესაძლებელია, კოდის ფრაგმენტის არსის სრულყოფილად ამსახველიც;
- ცალკეულ ოპერატორს, თუ ეს საჭიროა, კომენტარი გაუკეთდეს მისივე დასრულებისათანავე, იმავე სტრიქონში;
- ბლოკების დანიშნულების აღსაწერად კი მათ მიეცეთ იერარქიული სტრუქტურის სახე, ანუ HTML ელემენტების შემადგენელი საწყისი და ბოლო ტეგების მსგავსად, გააჩნდეთ როგორც დასაწყისი, ასევე დაბოლოება. პირველი მათგანი განთავსდეს HTML კოდისა თუ PHP პროგრამის შესაბამისი ფრაგმენტის თავში (დასაშვებია, პირველი სტრიქონის ბოლოშიც), ხოლო მეორე კი – უკანასკნელი ტეგის შემდეგ ან მის გვერდზე. ამასთან, უმჯობესია ორივე მათგანში განთავსებული, მხოლოდ მომხმარებლისათვის განკუთვნილი ინფორმაციები იდენტური იყოს, გარდა ერთადერთი სიმბოლოსა თუ სიტყვისა, რომელიც ახდენენ მარკირებას, არის აღმნიშვნელი პროგრამული ფრაგმენტის დასაწყისის თუ დაბოლოების;
- პროგრამაში შინაარსობრივად იერარქიული სახის კომენტარების განლაგებისას, პროგრამული ტექსტის აღქმადობის გასაუმჯობესებლად, ფრიად სასურველია კომენტარების სისტემას ვიზუალურადაც მიეცეს იერარქიული სტრუქტურის სახე, პროგრამის ცალკეული უბნებისათვის განკუთვნილი კომენტარების ფურცლის კიდიდან სხვადასხვა მანძილით დაცილების გზით.

მოვიყვანოთ მაგალითი:

```
// თავი – 1 მოდულის დანიშნულება
   ------------------------
    // თავი  – 1.1 მოდულის დანიშნულება
      ------------------------
    // ბოლო – 1.1 მოდულის დანიშნულება
    // თავი  – 1.2 მოდულის დანიშნულება
      ------------------------
    // ბოლო  – 1.2 მოდულის დანიშნულება
   ------------------------
// ბოლო  – 1 მოდულის დანიშნულება

// თავი  – 2 მოდულის დანიშნულება
   ------------------------
// ბოლო – 2 მოდულის დანიშნულება
```

ცვლადები

PHP-ში ცვლადის სახელი $ სიმბოლოთი იწყება! მეორე პოზიციაზე შეიძლება გამოვიყენოთ დიდი და პატარა ლათინური სიმბოლოები, ციფრები და ხაზგასმის ნიშანი, ხოლო მესამეზე და ა. შ. – ციფრებიც. სხვა სიმბოლოების, მათ შორის **ჰარის** *(ხარვეზის, ცარიელი არის)* გამოყენება არ დაიშვება.

მოვიყვანოთ ცვლადების სწორად დასახელების მაგალითები:

$a, $_A_B_C, $b555 და ა.შ.

ცვლადის სახელი რეგისტრის მიმართ მგრძნობიარეა, ანუ $abc, $Abc და $ABC სამი სხვადასხვა ცვლადია. აქვე წინსწრებით შევნიშნოთ, რომ PHP ენაში გასაღებური სიტყვები და ფუნქციების სახელები რეგისტრის მიმართ არამგრძნობიარეა.

ცვლადებს ენიჭებათ მნიშვნელობები. ეს მნიშვნელობები და, შესაბამისად ცვლადებიც, შეიძლება იყოს სხვადასხვა ტიპის. კერძოდ, ცვლადებში დასაშვებია შევინახოთ:

რიცხვები, სიმბოლოთა სტრიქონები, ობიექტები, მასივები, ლოგიკური მნიშვნელობები.

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

```
$num1 = 8;
$name = “გია”;
```

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

```
print $num1;
```

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

დინამიკური ცვლადები

PHP-ში ცვლადის სახელი შეიძლება დავიმახსოვროთ, როგორც სხვა ცვლადის მნიშვნელობა! მაგალითად, გამოსახულება

```
$user=”bob”;
```

შეიძლება შეიცვალოს შემდეგი ორი გამოსახულებით:

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

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

```
print “$$holder”;
```

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

PHP-ში დინამიკური ცვლადი შეიძლება გამოცხადდეს სტრიქონის *(სტრიქონული მუდმივის)* მეშვეობითაც *(გამოიყენება განსხვავებული სინტაქსი):*

$ {“user”}=”bob”;

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

ცხადია, ფიგურულ ფრჩხილებში ცვლადიც შეიძლება განთავსდეს:

$ {$holder}

ქვემოთ, პროგრამაში მოყვანილია დინამიკური ცვლადის შექმნის სხვადასხვა მაგალითი.

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

შემდეგ, მოყვანილია აღნიშნული ცვლადების მნიშვნელობების ეკრანზე გამოყვანის ასევე სხვადასხვა ხერხი:

ლისტინგი 2

```
<html>
<head>
 <title> დინამიკური ცვლადის შექმნა და მისდამი მიმართვა</title>
</head>
<body>
<?php
  $holder=”user”;
 $$holder=”bob”;
 print   “$user<br>”;          // გამოჰყავს ”bob”
 print   “$$holder”;           // გამოჰყავს ”$user”
 print   “<br>”;
 print   “$ {$holder}<br>”;    // გამოჰყავს ”bob”
 print   “S {‘user’}<br>” ;    // გამოჰყავს ”bob”
?>
```

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

დინამიკური ცვლადების შემოღება მნიშვნელოვნად აადვილებს დაპროგრამების პროცესს, მაგალითად, ფაილის სახელის პროგრამაშივე გენერირების გზით მარტივდება დასამუშავებელი ფაილებისადმი მიმართვა და მათზე სხვადასხვა ოპერაციის ჩატარება (ამ შესაძლებლობას ერთ-ერთ მომდევნო პარაგრაფში განვიხილავთ).

შემდეგ, PHP ასევე საშუალებას იძლევა, ერთი ცვლადი მეორეს დაეყრდნოს (ასეთი გადაწყვეტაც დამახასიათებელია თანამედროვე დაპროგრამების ენებისათვის და მათი სრულყოფის მიზანს ემსახურება).

მოვიყვანოთ ცვლადის ცვლადზე დაყრდნობის მაგალითი:

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

მონაცემთა ტიპები

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

ჩამოვთვალოთ PHP-ში გათვალისწინებული მონაცემთა ტიპები:

- String – სტრიქონული;
- Integer – მთელრიცხვა;
- Float (floating point numbers - double) – მცურავწერტილიანი;
- Boolean – ლოგიკური;
- Array – მონაცემთა ერთობლიობა მასივის სახით;
- Object – ობიექტი - რაიმე კლასის კონკრეტული ეგზემპლარი წარმოადგენს ამ ეგზემპლარის თვისებების (ანუ მონაცემთა) და ამ თვისებებთან მომუშავე მეთოდების ერთიანობას;
- NULL – ტიპის სახელწოდებამ შეცდომაში არ შეგვიყვანოს! ნულოვანი ტიპი და ამავე დროს NULL მნიშვნელობა ავტომატურად განსაზღვრება ცვლადს, თუ მას (ჯერ) კონკრეტული მნიშვნელობა არ მინიჭებია.

- Resource – ამ ტიპის მქონედ თვლიან დაყრდნობებს იმ გარე ფუნქციებსა და სხვა საშუალებებზე, რომლებისადმიც, მაგალითად, მონაცემთა ბაზისადმი, PHP ენაზე დაწერილი პროგრამიდან შეიძლება მოხდეს მიმართვები.

ამრიგად, ტიპი ენიჭება ცვლადს ან ცვლადთა ერთობლიობას. აქ აღსანიშნავია ერთი მომენტი:

გამორიცხულია არაა, რომ პროგრამების მიერ ერთმანეთისათვის მონაცემების გადაგზავნისას აღმოჩნდეს, რომ ესა თუ ცვლადი არ არსებობს ანდა მას მნიშვნელობა არ ჰქონია მინიჭებული (ის ცარიელია) და შესაბამისად, მნიშვნელობად (და ტიპადაც) ითვლება NULL.

ასეთ შემთხვევაში სიტუაციაში გასარკვევად გამოიყენება ფუნქცია isset(), რომელიც გვიბრუნებს true მნიშვნელობას, თუ ფუნქცია არსებობს და ამავე დროს მისი მნიშვნელობა არ გახლავთ NULL, საწინააღმდეგო შემთხვევაში კი - false-ს.

მსგავსი როლი აკისრია empty() ფუნქციას, რომელიც ამოწმებს, ცარიელია თუ არა მისთვის პარამეტრად გადაცემული ცვლადი. შესაბამისად, თუ ცვლადი არსებობს და მისი მნიშვნელობა არის 0 (და არა NULL), false ან ცარიელი სტრიქონი, ის გვიბრუნებს true მნიშვნელობას, სხვა შემთხვევაში კი – false-ს. აქ გასათვალისწინებელია შემდეგი გარემოება - empty() ფუნქციის მიერ false მნიშვნელობის დაბრუნება ხდება მაშინაც, როცა ცვლადი საერთოდ არ არსებობს.

is_null() ფუნქცია კი true მნიშვნელობას გვიბრუნებს იმ შემთხვევებში, როცა ცვლადის მნიშვნელობა ცარიელია ანდა მას მინიჭებული ჰქონდა სპეციალური, ზემოთ აღნიშნული NULL მნიშვნელობა. არსებობს კიდევ ასეთი ვარიანტიც – პროგრამაში ადრე არსებული ცვლადი წაშლილი იქნა unset() ფუნქციის დახმარებით.

პროგრამისტს შეუძლია gettype() ფუნქციის მეშვეობით შეამოწმოს მონაცემთა ტიპი:

ლისტინგი 3

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

```
  print gettype ($testing );      // string
  print (“<br>”);
    $testing = 5.0;
  print gettype ($testing );      // double
  print (“<br>”);
    $testing = true;
  print gettype ($testing );      // boolean
  print  “<br>”;
?>
</body>
</html>
```

ეს პროგრამა გამოიტანს შემდეგ შეტყობინებებს:

```
integer
string
double
boolean
```

მონაცემთა ტიპის შეცვლა

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

ლისტინგი 4

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

```
    settype( $undecided, boolean );
print gettype ( $undecided );                    // boolean
print  " -- $undecided<br>";                     // 1
?>
</body>
</html>
```

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

მოვიყვანოთ ასეთი გარდაქმნების მაგალითი:

ლისტინგი 5

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

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

გამოსახულებები PHP-ში სხვა ენების მსგავსად იქმნება. გამოიყენება

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

სტრიქონის სტრიქონზე გადაბმა კი აქ წერტილის მეშვეობით ხდება:

```
Print “hello ”.”world”;
```

ამ ინსტრუქციის შედეგად ეკრანზე აისახება hello world სტრიქონი.

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

```
x  += 4;
```

ცხადია, გამოიყენება

-=, /=, *= და %= ოპერატორებიც.

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

გამოსახულებებში გამოიყენება ჩვენთვის ნაცნობი შედარების ოპერატორები:

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

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

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

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

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

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

ზოგჯერ საჭიროა, პროგრამაში თავიდან ავიცილოთ ცვლადის მნიშვნელობის და ტიპის შეცვლა. მაშინ ცვლადს მხოლოდ ერთხელ, პროგრამის დასაწყისში მიენიჭება მნიშვნელობა *(ამ მნიშვნელობის მიხედვით - ტიპიც)* და მერე ის აღარ იცვლება.

ცვლადის აღწერილი ნაირსახეობა, ფაქტობრივად, კონსტანტაა. PHP ენაში მას ასეც უწოდებენ. მისი განსაზღვრისას $ სიმბოლოს არ იყენებენ და სახელს დიდი ასოებისაგან ადგენენ.

იხ. მაგალითი:

ლისტინგი 6

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

მივაქციოთ ყურადღება - გამობეჭდვისას კონსტანტა სტრიქონს მივუერთეთ კონკატენციის ოპერატორის მეშვეობით.

აღვნიშნოთ, რომ PHP ენა პროგრამისტს აწვდის ე. წ. **წინასწარ განსაზღვრული კონსტანტების** გამოყენების საშუალებასაც.

მაგალითად:

FILE კონსტანტა შეიცავს იმ ფაილის სახელს, რომელსაც მოცემულ მომენტში ამუშავებს ინტერპრეტატორი,

LINE კი - ფაილის მიმდინარე სტრიქონის ნომერს და სხვ.

სტრიქონულ მონაცემებთან მომუშავე ფუნქციები

გავეცნოთ სტრიქონებთან მომუშავე ყველაზე მნიშვნელოვან ფუნქციებს:

*(**შენიშვნა**: ამ ფუნქციების სრული სიისა და შესაძლებლობების გასაცნობად იხ.: https://www.w3schools.com/php/php_ref_string.asp).*

1. სტრიქონის სიგრძის განსაზღვრისათვის გამოიყენება `strlen()`ფუნქცია;
2. სტრიქონში სიტყვების დათვლა ხდება `strrev()`ფუნქციის დახმარებით;
3. `strops()`ფუნქციით განისაზღვრება იმ პოზიციის ნომერი, რომლიდანაც ტექსტის სახით მოცემულ პირველ არგუმენტში იწყება მერე არგუნენტად მოცემული სტრიქონი. მაგალითად:
   ```
   <?php
    echo strpos("Hello world!", "world"); // გამოგვივა 6, რადგანაც სტრიქონში პირველი სიმბოლოს პოზიცია ნულოვანად ითვლება.
   ?>
   ```
4. სამარგუმენტიან `str_replace` ფუნქციაში მესამე არგუმენტად მოყვანილ ტექსტში პირველი არგუმენტის ტექსტი ყველგან შეიცვლება მეორე არგუმენტ-სტრიქონით:
   ```
   <?php
     echo str_replace("world", "Dolly", "Hello world!");
     // გამოვა Hello Dolly!
   ?>
   ```

ნაკადის მართვა

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

მოვიყვანოთ IF ინსტრუქციის გამოყენების მაგალითი:

ლისტინგი 7

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

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

პროგრამა ოდნავ გავართულოთ else ბლოკის დანიშნულების სადემონსტრაციოდ:

ლისტინგი 8

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

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

ლისტინგი 9

```
<html>
<head>
    <title> else  და  elseif  ბლოკების გამოყენება </title>
</head>
<body>
<?php
    $mood = “sad”;
```

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

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

ლისტინგი 10

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

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

შენიშვნა: switch ინსტრუქციასა და მომდევნო პარაგრაფებში განხილულ ციკლის ოპერატორებში break ოპერატორის ნაცვლად შეიძლება გამოვიყენოთ (ადრე შერისხული) და php ენის უკანასკნელ ვერსიებში დაბრუნებული goto ოპერატორი. ცხადია, მასში უნდა მიეთითოს პროგრამის იმ უბნის მომნიშვნელი ჭდე, რომელზეც მოხდება გადასვლა:

```
<?php
 ------------------------
  goto abc;
 ------------------------
 ------------------------
  abc: ოპერატორი;
-------------------------
?>
```

შემდეგ, დიდი პოპულარობით არ სარგებლობს, მაგრამ უნდა ვიცნობდეთ ? ოპერატორსაც:

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

ვაჩვენოთ ამ ოპერატორის ფუნქციონირება შემდეგი პროგრამის მაგალითზე:

ლისტინგი 11

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

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

ციკლები

ციკლი while

ლისტინგი 12

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

ციკლი do… while

ლისტინგი 13

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

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

ხაზი უნდა გავუსვათ შემდეგ მოთხოვნას – ციკლის ბოლოს აუცილებლად უნდა დავსვათ ; (წერტილ-მძიმე) სიმბოლო.

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

For ციკლი

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

ქვემო მაგალითში 2-ზე მრავლდება პირველი 12 ნატურალური რიცხვი:

ლისტინგი 14

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

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

ლისტინგი 15

```
<html>
<head>
      <title> break ინსტრუქციის გამოყენება </title>
</head>
<body>
```

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

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

შენიშვნა: ზემოთ, switch ინსტრუქციის განხილვისას უკვე აღვნიშნეთ, რომ აქაც, ციკლის ოპერატორებში break ოპერატორის ნაცვლად შეიძლება გამოვიყენოთ უბნიდან უბანზე გადასვლისათვის განკუთვნილი goto ოპერატორი. აქ დამატებით ვიტყვით, რომ goto ოპერატორს მხოლოდ გამოვყავართ ციკლიდან, ხოლო გარედან, ციკლის რომელიმე უბანზე მისი დახმარებით შეღწევა დაუშვებელია.

ლისტინგი 16

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

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

ერთმანეთში ჩადგმული ციკლები

ციკლის სხეული შეიძლება სხვა ციკლსაც შეიცავდეს. ასეთი კონსტრუქციის გამოყენება განსაკუთრებით ხელსაყრელია ცხრილებთან მუშაობისას. მოვიყვანოთ ჩადგმული ციკლების მაგალითი, რომელთაც ბროუზერის ეკრანზე გამოჰყავს გამრავლების ტაბულა:

ლისტინგი 17

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

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

შენიშვნა: *სადღეისოდ* print *ინსტრუქციის (ოპერატორის) ნაცვლად უფრო ხშირად იყენებენ* echo *ოპერატორს.*

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

პროგრამის მუშაობის საბოლოო შედეგად მიიღება გამრავლების ცხრილი.

ფუნქციები

PHP ენაში გამოყენებულ ზოგიერთ ფუნქციას ჩვენ უკვე ზემოთ გავეცანით (საერთოდ უნდა აღნიშნოს, რომ ნებისმიერი, მეტ-ნაკლებად "სერიოზული" პროგრამა, როგორც წესი, იყენებს ფუნქციებს).

უფრო დაწვრილებით შევისწავლოთ ეს საკითხი.

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

განსაკუთრებით სასურველია ფუნქციების გამოყენება მრავალჯერ განმეორებადი მოქმედებების შესასრულებლად.

ფუნქცია განიმარტება, როგორც ინსტრუქციების ბლოკი, რომელიც სრულდება პროგრამის მიერ მხოლოდ მისი გამოძახების შემთხვევაში. მისი სახელი იწყება ასოთი ან ქვედა ტირეთი და, ცვლადებისგან განსხვავებით, რეგისტრისადმი მგრძნობიარე არ არის.

შენიშვნა:. ჩვენ მიერ უკვე შესწავლილ Javascript ენაში რეგისტრი-სადმი მგრძნობიარეა, როგორც ცვლადების, ასევე ფუნქციების სახელებიც.

ფუნქცია ორი სახისაა:

სისტემაში (აქ PHP-ში) ჩაშენებული და *მომხმარებლის მიერ შექმნილი.*

უმეტესწილად ფუნქციას მიეწოდება არგუმენტი *(არგუმენტები).* ისინი უნდა განლაგდნენ მრგვალ ფრჩხილებში *(როგორც უკვე ვიცით, გამონაკლისს წარმოადგენს* print *და* echo *ფუნქციები).* იმ იშვიათ შემთხვევებშიც კი, როცა რომელიმე ფუნქცია არგუმენტებს არ

საჭიროებს, ფრჩხილების დასმა მაინც აუცილებელია. სიაში არგუმენტები ერთმანეთისაგან მძიმით გამოიყოფა.

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

ლისტინგი 18

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

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

ლისტინგი 19

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

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

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

ახლა კი ვაჩვენოთ არგუმენტის შემცველი ფუნქციის მაგალითი:

ლისტინგი 20

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

შემდეგი ნაბიჯი იქნება ისეთი ფუნქციის შექმნა, რომელიც გვიბრუნებს მნიშვნელობას:

ლისტინგი 21

```
<html>
<head>
   <title> მნიშვნელობის დამბრუნებელი ფუნქციის შექმნა </title>
</head>
<body>
<?php
function addNums($firstnum, $secondnum);
  {
  $result = $firstnum + $secondnum;
```

```
  return $result;
  }
print  addNums(3,5);       // დაიბეჭდება  8
?>
</body>
</html>
```

return ოპერაციას შეუძლია შედეგი სხვადასხვა სახით დააბრუნოს:

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

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

ლისტინგი 22

```
<html>
<head>
   <title> ფუნქციის დინამიკურად გამოძახება </title>
</head>
<body>
<?php
      /* აქ პროგრამა გამარტივებულია, პრაქტიკაში კი მას ფუნქციის
      სახელად "sayHello" ან რაიმე სხვა მნიშვნელობა მომხმარებლის
      მიერ მიეწოდება გარედან დიალოგის რეჟიმში */
function sayHello( );
  {
  print  ("Hello<br>");
  }
$function_holder =  "sayHello";
$function_holder ( );          // მივაქციოთ ყურადღება გამოძახების წესს!
?>
</body>
</html>
```

ცვლადების ხილვადობის უბნები

ფუნქციაში შექმნილი ცვლადი მხოლოდ მის არეალში მოქმედებს, ანუ ის **ლოკალური** მოქმედებისაა და შესაბამისად, **ლოკალურ ცვლადად** იწოდება. ფუნქციის შიგნით ავტომატურად ვერ მივმართავთ მის გარეთ შექმნილ ცვლადსაც.

ასეთი მიდგომები თავიდან გვაცილებს სხვადასხვა ადამიანის მიერ ერთი დიდი პროგრამის შექმნისას მოსალოდნელ გაუგებრობებს. მაგრამ, ძალიან ხშირად საჭირო ხდება ისეთი ცვლადების შექმნაც, რომელთაც პროგრამის ნებისმიერი უბნიდან შეიძლება მივმართოთ. ანუ დგება საკითხი ე.წ. **გლობალური** მოქმედების ცვლადების შექმნისა. ამ მიზნით PHP იყენებს global ინსტრუქციას.

მოვიყვანოთ გლობალური ცვლადების გამოყენების მაგალითები:

ლისტინგი 23

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

საკითხის ასეთი წესით გადაწყვეტის შედეგად ეკრანზე აისახება შეტყობინება: The meaning of Life is 42

ლისტინგი 23–1

```
<html>
<head>
   <title> გლობალური ცვლადების გამოყენება </title>
```

```
</head>
<body>
<?php
 $saxeli="გიორგი<br/>";
 echo  $saxeli;
 function f1() {global $saxeli; echo "ფუნქციაში ვიყენებთ გლობალური ცვლადის მნიშვნელობას! <br/>ცვლადი 'სახელის' მნიშვნელობაა:  ". $saxeli;
                };
 f1();
?>
</body>
</html>
```

გამოძახებებს შორის ფუნქციის მდგომარეობის დამახსოვრება

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

ლისტინგი 24

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

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

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

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

ლისტინგი 25

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

ცხადია, რომ ასეთი ფუნქცია კი უკვე დამოუკიდებელი სახისაა. შესაბამისად, ამჯერად უკვე საჭირო აღარ არის გლობალური ცვლადების შექმნაზე ზრუნვა, შედეგი კი ისეთივე იქნება, როგორსაც უზრუნველყოფდა წინა პროგრამა.

კვლავ ფუნქციის არგუმენტების შესახებ, მათი მნიშვნელობის განსაზღვრა დუმილის პრინციპით

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

ლისტინგი 26

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

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

დაყრდნობის გზით არგუმენტის გადაცემა

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

ქვემოთ ნაჩვენებია ეს ორივე ვარიანტი:

ლისტინგი 27

```
<html>
<head>
  <title>ფუნქციისთვის დაყრდნობის გზით არგუმენტის გადაცემა მისი
         გამოძახებისას
   </title>
</head>
<body>
<?php
 function addFive($num)
 {
    $num += 5;
 }
 $orignum = 10;
 addFive(&$orignum);
 print $orignum;                // 15
?>
</body>
</html>
```

ლისტინგი 28

```
<html>
<head>
    <title>არგუმენტის დაყრდნობა სხვა ცვლადზე ფუნქციის
        განსაზღვრისთანავე
    </title>
</head>
<body>
<?php
 function addFive($num)
{
   $num += 5;
}
  $orignum = 10;
  addFive (&$orignum);
  print $orignum;
?>
</body>
</html>
```

აქვე შევნიშნოთ, რომ უფრო "პროგრამაგენურად" გამოიყურება მეორე მიდგომა.

დავალება–1:

დაწერეთ პროგრამა, რომელიც გამოგვითვლის რიცხვების საშუალო არითმეტიკულს შესაბამისი ფუნქციის გამოძახების შედეგად.

დავალება–2:

დაწერეთ პროგრამა, რომელიც გამოგვითვლის n-ის ფაქტორიალს, შესაბამისი ფუნქციის გამოძახების შედეგად.

მასივები

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

დავუშვათ, რომ საჭიროა, პროგრამაში ინახებოდეს რაიმე 100 მნიშვნელობა. შეიძლებოდა თითოეული მათგანისთვის განგვესაზღვრა ცვლადი *(შესაბამისი მნიშვნელობით),* მაგრამ, გაცილებით აადვილებს ამ მნიშვნელობებთან მუშაობას მათი გაფორმება მასივის სახით *(მაგალითად, ამ მონაცემების დასახარისხებლად, ციკლების გამოყენებისას და ა.შ).*

მასივიდან ელემენტის ამორჩევა, უმეტესწილად, ნომრის ჩვენებით ხდება *(სხვა შემთხვევები ქვემოთ იქნება განხილული).* PHP-ში მასივის პირველი ელემენტის ნომერია არა “1”, არამედ “0”. *(გავიხსენოთ, რომ JavaScript-შიც ეს გადაწყვეტილება იყო მიღებული).*

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

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

II - $users[] “გია”;

$users[] “მზია”;

$users[] “ლია”;

$users[] “გიგი”;

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

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

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

$users[] = “ჯონი”;

$users[] = “ჯიმი”;

ასოცირებული მასივები

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

PHP-ში, სხვა მხრივ, ჩვეულებრივ და ასოცირებულ მასივებს შორის არავითარი განსხვავება არ არსებობს. თუმცა, ცხადია, ასოცირებული

მასივების აღნიშნული თავისებურება მათი დამუშავებისას განსხვავებულ მიდგომებს მოითხოვს.

ასოცირებული მასივების შექმნაც ზემოთ მოყვანილი ორივე ხერხით არის შესაძლებელი:

```
$character = array (saxeli => “ვეფხია”,
                    gvari => “ლომიძე”,
                    asaki => 22,
                    “damatebiti informacia” => “ნიანგებზე მონადირე”);
```

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

ახლა უკვე შეგვიძლია მივმართოთ მასივის ნებისმიერ ელემენტს, მაგალითად, ამგვარი წესით:

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

ვაჩვენოთ მეორე გზაც:

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

მრავალგანზომილებიანი მასივები

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

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

ლისტინგი 29

```
<html>
<head>
<title>მრავალგანზომილებიანი მასივის შექმნა</title>
</head>
<body>
<?php
  $characters = array (
        array ( name=>"bob",
```

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

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

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

გადმოცემული ზომის 1-ით შემცირება გამოწვეულია იმ მიზეზით, რომ **მასივის ელემენტების დანომვრა 0-დან იწყება.**

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

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

ასოცირებული მასივის ციკლში ჩათვალიერება კი რამდენადმე განსხვავებულ მიდგომას მოითხოვს:

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

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

მოვიყვანოთ მაგალითი:

ლისტინგი 30

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

პროგრამის შესრულების შედეგად ეკრანზე აისახება შემდეგი ინფორმაცია:

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

მრავალგანზომილებიანი მასივის გამოტანა

ზემოთ განხილული მეთოდების გამოყენებით შეგვიძლია ეკრანზე ავსახოთ მრავალგანზომილებიანი მასივებიც:

ლისტინგი 31

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

ეკრანზე გამოვა ასოცირებული მასივების მასივი:

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

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

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

ამ პროგრამაში გარე ციკლი ჩაათვალიერებს ნომრების მიხედვით მოწესრიგებულ $users მასივს და მის თითოეულ ელემენტს მოათავსებს $val ცვლადში. $users-ის თითოეული ელემენტი თვითონ არის მასივი, მაგრამ, რადგან ეს მასივი ასოცირებული გახლავთ, შიდა ციკლით მისი ჩათვალიერებისას გამოიყენება უკვე განხილული მიდგომა – თითოეული ქვეელემენტის სახელი და მნიშვნელობა განთავსდება $key და $final_val ცვლადებში.

მასივების გაერთიანება

მასივების გაერთიანებას ემსახურება array_merge ჩაშენებული ფუნქცია:

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

მასივისთვის ელემენტების დამატება

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

მაგალითად:

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

საინტერესოა, რომ ამ დროს ხდება არა მარტო $first მასივისთვის სამი ახალი ელემენტის დამატება, არამედ $total ცვლადისათვის მოდიფიცირებული მასივის სიგრძის მნიშვნელობის მინიჭებაც. ამოვბეჭდოთ ეს სიდიდე და ეკრანზე ავსახოთ მასივის ელემენტები:

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

ზედა მაგალითში გამოყენებულია შენიღბვის (მასკირების) ხერხი – “\” სიმბოლოს დასმა “$” სიმბოლოს წინ ინტერპრეტატორს აცნობებს, რომ დახრილი ხაზის მომდევნო სიტყვა $first მან უნდა აღიქვას არა როგორც ცვლადი, არამედ როგორც “$first” შიგთავსის მქონე სტრიქონი.

მასივის პირველი ელემენტის ამოგდება

აღნიშნულ მიზანს ემსახურება array_shift() ფუნქცია.

ქვემოთ, პროგრამაში ციკლის მეშვეობით ხდება მასივისთვის პირველი ელემენტის მოცილება მანამ, სანამ ის არ დაცარიელდება, რაც მოწმდება count() ფუნქციით. ამოგდებული ელემენტის მნიშვნელობა ენიჭება $val ცვლადს. ციკლის თითოეულ ბიჯზე ხდება ამ ცვლადისა და მასივში დარჩენილი ელემენტების რიცხვის ამობეჭდვა.

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

მასივიდან ქვემასივის მიღება

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

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

მასივების სორტირება

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

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

შესაძლებელი არის ასოცირებული მასივის სორტირება ველების სახელების მიხედვითაც, რაშიც დაგვეხმარება ksort() ფუნქცია:

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

დავალება–1:

დაწერეთ პროგრამა, რომელშიც რამდენიმე მასივზე ჩატარდება ყველა ზემოთ აღწერილი ოპერაცია.

დავალება–2

ინტერნეტში მოძებნეთ shufle() ფუნქცია, რომელიც თავის თავზე აიღებს მასივის ელემენტების შემთხვევითი წესით განლაგებას - მათ “აჭრას” და გამოიყენეთ მისი შესაძლებლობები სხვადასხვა ტიპის ელემენტების შემცველი მასივებისათვის.

ობიექტები

დაპროგრამებაში შემდგომი, მნიშვნელოვანი ნაბიჯი გადაიდგა წინ, როცა შემოღებული იქნა ობიექტის ცნება.

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

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

თვით ობიექტი წარმოგვიდგება, როგორც გარკვეული კლასის კონკრეტული ეგზემპლარი. ამრიგად, კლასი გამოდის სპეციალური შაბლონის როლში, რომლის მიხედვითაც შესაძლებელია ობიექტის შექმნა.

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

ობიექტებისთვის კი, როგორც წესი, ხდება მათი დამახასიათებელი თვისებების კონკრეტული მნიშვნელობების განსაზღვრა.

აქვე აღვნიშნოთ, რომ დასაშვებია თვისებებისთვის მნიშვნელობების განსაზღვრა კლასის გამოცხადების დროსაც. თუ კლასის ეგზემპლარის შექმნის მომენტში ასეთი თვისებისათვის გარედან გადმოცემული მნიშვნელობის მინიჭება არ მოხდა, ის მიიღებს კლასის შექმნისას მისთვის დუმილით დანიშნულ მნიშვნელობას.

გამოვაცხადოთ რაიმე მარტივი კლასი და მის საფუძველზე განვსაზღვროთ კლასის ეგზემპლარები – ობიექტები – ქვემოთ მოყვანილი წესით:

```
class first_class
    {
     var $name="harry";
    }
$obj1=new first_class();
$obj2=new first_class();

// ახლა მივანიჭოთ მნიშვნელობა პირველი ობიექტის name თვისებას.

// ყურადღება მივაქციოთ თვისების განსაზღვრის წესს!

$obj1 -> name="bob";

// ეკრანზე ავსახოთ ობიექტების თვისების მნიშვნელობები.
```

Print “$obj1 -> name
”; *// გამოდის ახალი მნიშვნელობა* ”bob”
Print “$obj2 -> name
”; *// გამოდის მნიშვნელობა* ”harry” *(დუმილით)*

აღვნიშნოთ, რომ ობიექტიდან თვისებაზე გასვლა ხდება –> ოპერატორის მეშვეობით, თვისების სახელის წინ კი დოლარის სიმბოლო აღარ ფიგურირებს!

ამჯერად განვსაზღვროთ კლასი, რომელიც მეთოდსაც შეიცავს:

ლისტინგი 32

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

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

ხაზგასასმელია ის გარემოება, რომ მეთოდს შეუძლია მიმართოს კლასში არსებულ ნებისმიერ შიდა ცვლადს (თვისებას).

აღნიშნულ შესაძლებლობას უზრუნველყოფს ცვლადისადმი მიმართვისას მისი სახელის წინ $this მაჩვენებლის დასმა.

მოვიყვანოთ მეთოდის შემცველი კლასის და ამ მეთოდით თვისებისადმი მიმართვის მაგალითი:

ლისტინგი 33

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

რადგან მეთოდი ფუნქცია გახლავთ, მას გამოძახებისას, ცხადია, არგუმენტიც (*არგუმენტები*) შეიძლება გადავცეთ. მაგრამ აქ ხაზი უნდა გავუსვათ ერთ მნიშვნელოვან გარემოებას:

თუ კლასში გამოცხადებული მეთოდისა და თვით კლასის სახელები ერთმანეთს ემთხვევა, მაშინ ობიექტის შექმნისას ავტომატურად ხდება ასეთი მეთოდის გამოძახებაც.

აღვნიშნოთ, რომ ამ დროს ჩვენ შეგვიძლია განვსაზღვროთ ობიექტის თვისებათა მნიშვნელობები შესაბამისი არგუმენტების გადაცემით.

ასეთ სპეციალურ ფუნქციებს **კონსტრუქტორები** ეწოდება.

მოვიყვანოთ კონსტრუქტორის მეშვეობით ობიექტის თვისებისათვის დუმილით განსაზღვრული მნიშვნელობის შეცვლის მაგალითი:

ლისტინგი 34

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

```
</head>
<body>
<?php
 class first_class
      {
      var  $name;
      function first_class( $n="anon" )
        {
        $this->name = $n;
        }
       function sayHello()
        {
        print "hello my name is $this->name<BR>";
        }
      }
 $obj1 = new first_class("bob");
 $obj2 = new first_class( "harry");
 $obj3 = new first_class();

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

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

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

დაბოლოს, კლასში თითოეული ობიექტისთვის გამოიძახება სხვა - sayHello() მეთოდიც, რომლის დანიშნულება არის ობიექტის სახელის შემცველი სტრიქონის ფორმირება და მისი ეკრანზე გამოტანა.

მაგალითი

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

დასასრულ, კიდევ ერთი მეთოდით ავსახოთ ცხრილი ეკრანზე.

კლასის თვისებები

პროგრამის აგებას ვიწყებთ ცხრილის შესაქმნელად განკუთვნილი კლასის ფორმირებით:

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

კონსტრუქტორი

ცხრილის სვეტების სახელთა შემცველი $headers მასივის მნიშვნელობები უნდა განისაზღვროს ობიექტის *(ანუ ამ ცხრილის)* შექმნისას. აღნიშნულ მიზანს ემსახურება კონსტრუქტორი-ფუნქცია. უკვე ვიცით, რომ კონსტრუქტორ-ფუნქციის სახელი უნდა ემთხვეოდეს გამოცხადებული კლასის სახელს, ანუ უნდა იყოს Table.

შინაარსობრივი მხარიდან გამომდინარე, ამ კონსტრუქტორ-ფუნქციის არგუმენტის *(რომელიც მასივს წარმოადგენს)* სახელად ავირჩიოთ $headers.

შემდეგ კი, ჩვენთვის უკვე ნაცნობი ხერხით უნდა განისაზღვროს კლასში გამოცხადებული $headers და $cols ცვლადებისათვის შესაბამისი მნიშვნელობები, ანუ დადგინდეს სვეტების სათაურთა მასივის შემცველობა და count() ფუნქციის მეშვეობით დათვლილი იქნას ამ მასივში ელემენტების რაოდენობა.

ამრიგად, კონსტრუქტორს ექნება ასეთი სახე:

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

ობიექტის თვისებებში განთავსებული ინფორმაცია მიწვდომადია ობიექტის ნებისმიერი მეთოდისთვის.

addRow() *მეთოდი*

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

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

array_push ფუნქცია ცხრილს, რომელიც ასევე მასივის სახით წარმოგვიდგება, ახალ სტრიქონს უმატებს. დასამატებელი ელემენტი *(სტრიქონი)* თვითონ გახლავთ მასივი, მაგრამ ის არსებულ მასივს ემატება, როგორც ელემენტი.

ამრიგად, ვიღებთ მრავალგანზომილებიან მასივს - მასივთა მასივს.

AddRowAssocArray() *მეთოდი*

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

პროგრამა შემდეგნაირად აიგება:

- ფუნქცია ზემოთ აღნიშნულ ასოცირებულ მასივს $row-assoc არგუ-მენტის სახით იღებს.
- ვქმნით $row ცარიელ მასივს, რომელიც თანდათანობით შეივსება საჭირო ელემენტებით ასოცირებული მასივიდან.

- $row მასივისათვის ელემენტის დასამატებლად foreach ციკლით ჩათვალიერდება $this–>headers სათაურთა მასივი და მისი ელემენტების მნიშვნელობა თანმიმდევრულად მიენიჭება header ცვლადს.
- isset() ჩაშენებული ფუნქციით ციკლში მოწმდება, არსებობს თუ არა $row-assoc ასოცირებულ მასივში $header სახელის მქონე ელემენტი. თუ ეს ასე არ არის, ნაკლული $row-assoc მასივი ივსება ამ სახელის მქონე ელემენტით, რომელსაც ცარიელი სტრიქონის მნიშვნელობა ენიჭება. ნებისმიერ შემთხვევაში ამას მოსდევს $row მასივის შესაბამისი ელემენტის ფორმირებაც.
- ციკლის დამთავრების შემდეგ განკარგულებაში გადმოგვეცემა $row მასივი, რომელშიც ფიგურირებს $row-assoc ასოცირებულ მასივში მოცემული ელემენტები. ზოგიერთი ადგილი კი მასში, როგორც ვნახეთ, ცარიელი მნიშვნელობითაც შეიძლება განისაზღვროს.
- პროგრამის ბოლოს array_push ჩაშენებული მეთოდით ხდება $table_array ცვლადში განლაგებული მასივისთვის $row მასივის, როგორც ელემენტის, დამატება.
- return true ინსტრუქციით ვიღებთ შეტყობინებას, რომ ფუნქციის მუშაობა ნორმალურად დამთავრდა.

output() მეთოდი

ამ მეთოდით პროგრამას $headers მასივიდან ეკრანზე გამოჰყავს სათაურები ასაგები ცხრილის სვეტებისათვის, ხოლო ცხრილის დანარჩენი უჯრებისათვისვის კი - მნიშვნელობები $table-array მრავალგანზომილებიანი მასივიდან.

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

გადავდივართ შემდეგ სტრიქონზე. foreach ცვლადში ჩათვალიერდება table-array მასივის ელემენტებიც, მაგრამ, რადგანაც ეს მასივი არის მრავალგანზომილებიანი *(მოცემულ შემთხვევაში - ორგანზომილებიანი),* საჭირო ხდება მისი თითოეული ელემენტის *(მასივის)* შიდა foreah ციკლში ჩათვალიერებაც და ცალკეული ელემენტების ეკრანზე გამოყვანა. ზედა დონის თითოეული ელემენტის *(ანუ მასივის)* ამობეჭდვის შემდეგ, პროგრამას კურსორი მომდევნო სტრიქონზე გადაჰყავს.

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

ლისტინგი 35

```
<html>
<head>
  <title> Table კლასი</title>
</head>
<body>
<?php
 class Table
      {
       var $table_array = array ();
       var $headers = array ();
       var $cols;
      function Table ($headers )
         {
          $this->headers = $headers;
          $this->cols = count ( $headers );
         }
      function addRow ($row )
       {
        if ( count ($row ) != $this->cols )
             return false;
             array_push($this->table_array, $row);
             return true;
        }
      function addRowAssocArray ($row_assoc)
         {
            $row = array();
            foreach ($this->headers as  $header)
              {
              if ( ! isset( $row_assoc[$header] ))
                 $row_assoc[$header] = " " ;
                 $row[] = $row_assoc[$header];
              }
            array_push($this->table_array, $row);
            return true;
```

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

პროგრამის მუშაობის შედეგს ექნება სახე:

a b c

1 2 3

4 5 6

6 0 3

კვლავ კლასების შესახებ

მემკვიდრეობითობა

კლასებს მომხმარებლისთვის მრავალფეროვანი სერვისის გაწევა შეუძლიათ. მაგალითად, ადვილად ხდება არსებულის ბაზაზე მოდიფიცირებული კლასების შექმნა. გარდა ამისა, ახალ კლასებს მემკვიდრეობით გადმოეცემათ მშობელი კლასის წარმომადგენლებისათვის დამახასიათებელი ფუნქციური როლებიც.

მოვიყვანოთ არსებულის ბაზაზე მემკვიდრეობით შექმნილი კლასის მაგალითი:

ლისტინგი 36

```
<html>
<head>
  <title>არსებული კლასის ბაზაზე ახლის შექმნა</title>
</head>
<body>
<?php
 class first_class
      {
      var  $name = "harry";

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

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

  class second_class extends first_class
      {
```

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

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

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

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

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

მშობელი კლასის მეთოდების სახეცვლილება

წინა მაგალითში წარმოებული კლასის ობიექტები ზუსტად ისევე იქცეოდნენ, როგორც მშობელი კლასის ობიექტები. რა თქმა უნდა, საჭიროების შემთხვევაში დასაშვებია, ეს ასე არც იყოს.

მაგალითად, შევქმნათ წარმოებული second-class კლასისათვის საკუთარი მეთოდი sayHello():

ლისტინგი 37

```
<html>
<head>
  <title>მეთოდის ხელახლა განსაზღვრა</title>
</head>
<body>
<?php
```

```
class first_class
    {
    var  $name = "harry";

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

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

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

მშობელი კლასის მეთოდის გამოძახება

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

ლისტინგი 38

```
<html>
<head>
 <title>მშობლიური კლასის მეთოდის გამოძახება</title>
</head>
<body>
<?php
 class first_class
      {
      var  $name = "harry";

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

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

  class second_class extends first_class
      {
                    function sayHello()
       {
          print "I'm not going to tell you my name --";
          first_class::sayHello();
       }
      }
        $test = new second_class("son of harry");
        $test->sayHello();
       / /  გამოჰყავს "I'm not going to tell you my name" - -Hello my name is
            son of harry"
?>
</body>
</html>
```

მემკვიდრეობის გადაცემის მაგალითი

მიზნად დავისახოთ ადრე განხილული Table კლასის ბაზაზე შევქმნათ ახალი, HTMLTable-დ წოდებული კლასის შექმნა, რომელსაც დამატებით ფუნქციებს დავაკისრებთ. კერძოდ, მათი მეშვეობით მოხდება ცხრილში გამოსატანი მონაცემების დაფორმატება და ეკრანზე მათი ჩვეულებრივი, html ცხრილის სახით ასახვა. ამ მაგალითში მომხმარებელს შეუძლია, აირჩიოს cellpadding და bgcolor ატრიბუტების მნიშვნელობები.

წინა პროგრამაში უნდა შევიტანოთ შემდეგი ცვლილებები:

ვაცხადებთ table-ს მემკვიდრე HTMLTable კლასს:

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

კონსტრუქტორის შექმნა

ახლად მიღებულ კლასში შევქმნათ საკუთარი კონსტრუქტორი *(საწინააღმდეგო შემთხვევაში ობიექტის შექმნისას მოხდებოდა მშობელი კლასის კონსტრუქტორის გამოძახება):*

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

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

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

SetCellPadding მეთოდის დამატება

კლასში დავამატოთ მისი საკუთარი მეთოდებიც.

მაგალითად, შევქმნათ მეთოდი, რომლის გამოძახების შედეგად cellpadding თვისებისათვის შეიცვლება მისთვის დუმილის წესით გათვალისწინებული მნიშვნელობა:

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

Output() მეთოდი

დაბოლოს, მოგვიწევს მშობლიურ კლასში არსებული Output() მეთოდის მთლიანად გარდაქმნა, რათა შევძლოთ მონაცემების HTML ცხრილის სახით გამოტანა:

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

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

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

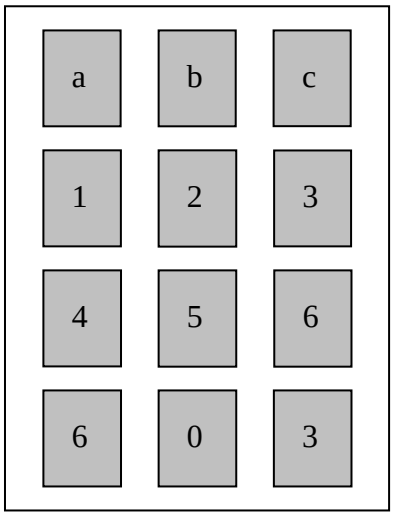
ლისტინგი 39

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

```
if ( ! isset( $row_assoc[$header] ))
    $row_assoc[$header] = " ";
  $row[] = $row_assoc[$header];
  }
array_push($this->table_array, $row);
}
function output ()
 {
    print "<pre>";
    foreach ( $this->headers as $header )
    print "<B>$header</B> ";
    print "\n";
    foreach ( $this->table_array as $y )
     {
       foreach ( $y as $xcell )
       print "$xcell ";
       print "\n";
      }
    print "</pre>";
 }
}
 class HTMLTable extends Table
{
    var $bgcolor;
    var $cellpadding = "2";
    function HTMLTable ( $headers, $bg="#ffffff" )
     {
       Table::Table($headers);
      $this->bgcolor=$bg;
    }
    function setCellpadding( $padding )
```

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

პროგრამის მუშაობის შედეგად ეკრანზე აისახება შემდეგი ცხრილი:



დავალება: *ზემოთ მოტანილ პროგრამაში შეიტანეთ ცალკეული ოპერატორების და/ან მათი ჯგუფების დანიშნულების აღმნიშვნელი კომენტარები.*

ფორმებთან მუშაობა

ჯერ გავიხსენოთ გლობალური ცვლადების დანიშნულება და აღვნიშნოთ, რომ PHP-ში თითოეული გლობალური ცვლადი ინახება ჩაშენებულ ასოცირებულ $GLOBALS მასივში, რომელშიც არსებულ ელემენტებს, ქვემოთ ნაჩვენები წესით, ფუნქციიდანაც შეიძლება მივმართოთ, ადრე განხილული global ინსტრუქციის გამოყენების გარეშე (მასივის ელემენტისათვის ინდექსის როლს ასრულებს ცვლადის სახელი).

მოგვყავს ამ შესაძლებლობის მაილუსტრირებელი მაგალითი

ლისტინგი 40

```
<?php
function test() {
    $foo = "მე გახლავართ ლოკალური ცვლადი.";
    echo '$foo ცვლადის მნიშვნელობა გლობალურ სივრცეში არის: ' .
$GLOBALS["foo"] . "\n";
    echo '$foo ცვლადის მნიშვნელობა მიმდინარე ხედვის არეში არის:  ' . $foo .
"\n";
}
  $foo = "მე გახლავართ გლობალური ცვლადი.";
  test();
?>
```

პროგრამის შესრულების შედეგი იქნება:

$foo ცვლადის მნიშვნელობა გლობალურ სივრცეში არის: მე გახლავართ გლობალური ცვლადი. $foo ცვლადის მნიშვნელობა მიმდინარე ხედვის არეში არის: მე გახლავართ ლოკალური ცვლადი.

შენიშვნა: მივაქციოთ ყურადღება, თუ როგორ აღიქვამს PHP–ს ინტერპრეტატორი ცვლადების მნიშვნელობებს ცალმაგ და ორმაგ ფრჩხილებში, ასევე - მათ გარეშე.

მოვიყვანოთ $GLOBALS მასივისადმი მიმართვის სხვა მაგალითიც:

ლისტინგი 40–1

```
<?php
  $x = 5;
  $y = 10;

 function myTest() {
    $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
 }
  myTest();
  echo $y;    //  15
?>
```

ახლა ვნახოთ, თუ როგორ ხერხდება $GLOBALS მასივში არსებული ცვლადების ჩათვალიერება:

ლისტინგი 41

```
<htmlL>
<head>
  <title> $GLOBALS მასივის ჩათვალიერება </title>
</head>
<body>

<?php

  $user1 = "Bob";
  $user2 = "Harry";
  $user3 = "Mary";
foreach  ( $GLOBALS as $key=>$value )
{
   print "\$GLOBALS[\"$key\"] == $value<br>";
}

?>
```

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

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

```
$GLOBALS["GLOBALS"] = Array
$GLOBALS["user1"] = Bob
$GLOBALS["user2"] = Harry
$GLOBALS["user3"] = Mary
$GLOBALS["value"] = Mary
$GLOBALS["key"] = value
```

$GLOBALS მასივში არსებულ, წინასწარგანსაზღვრულ ცვლადებს სუპერგლობალურ ცვლადებსაც უწოდებენ.

ვნახეთ, რომ ეს ცვლადები ხელმისაწვდომი არიან პროგრამის ნებისმიერი ადგილიდან. არსებობს სხვა სუპერგლობალური ცვლადებიც, მაგალითად, $_POST, S_GET, S_SERVER (იხ. დანართი) და სხვ. მათ შემდგომ გავეცნობით.

მომხმარებლის მიერ შევსებული ფორმის დამუშავების პროგრამა

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

ლისტინგი 42

```
<html>
<body>
 <form action="welcome.php" method="post">
  სახელი: <input type="text" name="name"><br/>
  E-მაილი: <input type="text" name="email"><br/>
  მისამართი: <textarea name="address" rows="5" cols="40"></textarea>
 <input type="submit">
 </form>
</body>
</html>
```

ამ მარტივ ფორმაში განლაგებულია "name" და “email” სახელის მქონე ტექსტური ველები, შედარებით მეტი მოცულობის ტექსტებისათვის textarea ტიპის "address" სახელწოდების ველი და მონაცემთა გადაცემის ღილაკი "submit".

ფორმის შევსების შემდეგ "submit" ღილაკზე დაწკაპუნებით გამოიძახება მიმდინარე საქაღალდეში მყოფი welcome.php პროგრამა, რომელიც დაამუშავებს ფორმიდან გადაცემულ მონაცემებს:

ლისტინგი 43

```
<html>
<body>
 მოგესალმებით <?php echo $_POST["name"]; ?>!<br/>
 თქვენი ელფოსტის მისამართია: <?php echo $_POST["email"]; ?><br/>
 თქვენი მისამართია: <?php echo $_POST["address"]; ?>
</body>
</html>
```

ზემოთ მოყვანილ პროგრამას პრინციპული სიახლე ახასიათებს – მისი გამოძახება ხდება არა უშუალოდ, არამედ HTML-ფორმიდან. პროგრამა სრულდება სერვერზე.

ფორმიდან გადაცემულ მონაცემებს პროგრამები გლობალური ცვლადების მეშვეობით იღებენ.

შესაძლებელია იგივე მონაცემები პროგრამიდან პროგრამას გადაეცეს GET მეთოდითაც. ასეთ შემთხვევაში ბროუზერის სამისამართო უბანში გადასაცემი მონაცემები აღარ ჩანს და, უსაფრთხოების დაცვის თვალსაზრისით, ასეთი მიდგომა უკეთესად ითვლება.

ლისტინგი 42–1

```
<html>
<body>
  <form action="welcome.php" method="get">
    სახელი: <input type="text" name="name"><br/>
    E-მაილი: <input type="text" name="email"><br/>
```

```
მისამართი: <textarea name="address" rows="5" cols="40"></textarea>
 <input type="submit">
</form>
</body>
</html>
```

ლისტინგი 43–1

```
<html>
<body>
მოგესალმებით <?php echo $_GET["name"]; ?>!<br/>
თქვენი ელფოსტის მისამართია: <?php echo $_GET["email"]; ?><br/>
თქვენი მისამართია: <?php echo $_GET["address"]; ?>
</body>
</html>
```

მრავალი მნიშვნელობის მქონე ელემენტების დამუშავება

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

ეს ამოცანა შემდეგნაირად წყდება – ელემენტის სახელის შემდეგ უნდა დაისვას ცარიელი კვადრატული ფრჩხილები:

ლისტინგი 44

```
<html>
<head>
 <title> HTML ფორმა Select ტეგით </title>
</head>
<body>
 <form action="welcome.php" method="post">
    <input type="text" name ="user">
    <br/>
```

```
  <textarea name="address" rows="5" cols="40"></textarea>
    <br/>
  <select name="produqts[]" multiple>
    <option>ვაშლი
    <option>ატამი
    <option>გოგრა
    <option>კომბოსტო
  </select>
   <br/>
   <input type="submit" value="hit it!">
  </form>
</body>
</html>
```

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

გამოძახებულ პროგრამას შეიძლება ასეთი სახე ჰქონდეს:

ლისტინგი 45

```
<html>
<head>
  <title> წინა ფაილიდან გადმოცემული მონაცემების დამუშავება </title>
</head>
<body>
<?php
  echo 'თქვენი სახელია: <b>' . $_POST["user"] . '</b><BR/>';
  echo 'თქვენი მისამართია: ' . $_POST["address"] . '<BR/><BR/>';
  echo 'არჩეული პროდუქტებია: ';
  echo "<ul>\n\n";
 foreach ( $_POST["produqts"] as $_POST["value"] )
  {
```

```
  echo '<li>' . $_POST["value"] . '<BR/>';
  }
 echo "</ul>";
 ?>
</body>
</html>
```

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

HTML-ტექსტისა და PHP-პროგრამების ერთ ფურცელზე განლაგება

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

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

დავიწყოთ HTML ფორმის შექმნა. ის შეიცავს ერთადერთ ტექსტურ ველს. ასეთ შემთხვევაში ფორმის გადასაცემად თანამედროვე ბროუზერებს აღარ სჭირდებათ Submit ღილაკის არსებობა – მიზნის მისაღწევად საკმარისია <Return> კლავიშზე ხელის დაჭერა.

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

ლისტინგი 46

```
<html>
<head>
  <title> თავისი თავის გამომძახებელი HTML ფორმა </title>
</head>
<body>
<form action="<?php print $_SERVER['PHP_SELF'] ?>" method="POST">
   შეიტანეთ რიცხვი: <input type="text" name="guess">
</form>
```

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

ცხადია, ეკრანზე არაფერი აისახება (გარდა ფორმის ელემენტში არსებული შეტანის ველისა), რის გამოც შემდეგი ნაბიჯი იქნება იმავე Web-ფურცელზე php-პროგრამის ტექსტის დამატება.

ამჯერად ვაჩვენოთ ცვლადის გამოყენების უფრო "შინაარსობრივი" მაგალითი:

```
<?php
if(isset($_POST['submit']))
{
   $name = $_POST['name'];
   echo "მომხმარებელმა შეტანის ველის მეშვეობით სერვერს გაუგზავნა სახელი: <b> $name </b>";
   echo "<br>შეგიძლიათ ფორმა გამოიყენოთ ახალი სახელის გადასაგზავნად.";
}
?>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
  <input type="text" name="name"><br>
  <input type="submit" name="submit" value="Submit Form"><br>
</form>
```

ქვემოთ მოყვანილ ამოცანაში კი მომხმარებელმა უნდა გამოიცნოს კომპიუტერის მიერ ჩაფიქრებული რიცხვი. უშედეგოდ დამთავრებული ყოველი ცდის შემდეგ ხელახლა ხდება php პროგრამის გამოძახება.

ამოცანა შემდეგნაირად წყდება:

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

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

შემდეგ კი პირობების შემმოწმებელ ოპერატორებში გარკვევა სირთულეს აღარ წარმოადგენს.

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

ლისტინგი 47

```
<html>
<head>
  <title>რიცხვის გამოცნობა</title>
</head>
<body>
<?php if ( ! isset( $guess ) )
{
$message = "მოგესალმებით! აბა, გამოიცანით, რომელი მთელი რიცხვი ჩავიფიქრე!<br><br><br>";
echo $message;
}
?>
<form action=<?php print $_SERVER['PHP_SELF'] ?> method="POST">
   აკრიფეთ რაიმე მთელი რიცხვი: <input type="text" name="guess">
</form>
</body>
</html>
```

```
<?php
 $num_to_guess = 42;
 $message = " ";
 $guess = (isset($_POST['guess']) ? $_POST['guess'] : null);

if   ( $guess > $num_to_guess )
{
```

```
$message = "თქვენ მიერ ნავარაუდევი რიცხვი ($guess) აჭარბებს ჩემ მიერ ჩაფიქრებულს! <br> გაითვალისწინეთ ეს და კიდევ სცადეთ გამოცნობა! <br><br><br> ";
}
elseif ( $guess < $num_to_guess )
{
$message = "თქვენ მიერ ნავარაუდევი რიცხვი ($guess) მცირეა ჩემ მიერ ჩაფიქრებულზე!<br> გაითვალისწინეთ ეს და კიდევ სცადეთ გამოცნობა!<br><br><br>";
}
else
{
  $message = "<br>გილოცავთ! თქვენ გენიოსი ბრძანდებით!";
  echo $message;
  exit;   // აღარ ხდება პროგრამის მიერ თავისი თავის ხელახლა გამოძახება! გაანალიზეთ ამ ოპერატორის გარეშე როგორი შედეგი გვექნებოდა და შეამოწმეთ ნავარაუდევი!
}
echo $message;
?>
```

დავალება:

ეს პროგრამა ასრულებს მასზე დაკისრებულ ფუნქციას, მაგრამ აქვს რამდენიმე ნაკლი. დააფიქსირეთ ისინი, გაანალიზეთ და გამოასწორეთ მოდიფიცირებულ ვარიანტში (ვარიანტებში).

მოგვყავს ამავე ამოცანის გადაწყვეტის ერთ-ერთი გაუმჯობესებული ვარიანტი:

ლისტინგი 47_1

```
<html>
<head>
  <title>რიცხვის გამოცნობა_1</title>
</head>
<body>
<center>
  <h2>მოგესალმებით! <br>
  <h3>აბა, გამოიცანით, რომელი მთელი რიცხვი ჩავიფიქრე!</h3></h2>
</center>
<form action=<?php print $_SERVER['PHP_SELF'] ?> method="POST">
```

```
    აკრიფეთ რაიმე მთელი რიცხვი: <input type="text" name="guess">
</form>
</body>
</html>

<?php
  $num_to_guess = 42;
  $message = " ";
  $guess = (isset($_POST['guess']) ? $_POST['guess'] : null);

if ( ! isset( $guess ) )
{
 $message = "დაიწყეთ!<br><br><br>";    // შეტყობინება გამოვა მხოლოდ პირველ ჯერზე ანუ რიცხვის აკრეფვის დაწყებამდე!
 }
elseif   ( $guess > $num_to_guess )
{
$message = "თქვენ მიერ ნავარაუდევი რიცხვი ($guess) აჭარბებს ჩემ მიერ ჩაფიქრებულს! <br> გაითვალისწინეთ ეს და კიდევ სცადეთ გამოცნობა! <br><br><br> ";
}
elseif ( $guess < $num_to_guess )
{
$message = "თქვენ მიერ ნავარაუდევი რიცხვი ($guess) მცირეა ჩემ მიერ ჩაფიქრებულზე!<br> გაითვალისწინეთ ეს და კიდევ სცადეთ გამოცნობა!<br><br><br>";
}
else
{
  $message = "<br/><center><strong>გილოცავთ! თქვენ გენიოსი ბრძანდებით!</strong></center>";
  echo $message;
  exit;
}
echo $message;
?>
```

მდგომარეობის დასაფიქსირებლად დამალული ველების გამოყენება

დავუშვათ, გვსურს დავიმახსოვროთ, თუ რამდენი ცდა გამოიყენა მომხმარებელმა. ამ საქმეში შეგვიძლია ე.წ. **დამალული ველი** დავიხმაროთ. ის ტექსტური ველია, რომელიც ეკრანზე უშუალოდ არ აისახება.

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

ლისტინგი 48

```
<?php
echo "<center><h2>მოგესალმებით! <br> <h3>აბა, გამოიცანით, რომელი მთელი რიცხვი ჩავიფიქრე!</h3></h2></center>";
  $num_to_guess = 42;
  $message = " ";
  $guess = (isset($_POST['guess']) ? $_POST['guess'] : null);
  $num_tries = (isset($_POST['num_tries']) ? (++$_POST['num_tries']) : 0);
  echo  "ცდათა რიცხვია " . $num_tries . "<br/></br/>";
if ( ! isset( $guess ) )
{
 $message = "დაიწყეთ!<br><br><br>";
 }
elseif   ( $guess > $num_to_guess )
{
$message = "თქვენ მიერ ნავარაუდევი რიცხვი ($guess) აჭარბებს ჩემ მიერ ჩაფიქრებულს! <br> გაითვალისწინეთ ეს და კიდევ სცადეთ გამოცნობა! <br><br><br> ";
}
elseif ( $guess < $num_to_guess )
{
$message = "თქვენ მიერ ნავარაუდევი რიცხვი ($guess) მცირეა ჩემ მიერ ჩაფიქრებულზე!<br> გაითვალისწინეთ ეს და კიდევ სცადეთ გამოცნობა!<br><br><br>";
}
```

```
    else
    {
      $message = "<br/><center><strong>გილოცავთ! თქვენ გენიოსი ბრძანდებით!</strong></center>";
      echo $message;
      exit;
    }
    echo $message;
    ?>
<html>
<head>
 <title>რიცხვის გამოცნობა_3</title>
</head>
<body>
 <form action=<?php print $_SERVER['PHP_SELF'] ?> method="POST">
  აკრიფეთ რაიმე მთელი რიცხვი: <input type="text" name="guess" >
  <input type="hidden" name="num_tries" value="<?php echo $num_tries ?>" >
</form>
</body>
</html>
```

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

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

ასევე ვიქცევით guess ველისთვისაც. მასაც იმავე წესით ვანიჭებთ მნიშვნელობას:

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

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

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

კლიენტის გადართვა სხვა Web-ფურცელზე

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

აი, ტიპური სათაურის მაგალითი, რომელსაც PHP-პროგრამა ბროუზერს უგზავნის (შესაძლებელია მეტი ინფორმაციის გადაცემაც):

ლისტინგი 49

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

შემდეგ, ჩვენ მიერ ზემოთ განხილულ პროგრამებს ერთი ასეთი ნაკლიც აქვს – მათ მაშინაც გამოჰყავს ეკრანზე შესავსები ფორმა, როცა კლიენტმა რიცხვი უკვე გამოიცნო. შესაძლებელია ამ პროგრამების შედეგნაირად გადაკეთება - რიცხვის გამოცნობის შემთხვევაში მომხმარებელი გადავრთოთ იმ WEB-ფურცელზე, რომელზეც განთავსებული იქნება მხოლოდ მილოცვის ტექსტის შემცველი შეტყობინება. ამ მიზნით ვიყენებთ header() ფუნქციას, რომელსაც არგუმენტად გადაეცემა შესაბამისი კონტენტის, დავუშვათ, congrats.html სახელწოდების ფაილი.

მოგვყავს პროგრამის მოდიფიცირებული ფრაგმენტის მაგალითი:

```
<?php
  ----------------------------
else
    {
      header ( "milocva.html" );
```

```
    exit;
  }
 ---------------------------
?>
```

აქვე აღვნიშნავთ, რომ, პროგრამული კომპონენტების ინსტალაციის თავისებურებებიდან გამომდინარე, header("Location: congrats.html") ოპერატორი ზოგჯერ სასურველ შედეგს ვერ იძლევა. ასეთ შემთხვევაში პრობლემა წყდება ცოტა უფრო რთული გზით:

```
<?php
 ---------------------------
/* Redirect to a different page in the current directory that was requested */
 $host  = $_SERVER['HTTP_HOST'];
 $uri   = rtrim(dirname($_SERVER['PHP_SELF']), '/\\');
 $extra = congrats.php';
 header("Location: http://$host$uri/$extra");
exit;
 ---------------------------
?>
```

მოგვყავს მოდიფიცირებული პროგრამის მაგალითი, რომელშიც header() ფუნქციის დახმარებით მომხმარებელს ვატყობინებთ მილოცვის ტექსტის შემცველი საიტის მისამართს:

ლისტინგი 50

```
<html>
<head>
 <title> რიცხვის გამოცნობა </title>
</head>
<body>

 <center><h2>მოგესალმებით! <br> <h3>აბა, გამოიცანით, რომელი მთელი რიცხვი ჩავიფიქრე!</h3></h2></center>

 <form action=<?php print $_SERVER['PHP_SELF'] ?> method="POST">
  აკრიფეთ რაიმე მთელი რიცხვი: <input type="text" name="guess">

 <input type="hidden" name="num_tries" value="<?php echo $num_tries ?>" >
</form>
```

```
</body>
</html>

<?php
 $num_to_guess = 42;
 $message = " ";

 $guess = (isset($_POST['guess']) ? $_POST['guess'] : null);

if ( ! isset( $guess ) )
{
 $message = "დაიწყეთ!<br><br><br>";    // შეტყობინება გამოვა მხოლოდ პირველ ჯერზე ანუ რიცხვის აკრეფვის დაწყებამდე!
 }
elseif   ( $guess > $num_to_guess )
{
$message = "თქვენ მიერ ნავარაუდევი რიცხვი ($guess) აჭარბებს ჩემ მიერ ჩაფიქრებულს! <br> გაითვალისწინეთ ეს და კიდევ სცადეთ გამოცნობა!

<br><br><br> ";
}
elseif ( $guess < $num_to_guess )
{
$message = "თქვენ მიერ ნავარაუდევი რიცხვი ($guess) მცირეა ჩემ მიერ ჩაფიქრებულზე!<br> გაითვალისწინეთ ეს და კიდევ სცადეთ გამოცნობა!

<br><br><br>";
}
else
{
 /* Redirect to a different page in the current directory that was requested */
  $host  = $_SERVER['HTTP_HOST'];
  $uri   = rtrim(dirname($_SERVER['PHP_SELF']), '/\\');
  $extra = 'milocva.php';
  header("Location: http://$host$uri/$extra");
                              exit;
}
echo $message;

?>
```

მოვიყვანოთ მილოცვის ფურცლის შემცველობის კოდიც:

ლისტინგი 50–1

```
<html>
<head>
  <title> მილოცვის ფურცელი</title>
  <meta charset="utf-8">
</head>
<body>
 <?php
  echo "<center><h1><strong>გილოცავთ! თქვენ გენიოსი
ბრძანდებით!</strong></h1></center>";
 ?>
</body>
</html>
```

php ენა შესაძლებლობას იძლევა, მასზე დაწერილმა პროგრამამ გამოიყენოს Javascript ენის კონსტრუქციებიც. მოვახდინოთ რიცხვის გამოცნობის პროგრამის მოდიფიცირება ამ შესაძლებლობის დემონსტრირებისათვის:

```
<html>
<head>
 <title>რიცხვის გამოცნობის პროგრამაში Javascript-ის გამოყენება</title>
</head>
<body>
 <div id="ABC">
  <form action=<?php print $_SERVER['PHP_SELF']; ?> method="POST">
   <center><strong>აბა, გამოიცანით, რომელი მთელი რიცხვი
ჩავიფიქრე!</strong></center>
   <br><br>აკრიფეთ რაიმე მთელი რიცხვი: <input type="text" name="guess">
  </form>
 </div>
```

```
<?php
 $num_to_guess = 42;
 $message="";
 $guess = (isset($_POST['guess']) ? $_POST['guess'] : null);
 if ( ! isset( $guess ) )
 {
  $message = "დაიწყეთ!<br><br><br>";   // შეტყობინება გამოვა მხოლოდ პირველ ჯერზე ანუ რიცხვის აკრეფვის დაწყებამდე!
 }
 elseif ( $guess > $num_to_guess )
  {
   $message = "თქვენ მიერ ნავარაუდევი რიცხვი ($guess) აჭარბებს ჩემ მიერ ჩაფიქრებულს! <br/> გაითვალისწინეთ ეს და კიდევ სცადეთ გამოცნობა! <br/><br/><br/> ";
   echo $message;
  }
 elseif ( $guess < $num_to_guess )
  {
   $message = "თქვენ მიერ ნავარაუდევი რიცხვი ($guess) მცირეა ჩემ მიერ ჩაფიქრებულზე!<br/> გაითვალისწინეთ ეს და კიდევ სცადეთ გამოცნობა!<br/><br/><br/>";
   echo $message;
   }
 else
   {  // მივაქციოთ ყურადღება, თუ როგორ ხდება ქვემოთ სკრიპტის მეშვეობით კოდში ცვლილებების შეტანა!
      $message = "<script>  document.getElementById('ABC').innerHTML =
    '<center><strong>გილოცავთ, თქვენ გენიოსი ბრძანდებით!</strong></center>';
</script>";
            echo $message;
```

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

ფაილების სერვერზე გადასაცემად საჭირო ფორმები და პროგრამები

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

```
enctype = "multipart/form-data";
```

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

ლისტინგი 51

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

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

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

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

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

ფორმის დახმარებით შესაძლებელია ერთდროულად რამდენიმე ფაილის გადაცემაც. მათთვის სერვერს დამატებითი ცვლადები აქვს გათვალისწინებული *(ამ საკითხს აქ არ განვიხილავთ).*

ახლა კი დავწეროთ პროგრამა, რომელიც ეკრანზე გამოიტანს ინფორმაციას გადაცემული ფაილის შესახებ (თუ ფაილის გაფართოება იქნება GIF, მაშინ ეკრანზე ნახატიც აისახება):

ლისტინგი 52

```
<html>
<head>
  <title> გადაცემული ფაილის დამუშავების პროგრამა </title>
</head>
<?php
  $file_dir= "/home/matt.htdocs/uploads";
  $file_url= "http://www.corrosive.co.uk/matt/uploads";

  if (isset( $fupload ) )
    {
     print "path: $fupload<br>\n";
     print "path: $fupload_name<br>\n";
```

```
    print "path: $fupload_size bytes<br>\n";
    print "path: $fupload_type<p>\n\n";
     if ($fupload_type == "image/gif")
       {
        copy ($fupload, "$file_dir/$fupload_name") or die ("Couldn't copy");
        print "<img src=\"$file_url/$fupload_name\"><p>\n\n";
       }
     }
?>
<body>
  <form enctype="multipart/form-data" action="<?php print $PHP_SELF ?>"
        method="POST">
    <input type="hidden" name="MAX_FILE_SIZE" value="51200">
    <input type="file" name="fulpload"><br>
    <input type="submit" value="Send file!">
  </form>
</body>
</html>
```

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

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

ფაილებთან მუშაობა

ტექსტური ფაილებით მანიპულირება

თავდაპირველად შევქმნათ რაიმე ტექსტური ფაილი, მაგალითად, teqsti.txt სახელის მქონე და განვათავსოთ ის სერვერზე იმავე საქაღალდეში, რომელშიც ჩავწერთ მასთან მომუშავე php პროგრამებს *(ცხადია, შესაძლებელია ფაილის განთავსება სერვერის ნებისმიერ სხვა ადგილზე ან ინტერნეტიდან გამოძახებაც).*

შევიტანოთ ტექსტურ ფაილში რამდენიმე სტრიქონი:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

ამ ფაილის შემცველობა ეკრანზე შეიძლება გამოვიტანოთ შემდეგი ოპერატორით:

```
<?php
 echo readfile("teqsti.txt");
?>
```

გავუშვათ პროგრამა შესრულებაზე და დავინახავთ, რომ სტრიქონები ეკრანზე ერთმანეთზე გადაბმულად აისახება.

საერთოდ, ფაილებით მანიპულირებისას ამჯობინებენ არა readfile(), არამედ fopen() ფუნქციის გამოყენებას. ასეთ შემთხვევაში პარამეტრების გადაცემის გზით მეტი შესაძლებლობებით ვსარგებლობთ.

მოგვყავს ამ ფუნქციის გამოყენების მაგალითი (მეორე პარამეტრი განსაზღვრავს ფაილთან მუშაობის რეჟიმს):

```
<?php
  $myfile = fopen("teqsti.txt", "r") or die("Unable to open file!");
  echo fread($myfile,filesize("teqsti.txt"));
  fclose($myfile);
?>
```

ვხედავთ, რომ თუ ფაილის გაღება ვერ ხერხდება, პროგრამა გასცემს შესაბამის შეტყობინებას.

fopen () ფუნქციისათვის გადაცემული მეორე, r პარამეტრის მნიშვნელობა (Open a file for read only) ფაილის მხოლოდ წაკითხვაზე იძლევა ნებას (ანუ მასში ცვლილებების შეტანა აკრძალულია). ამის შემდეგ კურსორი გადაადგილდება ფაილის დასაწყისში.

მეორე პარამეტრმა შეიძლება მიიღოს სხვა მნიშვნელობებიც:

- w (Open a file for write only) - ფაილი სუფთავდება ჩაწერისათვის. თუ ის არ არსებობს, იქმნება და კურსორი გადაადგილდება ფაილის დასაწყისში.
- a (Open a file for write only) – არსებული ფაილი არ იშლება. კურსორი ექცევა ფაილის ბოლოში. თუ ფაილი არ არსებობს, ის იქმნება.
- x (Creates a new file for write only) – ფაილი იქმნება და იხსნება მხოლოდ ინფორმაციის ჩასაწერად. თუ ფაილი ასეთი სახელით უკვე არსებობს, პასუხად გვიბრუნდება false, თუმცა პროგრამა ფუნქციონირებას აგრძელებს.
- r+ (Open a file for read/write) - ფაილი იღება წაკითხვა-ჩაწერისათვის. კურსორი გადადის ფაილის სათავეში.
- w+ (Open a file for read/write) – ეს ვარიანტი w რეჟიმისგან იმით განსხვავდება, რომ მოცემულ შემთხვევაში დამატებით შესაძლებელია ფაილის შემცველობის წაკითხვაც.
- a+ (Open a file for read/write) – a რეჟიმისგან იმით განსხვავდება, რომ აქ დამატებით ნებადართულია ფაილის შემცველობის წაკითხვაც.
- x+ (Creates a new file for read/write) – x რეჟიმისგან ის განასხვავებს, რომ ამ შემთხვევაში დამატებით შესაძლებელია ფაილის შემცველობის წაკითხვაც.

ქვემოთ მოგვყავს php-პროგრამის ფაილებთან მუშაობის მაგალითები:

```
<?php
  $myfile = fopen("teqsti.txt", "r");
  // ჩავწეროთ რაიმე შესრულებადი კოდი...
  fclose($myfile);
?>
```

```
<?php
  $myfile = fopen("teqsti.txt", "r") or die("Unable to open file!");
   echo fgets($myfile)."<br/>";  // გამოჰყავს თითო-თითო სტრიქონი
   echo fgets($myfile)."<br/>";  //  გამოჰყავს მეორე სტრიქონი და ა.შ.
   echo fgets($myfile)."<br/>";
fclose($myfile);
?>
```

feof () ფუნქცია ამოწმებს, მიღწეული იქნა თუ არა ფაილის ბოლო (EOF). ციკლში ამ ფუნქციის გამოყენება უპრიანია მაშინ, როდესაც გადასარჩევ სიმრავლეში ელემენტების რაოდენობა წინასწარ ცნობილი არ არის.

ქვემოთ მოგვყავს აღნიშნული ფუნქციით სარგებლობის მაგალითი:

```
<?php
$myfile = fopen("teqsti.txt", "r") or die("Unable to open file!");
// გამოდის თითო-თითო სტრიქონი ფაილის ბოლომდე (end-of-file)
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

პროგრამის შესრულების შედეგი იქნება:

AJAX = Asynchronous JavaScript and XML

CSS = Cascading Style Sheets

HTML = Hyper Text Markup Language

PHP = PHP Hypertext Preprocessor

SQL = Structured Query Language

SVG = Scalable Vector Graphics

XML = EXtensible Markup Language

fgetc() ფუნქცია კი ფაილიდან ერთმანეთის მიყოლებით კითხულობს თითო-თითო სიმბოლოს. feof()-ს ფუნქციას, ცხადია, აქაც იგივე როლი აკისრია.

მოგვყავს fgetc() ფუნქციით სარგებლობის მაგალითი:

```
<?php
 $myfile = fopen("teqsti.txt", "r") or die("Unable to open file!");
 // თითო-თითო სიმბოლოს გამოტანა, სანამ არ გავალთ ფაილის ბოლომდე
 while(!feof($myfile)){
 echo fgetc($myfile);
                      }
 fclose($myfile);
?>
```

ამჯერად ეკრანზე მივიღებთ ასეთ სურათს:

AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics XML = EXtensible Markup Language

შენიშვნა: ფაილებთან მომუშავე ყველა ფუნქციის შესასწავლად იხ. დანართი 6, ხოლო ბოლო ცვლილებებთან გასაცნობად საიტი:

https://www.w3schools.com/php/php_ref_filesystem.asp

ფაილების ჩართვა PHP-დოკუმენტში

ნებისმიერ php დოკუმენტში სპეციალური დირექტივების დახმარებით შესაძლებელია სხვადასხვა ტიპის ფაილების ჩართვა, მაგალითად, ტექსტურის, html გაფართოების და რაც განსაკუთრებით მნიშვნელოვანია, თვით php პროგრამული მოდულებისაც. ამ გზით გვეძლევა საშუალება, ჩასართავი ფაილი გამოძახებული იქნეს მრავალი სხვა php დოკუმენტის მიერაც, რაც ფრიად აადვილებს დაპროგრამების პროცესს. კერძოდ, თუ საჭიროა პროგრამულ ფაილში ინფორმაციის კორექტირება, შესაძლებელი ხდება პროცესი განხორციელდეს მხოლოდ ამ ფაილში, ანუ ერთადერთ ადგილას და არა ყველგან.

PHP ენა პროგრამაში სხვა ფაილების ჩასართავად იყენებს ორ დირექტივას (ფუნქციას, ოპერატორს):

include() და require().

ცხადია, ორივე მათგანს სჭირდება ფაილის სახელისა და მისკენ მიმავალი მარშრუტის ჩვენება. აღვნიშნავთ, რომ ამ დირექტივების დანიშნულებებს შორის არსებობს პრინციპული სხვაობა:

require()-ს მივმართავთ მაშინ, როდესაც წინასწარ არის ცნობილი ძირითად PHP დოკუმენტში სხვა ფაილის (ფაილების) ჩართვის აუცილებლობა და, აქედან გამომდინარე, PHP ფაილის კოდში გამოსაძახებელი ფაილების ჩართვის მოთხოვნებს შესაბამის ადგილებზე განვათავსებთ (მაგალითად, ძირითადი პროგრამის თავსა და ბოლოში ვრთავთ შაბლონებად ქცეულ, შესაბამისი დანიშნულების მქონე html ფაილების კოდებს):

```
<?php
 require 'head.html';        // დაწყება
 // შემდეგ მოდის ძირითადი პროგრამის ტექსტი
 require 'php_prog1.php';
 // გრძელდება ძირითადი პროგრამის ტექსტი
 require 'bottom.html';    // დაბოლოება
?>
```

რაც მთავარია, ამ მიდგომისას ძირითადი პროგრამისა და მასში ჩართული PHP ფაილების ინტერპრეტირება ხდება ერთდროულად, რაც, ცხადია, მნიშვნელოვნად ამცირებს პროგრამის შესრულების დროს, განსხვავებით include() დირექტივისადმი მიმართვისგან. ბოლო შემთხვევაში ამა თუ იმ მოდულის გამოძახება-დამუშავების საჭიროება, უმეტეს წილად წინასწარ ცნობილი არ არის, რის გამოც პროცესი დინამიკურად წარიმართება, საჭირო ფაილის გამოძახებისას ძირითადი პროგრამის შესრულება წყდება და ის მანამდე არ განახლდება, სანამ გამოძახებული მოდული არ დამუშავდება.

მოვიყვანოთ ამ ფუნქციების გამოყენების მაგალითები:

ლისტინგი 53

```
<!DOCTYPE html>
<html>
<head>
  <title>include ფუნქციის გამოყენება</title>
</head>
<body>
   <h1>Welcome to my home page!</h1>
   <p>Some text.</p>
   <p>Some more text.</p>
```

<?php include 'footer.php'; // ამ ადგილას ჩაირთვება გამოძახებული პროგრამა, რომლის დანიშნულებაა მიმდინარე წლის გამოთვლა და გამოსატან ტექსტში ჩასმა ?>

</body>

</html>

თუ გამოძახებული ფაილი, ვთქვათ, მხოლოდ ასეთი სტრიქონის შემცველია: I have been included! - ის დამუშავდება, როგორც უბრალო HTML-ტექსტი და ასევე აისახება ეკრანზე. მაგრამ თუკი გვსურს, რომ გამოძახებული ფაილი აღქმული იქნეს, როგორც PHP პროგრამა, ის შესაბამის ტეგებში უნდა მოვათავსოთ. განსახილველ შემთხვევაში საჭიროა, სწორედ ასე მოვიქცეთ:

```
<?php
 echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";
  // მიმდინარე თარიღიდან გამოიყოფა წელი
?>
```

განვიხილოთ მოცემულ ფაილში გამოძახებული პროგრამის ჩართვის კიდევ რამდენიმე მაგალითი:

ლისტინგი 53–1

```
<html>
<body>
  <div class="menu">
  <?php include 'menu.php'; // ამ ადგილას ჩაირთვება გამოძახებული პროგრამა, რომლის დანიშნულებაა დისფლეიზე მენიუს გამოტანა ?>
  </div>
   <h1>Welcome to my home page!</h1>
   <p>Some text.</p>
   <p>Some more text.</p>
</body>
</html>
```

გამოსაძახებელ პროგრამაში ასახულ მენიუს შეიძლება ჰქონდეს, მაგალითად, ასეთი სახე:

```
<?php
echo '<a href="/default.asp">Home</a> -
```

```
        <a href="/html/default.asp">HTML Tutorial</a> -
        <a href="/css/default.asp">CSS Tutorial</a> -
        <a href="/js/default.asp">JavaScript Tutorial</a> -
        <a href="default.asp">PHP Tutorial</a>';
    ?>
```

ლისტინგი 53–2

```
<html>
<body>
<h1>Welcome to my home page!</h1>
<?php include 'vars.php';  echo "I have a $color $car.  ";
// ჩაირთვება პროგრამა, რომლის დანიშნულებაა echo ოპერატორში გამოყენებული ცვლადებისათვის მნიშვნელობების განსაზღვრა.
?>
</body>
</html>
```

ხოლო პროგრამას შეიძლება ჰქონდეს ასეთი სახე:

```
<?php
  $color='red';
  $car='BMW';
?>
```

კიდევ ერთი მაგალითი:

ლისტინგი 54

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

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

მოვიყვანოთ მაგალითები:

ლისტინგი 55

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

ლისტინგი 56

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

{ ეს ტექსტი ეკრანზე, ცხადია, ვერ აისახება! }

უნდა აღვნიშნოთ, რომ include() ინსტრუქციის გამოყენება შესაძლებელია პირობით ოპერატორებსა და ციკლებშიც:

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

ლისტინგი 57

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

```
    print "ვრთავთ $incfile  ფაილს <br>";
    include ("$incfile");
    print "<p>";
   }
?>
</body>
</hmtl>
```

ამ პროგრამის შესრულების შედეგად Web-ფურცლის ტექსტში ჩაემატება 3 ფაილი:

incfile1.txt, incfile2.txt და incfile3.txt.

ვხედავთ, რომ მოხდა გამოსაძახებელი ფაილების სახელების ავტომატური გენერირება.

შენიშვნა: *დიდი მოცულობის პროგრამებში გამორიცხული არ არის, შეცდომის შედეგად ერთი და იგივე ფაილი რამდენჯერმე იქნეს გამოძახებული, რაც, ცხადია, შეანელებს სისტემის მუშაობას. ამ არასასურველი მოვლენის თავიდან აცილება შესაძლებელი ხდება* ***require()*** *და* ***include()*** *დირექტივების ნაცვლად* ***require_once()*** *და* ***include_once*** *დირექტივების გამოყენებით.*

სანამ ფაილთან მუშაობას დავიწყებდეთ, სასურველია ფუნქციების დახმარებით ამ ფაილის შესახებ ინფორმაციის მიღება-შემოწმებაც. მაგალითად, ფუნქცია file_exists ("*ფაილის_სრული_მისამართი* ან მხოლოდ *ფაილის_დასახელება*") ამოწმებს ფაილის არსებობას. გაცემულ კითხვაზე დადებითი პასუხი გამოიხატება ფუნქციის მიერ true-ს დაბრუნებით, საწინააღმდეგო შემთხვევაში კი გადმოგვეგზავნება false:

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

არსებობს ფაილის სხვა თვისებების *(სტატუსის, სიგრძის, შექმნის თარიღის და ა.შ.)* შესახებ ინფორმაციის მომწოდებელი ფუნქციებიც. ასევე, PHP-პროგრამების მეშვეობით შესაძლებელია კატალოგების შექმნა, კორექტირება და განადგურება *(აქ ამ საკითხებს არ განვიხილავთ).*

დავალება:

ინტერნეტში მოიძიეთ ეს ფუნქციები და დაწერეთ პროგრამები, რომლებშიც გამოიყენებთ მათ შესაძლებლობებს.

გუნდების ქვეჯგუფებში განაწილების ამოცანა

წინა პარაგრაფებში შესწავლილი მასალების საფუძველზე შესაძლებელია PHP ენაზე დავწეროთ პროგრამები საშუალო სირთულის ამოცანებისათვის, მაგალითად, ასეთისათვის:

მოითხოვება n რაოდენობის ქვეჯგუფში თანაბრად (ან დაახლოებით თანაბრად) განაწილდეს m რაოდენობის გუნდი.

პირობები:

ა) გუნდები რანჟირებულია სიძლიერის მიხედვით და განაწილებისას დაცული უნდა იქნეს ე. წ. კალათური პრიორიტეტების პრინციპი. ამასთან, ცხადია, რომ $m > n$.

ბ) ამოცანის გადაწყვეტის გასამარტივებლად ჩავთვალოთ, რომ m ზემოდან შემოზღუდული რიცხვია.

ქვემოთ მოგვყავს ამ ამოცანის გადაწყვეტის ერთ-ერთი ვარიანტი *(გუნდებისათვის კალათის არჩევისას ვსარგებლობთ შემთხვევითი რიცხვების გენერატორით. კომენტარები გამოყენებული არის ამოცანის შინაარსობრივი მხარის აღსაწერად):*

```
<html>
<head>
  <title>კენჭისყრა</title>
</head>
<meta charset = "UTF-8">
<body bgcolor = 'purple'>
 <form action="index.php" method="post">
 <center>
 <p><font size=6> <b>გუნდების რიცხვი: </b> <input type="text" name="გუნდებისრაოდ" /></font></p>
 <p><font size=6> <b>ჯგუფების რიცხვი: </b> <input type="text" name="ჯგუფებისრაოდ" /></font></p>
 <p><input type="submit" /></p>
 </center>
</form>

<?php
```

```
if ($_SERVER['REQUEST_METHOD'] == 'POST')
{
 class Table
      {
        var $table_array = array();
        var $headers = array();
        var $cols;
      function Table ($headers)
        {
         $this->headers = $headers;
         $this->cols = count($headers);
        }
      function addRow ($row )
       {
        if ( count($row) != $this->cols )
             return false;
             array_push($this->table_array, $row);
             return true;
        }
      function addRowAssocArray($row_assoc)
        {
           $row = array();
           foreach ($this->headers as  $header)
             {
             if ( ! isset( $row_assoc[$header] ))
                $row_assoc[$header] = " " ;
             $row[] = $row_assoc[$header];
             }
           array_push($this->table_array, $row);
           return true;
        }
       function output()
        {
           print "<pre>";
           foreach ($this->headers as $header)
             print  "<b>$header</B> ";
```

```
            print "\n";
            foreach ($this->table_array as $y)
              {
                 foreach ($y as $xcell)
                    print "$xcell ";
                    print  "\n";
              }
          print "</pre>";
       }
     }
```

$gundraod=$_POST["გუნდებისრაოდ"]; // ცვლადს ენიჭება გადმოცემული მნიშვნელობა.

$jgufraod=$_POST["ჯგუფებისრაოდ"]; // ცვლადს ენიჭება გადმოცემული მნიშვნელობა. მეორე სტრიქონიდან (კალათიდან) დაწყებული, ყოველ ჯერზე $jgufraod სიდიდით იზრდება მოცემულ კალათაში გასანაწილებელი გუნდების ნომრები წინა კალათასთან შედარებით.

$k=ceil($gundraod/$jgufraod); // გამოითვლება ჯგუფში მოხვედრილი გუნდების შესაძლო მაქსიმალური რიცხვი, რომლის დამრგვალება ხდება მეტობით.

$mas=array(1,2,3,4); // გუნდების ნომრების შემცველი დამხმარე მასივი (თავდაპირველი მნიშვნელობებით).
$el=0; // დამხმარე ელემენტი, გამოყენებული მასივის აჭრისას.

$test = new table (array("a","b","c", "d")); // კლასის ეგზემპლარის შექმნა. ცხრილის სვეტების სათაურების გადაცემა.

// begin - $k რაოდენობის სტრიქონის (კალათის) ფორმირება და ცხრილში დამატება.

```
for ($x=1; $x<=$k; $x++)
{
   // begin - $mas მასივის რანდომიზება

       for ($m1=0; $m1<=25; $m1++) // "კარტის" აჭრა ხდება 26-ჯერ!
      {
```

```
        $gr = rand(1,$jgufraod)-1;      // ერთის გამოკლება აქ და მომდევნო
        ოპერატორში (ასევე - უფრო ქვემოთაც) ხდება იმის გამო, რომ PHP-ის მასივში
        პირველი ელემენტის ინდექსი ნულია!!!
       $gr1= rand(1,$jgufraod)-1;

         // begin - მასივში შემთხვევითობის წესით ამორჩეული 2 ელემენტი
           ადგილებს ცვლის.
          $el=$mas[$gr1];
           $mas[$gr1]=$mas[$gr];
            $mas[$gr]=$el;
        // end - მასივში შემთხვევითობის წესით ამორჩეული 2 ელემენტი ადგილებს
         ცვლის.
       }
   // end - $mas მასივის რანდომიზება

   $test->addRow($mas); // ცხრილში სტრიქონის (კალათის) დამატება

    // begin - მომდევნო დასამატებელი სტრიქონისათვის (კალათისათვის) ბაზის
       მომზადება
       for ($p=0; $p<=$jgufraod-1; $p++)
      {
       $mas[$p]=$mas[$p]+$jgufraod; // კალათაში მოსახვედრი გუნდებისათვის
           ნომრების ზრდა - საწყისი პოზიციის განსაზღვრა
         // შემოწმება ბოლო, შესაძლოა  ნაკლული ჯგუფისათვის და ასეთ
         შემთხვევაში ზედმეტი ნომრების განულება:
          if ($mas[$p]>$gundraod) {$mas[$p]=0;}
       }

    // end  -  მომდევნო დასამატებელი სტრიქონისათვის (კალათისათვის) ბაზის
       მომზადება

}
// end - $k რაოდენობის სტრიქონის (კალათის) ფორმირება და ცხრილში დამატება

     $test->output();  // საბოლოო შედეგის (კლასის ეგზემპლარის - ჯგუფებში
                       განაწილებული გუნდების ცხრილის) გამოტანა
}

?>
```

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

შენიშვნა:

რადგანაც პროგრამის კოდი „ვორდშია“ გადმოტანილი, ფურცლის არის შეზღუდულობის გამო გრძელი კომენტარები ზოგჯერ მომდევნო სტრიქონზე გადადის და მათთვის კომენტარის აღმნიშვნელი ნიშანი (//) აქ აღარ ფიგურირებს!

DBM-ფუნქციებთან მუშაობა

PHP პროგრამებს შეუძლია მუშაობა ისეთ მძლავრ მონაცემთა ბაზებთან, როგორიცაა, მაგალითად, ORACLE, SQL, MySQL და სხვ., თუმცა მონაცემების შენახვა ბაზის ფაილების სახით და მათთან მუშაობა მაშინაც არის შესაძლებელი, როცა ზემოთ ჩამოთვლილი ბაზები ჩვენს განკარგულებაში არ იმყოფება. საქმე ისაა, რომ PHP-ს ე. წ. DBM-ფუნქციების მეშვეობით შეუძლია მონაცემთა ბაზის ფუნქციების ემულირება. ცხადია, ასეთი გზით შექმნილ ელემენტარულ ბაზებს ზემოთ დასახელებული ბაზების ყველა შესაძლებლობა არ გააჩნია, მაგრამ რიგ პრაქტიკულ შემთხვევაში მათი დახმარებით წარმატებით წყდება ჩვენ მიერ დასახული ამოცანები.

DBM-მონაცემთა ბაზის გახსნა-დახურვა

ამ მიზანს ემსახურება dbmopen() ფუნქცია. მას გადავცემთ შემდეგ ორ არგუმენტს:

DBM-ფაილებამდე გზას და ალმების სტრიქონს.

განსახილველი ფუნქციის გამოსაყენებლად ვქმნით ცვლადს - გაღებული მონაცემთა ბაზის იდენტიფიკატორს, რომლითაც შემდგომ სარგებლობს სხვა DBM-ფუნქციებიც.

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

ქვემოთ ჩამოთვლილი ალმები განსაზღვრავს მონაცემთა ბაზასთან მუშაობის რეჟიმებს:

- r – *მონაცემთა ბაზა იღება მხოლოდ წაკითხვისათვის.*
- w – *მონაცემთა ბაზა იღება წაკითხვისა და ჩაწერისთვის.*

- c – *დასაშვებია მონაცემთა ბაზის შექმნა (ანდა მისი წაკითხვა და კორექტირება, თუ ის უკვე არსებობს).*
- n – *იქმნება ახალი მონაცემთა ბაზა (და ნადგურდება ამ სახელის მქონე ბაზის შემცველობა, თუკი ის უკვე არსებობს).*

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

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

მონაცემთა ბაზასთან მუშაობის დასამთავრებლად ფუნქციაში მიეთითება გაღებული ბაზის იდენტიფიკატორი. მოცემულ შემთხვევაში მას ექნება ასეთი სახე:

```
dbmclose($dbh);
```

ბაზებში მონაცემების დამატება

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

ლისტინგი 58

```
<html>
<head>
  <title> DBM_მონაცემთა ბაზაში მონაცემთა დამატება </title>
</head>
<body>
 Adding produts now...
 <?php
  $dbh = dbmopen( "./data/products", "c") or die( "couldn't open DBM" );
  dbminsert($dbh, "snic crewdriver", "23.20" );
  dbminsert($dbh, "Tricorder", "55.50" );
  dbminsert($dbh, "ORAC AI", "2200.500" );
  dbminsert($dbh, "HAl 2000", "4500.50" );
  dbmclose($dbh);
 ?>
</body>
</html>
```

ხაზი გავუსვათ შემდეგ გარემოებას:

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

თუ ამის შემდეგ შეცდომით იგივე სახის მქონე ველის დამატებას შევეცდებით, ქმედებას მოჰყვება ასეთი შედეგი:

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

ბაზის მონაცემთა კორექტირება

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

ლისტინგი 59

```
<html>
<head>
  <title> DBM_მონაცემთა ბაზაში მონაცემთა დამატება ან შეცვლა
</title>
</head>
<body>
Adding produts now...
 <?php
  $dbh = dbmopen( "./data/products", "c") or die( "couldn't open DBM" );
   dbmreplace($dbh, "snic screwdriver", "25.20" );
   dbmreplace($dbh, "Tricorder", "56.50" );
   dbmreplace($dbh, "ORAC AI", "2209.50" );
   dbmreplace($dbh, "HAl 2000", "4535.50" );
   dbmclose($dbh);
 ?>
</body>
</html>
```

ბაზიდან მონაცემების წაკითხვა

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

მაგალითად, Tricorder ელემენტის ფასის გასაგებად ვწერთ შემდეგ ოპერატორს:

```
$price = dbmfetch ($dbh, "Tricorder");
```

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

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

ვთქვათ, გვაინტერესებს ბაზაში შეტანილი ყოველი პროდუქტის ფასი. ასეთ შემთხვევებში დაგვეხმარება dbmfirstkey() ფუნქცია, რომელიც გვიბრუნებს ბაზაში პირველი ელემენტის სახელს. შესაძლებელია, ეს ელემენტი პირველად შეტანილი არც იყოს *(ასეთია DBM ბაზის შიდა ორგანიზების თავისებურება),* თუმცა ეს საქმეს დიდად არ გვირთულებს - dbmnexthey() ფუნქციის მეშვეობით შეგვიძლია მივადგეთ ბაზის სხვა ელემენტებსაც *(აქ მათ გასაღებებს უწოდებენ).* ელემენტის მნიშვნელობის წაკითხვა კი, როგორც ვიცით, ხორციელდება dbmfetch() ფუნქციით.

სამივე ფუნქციისათვის საჭირო არის მონაცემთა ბაზის იდენტიფიკატორის მითითება.

მივიღოთ ინფორმაცია ბაზაში არსებული ყოველი საქონლის შესახებ:

ლისტინგი 60

```
<html>
<head>
   <title> DBM_მონაცემთა ბაზიდან მონაცემთა წაკითხვა </title>
</head>
<body>
Here at the Impossible Gadget Shop
we're offering the following exciting
products:
<p>
<table border=1 cellpadding ="5">
  <tr>
   <td aling="center"> <b>product</b></td>
   <td aling="center"> <b>price</b></td>
  </tr>
 <?php
```

```
$dbh = dbmopen( "./data/products", "c") or die( "couldn't  open DBM" );
   $skey = dbmfirstkey($dbh);
   while ($skey !="")
   {
      $value=dbmfetch($dbh,$key);
      print "<tr><td align =\"left\">$skey </td>";
      print "<td align = \"right\">\$$value </td></tr>";
      $key =dbmnextkey($dbh,$key);
}
 dbmclose($dbh);
 ?>
 </table>
</body>
</html>
```

ბაზაში ელემენტის არსებობის შემოწმება

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

```
if (dbmexists ($dbh, "Tricorder"))
    print dbmfetch ($dbh, "Tricorder"))
```

მონაცემთა ბაზიდან ელემენტის ამოგდება

მარტივად ხორციელდება dbmdelete() ფუნქციის მეშვეობით.

მოვიყვანოთ მაგალითი:

```
dbmdelete($dbh, "Tricorder")
```

ბაზაში უფრო რთული სტრუქტურების შენახვა

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

PHP პროგრამის მონაცემთა ბაზებთან კავშირი MySQL-ის მაგალითზე

MySQL არის მონაცემთა ის ბაზა, რომელსაც უმრავლეს შემთხვევაში იყენებენ PHP ენაზე დაწერილი პროგრამები, რაც განპირობებული არის ამ ბაზის შემდეგი ღირსებებით:

- MySQL-ს განათავსებენ სერვერზე;
- ის წარმატებით მუშაობს როგორც ლოკალურ, ასევე გლობალურ ქსელებში, ინტერნეტის ჩათვლით;
- ერთნაირად მაღალი ეფექტიანიანობით ხასიათდება როგორც მცირე, ასევე დიდი ზომის დანართების შექმნისას;
- გამოირჩევა მაღალი სწრაფქმედებით, გამოყენების სიმარტივით და დიდი საიმედოობით;
- მონაცემებთან მუშობისას იყენებს სტანდარტული ენის შესაძლებლობებს;
- კომპილირებადია და შეფერხებების გარეშე მუშაობს, ფაქტობრივად, ყველა პლატფორმასთან;
- ხასიათდება კროსპლატფორმირებულობის თვისებით – Windows-ზე შექმნილს შეუძლია შეუფერხებლად იმუშაოს Unix პლატფორმაზეც;
- ბაზასთან მუშაობა შესაძლებელია ავტონომიურ რეჟიმშიც ანუ PHP თუ სხვა ენაზე დაწერილი პროგრამიდან მიმართვების გარეშეც; ამასთან, გრაფიკული ინტერფეისის გამოყენებით;
- მისი გადმოწერა შესაძლებელია უფასოდ შემდეგი მისამართიდან: www.mysql.com;
- MySQL, ფაქტობრივად, სტანდარტია ისეთი საიტებისათვის, რომლებთაც სჭირდება უზარმაზარი მოცულობის და მეტად ეფექტიანად მომუშავე მონაცემთა ბაზები, როგორიცაა, მაგალითად, Facebook, Twitter და Wikipedia;
- დასასრულ, MySQL-ის შემქმნელი არის ცნობილი კორპორაცია Oracle.

MySQL ბაზასთან მუშაობისას PHP საშუალებას იძლევა გამოვიყენოთ მასთან ურთიერთობის 3 ხერხი:

- MySQLi - ობიექტზე ორიენტირებული;
- MySQLi - პროცედურული;
- PDO (PHP Data Objects) – ესეც ობიექტზე ორიენტირებული მიდგომა (თუ რა განასხვავებს პირველი ვარიანტისაგან, იხ. ქვემოთ).

შევნიშნავთ, რომ MySQLi სახელწოდებაში i სიმბოლო აღნიშნავს "გაუმჯობესებულს" (2012 წლამდე არსებული PHP ენის ვერსიები იყენებდნენ MySQL გაფართოებას). როდესაც ვსარგებლობთ php5 MySQL პაკეტით, როგორც წესი, ავტომატურად იგულისხმება, რომ საქმე გვაქვს MySQLi გაფართოებასთან

PDO ვარიანტს, წინა ორისაგან განსხვავებით, ის უპირატესობა აქვს, რომ მას შეუძლია კავშირი დაამყაროს არა მარტო MySQL, არამედ 12 სხვადასხვა მონაცემთა ბაზასთან. MySQLi მიდგომის ღირსება კი არის პროცედურული API-ის გამოყენების შესაძლებლობის ფლობა.

მონაცემთა ბაზასთან MySQLi გაფართოებით წვდომის შესაძლებლობა პაკეტის დაყენებისას, როგორც წესი, უკვე ჩადებულია, მაგრამ თუ ასე არაა, მივმართავთ საიტს:

http://php.net/manual/en/mysqli.installation.php

ხოლო PDO შესაძლებლობის ინსტალირებისათვის კი:

http://php.net/manual/en/pdo.installation.php.

ამის შემდეგ, პირველი, რაც უნდა გავაკეთოთ, ეს გახლავთ სერვერთან დაკავშირება. თავიდანვე აღვნიშნოთ, რომ როგორც სერვერთან დამაკავშირებელი, ისე mysqli ბაზასთან მუშაობისათვის განკუთვნილი სხვა ოპერაციები შესაძლებელია შესრულდეს არა მარტო უშუალოდ პროგრამიდან:

დასაშვებია, ბროუზერის სამისამართო სტრიქონში ჩავწეროთ localhost/phpmyadmin და ეკრანზე გამოვა ამ მიზნებისათვის განკუთვნილი ფანჯარა. მასში განთავსებულია ყველა საშუალება ბაზასთან დაგეგმილი ოპერაციების წარმართვისათვის. ეს ოპერაციები, როგორც წესი, გრაფიკული ინტერფეისის დახმარებით, ხორციელდება საკმაოდ მარტივად. თუმცა, დინამიკაში იმავე ოპერაციების ჩასატარებლად, რაც, უწინარეს ყოვლისა, მოეთხოვება სერვერულ პროგრამებს, PHP ენა ფლობს ყველა შესაბამის ფუნქციას (ქვემოთ გავეცნობით მათგან უფრო ხშირად გამოყენებულებს).

ზემოთ ხსენებული მიზნის მისაღწევად (სერვერთან დაკავშირება) PHP-ში გათვალისწინებული არის mysqli_connect() ფუნქცია, რომელსაც უნდა გადაეცეს შემდეგი სამი არგუმენტი:

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

თუ სერვერთან დაკავშირება მოხერხდა, ფუნქცია გვიბრუნებს **მიერთების იდენტიფიკატორს,** რომელსაც იმახსოვრებენ ცვლადის მნიშვნელობად და იყენებენ ბაზასთან შემდგომი მუშაობისთვის.

აღნიშნულ ფუნქციაზე დაყრდნობით, ზემოთ განხილული სამივე შემთხვევისათვის ვაჩვენოთ იმ სერვერთან შეერთების მაგალითები, რომელზეც განთავსებულია (ან ამას ვაპირებთ) ჩვენთვის საინტერესო მონაცემთა ბაზა.

სერვერთან შეერთება

MySQLi - ობიექტზე ორიენტირებული

ლისტინგი SQL_1 (61)

```
<?php
  // Example (MySQLi Object-Oriented)
  echo "წინ, პროგრამირებაში ახალი მწვერვალების დასაპყრობად! ";
  $servername = "localhost";
  $username = "root"; // მომხმარებლისთვის root სახელი დუმილით არის გათვალისწინებული.
  $password = "";          // root სახელის მქონე მომხმარებლისათვის თავდაპირველად დასაშვებია, პაროლი არ მიეთითოს.
 // ვქმნით სერვერთან კავშირს!
  $conn = new mysqli($servername, $username, $password);
 // ვამოწმებთ შედეგს!
  if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
                                }
   echo "სერვერთან კავშირი დამყარდა! ";
  //  $conn->close(); // კავშირი ავტომატურად წყდება სცენარის დამთავრებისთანავე, მაგრამ  თუ ამის გაკეთება მანამდეა საჭირო, მაშინ გამოიყენება სწორედ ეს $conn->close() ოპერატორი.
?>
```

აღვნიშნოთ, რომ კომპიუტერის სახელად "localhost"-ის გამოყენებით კავშირს ვამყარებთ ჩვენ კომპიუტერზე განთავსებულ MySQL მონაცემთა ბაზასთან.

MySQLi – პროცედურული

ლისტინგი SQL_2 (62)

```
<?php
 // Example (MySQLi Procedural)
 echo "წინ, პროგრამირებაში ახალი მწვერვალების დასაპყრობად!";
 $servername = "localhost";
 $username = "root";
 $password = "";
 //  ვქმნით სერვერთან  კავშირს!
 $conn = mysqli_connect($servername, $username, $password);
 // ვამოწმებთ შედეგს!
 if (!$conn) {
   die ("სერვერთან კავშირი ვერ დამყარდა:  " . mysqli_connect_error());
                }
  echo "სერვერთან კავშირი დამყარდა! ";
  // mysqli_close($conn);     // კავშირი ავტომატურად წყდება სცენარის დამთავრებისთანავე, მაგრამ,  თუ ამის გაკეთება მანამდეა საჭირო, მაშინ გამოიყენება სწორედ ეს mysqli_close($conn)  ოპერატორი.
?>
```

PDO (PHP Data Objects)

ლისტინგი SQL_3 (63)

```
<?php
 // Example (PDO)
 echo "წინ, პროგრამირებაში ახალი მწვერვალების დასაპყრობად!";
 $servername = "localhost";
 $username = "username";
```

```
 $password = "password";

try {
   // ვქმნით კავშირს!
   $conn = new PDO("mysql:host=$servername;dbname=myDB",
$username, $password);
   // შეცდომის აღმოჩენის რეჟიმის დაყენება
   $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
   echo "კავშირი დამყარდა!";
   }
catch(PDOException $e)
   {
   echo "კავშირი ვერ დამყარდა: " . $e->getMessage();
   }
//    $conn = null;    // კავშირი ავტომატურად წყდება სცენარის
დამთავრებისთანავე, მაგრამ  თუ ამის გაკეთება მანამდეა საჭირო, მაშინ
გამოიყენება სწორედ ეს $conn = null ოპერატორი.
?>
```

მონაცემთა ბაზის შექმნა

სერვერთან შეერთების შემდეგ შესაძლებელია შევქმნათ მონაცემთა ბაზა. ამ მიზნის მისაღწევად MySQL–ში გათვალისწინებულია CREATE DATABASE ოპერატორი.

ვაჩვენოთ MyDB სახელის მქონე ბაზის შექმნის მაგალითები ზემოთ განხილული სამივე მიდგომისათვის:

MySQLi - ობიექტზე ორიენტირებული

ლისტინგი SQL_4 (64)

```
<?php
 $servername = "localhost";
 $username = "username";
 $password = "password";
```

```
   // ვქმნით სერვერთან  კავშირს!
 $conn = new mysqli($servername, $username, $password);
   //  ვამოწმებთ შედეგს!
 if ($conn->connect_error) {
    die("კავშირი ვერ დამყარდა: " . $conn->connect_error);
                                    }
   //  მონაცემთა  ბაზის  შექმნა
 $sql = "CREATE DATABASE myDB";
 if ($conn->query($sql) === TRUE) {
    echo "მონაცემთა ბაზა შეიქმნა";
                                          }
 else {
    echo "ბაზის შექმნის მცდელობისას მოხდა შეცდომა: " . $conn->error;
       }
 $conn->close();
?>
```

MySQLi – პროცედურული

ლისტინგი SQL_5 (65)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
  // ვქმნით სერვერთან  კავშირს!
$conn = mysqli_connect($servername, $username, $password);
//  ვამოწმებთ შედეგს!
if (!$conn) {
   die("კავშირი ვერ დამყარდა შემდეგი შეცდომის გამო: " .
mysqli_connect_error());
              }
  //  მონაცემთა  ბაზის  შექმნა
```

```
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
  echo "მონაცემთა ბაზა შეიქმნა";
                                    }
 else {
    echo "ბაზის შექმნის მცდელობისას მოხდა შემდეგი შეცდომა: " .
mysqli_error($conn);
       }
```

PDO (PHP Data Objects)

ამ მომდევნო მიდგომას წინა ორთან შედარებით აქვს მნიშვნელოვანი უპირატესობა - ხდება მონაცემთა ბაზისადმი ნებისმიერი მოთხოვნის უფრო დაწვრილებით გაანალიზება, შეცდომის აღმოჩენის შემთხვევაში სკრიპტი წყვეტს მუშაობას და, პრობლემის გადაწყვეტის მიზნით, მართვა გადაეცემა გამონაკლისების კლასით გათვალისწინებულ შესაბამის catch ბლოკს.

ლისტინგი SQL_6

```
<?php
 $servername = "localhost";
 $username = "username";
 $password = "password";
try {
   $conn = new PDO("mysql:host=$servername;dbname=myDB", $username,
$password);
   // შედეგის შემოწმების რეჟიმის დაყენება
   $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
   // მონაცემთა ბაზის შექმნა
   $sql = "CREATE DATABASE myDBPDO";
   // წარუმატებლობის შემთხვევაში exec()  ფუნქციის გამოძახება
   $conn->exec($sql);
   echo "მონაცემთა ბაზა შეიქმნა"<br>";
```

```
  }
catch(PDOException $e)
  {
  echo $sql . "<br>" . $e->getMessage();
   }
$conn = null;
?>
```

ცხრილის შექმნა

თავდაპირველად აღვნიშნოთ, რომ MySQL მონაცემთა ბაზაში ცხრილი იქმნება create table ოპერატორის მეშვეობით.

მოვიყვანოთ მაგალითი მონაცემთა ბაზის სტუმრებისათვის 5 ველის (სვეტის) შემცველი ცხრილის შექმნისა. სინტაქსი ასეთი სახისაა:

```
CREATE TABLE MyGuests (
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  firstname VARCHAR(30) NOT NULL,
  lastname VARCHAR(30) NOT NULL,
  email VARCHAR(50),
  reg_date TIMESTAMP
)
```

ცხრილში შემავალი ველის დასახელების შემდეგ აუცილებელია მონაცემთა ტიპის განსაზღვრა. ხშირ შემთხვევაში ხდება ასევე რამდენიმე (არასავალდებულო) ატრიბუტის ჩვენებაც:

- NOT NULL – მოცემული ველისათვის (სვეტისათვის) აუცილებელი რაიმე მნიშვნელობის ჩვენება;
- დუმილით გათვალისწინებული მნიშვნელობის ავტომატურად მინიჭება, როდესაც სხვა მნიშვნელობა არ მიეთითება;
- UNSIGNED - უზრუნველყოფს მთელი რიცხვებისათვის მნიშვნელობების დიაპაზონად (-2147483648 – 2147483647)-ის ნაცვლად (0 – 4294967295) არის განსაზღვრას;
- AUTO_INCREMENT – ამ ატრიბუტით მონიშნული ველის მნიშვნელობა ცხრილში ახალი ჩანაწერის დამატებისას ავტომატურად იზრდება ერთი ერთეულით;
- PRIMARY KEY - ახდენს ცხრილში სტრიქონების ცალსახა იდენტიფიკაციას. რეალურად წარმოადგენს ჩანაწერის

საიდენტიფიკაციო ნომერს და ხშირად ზემოთ განმარტებულ AUTO_INCREMENT ატრიბუტთან ერთად გამოიყენება.

ვაჩვენოთ PHP პროგრამიდან ცხრილის შექმნის მაგალითები ზემოთ განხილული სამივე შემთხვევისათვის:

MySQLi - ობიექტზე ორიენტირებული

ლისტინგი SQL_7 (67)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

  // ვქმნით სერვერთან კავშირს!
$conn = new mysqli($servername, $username, $password, $dbname);

  //  ვამოწმებთ შედეგს!
if ($conn->connect_error) {
   die("კავშირი ვერ დამყარდა:  " . $conn->connect_error);
}

  // sql ცხრილის შექმნა
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
   echo "MyGuests ცხრილი წარმატებით შეიქმნა";
                                    }
```

```
 else {
  echo "ცხრილის შექმნის მცდელობისას მოხდა შეცდომა " . $conn->error;
        }
$conn->close();
?>
```

MySQLi – პროცედურული

ლისტინგი SQL_8 (68)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// ვქმნით სერვერთან  კავშირს!
$conn = mysqli_connect($servername, $username, $password, $dbname);
//  ვამოწმებთ შედეგს!
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
// sql ცხრილის შექმნა
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";
if (mysqli_query($conn, $sql)) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . mysqli_error($conn);
```

```
}
mysqli_close($conn);
?>
```

PDO (PHP Data Objects)

ლისტინგი SQL_9 (69)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
   // შედეგის შემოწმების რეჟიმის დაყენება
   $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
   // sql to create table
   $sql = "CREATE TABLE MyGuests (
   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   firstname VARCHAR(30) NOT NULL,
   lastname VARCHAR(30) NOT NULL,
   email VARCHAR(50),
   reg_date TIMESTAMP
   )";
   // წარუმატებლობის შემთხვევაში  exec() ფუნქციისადმი მიმართვა
   $conn->exec($sql);
   echo " MyGuests ცხრილი წარმატებით შეიქმნა";
   }
catch(PDOException $e)
   { echo $sql . "<br>" . $e->getMessage(); }
```

```
$conn = null;
?>
```

საერთოდ, უნდა აღინიშნოს, რომ ისეთი ოპერაციები, როგორიცაა: მონაცემთა ბაზის შექმნა, მასში ცხრილების ფორმირება და ასევე ამ ცხრილების შევსება უკვე არსებული მონაცემებით, სამუშაოს გაადვილების თვალსაზრისით, უმჯობესია შევასრულოთ არა PHP პროგრამიდან, არამედ ადგილზევე - უშუალოდ MySQL ბაზაში, რომელიც, ამ მიზნების მისაღწევად, ფლობს მეტად მოხერხებულ გრაფიკულ ინტერფეისს. აღნიშნული შესაძლებლობების გამოსაყენებლად ბროუზერის სამისამართო უბანში უნდა ჩავწეროთ შემდეგი მისამართი:

localhost/phpadmin

რაც შეეხება ბაზაში შემდგომ შესატან ცვლილებებს, მათ მოვახდენთ PHP პროგრამიდან. ქვემოთ განვიხილავთ ამ მხრივ PHP ენის ყველაზე საჭირო შესაძლებლობებს.

ცხრილში მონაცემების ჩამატება

მონაცემთა ბაზაში ცხრილის (ცხრილების) შექმნის შემდეგ შესაძლებელია მასში ჩავამატოთ მონაცემები. დაცული უნდა იქნეს ასეთი სინტაქსური წესები:

– ჩანაწერების დამატების მოთხოვნა ჩამოვაყალიბდეს PHP პროგრამაში, რომელშიც მიეთითება ველები და მათი მნიშვნელობები:

– AUTO_INCREMENT და TIMESTAMP ოფციანი ველებისათვის საჭირო არ არის მოთხოვნაში ავტომატურად კორექტირებადი მონაცემების ჩვენება.

SQL მოთხოვნის ზოგადი სახეა:

INSERT INTO table_name (column1, column2, column3,...) VALUES (value1, value2, value3,...)

დავწეროთ წინა თავში შექმნილი MyGuests ცარიელი ცხრილისათვის მასში ჩანაწერების დამატების შემდეგი მოთხოვნა:

MySQLi - ობიექტზე ორიენტირებული

ლისტინგი SQL_10 (70)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
```

```
$dbname = "myDB";

  // ვქმნით სერვერთან  კავშირს!
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    echo "ახალი ჩანაწერი წარმატებით შეიქმნა";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

MySQLi – პროცედურული

ლისტინგი SQL_11 (71)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
  // ვქმნით სერვერთან  კავშირს!
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
```

```
    die("Connection failed: " . mysqli_connect_error());
}
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
    echo ""ახალი ჩანაწერი წარმატებით შეიქმნა";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

PDO (PHP Data Objects)

ლისტინგი SQL_12 (72)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    // შედეგის შემოწმების რეჟიმის დაყენება
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $sql = "INSERT INTO MyGuests (firstname, lastname, email)
    VALUES ('John', 'Doe', 'john@example.com')";
    // წარუმატებლობის შემთხვევაში  exec() ფუნქციისადმი მიმართვა
    $conn->exec($sql);
```

```
   echo "ახალი ჩანაწერი წარმატებით შეიქმნა";
   }
catch(PDOException $e)
   {
   echo $sql . "<br>" . $e->getMessage();
   }
$conn = null;
?>
```

განვიხილოთ ცხრილში მონაცემების ჩამატების ასეთი მაგალითიც:

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

Sample მონაცემთა ბაზაში შევქმნათ ცხრილი domains სახელითა და 4 ველით:

```
create  table domains (
  id INT NOT NULL AUTO_INCREMENT,
  PRIMARY EY ( id ),
  domain VARCHAR (20),
  sex CHAR (1),
  mail VARCHAR (20)
 );
```

ცხრილში კონკრეტული მონაცემების ჩასამატებლად საჭიროა, ბაზას მივმართოთ SQL-მოთხოვნით. ამისათვის PHP-ში გათვალისწინებულია mysqli-query() ფუნქცია. მას გადაეცემა **მოთხოვნის სტრიქონი** და *(არააუცილებელი)* **მიერთების იდენტიფიკატორი:**

ლისტინგი SQL_13 (73)

```
<html>
<head>
  <title> ცხრილში ჩანაწერის დამატება </title>
</head>
<body>
<?php
  $user = "harry";
```

```
  $pass = "elbomonkey";
  $db = "sample";
  $link = mysqli_connect( "localhost", $user, $pass );
if ( ! $link )
    die( "Coudn't connect to MySQL");
mysqli_select_db( $db, $link)
    or  die ("Couldn't open $db: ".mysqli_error() );
$query=”INSERT INTO domains ( domain, sex, mail )
        values( ‘123xyz.com’, ‘F’, sharp@adomain.com’ )”;
mysqli_query( $query, $link)
    or  die ("Couldn't add data to \”domains\”  table: “.mysqli_error() );
mysqli_close( $link);
?>
</body>
</html>
```

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

ამ პროგრამას რამდენჯერაც შევასრულებთ, ცხრილს იმდენჯერვე დაემატება ერთი და იმავე შემცველობის მქონე ახალი ჩანაწერი *(განსხვავება იქნება მხოლოდ id ველის მნიშვნელობაში).*

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

ლისტინგი SQL_14 (74)

```
<html>
<head>
  <title> ბაზაში ინფორმაციის დამატება  </title>
</head>
<body>
<?php
if ( isset( $domain ) && isset( $sex ) &&  isset( $mail ) )
 {
```

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

function add_to_database( $domain, $sex, $mail, &$dberror)
{
  $user = "harry";
  $pass = "elbomonkey";
  $db = "sample";
  $link = mysqli_pconnect( "localhost", $user, $pass );
if ( ! $link )
  {
    $dberror = "Couldn't connect to MySQL server";
    return false;
  }
if (! mysqli_seleqt_db( $db, $link ) )
  {
    $dberror =mysqli_error();
    return false;
   }
$query = "INSERT INTO domains ( domain, sex, mail )
values( '$domain','$sex','$mail' )";
if ( ! mysqli_query( $query, $link ) )
  {
```

```
    $dberror = mysqli_error();
    return false;
  }
return true;
}
function write_form()
{
    global $PHP_SELF;
    print "<form action=\"$PHP_SELF\" method=\"POST\">\n";
    print "<input type=\"text\" name=\"domain\"> ";
    print "The domain you would like<p>\n";
    print "<input TYPE=\"text\" name\"mail\"> ";
    print "Your mail address<p>\n";
    print "<select name=\"sex\">\n";
    print "\t<option value=\"F\" Female\n";
    print "\t<option value=\"M\"> Male\n";
    print "\select>\n";
    print "<input type=\"submit\" value=\"submit!\">\n</form>\n";
}
?>
 </body>
</html>
```

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

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

add_to_database() ფუნქციას 4 არგუმენტი აქვს - მომხმარებლის მიერ გადმოცემულ 3 მნიშვნელობას ემატება $dberror ცვლადთან დაკავშირებული $dberror სტრიქონიც. რომელიმე ოპერაციის უშედეგოდ

დასრულებისას ხდება შეცდომის შესახებ ინფორმაციის ჩაწერა ჩვენ მიერ გამოცხადებულ $dberror გარე ცვლადშიც *(და არა მარტო მის ასლში).*

როგორც ვხედავთ, add_to_database() ფუნქციაში ხდება მცდელობები:

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

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

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

ავტომატურად კორექტირებადი ველის მნიშვნელობის გაგება

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

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

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

mysqli_query( $query, $link );

$id=mysqli_insert_id ();

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

ბაზიდან მონაცემების წაკითხვა

ამ მიზნით, ვიყენებთ Select ტიპის SQL მოთხოვნას. მისი წარმატებით შესრულებისას mysqli_query() ფუნქცია აბრუნებს **მოთხოვნის შედეგის იდენტიფიკატორს,** რომელიც შეიძლება გადავცეთ მიღებული შედეგის დამამუშავებულ ფუნქციებს.

სვეტში (ველში) შეტანილი მონაცემების ამოსარჩევად ვწერთ:

SELECT სვეტის_სახელი(s) FROM ცხრილის_სახელი

ხოლო ცხრილში არსებული ყველა სვეტისადმი მიმართვა ხდება ამგვარად:

SELECT * FROM ცხრილის_სახელი

შენიშვნა: SQL მოთხოვნების შედგენის წესების გადასამეორებლად მიმართეთ საიტს:

https://www.w3schools.com/sql/default.asp

SQL-მოთხოვნით მოძებნილი ჩანაწერების რიცხვის განსაზღვრა

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

ლისტინგი SQL_15 (75)

```
<html>
<head>
 <title>სტრიქონების რიცხვის განსაზღვრა</title>
</head>
<body>
<?php
 $user = "harry";
 $pass = "elbomonkey";
 $db = "sample";
 $link = mysqli_connect( "localhost", $user, $pass );
if ( ! $link )
 die ( "Could't connect to MySQL");
  mysqli_select_db($db, $link)
      or die ("Couldn't open $db: ".mysqli_error() );
  $result=sql_query("SELECT *FROM  domains");
  $num_rows=mysqli_num_rows( $result);
 print "There are currently $num_rows rows in the table<P>";
```

```
 mysqli_close ($link);
?>
</body>
</html>
```

მოთხოვნის შედეგების ჩათვალიერება

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

ლისტინგი SQL_16 (76)

```
<html>
<head>
 <title>ცხრილის ყველა ჩანაწერის გამოყვანა</title>
</head>
<body>
<?php
 $user = "harry";
 $pass = "elbomonkey";
 $db = "sample";
 $link = mysqli_connect( "localhost", $user, $pass );
if ( ! $link )
  die ( "Could't connect to MySQL");
   mysqli_select_db($db, $link)
       or die ("Couldn't open $db: ".mysqli_error() );
   $result=sql_query("SELECT *FROM  domains");
   $num_rows=mysqli_num_rows( $result);
  print "There are currently $num_rows rows in the table<P>";
```

```
print "<table border=1>\n";
while ( $a_row = mysqli_fetch ($result))
  {
  print "<tr>\n";
  foreach ($a_row as $field)
     print "\t<td>$field</td>\n";
     print"</tr>\n";
  }
print "</table>\n";
mysqli_close ($link);
?>
</body>
</html>
```

while ციკლში mysqli_fetch() ფუნქციის მიერ დაბრუნებულ მნიშვნელობას ვანიჭებთ $a_row ცვლადს.

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

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

- პირველი გულისხმობს while-ის სათაურში mysqli_fetch() ფუნქციის ნაცვლად mysqli_fetch_array-ის გამოყენებას *(ის ასოცირებულ მასივს გვიბრუნებს);*
- მეორე – სტრიქონის აღქმას ობიექტად და მისი თვისებების mysqli_fetch_object() ფუნქციით წაკითხვას:

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

```
print "</TABLE>\n";

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

მონაცემების შეცვლა

mysqli_query() ფუნქციას ვიყენებთ ცხრილში მონაცემების შესაცვლელადაც.

მოთხოვნაში ფიგურირებს UPDATE ინსტრუქცია. მას შეიძლება, მაგალითად, ასეთი სახე ჰქონდეს:

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

UPDATE ინსტრუქციის წარმატებით შესრულება ჯერ კიდევ არ ნიშნავს მონაცემების ადგილზე ფაქტობრივად შეცვლას, რაც შეიძლება mysqli_affected_rows() ფუნქციით შემოწმდეს *(ეს ფუნქციაც მხოლოდ მიერთების იდენტიფიკატორს საჭიროებს არგუმენტად და აქაც, თუ ამ იდენტიფიკატორს არ მივუთითებთ, გამოყენებული იქნება უკანასკნელი მიერთების იდენტიფიკატორი).*

შევნიშნოთ, რომ mysqli-query() ფუნქციის გამოყენება შეიძლება მონაცემებში ნებისმიერი ცვლილებების შეტანისას.

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

ლისტინგი SQL_17 (77)

```
<html>
<head>
  <title>mysqli ფუნქციის გამოყენება ცხრილში მონაცემების
      შესაცვლელად
```

```
</title>
</head>
<body>
<?php
 $user = "harry";
 $pass = "elbomonkey";
 $db = "sample";
 $link = mysqli_connect( "localhost", $user, $pass );
if ( ! $link )
  die ( "Could't connect to MySQL");
   mysqli_select_db($db, $link)
        or die ("Couldn't open $db: ".mysqli_error() );
   if ( isset ($domain) && isset ($id))
       {
        $query="UPDATE domains SET domain = '$domain' where id=$id";
       $result = mysqli_query ($query);
       if  (! $result)
          die ("Couldn't update: ".mysqli_error());
      print "<h1>Table updated ". mysqli_affected_rows().
          "row(s) hanged</h1><p>";
   ?>
  <form action = "<? print $PHP_SELF ?>" method = 'POST'>
  <select name = "id">
   <?php
    $result=mysqli_query("SELECT domain, id FROM domains");
    while ($a_row = mysqli_fetch_object ($result))
       {
        print "<OPTION VALUE = \"$a_row->id\"";
        if ( isset ($id) && $id == $a_row->id)
           print "SELECTED";
        print" > $a_row->domain\n";
```

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

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

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

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

გარემოს შესახებ ინფორმაციის შემცველი ცვლადები

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

ამ ცვლადების მეშვეობით სერვერზე განთავსებული PHP პროგრამიდან მომხმარებელს შეეძლება მიიღოს ინფორმაცია, მაგალითად, საკუთარი საიტის მნახველების შესახებ. მაგრამ, რადგანაც სისტემაში ან სერვერზე არსებულ რესურსებთან შეღწევა ყოველთვის ვერ ხერხდება, ჯობია, წინასწარ შევამოწმოთ მცდელობათა შედეგები. მოვიყვანოთ შესაბამისი მაგალითი:

ლისტინგი 78

```
<html>
<head>
 <title>ბროუზერის ფანჯარაში ცვლადების მნიშვნელობების
       გამოტანა
</title>
</head>
<body>

<?php
$envs = array ( "HTTP_REFERER", "HTTP_USER_AGENT",
"REMOTE_ADDR", "REMOTE_HOST",  "QUERY_STRING", "PATH_INFO");

foreach ($envs as $env)
   {
    if  ( isset ($$env))
       print "$env: ${$env}<br>";
   }
?>
</body>
</html>
```

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

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

<A HREF='"ფაილის სახელი".php/my_path_info?query_key=query_value'> go

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

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

მოთხოვნის სტრიქონი შედგება &-სიმბოლოებით გაყოფილი შემდეგი წყვილებისგან:

სახელი/მნიშვნელობა

ორაზროვნობის თავიდან ასაცილებლად ეს წყვილები ექვემდებარება ე.წ. **URL-კოდირებას** – "საეჭვო" სიმბოლოები იცვლება მათი 16-ობითი ეკვივალენტებით.

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

PHP ენისა და მასთან დაკავშირებული მთელი სამუშაო გარემოს შესახებ ამომწურავი ინფორმაციის მისაღებად სისტემაში გათვალისწინებული არის ფუნქცია phpinfo(). მისი მეშვეობით შესაძლებელია დისფლეიზე გამოვიყვანოთ დიდი მოცულობის ინფორმაცია PHP პროგრამების კომპილაციასთან დაკავშირებული პარამეტრების, გაფართოებების, ენის ვერსიის, ასევე ოპერაციული სისტემის, ვებსერვერის, http პროტოკოლისა და სხვა სისტემური მონაცემების შესახებ:

```
<?php
echo "გამოგვყავს ინფორმაცია PHP-ს და მისი მომცველი გარემოს მდგომარეობის შესახებ";
 phpinfo();
 //  bool phpinfo ([ int $what = INFO_ALL ] );
?>
```

იმის გამო, რომ php და მისი გარემომცველი სივრცე სხვადასხვაგვარად შეიძლება იყოს კონფიგურირებული, მოწოდებული ინფორმაცია დაგვეხმარება ჩვენთვის საჭირო შესაბამისი გადაწყვეტილებების მიღებაში.

ამავე დროს შესაძლებელია, ფუნქციისათვის მიერ გამოტანილ ინფორმაციას ჩვენთვის სასურველი სახე მივცეთ $what პარამეტრისათვის დუმილით გათვალისწინებული INFO_ALL მნიშვნელობის სხვათი შეცვლით.

ესენია:

INFO_GENERAL,

INFO_CREDITS,

INFO_CONFIGURATION,

INFO_MODULES,

INFO_ENVIRONMENT,

INFO_VARIABLES,

INFO_LICENSE..

მოვიყვანოთ ერთ-ერთი რეჟიმის დაყენების მაგალითი:

```
<?php
  // გამოიტანე მთელი ინფორმავია (INFO_ALL)
  phpinfo();
  // გამოდის ინფორმაცია მხოლოდ მოდულის შესახებ
  phpinfo(INFO_MODULES);
?>
```

ამ და სხვა პარამეტრების დანიშნულების შესახებ უფრო დაწვრილებითი ინფორმაციის მისაღებად იხ, მაგალითად, საიტი:

http://php.net/manual/en/function.phpinfo.php

კლიენტის სერვერთან HTTP ოქმით შეერთების ზოგიერთი საკითხი

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

მოთხოვნა

კლიენტის მიერ სერვერიდან მონაცემების მოთხოვნა სამ კომპონენტს შეიძლება მოიცავდეს:

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

მოთხოვნის სტრიქონში მითითებულია:

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

მოთხოვნის სტრიქონის ტიპური მაგალითია:

```
GET / mydoc.html  HTTP.1.0
```

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

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

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

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

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

ცხრილში მოყვანილია ინფორმაცია ზოგიერთი გასაღების შესახებ:

სახელი	**აღწერა**
Accept	ინფორმაციის მატარებელთა ტიპები, რომლებთანაც შეუძლია კლიენტს მუშაობა.
Accept-Encoding	მონაცემთა შეკუმშვის ტიპები, რომლებთანაც შეუძლია კლიენტს მუშაობა.
Accept-Charset	სიმბოლოთა ცხრილები, რომელთაც კლიენტი უპირატესობას ანიჭებს.
Accept-Language	კლიენტისათვის სასურველი ენა (მაგალი-

	თად, “en” - ინგლისური)
Host	კომპიუტერის ქსელური სახელი, რომელსაც მომხმარებელის მოთხოვნა ეგზავნება.
Referer	დოკუმენტი, რომლიდანაც მოხდა მოთხოვნის გაცემა.
User-Agent	ინფორმაცია პროგრამა-კლიენტის ტიპისა და ვერსიის შესახებ.

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

რაც შეეხება მოთხოვნის სხეულს, ის ძალიან ჰგავს მოთხოვნის სტრიქონს - მასშიც ფიგურირებს URL კოდირებული წყვილები:

სახელი/მნიშვნელობა

ქვემოთ მოყვანილია ბროუზერის მიერ სერვერისადმი გადაგზავნილი მოთხოვნის ტიპური ვარიანტი:

ლისტინგი 79

GET / HTTP / 1.0

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

Connection: Keep-Alive

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

Host : www.corrosive.co.uk

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

/

Accept-Encoding: gzip

Accept-Language: en

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

პასუხი

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

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

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

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

მოვიყვანოთ ზოგიერთი კოდი თავისი აღწერილობით:

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

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

სერვერიდან გადმოგზავნილი პასუხების ტიპური სათაურებია:

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

ქვეთავის დასასრულს განვიხილოთ HTML, CSS, Javascript, PHP ენების და HTTP ოქმის თანამშრომლობის მაგალითი:

HTML ფორმაში ინფორმაციის შეტანისას ხდება სერვერზე არსებულ PHP პროგრამისადმი მიმართვა და მისგან სავარაუდო შესატანი მონაცემების გადმოგზვნა, რომელიც გამოყენების შემთხვევაში ფორმისადმი შემდეგი მიმართებისას ფიგურირებს უკვე ადგილობრივ, კლიენტის მხარეს მყოფ მენიუში:

ლისტინგი 80

```
<html>
<head>
  <title>gethint</title>
</head>
<body>
<script language=Javascript>
var xmlHttp;
function showHint(str)
{
  if (str.length==0)
  {
    document.getElementById("txtHint").innerHTML="";
    return;
  }
  xmlHttp=GetXmlHttpObject();
 if (xmlHttp==null)
 {
   alert ("HTTP  მოთხოვნების შესრულებას ეს ბროუზერი ვერ ახერხებს!");
```

```
  return;
 }

var url="gethint.php";

url=url+"?q="+str;
url=url+"&sid="+Math.random();
xmlHttp.onreadystatechange=stateChanged;
xmlHttp.open("GET",url,true);
xmlHttp.send(null);
}

function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
   document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}

function GetXmlHttpObject()
{
 var objXMLHttp=null;
 if (window.XMLHttpRequest)
 {
   objXMLHttp=new XMLHttpRequest()
 }
 else if (window.ActiveXObject)
 {
  objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
 return objXMLHttp
```

```
}
</script>
 <form>
   სახელი
   <input type="text" id="txt1"
   onkeyup="showHint(this.value)">
 </form>
 <p>უკვე შეტანილი სახელების სიის გამოსატანად ტექსტურ ველზე დააწკაპუნეთ ორჯერ.</p>
 <p>თუ სიიდან ირჩევთ რომელიმე სახელს, დააწკაპუნეთ მასზე.</p>
 <p>შეგიძლიათ აკრიფოთ ახალი სახელიც. <span id="txtHint"></span></p>
<?php
$a[]="Anna";
$a[]="Brittany";
$a[]="Cinderella";
$a[]="Diana";
$a[]="Eva";
$a[]="Fiona";
$a[]="Gunda";
$a[]="Hege";
$a[]="Inga";
$a[]="Johanna";
$a[]="Kitty";
$a[]="Linda";
$a[]="Nina";
$a[]="Ophelia";
$a[]="Petunia";
$a[]="Amanda";
$a[]="Raquel";
$a[]="Cindy";
$a[]="Doris";
```

```
$a[]="Eve";
$a[]="Evita";
$a[]="Sunniva";
$a[]="Tove";
$a[]="Unni";
$a[]="Violet";
$a[]="Liza";
$a[]="Elizabeth";
$a[]="Ellen";
$a[]="Wenche";
$a[]="Vicky";
// URL-იდან გამოვაცალკევებთ q პარამეტრს:
$q=$_GET["q"];
// თუ მასივის სიგრძე  q>0,  განვიხილავთ რეკომენდაციებს:
if (strlen($q) > 0)
{
  $hint="";
  for($i=0; $i<count($a); $i++)
  {
  if (strtolower($q)==strtolower(substr($a[$i],0,strlen($q))))
    {
    if ($hint=="")
      {
      $hint=$a[$i];
      }
    else
      {
      $hint=$hint." , ".$a[$i];
      }
    }
  }
}
```

```
if ($hint == "")
{
$response="ძიება უშედეგოდ დამთავრდა!";
}
else
{
$response=$hint;   //  შესაბამისი მნიშვნელობის მინიჭება
}
//  პასუხის გამოტანა
echo $response;
?>
</body>
</html>
```

AngularJS

1. შესავალი

*რას წარმოადგენს **AngularJS**?*

Angular არის JavaScript ენაზე დაწერილი, თვისებრივად გაუმჯობესებული ვებგამოყენების (დანართების) შესაქმნელად განკუთვნილი MVW (Model-View-Whatever) ფრეიმვორკი.

ის დამუშავებულია და ვითარდება Google Inc კორპორაციის მიერ და უშუალო დანიშნულების გარდა, წარმოდგენას გვიქმნის, თუ როგორი იქნება ვებსივრცის მომავალი.

ტერმინ MVW (Model-View-Whatever)-ის არსი დაახლოებით ასე შეიძლება განიმარტოს:

მისი მთავარი ღირსებაა მოქნილობა - ნებისმიერი გარემოს ამსახველი ინფორმაციული ბლოკის სტრუქტურული ელემენტების ამა თუ იმ შაბლონის სახით წარმოდგენა და მათი არჩევის ხელმისაწვდომობა, რაც მნიშვნელოვნად ამარტივებს ვებგამოყენების შექმნის პროცესს. მაგალითად, ამ მიზნით,

შეგვიძლია ავირჩიოთ MVC (Model-View-Controller, იხ. დანართი_1) ან MVVM (Model-View-ViewModel) მოდელი.

AngularJS ფრეიმვორკს ახალ დონეზე აჰყავს HTML ენისა და მასთან ურთიერთობის შესაძლებლობები, ამ ენის ელემენტებს სძენს დინამიკურობას.

AngularJS თავის თავში მოიცავს როგორც კლიენტის მხარეს მოქმედი JavaScript ენაზე დაწერილი სცენარების, ასევე სერვერული ენებისათვის დამახასიათებელ კონცეფტუალურ შესაძლებლობებსაც.

ამ ფრეიმვორკში მონაცემებთან მუშაობისათვის გამოყენებული მიდგომა განსხვავდება ისეთი ფრეიმვორკების აგების კონცეფციისაგან, როგორიცაა, მაგალითად, Backbone.js და Ember.js. AngularJS არ არის მიბმული კონკრეტულ DOM-ზე და ოპერირებს სიტუაციის მიხედვით: DOM-ს განაახლებს JavaScript-ის ობიექტებში მოდელის შესახებ არსებულ ინფორმაციაზე დაყრდნობით. ამრიგად, აქ ეკრანზე გამოსატანი HTML მონაცემები მოდელში ინახება მცირე ზომის შაბლონის სახით. შედეგად, ელემენტზე ჩატარებული საჭირო გარდაქმნები ადვილად ტირაჟირდება ყველა საჭირო შემთხვევაში. მაშასადამე, AngularJS არ არის დამოკიდებული HTML-ზე, მეტიც - ის მას აუმჯობესებს, სძენს ახალ თვისებებს, რადგანაც საქმე გვაქვს მონაცემების დაფიქსირების **ორ შრესთან:**

მოდელთან და ეკრანზე გამოსულ **სახესთან.**

ამ შრეებს შორის კავშირი ორმხრივია - მოდელში განხორციელებული ცვლილებები გავლენას ახდენს ეკრანზე გამოსატანი მონაცემის სახეზე და პირიქით. მოდელისა და სახის სინქრონიზაციისათვის კი გამოიყენება JavaScript-ში არსებული მარტივი ობიექტები.

შესაბამისი გარდაქმნებისათვის კი AngularJS მიმართავს JSON ფორმატს. ამასთანვე აღსანიშნავია, რომ მის მიერ განსაკუთრებით ეფექტიანად გამოიყენება REST მეთოდი.

2. ერთფურცლოვანი დანართები **(Single Page Application, SPA), მათი მდგომარეობის მართვა და Ajax (HTTP)**

SPA **ერთფურცლოვანი დანართის** ფუნქციონირებისათვის საჭირო კოდი (HTML, CSS და JavaScript) სერვერიდან ან ერთბაშად გადმოიტვირთება კლიენტის მხარეს, ანდა ის ჩაიტვირთება დინამიკურად, როგორც წესი, მომხმარებლის ქმედებიდან გამომდინარე. ხშირად SPA დანართის სერვერთან ურთიერთობის აღნიშნული პროცესი სრულდება ფონურ რეჟიმში, ვებფურცლის გადატვირთვის გარეშე.

ადრინდელი მიდგომებისას, როდესაც ინფორმაცია პროგრამის მდგომარეობის შესახებ სერვერზე ინახებოდა, ზოგჯერ ქსელში იგვიანებდა მომხმარებლის ქმედებებზე დროულად რეაგირება, რის გამოც კლიენტის მხარეს ფურცლის სახე და სერვერზე არსებული მისი მოდელი ერთმანეთისაგან განსხვავდებოდა. ამასთანავე, ზოგჯერ ვერ ხერხდებოდა ამ ცვლილებების დამახსოვრებაც, ისინი იკარგებოდნენ და აუცილებელი ხდებოდა ვებფურცლის განახლება, სერვერიდან მისი ხელახლა მთლიანად გადმოქაჩვით.

AngularJS ფრეიმვორკის საშუალებებზე დაყრდნობით კი ინფორმაცია დანართის მდგომარეობის შესახებ ინახება ბროუზერში, ხოლო ფურცელზე შეტანილი ცვლილებების შესახებ შეტყობინებები სერვერს გადაეცემა Ajax (HTTP)-ის მეშვეობით, რომელიც ამ მიზნით იყენებს GET, POST, PUT და DELETE მეთოდებს.

API დანართების შექმნისას AngularJS-ის საშუალებების გამოყენებაზე ორიენტირებული მიდგომის უდავო ღირსებაა ის, რომ ამ დანართების სტრუქტურის ფორმირებისა და გამართვა-ტესტირების პროცესები ერთმანეთის მსგავსი და გამარტივებულია.

3. მოდულები

Angular გამოყენებებს (დანართებს) ქმნის მოდულების მეშვეობით. მოდული შესაძლებელია იყოს ავტონომიური ან სხვა მოდულებზე დამოკიდებული. ის დანართის სხვადასხვა ნაკვეთისათვის ასრულებს არაერთხელ გამოყენებადი კონტეინერის როლს.

მოდულის შესაქმნელად გამოიყენება გლობალური ობიექტი Object, ფრეიმვორკის სახელთა სივრცე და module მეთოდი.

3.1 სეტერები (setters)

გამოყენებისათვის გათვალისწინებულია (დანიშნულია) ერთი, app სახელწოდების მქონე მოდული:

```
angular.module('app',[]);
```

მეორე არგუმენტი მასივია და ჩვეულებრივ მიუთითებს იმ მოდულებზე, რომლებზეც დამოკიდებული არის მოცემული მოდული და რომლებიც მიუერთდებიან მოცემულს (აქ მასივი ცარიელია).

3.2 გეტერები (Getters)

Controllers, Directives, Services და სხვა შესაძლებლობებისადმი მიმართვისათვის უნდა დავეყრდნოთ არსებულ მოდულს. ამ შემთხვევაში მეორე არგუმენტი აღარ გამოიყენება:

```
angular.module('app');
```

3.3 მოდულებთან მუშაობა

მოდული ამგვარად გამოიძახება და შეინახება ცვლადის დახმარებით:

```
var app = angular.module('app', []);
```

ამის შემდეგ შესაძლებელი არის app ცვლადის გამოყენება დანართის შესაქმნელად.

3.4 HTML ბუტსტრაპი

დანართი, ჩვეულებრივ, DOM-ში ინახება <html> ელემენტის მიერ შესაბამისი ატრიბუტის გამოყენების შედეგად:

```
<html ng-app="app">
  <head></head>
  <body></body>
</html>
```

თუ JavaScript-ის სცენარების შემცველი ფაილების ჩატვირთვა ასინქრონულად ხდება, app დანართი ჩაიტვირთება ოპერატორით:

```
angular.bootstrap(document.documentElement, ['app']);
```

4. რა არის $scope?

$scope განიმარტება ხილვადობის უბნად, ასედაც - DOM-ში არსებულ მონაცემთა და JavaScript-ის სცენარს შორის გადებულ ავტომატურ ხიდად. ითვლება, რომ $scope სცენარში ასრულებს მოდელის წარმომადგენლის (ViewModel) როლს. ის გამოიყენება მხოლოდ კონტროლერების შიგნით და ახდენს მათში გამოცხადებული მონაცემების მიბმას ამ მონაცემების ეკრანულ წარმოდგენასთან (View-სთან) .

მოგვყავს კონტროლერში მონაცემების გამოცხადების მაგალითი:

```
$scope.someValue = 'Hello';
```

შემდეგ დგება საკითხი DOM-ში ამ `someValue` მნიშვნელობის ასახვისა. ის წყდება Angular-სთვის DOM-ში მონაცემის ჩასადგმელი ადგილის მითითებით, ანუ HTML-ის ელემენტთან კონკრეტული კონტროლერის მიბმით, მაგალითად, ამგვარად:

```
<div ng-controller="AppCtrl">
  {{ someValue }}
</div>
```

ცხადია, JavaScript-ის სცენარში არსებული ნებისმიერი ტიპის მქონე მონაცემებისათვის რაც მეტი კონტროლერი შეიქმნება და DOM-ის მონაცემებთან კავშირი დამყარდება, საქმე გვექნება მით მეტ ხედვის უბანთან და მათ იერარქიაში გასარკვევად მიმართავენ $rootScope ცვლადს.

4.1 $rootScope

$rootScope ცვლადის არსს განმარტავენ, როგორც საწყისი (ფესვური) დონის $scope ობიექტს. სწორედ მისგან გავდივართ ყველა სხვა დონის ხედვის უბანზე. $rootScope შესაძლებელია გამოყენებული იქნეს ხედვის ერთი უბნიდან მეორეში მონაცემების გადასაცემად.

5. კონტროლერები

კონტროლერი ორმხრივ კავშირს ამყარებს წარმოდგენასა და მოდელს შორის - ცვლილებები ერთ მათგანში აისახება მეორეშიც. ამრიგად, კონტროლერი წარმოდგენასა და მოდელს შორის შუამავალია - ის იღებს მოთხოვნას მომხმარებლისაგან, აანალიზებს მას და შემდგომი დამუშავებისათვის გადასცემს სისტემის შესაბამის რგოლს, რითაც ხდება ერთმანეთისაგან განცალკევებული ბიზნესლოგიკისა და წარმოდგენის (პრეზენტაციის) ლოგიკის შეთანხმებული მუშაობა.

როგორც უკვე ვნახეთ, კონტროლერის შექმნა ამგვარად ხდება:

```
angular
  .module('app', [])
  .controller('MainCtrl', function () {
});
```

ჩანს, რომ კონტროლერს გადაეცემა ორი არგუმენტი - გამოძახებისათვის განკუთვნილი, მისი მაიდენტიფიცირებელი სახელი და შესასრულებელი ფუნქცია, რომელიც რეალურად კონტროლერის სხეულს წარმოადგენს.

შესაძლებელია, ზემოთ მოტანილ ჩანაწერს ასეთი, უფრო გამჭვირვალე სახეც მივცეთ (ქვემოთ მოყვანილ მაგალითში ფუნქცია Angular-ის გარეთ არის გატანილი და საკუთარი სახელი აქვს მიცემული. ნათლად ჩანს, რომ ფუნქცია დამოუკიდებელია Angular-ისა და მისი სინტაქსისაგან):

```
function MainCtrl () {
}
angular
  .module('app', [])
  .controller('MainCtrl', MainCtrl);
```

ქვემოთ მოყვანილ მაგალითებში იგულისხმება, რომ Angular-ის მოდული უკვე შექმნილია.

5.1 მეთოდები და პრეზენტაციების ლოგიკა

პრეზენტაციების ფორმატში ბიზნესლოგიკის ინტერპრეტირებისათვის კონტროლერი მიმართავს **სერვისს** და $scope ობიექტის მეშვეობით ახდენს სერვერიდან მონაცემების (პირდაპირი ან მოდიფიცირებული სახით) ადგილზე (View-ში) გადაცემას. განახლებული View კი უკუპროცესით იმავე სერვისის მეშვეობით ფიქსირდება სერვერზეც.

კონტროლერის ფუნქციონირებაში უკეთ გარკვევის მიზნით, ჯერ მასში ვქმნით რამდენიმე ობიექტს (ბიზნესლოგიკის საკითხს შემდგომ განვიხილავთ):

```
function MainCtrl ($scope) {
  $scope.items = [{
    name: 'მენიუ',
    id: 7297510
  },{
    name: 'კერძი_1',
    id: 0278916
  },{
```

```
    name: 'კერძი_2',
    id: 2389017
  },{
    name: 'კერძი_3',
    id: 1000983
  }];
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

$scope-ის მეშვეობით იქმნება მასივი, რომელიც შემდგომ შესაძლებელია Angular-ის ng-repeat დირექტივის მეშვეობით ციკლში ჩათვალიერდეს და, რაიმე შაბლონზე დაყრდნობით, ამა თუ იმ სტრუქტურის სახით, DOM-ში წარმოგვიდგეს:

```
<div ng-controller="MainCtrl">
  <ul>
      <li ng-repeat="item in items">
          {{ item.name }}
      </li>
  </ul>
</div>
```

5.2 ახალი, controllerAs სინტაქსი

კონტროლერები კლასებს წააგავს, მაგრამ არის განსხვავებაც - მათი გამოყენება ხდება არა this გასაღებური სიტყვის, არამედ $scope ობიექტების მეშვეობით, თუმცა Angular-ის დამპროექტებლებმა ასეთი მიდგომა დაუშვეს სპეციალურად შემოღებული controllerAs გადაწყვეტისათვის, რაც ამგვარად აისახა სინტაქსში (მივაქციოთ ყურადღება - გამოტოვებულია $scope-ის გამოძახება და მის ნაცვლად გამოიყენება this პრეფიქსი):

```
function MainCtrl () {
  this.items = [{
    name: 'მენიუ',
```

```
    id: 7297510
  },{
    name: 'კერძი_1',
    id: 0278916
  },{
    name: 'კერძი-2',
    id: 2389017
  },{
    name: 'კერძი_3',
    id: 1000983
  }];
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

ამის შემდეგ DOM-ის იმ ადგილებში, სადაც საჭიროა კონტროლერის ეგზემპლარის შექმნა (მისი გამოყენება ამ წესით ხდება), ვამატებთ as ტერმინს.

ჩანაწერი MainCtrl as main მიგვითითებს, რომ ყველა მონაცემი ინახება main ცვლადში, რის გამოც წინა მაგალითში გამოყენებულ items ტერმინს ვცვლით main.items-ზე.

```
<div ng-controller="MainCtrl as main">
  <ul>
      <li ng-repeat="item in main.items">
          {{ item.name }}
      </li>
  </ul>
</div>
```

შედეგად, ხდება არა მარტო იმის გარკვევა, თუ რომელ კონკრეტულ კონტროლერს მიეკუთვნება ესა თუ ის თვისება, არამედ ასეთი მიდგომით თავიდან ვიცილებთ ერთნაირი სახელების მქონე თვისებებს შორის კონფლიქტური სიტუაციის წარმოშობის შესაძლებლობასაც:

მართლაც, controllerAs-ის გამოყენებით საჭირო აღარაა თითოეული $scope-ის მეთოდისათვის მშობელი $parent ობიექტის მითითება, ხოლო უფრო მაღალი დონისათვის - $parent.$parent-ის და ა.შ. მათ ნაცვლად გამოიყენება მხოლოდ ცვლადის სახელი.

6. სერვისები და ფაბრიკები

სერვისი მეთოდია, რომელიც ქმნის ახალ ობიექტს, მასში მოდელის მონაცემებისა და ბიზნესლოგიკის შენახვის შესაძლებლობით.

6.1 service მეთოდი.

ვნახოთ, როგორ იქმნება სერვისი (ქვემოთ მოყვანილ მაგალითში სერვისის სახელია 'UserService') და მასთან კავშირდება შესაბამისი ფუნქცია:

```
function UserService () {
  this.sayHello = function (name) {
    return 'მოგესალმებით ' + name;
  };
}
angular
  .module('app')
  .service('UserService', UserService);
```

ამის შემდეგ შესაძლებელი ხდება, სერვისი კონტროლერში ჩაიდგას:

```
function MainCtrl (UserService) {
  this.sayHello = function (name) {
    UserService.sayHello(name);
  };
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

აღვნიშნავთ, რომ სერვისის წინ ვერ ხერხდება კოდის შესრულება, რადგანაც მეთოდები იქმნება ობიექტების სახით, ქვემოთ განხილული ფაბრიკებისაგან განსხვავებით.

6.2 ფაბრიკული მეთოდები

ფაბრიკული მეთოდები გვიბრუნებს ობიექტს ან ფუნქციას, მაშასადამე, ამ შემთხვევაში შესაძლებელი ხდება:

- გამოყენებული იქნეს უკუჩართვის მექანიზმი;
- დაიშვება `host` ობიექტის დაბრუნება, რომელსაც მიებმება მეთოდები;
- აგრეთვე მიმართავენ პრივატული და საჯარო ხილვადობის უბნების შექმნასაც.

ფაბრიკებს ხშირად სერვისების სახელითაც მოიხსენიებენ. ქვემოთ ფაბრიკული მეთოდის გამოყენებით ვქმნით UserService-ს:

```
function UserService () {
  var UserService = {};
  function greeting (name) {
    return 'მოგესალმებით ' + name;
  }
  UserService.sayHello = function (name) {
    return greeting(name);
  };
  return UserService;
}
angular
  .module('app')
  .factory('UserService', UserService);
```

პირველი მიდგომის გამოყენებისას ახდენენ ხილვადობის პრივატული უბნის ემულირებას. საერთოდ კი, შესაძლებელი იყო იმავე მექანიზმის გამოყენებით service constructor მეთოდის შიგნით ასეთივე შედეგის მიღება, მაგრამ, მაშინ ნათლად აღარ გამოჩნდებოდა, რა გვიბრუნდება და რა რჩება სერვისის ხილვადობის უბნის შიგნით. ასევე, შესაძლებელია პრივატული დამხმარე ფუნქციების შექმნაც, რომლებიც ხილვადობის არეში რჩება ფუნქციის შედეგის დაბრუნების შემდეგაც და დასაშვებია მათი გამოძახება გარე მეთოდებიდან.

ამგვარივე წესით არის შესაძლებელი პრივატული დამხმარე ფუნქციების შექმნაც, რომლებიც დარჩება ხედვის არეში ფუნქციის შესრულების შემდეგაც. მათი გამოყენება შესაძლებელია საჯარო მეთოდებიდანაც. შევნიშნავთ, რომ

კონტროლერის შიგნითაც დასაშვებია ამ სერვისების ზუსტად ასეთივე ხერხებით გამოყენება.

```
function MainCtrl (UserService) {
  this.sayHello = function (name) {
    UserService.sayHello(name);
  };
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

სერვისები, როგორც წესი, გამოიყენება არა პრეზენტაციების ლოგიკის, არამედ ბიზნესლოგიკის შრისათვის, რომელთანაც კავშირი მყარდება Ajax-ისა და REST მეთოდის (პროტოკოლის) მეშვეობით.

7. შაბლონების გამოყენება Angular-ის ბირთვისადმი მიმართვით

აქამდე განვიხილავდით Angular-ის პროგრამისტულ მხარეს, ამჯერად კი ვაჩვენებთ, თუ როგორ ხერხდება მისი გამოყენება HTML კოდიდან. აქცენტირება ხდება შაბლონების გამოყენების მძლავრ შესაძლებლობებზე.

7.1 გამოსახულებები

გამოსახულებები Angular-ში ორმაგ ფიგურულ ფრჩხილებში ექცევა - {{}}. ციკლები და პირობითი ოპერატორები მათში არ გამოიყენება, თუმცა დაშვებულია ლოგიკური (მაგალითად, || და &&) და ტერნალური ოპერატორის (value? true: false) ჩართვა.

```
function MainCtrl () {
  this.items = [{
    name: 'მენიუ',
    id: 7297510
  },{
    name: 'კერძი_1',
    id: 0278916
  },{
```

```
    name: 'კერძი_2',
    id: 2389017
  },{
    name: 'კერძი-3',
    id: 1000983
  }];
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

აი, როგორ შეიძლება მივიღოთ ინფორმაცია მასივის სიგრძის შესახებ:

```
<div ng-controller="MainCtrl as main">
  მასივის სიგრძეა {{ main.items.length }}
</div>
```

length თვისებას Angular შეცვლის მისი მნიშვნელობით, რომელიც ეკრანზეც აისახება.

7.2 ძირითადი დირექტივების გამოყენება

ზემოთ განხილული ქმედებების შედეგად მასივში შეტანილი ცვლილებები ავტომატურად აისახება DOM-ში და, ცხადია, მონიტორის ეკრანზეც. მაგალითად, მასივიდან ერთი ელემენტის ამოგდებისას დაიწერება: „მასივის სიგრძეა 3“.

საერთოდ, Angular იძლევა ng-* პრეფიქსის მქონე მრავალი დირექტივის გამოყენების საშუალებას, რითაც მომხმარებელი ქმნის უფრო მეტი შესაძლებლობის მქონე HTML ელემენტებს და ატრიბუტებს.

თავდაპირველად მოკლედ აღვწეროთ ზოგიერთი უმნიშვნელოვანესი დირექტივის დანიშნულება:

ng-app - აცხადებს დანართისათვის ფესვურ ელემენტს;

ng-bind - HTML ელემენტის ტექსტს ავტომატურად ანიჭებს გადაცემულ მნიშვნელობას;

ng-model - წინას ანალოგიურია, იმ განსხვავებით, რომ ელემენტსა და მოდელის მნიშვნელობებს შორის მყარდება ორმხრივი კავშირი-შესაბამისობა;

ng-class - განსაზღვრავს დინამიკური ჩატვირთვის კლასებს;
ng-controller - აფორმირებს JavaScript-კონტროლერს HTML-გამოსახულებების გამოსათვლელად;
ng-repeat - კოლექციის თითოეული ელემენტისათვის ქმნის ეგზემპლარს;
ng-show და ng-hide - ლოგიკურ გამოსახულებაზე დაყრდნობით ეკრანზე გამოჰყავს ელემენტი ან მალავს მას;
ng-switch - მოქმედებს დაპროგრამებაში კარგად ცნობილი switch გადამრთველის ანალოგიურად;
ng-view - ეს გაზისური დირექტივა კონტროლერებით მართვადი შაბლონების მეშვეობით ამუშავებს ისეთ მარშრუტებს, რომლებიც იმართება JSON-ზე დაყრდნობით.

დირექტივებთან უფრო საფუძვლიანი გაცნობა დავიწყოთ ng-click-დან:

```
<div ng-controller="MainCtrl as main">
  <div>
    {{ main.items.length }} ელემენტია თქვენს
განკარგულებაში.
  </div>
  <ul>
      <li ng-repeat="item in main.items" ng-
click="main.removeFromStock(item, $index)">
          {{ item.name }}
      </li>
  </ul>
</div>
```

ng-click დირექტივის მეშვეობით ვუკავშირდებით ფუნქცია main.removeFromStock()-ს, რომელსაც გადავცემთ ელემენტს ციკლში დასამუშავებლად. შესაბამისი თვისება გვეხმარება ელემენტის მასივიდან ამოგდებაში - საჭირო აღარაა „ხელით“ გამოვთვალოთ მიმდინარე ელემენტის ინდექსი.

ამის შემდეგ უკვე შესაძლებელია, ფუნქცია კონტროლერს დავუმატოთ. ამ ფუნქციას გადაეცემა $index და მასივის ელემენტი, მეთოდი კი ასე მოქმედებს:

```
function MainCtrl () {
  this.removeFromStock = function (item, index) {
    this.items.splice(index, 1);
  };
  this.items = [...];
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

მეთოდის შექმნისას გასათვალისწინებელია, რომ this-ის მნიშვნელობა კონტექსტის მიხედვით სხვადასხვა მნიშვნელობას იღებს. თუ გამოვიყენებთ ასეთ მიდგომას და კონტროლერზე შევქმნით დაყრდნობას:

var vm = **this;** (აქ vm აღნიშნავს ViewModel-ს), მაშინ ეს დაყრდნობა აღარ დაიკარგება.

გადამუშავებულ კონტროლერს ასეთი სახე ექნება:

```
function MainCtrl () {
  var vm = this;
  vm.removeFromStock = function (item, index) {
    vm.items.splice(index, 1);
  };
  vm.items = [...];
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

ამ წესით იქმნება ინტერფეისთან მომუშავე პრეზენტაციების ლოგიკა. გამოტოვებულია მხოლოდ ერთი ბიჯი - მოდელის განახლება.

vm.items-დან ელემენტის ამოგდების წინ უნდა გაიგზავნოს DELETE მოთხოვნა ბეკ-ენდზე. დადებითი პასუხის მიღების შემდეგ მოხდება ელემენტის ამოგდება მასივიდან. ამრიგად, DOM განახლდება მხოლოდ მოთხოვნის შესრულებისას და მომხმარებელს შეცდომა აღარ მოუვა.

8. დირექტივები

დირექტივები აადვილებს მომავალში პროგრამების დაწერას. ისინი ორი სახისაა:

1. ერთნი უკვე ჩაშენებულია Angular-ში და მათზე გავდივართ კავშირების დახმარებით;
2. მეორენი იქმნება მომხმარებელთა მიერ მათთვის სასურველი კონკრეტული შედეგების მისაღებად.

ჯერ გავეცნოთ არსებულთ და შემდეგ გადავიდეთ საკუთარი დირექტივების შექმნის საკითხზე.

8.1 ng-repeat

ამ დირექტივას უკვე გავეცანით:

```
<ul>
  <li ng-repeat="item in main.items">
    {{ item }}
  </li>
</ul>
```

ng-repeat ახდენს ელემენტის კოპირებას და შემდეგ მის აღწარმოებას, მასივიდან აღებული ობიექტების მონაცემებზე დაყრდნობით. თუ მასივიდან ელემენტს ამოვაგდებთ, DOM ავტომატურად განახლდება.

8.2 ng-model

გამოიყენება როგორც ახლად შესაქმნელი მოდელის ინიციალიზაციისათვის, ისე უკვე არსებულთან მის მისაბმელად.

შეტყობინება: {{ main.message }}

თუ $scope-ის main.message თვისებაში ინახება მნიშვნელობა, ის გადაიცემა input-ში. საწინაღმდეგო შემთხვევაში, უბრალოდ, მოხდება მისი ინიციალიზება. შესაძლებელია ამ მნიშვნელობების სხვა, მაგალითად, ng-click დირექტივაში გადაცემა.

8.3 ng-click

onClick=”...” ხდომილობისაგან განსხვავებით, ng-click დირექტივა გლობალური სახის არ არის - ის მოქმედებს მხოლოდ საკუთარი ხედვის

არეში. ელემენტთან მისი მიბმა და დირექტივისათვის რაიმე პირობის შესრულებისას შესაბამისი ხდომილობის დამმუშავებლის გამოძახება კი ავტომატურად ხდება.

```
<input type="text" ng-model="main.message">
<a href=" ng-click="main.showMessage(main.message);">
      აჩვენეთ შეტყობინება
</a>
```

ზემო მაგალითში main.message გადაიცემა main.showMessage მეთოდში, რომელშიც Angular მას დაამუშავებს, როგორც ჩვეულებრივ JavaScript ობიექტს.

აქ იკვეთება Angular-ის მნიშვნელოვანი ღირსება - DOM-ში არსებული მონაცემების ჯგუფები წარმოგვიდგება ადვილად მოსაძიებელი ობიექტების სახით, რომლებზეც შესაძლებელია ჩატარდეს საჭირო მოქმედებები, კერძოდ, გარდაქმნები Json ფორმატში და გადაგზავნები.

8.4 ng-href/ng-src

Angular ფრეიმვორკი ბროუზერებთან ურთიერთობისათვის href და src პარამეტრების ნაცვლად იყენებს ng-href და ng-src დირექტივებს:

```
<a ng-href="{{ main.someValue }}">Go</a>
<img ng-src="{{ main.anotherValue}}" alt="">
```

8.5 ng-class

elem.addClass(className) და elem.removeClass(className) ტრადიციული მიმართვების (გამოძახებების) ნაცვლად Angular ფრეიმვორკი, კლასების დამატებისა და ამოგდების მიზნით, იყენებს სწორედ ამ დირექტივას:

```
<div class="notification" ng-class="{
  warning: main.response == 'error',
  ok: main.response == 'success'
}">
  {{ main.responseMsg }}
</div>
```

ის აანალიზებს main.response მნიშვნელობას და მის საფუძველზე შესაბამისად მოქმედებს.

8.6 ng-show/ng-hide

ამ დირექტივებით ხდება ელემენტის დამალვა-გამოჩენა, მისი რომელიმე თვისების მნიშვნელობიდან გამომდინარე.

გადართვა ხდება ng-click ხდომილობისადმი მიმართვით:

```
<a href=" ng-click="showMenu = !showMenu"> მენიუს
გადართვა!</a>
<ul ng-show="showMenu">
  <li>1</li>
  <li>2</li>
  <li>3</li>
</ul>
```

მათ შორის განსხვავება ვლინდება მითითებაში, თავდაპირველად როგორ მდგომარეობაში უნდა იმყოფებოდეს ელემენტი.

8.7 ng-if

ng-if დირექტივა არათუ მალავს, არამედ ანადგურებს კიდეც ელემენტს. ოღონდ, ელემენტის ამა თუ იმ თვისების მიერ შესაბამისი მნიშვნელობის მიღებისას, შესაძლებელი ხდება ახალი $scope-ის ბაზაზე ამ ელემენტის აღდგენაც.

შევნიშნავთ, რომ ng-if დირექტივის გამოყენების მეშვეობით ფრეიმვორკის სწრაფმოქმედება იზრდება.

```
<div ng-if="main.userExists">
  შეიტანეთ ლოგინი!
</div>
```

8.8 ng-switch

ng-switch დირექტივა იმგვარადვე მოქმედებს, როგორც პროგრამირებაში კარგად ცნობილი case / switch ოპერატორი. ამრიგად, შესამოწმებელი მნიშვნელობის მიხედვით, $scope-ში რამდენიმე ელემენტიდან ამოირჩევა ერთ-ერთი:

```
<div ng-switch on="main.user.access">
  <div ng-switch-when="admin">
    <!-- code for admins -->
  </div>
  <div ng-switch-when="user">
    <!-- code for users -->
  </div>
  <div ng-switch-when="author">
    <!-- code for authors -->
  </div>
</div>
```

8.9 ng-bind

როგორც ვნახეთ, DOM-ში მნიშვნელობების ჩასმა ხდება {{ value }} სინტაქსის მეშვეობით, თუმცა არსებობს სხვა გზაც - ng-bind დირექტივაც, რომლის დადებითი მხარეა ის, რომ ფურცლის ჩატვირთვისას შედეგის გამოთვლამდე კონტენტი არ ეკრანზე აისახება და ამის გამო ციმციმს ადგილი აღარ ექნება.

```
<p>{{ main.name }}</p>
<p ng-bind="main.name"></p>
```

8.10 ng-view

რათა ერთფურცლოვანი გამოყენების განახლება ავტომატურად მოხდეს ფურცლის ხელახლა ჩატვირთვის გარეშე, XMLHttpRequest მოთხოვნით გადმოგზავნილი, დინამიკურად ჩასადგამი კოდის მისაღებად ვიყენებთ ng-view ატრიბუტს ცარიელი ელემენტით.

ng-view-ის ადგილას ჩაიდგმება სხვადასხვა view (URL-ში მითითებული გზის მიხედვით), მაგალითად, Angular ჩადგამს login.html-ს, თუ URL-ის მნიშვნელობა იქნება myapp.com/#/login და ა.შ.

8.11 HTML-ის გაფართოება

როგორც ვნახეთ, ჩაშენებული დირექტივები მნიშვნელოვნად ამაღლებს HTML-ის ფუნქციონირების დონეს, თუმცა, ზოგიერთი შემთხვევისათვის მათი შესაძლებლობები საკმარისი აღარაა და ენისადმი გაზრდილი მოთხოვნების

დასაკმაყოფილებლად Angular-ის დამპროექტებლებმა კიდევ უფრო სრულყოფილი გახადეს ეს ფრეიმვორკი მასში მომხმარებელების მიერ საკუთარი დირექტივების შექმნის შესაძლებლობის დამატებით.

ქვემოთ განიხილება API ასეთი დირექტივების შესაქმნელად.

9. გაწყობადი დირექტივები

Angular-ისათვის განკუთვნილი გაწყობადი დირექტივები იყოფა სამ ჯგუფად:

- გაწყობადი ელემენტები,
- ჩრდილოვანი DOM,
- HTML-ის იმპორტი.

9.1 გაწყობადი ელემენტები

აღსანიშნავია, რომ გაწყობადი ელემენტების ჯგუფს მიაკუთვნებენ არა მარტო HTML ელემენტებს, არამედ ასევე - ატრიბუტებს, კლასებს და კომენტარებსაც.

პრაქტიკაში ყველაზე მეტად პოპულარულობა მოიპოვა გაწყობადმა ატრიბუტებმა მათი არაერთხელ და შაბლონის სახით გამოყენების შესაძლებლობის გამო. აქვე აღვნიშნავთ, რომ ისინი სხვადასხვა სახის ბროუზერებისათვის ყველაზე ნაკლებ პრობლემას ქმნის.

აი, გაწყობადი ელემენტებისადმი (საკუთრივ Element, ასევე, Attribute, Class და Comment) მიმართვის 4-ვე წესი:

```
<my-element></my-element>
<div my-element></div>

<div class="my-element"></div>

<!-- directive: my-element -->
```

ამ დირექტივების კუთვნილი restrict თვისებით შესაძლებელია მათი გამოყენების ვარიანტების შეზღუდვა. კერძოდ, restrict თვისებისთვის 'EA' მნიშვნელობის მინიჭებით გათვლა ხდება მხოლოდ ელემენტებსა და ატრიბუტებთან, 'C'-ით - კლასებსა, ხოლო 'M'-ით - კომენტარებთან მუშაობაზე.

9.2 ჩრდილოვანი DOM

ჩრდილოვანი DOM უზრუნველყოფს მასში იმპორტირებული როგორც წმინდა HTML კოდის, ასევე კონტენტის ასახვას გაწყობად ელემენტში. შედეგად, ჩვეულებრივ, DOM-ში შესაძლებელი ხდება ჩადგმული დოკუმენტების გადაგზავნაც.

აი, გაწყობად ელემენტში ტექსტის განთავსების მაგალითი:

```
<my-element>
  მოგესალმებით!
</my-element>
```

ამ ტექსტის გადმოწერა შესაძლებელი ხდება ჩრდილოვანი DOM-იდან.

9.3 HTML კოდის შაბლონიზება და იმპორტი

ქვემოთ განიხილება დირექტივებში შაბლონების გამოყენების 3 წესი:

9.3.1 template თვისება

template თვისებით გამოცხადებული სტრიქონის სახის მქონე შაბლონი კომპილირდება Angular-ის მიერ და DOM-ში ჩაისმება.

მაგალითი:

```
{
  template: '<div>' +
    '<ul>' +
      '<li ng-repeat="item in vm.items">' +
        '{{ item }}' +
      '</li>' +
    '</ul>' +
  '</div>'
}
```

ზედა ფრაგმენტს შესაძლებელია ასეთი სახეც მიეცეს:

```
  {
    template: [
      '<div>',
```

```
      '<ul>',
        '<li ng-repeat="item in vm.items">',
          '{{ item }}',
        '</li>',
      '</ul>',
    '</div>'
  ].join('')
}
```

JavaScript-იდან ჩვენთვის ნაცნობი [].join('') კონსტრუქციის გამოყენება კოდს უფრო ადვილად წაკითხვადს ხდის.

9.3.2 templateUrl თვისება

templateUrl თვისება მიუთითებს იმ გარე რესურსსა თუ ელემენტზე, რომელიც საჭირო შაბლონს შეიცავს.

დავუშვა, მოცემულია:

```
{
  templateUrl: 'items.html'
}
```

Angular ჯერ ეცდება DOM-ში მოძებნოს შესაბამისი id-ის მქონე ელემენტი, წარუმატებლობისას კი ამ ფაილზე მოთხოვნას HTTP GET-თი სერვერზე გადააგზავნის.

```
<script type="text/ng-template" id="/hello.html">
  <div>
    <ul>
      <li ng-repeat="item in vm.items">
        {{ item }}
      </li>
    </ul>
  </div>
</script>
```

ზედა მაგალითში თავდაპირველად შაბლონს ტიპად განესაზღვრება text/ng-template მნიშვნელობა, რათა ბროუზერმა, JavaScript-ის მსგავსად, გზავნილი ტექსტად არ აღიქვას. ამ გზით შესაძლებელი ხდება დანიშნულების

ადგილას რამდენიმე შაბლონის ერთი ფაილის სახით გადაგზავნა. ამასთანავე, დასაშვებია `template` თვისების გამოყენებაც, როდესაც შაბლონი სტრიქონში ინახება. Angular იმახსოვრებს ჩატვირთულ შაბლონს, რომლის გამოყენება შემდგომ შესაძლებელი ხდება ng-include და ng-view დირექტივებით. თუ შაბლონი ადგილზე ვერ მოიძებნა, Angular სერვერს მიმართავს GET-მოთხოვნით.

აღვნიშნავთ, რომ დანართი $templateCache ყველა ჩატვირთულ შაბლონს მეხსიერებაში მუდმივად ინახავს.

9.4 API დირექტივების შექმნა

განვიხილოთ საკუთარი დირექტივების შექმნის უმარტივესი მაგალითი მხოლოდ `return` ინსტრუქციის დახმარებით, რომელიც ობიექტს გვიბრუნებს:

```
function someDirective () {
  return {
  };
}
angular
  .module('app')
  .controller('someDirective', someDirective);
```

ქვემოთ ნაჩვენებია `someDirective` დირექტივისადმი მიმართვისას `return` ინსტრუქციით დაბრუნებული ობიექტის ყველაზე პოპულარული თვისებების გადმოცემის მაგალითი:

```
function someDirective () {
  return {
    restrict: 'EA',
    replace: true,
    scope: true,
    controllerAs: 'something',
    controller: function () {
    },
    link: function ($scope, $element, $attrs) {
    },
```

```
    template: [
      '<div class="some-directive">',
        'My directive!',
      '</div>'
    ].join('')
  };
}
angular
  .module('app')
  .controller('someDirective', someDirective);
```

9.4.1 restrict

როგორც უკვე ვნახეთ, `restrict` თვისებისათვის `A', 'E', 'M'` ან `'C'` მნიშვნელობების მიცემის შედეგად, იზღუდება დირექტივის მოქმედების არეალი.

9.4.2 replace

`replace`-ის მეშვეობით დირექტივის თავდაპირველი ელემენტი შეიცვლება სასურველით - სკრიპტის მუშაობის შედეგით. მაგალითად, თუ ვიყენებთ `<some-directive></some-directive>` დირექტივას, `replace: true`-ს მეშვეობით ფურცლის შექმნის შემდეგ მოხდება თავდაპირველი ელემენტის შეცვლა.

9.4.3 scope

საშუალებას იძლევა, გამოყენებული იქნეს მოცემული დირექტივისათვის მიმდინარე ან მშობელი არის (კონტექსტის) `$scope`-ის მემკვიდრეობა. შესაძლებელია,აგრეთვე, იზოლირებული `$scope`-ის შექმნაც და გაწყობადი დირექტივებით მისთვის რაიმე მნიშვნელობების გადაცემა.

9.4.4 controllerAs

ჩვენ უკვე გავეცანით ამ თვისებას - ის დირექტივის შიგნით განსაზღვრავს კონტროლერის სახელს. თუ ვთქვათ, ამ მიზნით ვწერთ *`controllerAs: 'something'`-ს,* კონტროლერისადმი მიმართვებს უნდა ჰქონდეთ ასეთი სახე:

```
something.myMethod();
```

9.4.5 controller

მისი მეშვეობით ხდება არსებული კონტროლერისადმი მიმართვა ან ახლის შექმნა.

თუ კონტროლერი `MainCtrl` სახელით უკვე არსებობს, მას განვსაზღვრავთ ასე:

```
controller: 'MainCtrl'.
```

ინკაფსულაციის შესანარჩუნებლად ახალ კონტროლერს ყოველ ჯერზე ამგვარად ვაცხადებთ:

```
controller: function () {}
```

კონტროლერის უკუგამოძახების ფუნქცია ამუშავებს ცვლილებებს `ViewModel`-ში და განაგებს სერვისებთან ურთიერთობას.

9.4.6 link

`link` ფუნქციისადმი მიმართვა შესაძლებელია ელემენტის კომპილირებისა და `DOM`-ში ჩადგმის შემდეგ. კონტროლერიდან `DOM`-ში ცვლილებების შეტანა არ ხდება, `link` ფუნქციაში კი დაშვებულია როგორც ეს ქმედება, ასევე, - `$scope`-ის, შაბლონის ფესვური `$element` ელემენტისა და `$attrs` ობიექტის ჩასმაც (ეს უკანასკნელი `DOM`-ის ელემენტის ყველა თვისებას შეიცავს).

`link` ფუნქციის შიგნით შესაძლებელია პლაგინების განსაზღვრა, ხდომილებათა დამმუშავებლების გამოძახება და `Angular`-ის სერვისების ჩადგმაც კი.

9.5 დირექტივების შექმნა

ქვემოთ მოგვყავს მაგალითი ისეთი დირექტივის შექმნისა, რომელიც იძლევა კოდში "`email`" კომპონენტის (შესაბამისი ველებიანად) ჩადგმის შესაძლებლობას.

ვქმნით `<compose-email>` ელემენტს, რომლის ადგილზე შემდგომ ჩაიდგმება შესაბამისი კონტენტი. `DOM`-ში ამ ელემენტის ჩადგმა დასაშვებია სხვადასხვა სასურველ ადგილას, თითოეულში კომპონენტის ცალკეული ეგზემპლარის სახით.

ვიწყებთ შაბლონებით:

```
function composeEmail () {
```

```
  return {
    restrict: 'EA',
    replace: true,
    scope: true,
    controllerAs: 'compose',
    controller: function () {
    },
    link: function ($scope, $element, $attrs) {
    },
    template: [
      '<div class="compose-email">',
        '<input type="text" placeholder="To..." ng-
model="compose.to">',
        '<input type="text" placeholder="Subject..." ng-
model="compose.subject">',
        '<textarea placeholder="Message..." ng-
model="compose.message"></textarea>',
      '</div>'
    ].join('')
  };
}
angular
  .module('app')
  .controller('composeEmail', composeEmail);
```

ამრიგად, უკვე შესაძლებელი ხდება `composeEmail` დირექტივის რამდენიმე ადგილზე ჩასმა - საჭირო აღარაა HTML-ის კოდის ფრაგმენტის კოპირება - Angular-ის მიერ დირექტივის სახელის პარსირების შედეგად `<compose-email></compose-email>` ელემენტი გადის HTML-ის კოდის საჭირო ფრაგმენტზე - კომპონენტის ეგზემპლარზე.

10. ფილტრები

ფილტრების მეშვეობით ხდება მათთვის გადაცემული ინფორმაციის რაიმე ლოგიკის შესაბამისად გადამუშავება და მისთვის სასურველი სახის მიცემა, მაგალითად:

- თარიღის საჭირო ფორმატში გამოტანა,
- გვარების სიიდან იმ ელემენტების ამორჩევა, რომლებიც განსაზღვრული ასოდან იწყება და სხვ.

არჩევენ 2 სახის ფილტრს:

- HTML სინტაქსით: `{{ filter_expression | filter : expression : comparator }}`
- და JavaScript სინტაქსით: `$filter('filter')(array, expression, comparator);`

პრაქტიკაში, ძირითადად ,იყენებენ პირველი სახის ფილტრს.

10.1 ფილტრები თარიღისათვის

Angular აადვილებს თარიღებთან მუშაობას ამ ამოცანისადმი შემდგომი მიდგომით (დრო აითვლება მილიწამებში):

```
$scope.timeNow = new Date().getTime():
<p>
  დღეს გვაქვს: {{ timeNow | date:'dd-MM-yyyy' }}
</p>
```

10.2 JSON ფილტრი

JSON-ის ჩადგმული ფილტრები JavaScript-ის ობიექტს გარდაქმნის JSON-სტრიქონად. მიმზიდველი სახით დასაფორმატებლად იყენებენ შემდეგ ტეგებს:

```
</code>
<source lang="html">
<pre>
{{ myObject | json }}
```

10.3 limitTo და orderBy

ზემოთ აღწერილი ფილტრების მუშაობა მარტივ წესებს ექვემდებარება: ფილტრი შესასვლელზე იღებს ერთ რაიმე მნიშვნელობას და მას სასურველ სახეს აძლევს.

ვნახოთ, როგორ მიმდინარეობს მონაცემთა კრებულებების ფილტრით დამუშავება. `limitTo` ლიმიტს აწესებს View-ში გადასაცემი მონაცემების რაოდენობაზე. მისი გამოყენება მოხერხებულია `ng-repeat`-ის შიგნით:

```
<ul>
  <li ng-repeat="user in users | limitTo:10">
    {{ user.name }}
  </li>
</ul>
```

ეს ფრაგმენტი ინფორმაციას გვაწვდის მხოლოდ 10 მომხმარებლის შესახებ. `orderBy` ახდენს გამოსაცემი მასივის ელემენტების სორტირებას ობიექტის რომელიმე თვისების მიხედვით.

```
{
  name: 'Todd Motto',
}
```

მოგვყავს ელემენტების ალფაბეტის მიხედვით სორტირებისა და ეკრანზე ასახვის მაგალითი:

```
<ul>
  <li ng-repeat=" user in users | orderBy:'name' ">
    {{ user.name }}
  </li>
</ul>
```

ქვემოთ გავეცნობით საკუთარი ფილტრების შექმნის მაგალითებს.

11. საკუთარი ფილტრები

საკუთარი ფილტრების შესაქმნელად Angular-ში იყენებენ შესაბამისი დანიშნულების მქონე API-ს. მონაცემთა შორის ორმხრივი კავშირი მარტივად მყარდება. შემუშავებული ფილტრების გამოძახება ხდება $digest ციკლში.

11.1 ფილტრები ერთადერთი მნიშვნელობისათვის

დავიწყოთ მარტივი მაგალითით - ფილტრს გადაეცემა ერთადერთი მნიშვნელობა და .filter() მეთოდით მოგვეწოდება გაფილტრული კონტენტი. თითოეული ასეთი ფილტრი გლობალური სახისაა და მიწვდომადია ხილვადობის ნებისმიერი უბნიდან.

ამ დანიშნულების ფილტრისათვის არსებობს შემდეგი სახის შაბლონი:

```
function myFilter () {
  return function () {
    // შედეგის დაბრუნება
  };
}
angular
  .module('app')
  .filter('myFilter', myFilter);
```

ფილტრებისათვის არგუმენტები ავტომატურად გადაიცემა. ქვემოთ ვქმნით ფილტრს ტექსტის ქვედა რეგისტრში გადასაყვანად (აქვე აღვნიშნავთ, რომ ასეთი დანიშნულების მქონე ჩაშენებული ფილტრი Angular-ში ისედაც არსებობს):

```
function toLowercase () {
  return function (item) {
    return item.toLowerCase();
  };
}
angular
  .module('app')
  .filter('toLowercase', toLowercase);
```

item პარამეტრი ფილტრს გადაეცემა, როგორც ლოკალური ცვლადი. შექმნილ ფილტრს ზუსტად ისევე მივმართავთ, როგორც - ჩაშენებულთ:

```
<p>{{ user.name | toLowercase }}</p>
```

11.2 მონაცემთა კრებულების დამმუშავებელი ფილტრები

შევქმნათ ფილტრი, რომელიც მოცემული კრებულიდან ირჩევს 'A' ასოთი დაწყებულ სიტყვებს:

```
function namesStartingWithA () {
  return function (items) {
    return items.filter(function (item) {
      return /$a/i.test(item.name);
    });
  };
}

angular
  .module('app')
  .filter('namesStartingWithA', namesStartingWithA);
```

გამოვიყენოთ ეს ფილტრი ng-repeat დირექტივაში:

```
<ul>
  <li ng-repeat="item in items | namesStartingWithA">
    {{ item }}
  </li>
</ul>
```

ფილტრში არგუმენტების გადაცემა შესაძლებელია ორწერტილის მეშვეობით:

```
<ul>
  <li ng-repeat="item in items | namesStartingWithA:something">
    {{ item }}
  </li>
</ul>
```

something ავტომატურად გადაიცემა ფილტრის ფუნქციაში:

```
function namesStartingWithA () {
  return function (items, something) {
    // შეღწევადია "items" და something-ც
  };
}

angular
  .module('app')
  .filter('namesStartingWithA', namesStartingWithA);
```

11.3 კონტროლერების ფილტრები

ფილტრები შესაძლებელია შეიქმნას `.filter()` მეთოდის გამოყენების გარეშეც, კონტროლერში ფილტრის როლის მქონე ფუნქციის ჩართვით. ასეთ შემთხვევაში ფილტრი მიწვდომადია მხოლოდ ამ კონტროლერიდან, ანუ ის გლობალური სახის ველარ იქნება.

კონტროლერისათვის ვქმნით `this.namesStartingWithA` ფუნქციას, მიზნის მისაღწევად ვიყენებთ `controllerAs` სინტაქსს:

```
function SomeCtrl () {
  this.namesStartingWithA = function () {
  };
}

angular
  .module('app')
    .controller('SomeCtrl', SomeCtrl);
```

DOM-ში ფილტრის გამოძახების სინტაქსი რამდენადმე განსხვავებული სახისაა:

```
<ul>

  <li ng-repeat="item in vm.items |
filter:namesStartingWithA">
    {{ item }}
  </li>
</ul>
```

12. დინამიკური როუტინგის ორგანიზება $routeProvider-ზე დაყრდნობით

(ერთფურცლოვანი) გამოყენების შესაქმნელად Angular იყენებს ngRoute სახელწოდების როუტერს, რომლის მდგომარეობა URL-ის შემცველობაზეა დამოკიდებული. საჭირო მოდულის მიერთება ამგვარად ხდება:

```
angular
  .module('app', [
    'ngRoute'
  ]);
```

ამის შემდეგ გზების მისათითებლად ხდება $routeProvider-ის ჩადგმა და გაწყობა .config()მეთოდში და უკვე შესაძლებელია მივმართოთ .when() მეთოდს.

დავუშვათ, ჩვენ განკარგულებაშია ელ. ფოსტის წამკითხველი დანართი და მოითხოვება, inbox-ზე დაწკაპუნებისას გამოგვივიდეს წერილების სია. ასეთ შემთხვევაში პირველი არგუმენტი უნდა იყოს შესაბამისი URL-ის აღმწერი სტრიქონი, მეორე კი - დამატებითი გაწყობების შესაძლებლობის მქონე ობიექტი.

არარსებული გზების დასამუშავებლად, აგრეთვე, გათვალისწინებული არის .otherwise() მეთოდიც.

```
function router ($routeProvider) {
  $routeProvider
  .when('/inbox', {})
  .otherwise({
    redirectTo: '/inbox'
  });
}
angular
  .module('app')
    .config(router);
```

გზის გაწყობისათვის თავდაპირველად ვირჩევთ რაიმე შაბლონს, მაგალითად, inbox.html-ს:

```
$routeProvider
.when('/inbox', {
  templateUrl: 'views/inbox.html'
```

```
})
.otherwise({
  redirectTo: '/inbox'
   });
```

თითოეულ view-ს ესაჭიროება კონტროლერი. ამ შესაძლებლობის უზრუნველსაყოფად Angular-სათვის გათვალისწინებული არის თვისებები: controller და controllerAs.

```
$routeProvider
.when('/inbox', {
  templateUrl: 'views/inbox.html',
  controller: 'InboxCtrl',
  controllerAs: 'inbox'
})
.otherwise({
  redirectTo: '/inbox'
});
```

მიზნად დავისახოთ კიდევ ერთი view-ის გაწყობა - დავუშვათ, გვსურს, შემომავალ წერილზე დაწკაპუნებისას ის წავიკითხოთ. ამ შემთხვევაში აუცილებელი ხდება დინამიკური როუტინგის გამოყენება, რადგანაც სხვადასხვა წერილს განსხვავებული URL აქვს. ამგვარი როუტინგის მონაცემების გადაცემისას, დინამიკური ჯგუფის სახელის წინ ისმება ორწერტილი. მაგალითად, id 173921938 იდენტიფიკატორიანი წერილი-სათვის გზა ასეთი სახით იქნება ნაჩვენები:

/inbox/email/173921938,

ხოლო გზის აღწერა კი შემდეგნაირად მოხდება:

'/inbox/email/: id'.

როდესაც დანართს გზის ასარჩევად გადაეცემა ესა თუ ის მნიშვნელობა (მოცემულ შემთხვევაში /inbox/email/173921938), Angular ჩამო-ტვირთავს იმავე შაბლონს და კონტროლერს, რომლებიც გამოიყენებოდა /inbox/email/902827312-ის შემთხვევაში.

საბოლოოდ გვექნება:

```
function router ($routeProvider) {
  $routeProvider
  .when('/inbox', {
    templateUrl: 'views/inbox.html',
    controller: 'InboxCtrl',
    controllerAs: 'inbox'
  })
   .when('/inbox/email/:id', {
    templateUrl: 'views/email.html',
    controller: 'EmailCtrl',
    controllerAs: 'email'
  })
  .otherwise({
    redirectTo: '/inbox'
  });
});
angular
  .module('app')
  .config(router);
```

ამის შემდეგ უნდა მიეთითოს, რომელ ადგილას უნდა ჩაიდგას ფურცელზე დამუშავებული შაბლონი, რისთვისაც გათვალისწინებულია `ng-view` დირექტივა:

```
<div ng-view></div>
```

`Angular` ფრეიმვორკი თვალყურს ადევნებს `URL`-ში განხორციელებულ ყველა ცვლილებას, არკვევს ფურცელზე სხვა შაბლონის დაყენების საჭიროებას და ახორციელებს შესაბამის ქმედებებს.

12.1 $routeParams

`$routeParams` სერვისი ავტომატურად ახდენს `URL` მისამართის პარსირებას და მისგან გამოყოფს პარამეტრების კრებულს, რომელსაც ობიექტად გარდაქმნის. წინა მაგალითში, დინამიკური როუტინგის `:id` სახით განსაზღვრისას, `URL`-იდან ამოირჩეოდა გადმოგზავნილი წერილის `id` იდენტიფიკატორი.

მოვიყვანოთ $routeParams-დან პარამეტრების გადამცემი კონტროლერის მაგალითი:

```
function EmailCtrl ($routeParams, EmailService) {
  // $routeParams { id: 20999851 }
  EmailService
  .get($routeParams.id) // გადაიცეს ობიექტი
  .success(function (response) {})
  .error(function (reason) {});
}

angular
  .module('app')
  .('EmailCtrl', EmailCtrl);
```

13. ფორმების შემოწმება

პირველ რიგში აუცილებელია ფორმას მიეცეს სახელი - ის ფორმისათვის განსაზღვრავს ხილვადობის არეს:

```
  <form name="myForm"></form>
```

Angular აფიქსირებს ფორმას და თვალყურს ადევნებს, დაიცვა თუ არა მომხმარებელმა მისი შევსებისას ყველა საჭირო წესი.

13.1 HTML5

HTML5 ენაში დამატებული იქნა pattern ატრიბუტი. ის ბროუზერს შესაძლებლობას აძლევს, შეამოწმოს კოდის შესაბამისობა ამ ენის მოთხოვნებთან. ასეთი კონტროლის შესაძლებლობა Angular-შიც არის გათვალისწინებული, დამატებით კი ეს ფრეიმვორკი ng-required დირექტივაში required-ით მონიშნული ველებისათვის განუწყვეტლივ ამოწმებს მოდელის მდგომარეობას.

ქვემოთ გავეცნოთ Angular-ის ზოგიერთ სხვა შესაძლებლობასაც.

13.2 $pristine

ფურცლის თავდაპირველად შექმნისას `Angular` ფორმისათვის ამატებს `$pristine` თვისებას და `ng-pristine` კლასში შეჰყავს ის ელემენტები, რომელთაც ჯერ ცვლილება არ განუცდია.

13.4 $dirty

`$dirty` თვისებით პირიქით - იდენტიფიცირდება შეცვლილი ელემენტები. ამასთან, მათ ემატება `ng-dirty` კლასებიც, ხოლო `ng-pristine` კლასები კი ნადგურდება. ფორმა ფურცლის გადატვირთვის გარეშე `$pristine` მდგომარეობას ვერ დაუბრუნდება.

13.5 $valid

`$valid` თვისების მეშვეობით შეტანის თითოეული ველი შესაძლებელია გამოცხადდეს, როგორც ვალიდური. მაგალითად, თუ ველს აქვს `ng-required` ატრიბუტი და მომხმარებელმა ის შეავსო, ველს ენიჭება `ng-valid` კლასი.

13.6 $invalid

წინას ანტიპოდია. დუმილით, ითვლება, რომ ყველა ფორმა `$invalid` მდგომარეობაში იმყოფება და მათ მინიჭებული აქვს `ng-invalid` კლასი. განხილულ 2 მდგომარეობას შორის გადართვები ხდება მომხმარებლის მიერ ინფორმაციის შეტანისას.

13.7 შეტანის ელემენტების ჩართვა-გამორთვა

ზოგჯერ მოითხოვება ინფორმაციის შეტანის ელემენტების (ტექსტური ველის, ღილაკის) ჩართვა-გამორთვა. მაგალითად, ქვემო ფრაგმენტში შეტანის ღილაკი ჩაირთვება მხოლოდ მაშინ, როდესაც მომხმარებელი შეტანის ველში რაიმე სიმბოლოს აკრეფს და თუ ეს არ მოხდა, ფორმა ვერ განახლდება:

```
<input type="text" ng-model="user.name"
placeholder="შეიტანეთ სახელი!">
<button ng-disabled="!user.name.length">
  განვაახლოთ სახელი!
</button>
```

ღილაკი ჩაირთვება მაშინ, როდესაც user.name.length მნიშვნელობა true გახდება. სანამ ფორმასთან მუშაობა მიმდინარეობს, ეს მნიშვნელობა განუწყვეტლივ მოწმდება.

14. სერვერთან ინფორმაციის გაცვლა $http და $resource დანართების (API) მეშვეობით

სერვერთან მაღალი დონის $http და $resource კავშირების საშუალებებითაც შესაძლებელია $digest ციკლების გაშვება და ჩვენი მონაცემების აქტუალიზება.

14.1 $http

სანამ $http მეთოდის დანიშნულებას გავეცნობოდეთ, სასურველია გავიხსენოთ, თუ როგორ მოქმედებდა jQuery-ში გათვალისწინებული $.ajax მეთოდი.

$http მეთოდი შესაძლებელია გამოვიყენოთ, როგორც ფუნქცია და ასევე, როგორც ობიექტი.

ქვემოთ მოგვყავს GET მარტივი HTTP-მოთხოვნის ფორმირების მაგალითი:

```
$http.get('/url')
.success(function (data, status, headers, config) {})
.error(function (data, status, headers, config) {});
```

შესაძლებელია კიდევ უფრო მარტივი მიდგომის გამოყენებაც:

```
$http.get('/url')
.success(function (response) {})
.error(function (reason) {});
```

არსებობს ასეთი გზაც:

```
 $http.get('/url')
.then(function (response) {
  // პასუხი წარმატებისას
}, function (reason) {
  // ინფორმაცია წარუმატებლობისას
```

```
});
```

jQuery ბიბლიოთეკისაგან განსხვავებით, Angular-ი *$http* გამოძახებებს განათავსებს $scope.$apply()-ში, რის შედეგად სერვერზე არსებული მონაცემებისათვის გაიშვება მოთხოვნების ციკლი და ხდება კავშირების განახლება.

14.2 $resource

$http მეთოდთან ერთად შესაძლებელი არის მოდულით სარგებლობაც, რომელშიც არსებული API $resource აადვილებს CRUD (create, read, update, delete) ოპერაციების ჩატარებას.

ეს რესურსი ქმნის ობიექტს, რომლის მეშვეობითაც მონაცემთა წყაროებთან ურთიერთობა მყარდება REST პროტოკოლებზე დაყრდნობით.

```
function MovieService ($resource) {
  return $resource('/api/movies/:id', { id: '@_id' },
    {
      update: {
        method: 'PUT'
      }
    }
  );
}
angular
  .module('app')
  .factory('MovieService', MovieService);
```

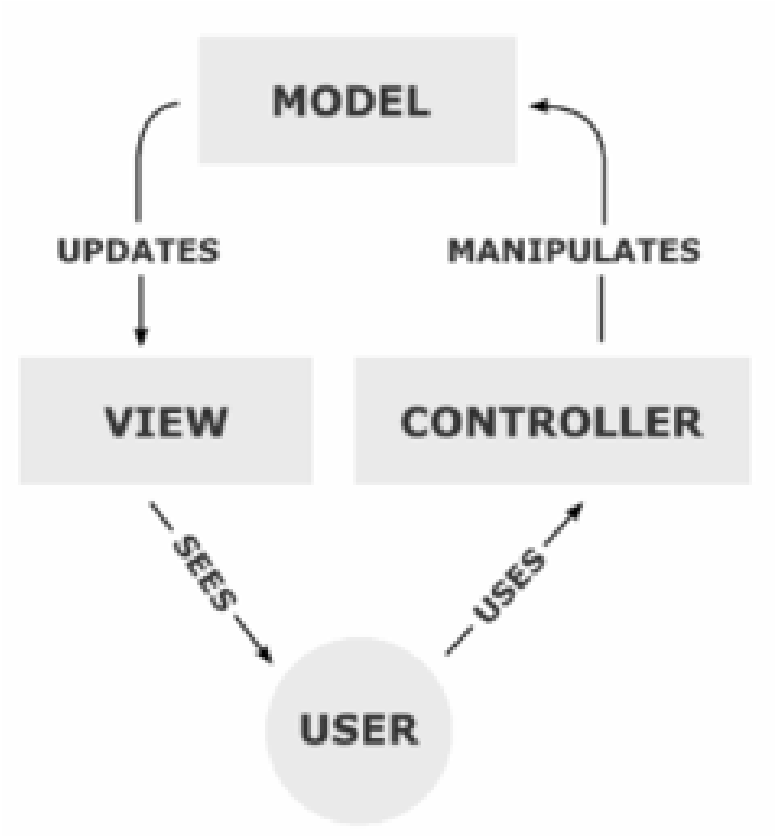
შესაძლებელია, დამოკიდებულება ჩაისვას მარტივ Movies ფაბრიკაში ჩვენ მიერ ფორმირებულ კონტროლერებში და შედეგად წვდომა გვქონდეს ყველა საჭირო მონაცემზე.

```
function MovieCtrl (MovieService) {
  var movies = new MovieService();
  // მოხდა განახლება
  movies.update(/* some data */);
}
```

```
angular
  .module('app')
  .controller('MovieCtrl', MovieCtrl);
```

დანართი_1

MVC (Model-View-Controller) მოდელის სქემა



დანართი_2
CGI-დაპროგრამება

ამ ორი ათეული წლის წინ ინტერნეტი მომხმარებელს, ძირითადად, სტატიკური სახის ინფორმაციას აწვდიდა – Web-კვანძებიდან ბროუზერში გადმოიგზავნებოდა "სუფთა" HTML-ფაილები. შემდგომ კი შესაძლებელი გახდა Web-კვანძებში განგვეთავსებინა Web-გამოყენებები, რომელთა საშუალებითაც შესრულებაზე ვუშვებთ სერვერზე არსებულ რომელიმე პროგრამას, სერვერზევე განთავსებულ მონაცემთა ბაზებიდან ვიღებთ სასურველ ინფორმაციას, ვსარგებლობთ ელექტრონული ფოსტის მომსახურებით და სხვ.

Web-ტექნოლოგიის თავისებურება ის გახლავთ, რომ აქ კლიენტი-კომპიუტერი უშუალოდ არ მიმართავს სერვერზე არსებულ ოპერაციულ სისტემას ამა თუ იმ, თუნდაც, ჩვეულებრივი გამოყენების, მაგალითად, Access-ის გასაშვებად. კერძოდ, მიმართვა ხდება კლიენტზე არსებული Web-ბროუზერიდან Web-სერვერ-პროგრამისადმი და სწორედ Web-სერვერ-პროგრამა გვევლინება ინფორმაციის გადამგზავნის როლში ბროუზერსა და Web-გამოყენებას შორის.

Web-სერვერ-პროგრამა განლაგებულია Web-სერვერ-კომპიუტერზე. აქვე ინახება Web-ფურცლები და Web-გამოყენებები. რაც შეეხება ოპერაციულ სისტემებს, ინტერნეტში ჩართული Web-სერვერ-კომპიუტერისთვის, როგორც წესი, უპირატესობას ანიჭებენ Unix-ს, ხოლო ლოკალური ქსელის სერვერზე კი განათავსებენ Windows სისტემას.

Web-ტექნოლოგიებიდან ამ 10-15 წლის წინ ერთ-ერთი პოპულარული გახლდათ CGI (Common Gateway Interface)-ტექნოლოგია. მისი მეშვეობით შექმნილ გამოყენებებს უწოდებენ Web-პროგრამებს. აღსანიშნავია, რომ CGI-ტექნოლოგია მხოლოდ Interface-ის (ურთიერთკავშირის) სტანდარტს განსაზღვრავს Web-სერვერსა და პროგრამას შორის. თვით Web-პროგრამა კი, ფაქტობრივად, ნებისმიერ დაპროგრამების ენაზე შეიძლება დაიწეროს, თუმცა, იმ პერიოდში ყველაზე ხშირად იყენებდნენ Perl-ენას.

ამრიგად, CGI-პროგრამის განთავსება ხდება Web-სერვერ-კომპიუტერზე, ხოლო მისი შესრულებისათვის კლიენტი ბროუზერის მეშვეობით მიმართავს Web-სერვერ-პროგრამას.

Web-გამოყენებას (პროგრამას) წაეყენება ორი მოთხოვნა:

1. Web-სერვერს უნდა შეეძლოს მისი გაშვება. ეს პირობა შესრულდება, თუ პროგრამა გაიშვება კომპიუტერის საბრძანებო სტრიქონიდან ამ პროგრამის მხოლოდ სახელის აკრეფითა და Enter-ზე ხელის დაჭერით.
2. პროგრამა Web-სერვერის მეშვეობით მომხმარებელს უნდა უბრუნებდეს სწორ მონაცემებს.

ახლა კი განვიხილოთ, როგორ ხდება სეანსის ორგანიზება HTTP-ოქმით მონაცემების გადასაცემად.

პროცესი იწყება იმით, რომ კლიენტი (ბროუზერი) Web-სერვერს მოთხოვნას უგზავნის რაიმე რესურსზე.

თუ რესურსი გახლავთ HTML-ფაილი, მოთხოვნას შეიძლება ჰქონდეს ამგვარი სახე:

HTTP://www.rc3.org/cgibook/index.html

ეს ჩანაწერი რესურსის URL მისამართია, www.rc3.org ის საქაღალდეა, რომელშიც დაყენებულია Web-სერვერ-პროგრამა.

თუ ფაილი მოიძებნა, Web-სერვერი მის ასლს გადაუგზავნის ბროუზერს.

დავუშვათ, რომ URL მიუთითებს CGI-პროგრამაზე, მაგალითად:

HTTP://www.rc3.org/cgi-bin/example.cgi

სერვერი ამ პროგრამას შესრულებაზე გაუშვებს და მიღებულ შედეგებს გადაუგზავნის ბროუზერს. მიღებული მოთხოვნის დასაკმაყოფილებლად სერვერი ამოწმებს:

1. არსებობს მოთხოვნილი ფაილი?
2. ეს ფაილი cgi-პროგრამა გახლავთ?
3. არსებობს ნებართვა ამ პროგრამის შესრულებაზე?
4. პროგრამა უშეცდომოდ შესრულდა?
5. შეესაბამება მიღებული შედეგები ბროუზერის მოთხოვნას?

მოთხოვნა რომ შესრულდეს, თითოეულ ამ კითხვაზე უნდა გაეცეს დადებითი პასუხი. საწინააღმდეგო შემთხვევაში მომხმარებელი მიიღებს შეტყობინებას შეცდომის შესახებ (ზოგჯერ კი არც ამ შეტყობინებასაც).

CGI-პროგრამის შესრულების შედეგად შეიძლება ბროუზერს სხვადასხვა სახის ინფორმაცია გადმოეგზავნოს. როცა CGI-პროგრამა ბროუზერს HTML-ტექსტს უგზავნის, მან ამ ტექსტს წინ უნდა წარუმძღვაროს შესაბამისი შეტყობინება – **დასაბრუნებელი მონაცემების ტიპის ამსახველი სათაური.**

შევნიშნოთ, რომ მოცემულ შემთხვევაში სათაურს ექნება ასეთი სახე:

Content–type: text/html

არსებობს დასაბრუნებელი მონაცემების სხვა ტიპებიც:

text/plain	უბრალო ტექსტი,
image/gif	gif-ნახატი,
image/jpg	jpg-ნახატი,
video/quicktime	quicktime-ანიმაცია,
application/octet-stream	ბინარული ფაილი.

ყველა ამ ტიპს ეწოდება MIME-ტიპი.

სათაურების უბანი სხვა სამსახურებრივ ინფორმაციასაც შეიცავს. აუცილებელია ამ მონაცემების დაცილება მომდევნო, HTML-ტექსტიდან **ერთადერთი სტრიქონით.**

მოვიყვანოთ სერვერიდან ბროუზერისათვის გადაგზავნილი პასუხის მაგალითი:

HTTP/ 1.1 200 OK
DATE Sun, 30 Jul 2002 05:23:47 GMT
Server: Apache / 1.3.3

Connection: close

Content type: text / html

<html>
<head>
<title> მაგალითი 1</title>
</head>
</body>
ეს გახლავთ მარტივი მაგალითი
</body>
</html>

რაც შეეხება ბროუზერიდან სერვერზე ინფორმაციის გადაგზავნის საშუალებებს, ამ მიზნით ძირითადად იყენებენ html-ფორმებს, მაგრამ არსებობს სხვა გზებიც.

სერვერისათვის ინფორმაციის გადაცემისას უნდა გავითვალისწინოთ, რომ ზოგიერთ სიმბოლოს მოთხოვნაში განსაკუთრებული როლი ენიჭება. მაგალითად: "?" სიმბოლო URL-მისამართში ერთმანეთისგან განაცალკევებს ფაილის სახელსა და მოთხოვნის სტრიქონს, ხოლო "+" სიმბოლო შუალედის, ანუ ჰარის შემცვლელად გამოიყენება. მაგრამ იგივე სიმბოლოები, ცხადია, მოთხოვნის სტრიქონში შეიძლება ტრივიალური დანიშნულებითაც გვხვდებოდეს. ასეთ შემთხვევებში შეცდომების თავიდან ასაცილებლად მიმართავენ ე. წ. Web-კოდირების ხერხს - ამ (და ზოგ სხვა) სიმბოლოებს მოთხოვნის სტრიქონში მხოლოდ კოდირებული სახით განათავსებენ (კოდირების ამ სახეს ჩვენ html-დან ვიცნობთ).

მოკლედ განვიხილოთ CGI-ის **დადებითი** და **უარყოფითი** მხარეები.

დადებითიდან პირველ რიგში აღსანიშნავია ის გარემოება, რომ, ფაქტობრივად, ნებისმიერი თანამედროვე დაპროექტების ენა შესაძლებელია გამოყენებული იქნას CGI-პროგრამების დასაწერად, თუმცა უფრო ხშირად უპირატესობას ანიჭებენ Perl-ენას. შევნიშნოთ, რომ ამ ენის შესწავლა განსაკუთრებულ სირთულეებთან არ არის დაკავშირებული. Perl-ენის უდავო ღირსება ის გახლავთ, რომ მასზე დაწერილი

პროგრამები ყოველგვარი ცვლილების გარეშე შეიძლება გამოვიყენოთ როგორც Unix, ისე Windows ოპერაციული სისტემების გარემოში. ესენი კი სწორედ ის ოპერაციული სისტემებია, რომლებიც გამოიყენება Web-სერვერ-კომპიუტერების უმეტესობაზე. CGI-დაპროგრამების დადებითი მხარეა კლიენტსა და სერვერს შორის მარტივი ინტერფეისის უზრუნველყოფაც.

ნაკლოვანი მხარეებიდან კი უნდა აღინიშნოს:

- ინტერპრეტატორის ხელახლა გაშვების აუცილებლობა CGI-პროგრამისადმი ყოველი მიმართვისას (თუ, რა თქმა უნდა, პროგრამა ინტერპრეტატორული ტიპის ენაზეა, მაგალითად, Perl-ზე გახლავთ დაწერილი), რაც, ცხადია, ეს ანელებს პროგრამის მუშაობას.
- html-ტექსტის გენერირების საჭიროება და არა უშუალოდ მისი ფურცელზე ჩადგმის შესაძლებლობა (რასაც უზრუნველყოფს, ვთქვათ, Microsoft კომპანიის მიერ შემოთავაზებული ASP-ტექნოლოგია. აქ პროგრამები იწერება VBScript ენაზე, რომელიც წარმოადგენს Visual Basic-ის ვერსიას).

რაც შეეხება უშუალოდ დაპროგრამების Perl ენას, მისი უარყოფითი მხარე (პროგრამისადმი ყოველი მიმართვისას ინტერპრეტატორის გაშვების საჭიროება) დამახასიათებელია ინტერპრეტირების ნებისმიერი ენისათვის. სამაგიეროდ, ამ ენაზე პროგრამირებას აქვს ბევრი დადებითი მხარე:

- პირველ რიგში აღვნიშნოთ, რომ მისთვის არსებობს მრავალი ფუნქციის შემცველი ბიბლიოთეკა (შედის CGI.pm მოდულის შემადგენლობაში).
- ძალიან მნიშვნელოვანია ის ფაქტიც, რომ Perl ენაზე უკვე მრავალი პროგრამაა დაწერილი და გარკვეული მოდიფიკაციის შემდეგ შესაძლებელია მათი გამოყენება დასახული მიზნების მისაღწევად.

სამუშაო ადგილის მომზადება

CGI-დაპროგრამება რამდენიმე ეტაპს მოიცავს:

1. CGI-პროგრამის დაწერა. ეს პროცესი, ფაქტობრივად, ნებისმიერ კომპიუტერზე შეიძლება განხორციელდეს. შესაძლებელია კოდი NotePad ტექსტურ რედაქტორში ავკრიფოთ, ფაილს ენიჭება pl გაფართოება.
2. პროგრამული კოდის სისწორის შესამოწმებლად ის შეგვიძლია ადგილზევე გავუშვათ შესრულებაზე საბრძანებო სტრიქონიდან. თუმცა, უმჯობესია, ეს მოხდეს Web-სერვერის მეშვეობით, რადგანაც სწორედ ეს გზა არის დამახასიათებელი მე-3 ეტაპისათვის. შეცდომების აღმოჩენის შემთხვევაში კოდში შეგვაქვს შესწორებები.
3. გამართულ პროგრამას Web-სერვერზე განვათავსებთ.

მე-3 ეტაპზე, უპირველეს ყოვლისა, გადასაწყვეტია საკითხი, მუშაობას ვაპირებთ საკუთარი ორგანიზაციის ლოკალური ქსელის ფარგლებში, თუ ვსარგებლობთ ინტერნეტის ქსელით.

პირველ შემთხვევაში ჩვენ მიერ დაწერილ და გაწყობილ CGI-პროგრამებს ლოკალური ქსელის სერვერზე განვათავსებთ, მეორე შემთხვევაში კი შეგვიძლია არჩევანის გაკეთება - კვლავ საკუთარი სერვერით ვისარგებლოთ (რომელიც ამჯერად ინტერნეტში იქნება ჩართული), თუ არენდით ავიღოთ ადგილი პროვაიდერის სერვერზე (შევნიშნოთ, რომ პროვაიდერების უმრავლესობა წინააღმდეგია მათ სერვერებზე მომხმარებლის CGI-პროგრამის განთავსების).

Web-სერვერ-კომპიუტერებზე გამოყენებული ოპერაციული სისტემების შესახებ ჩვენ უკვე გვქონდა მსჯელობა. როგორც წესი, უპირატესობას ანიჭებენ:

- დიდი სისტემებისათვის - Unix-ს;
- სტანდარტული პერსონალური კომპიუტერებისათვის – Linux ან Windows-ს.

CGI-პროგრამის შემოწმებისა და ფუნქციონირებისათვის საჭიროა ჩვენ განკარგულებაში იმყოფებოდეს:

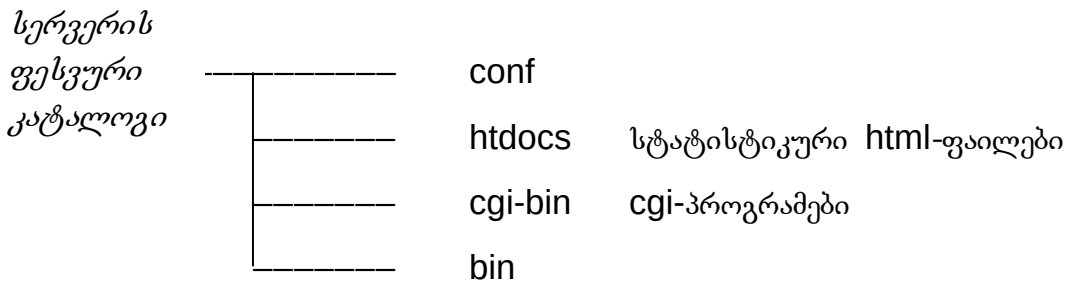
1. Web-სერვერ-პროგრამა. ცხადია, მას უნდა შეეძლოს CGI პროგრამებთან ურთიერთობა, რაც თანამედროვე სისტემებისათვის პრობლემას არ წარმოადგენს.
2. CGI-პროგრამირებისათვის გამოყენებული ენის ინტერპრეტატორი (ან კომპილატორი). თუ პროგრამა სცენარების ენაზე იწერება, ის განლაგებული უნდა იყოს bin/sh საქაღალდეში.
3. ბიბლიოთეკა, რომელშიც განთავსებული იქნება არჩეული ენისათვის დამახასიათებელი მონაცემების გარდაქმნის საშუალებები, რომელთა გამოყენება ამარტივებს დაპროგრამების პროცესს.

რაც შეეხება Web-სერვერებს, მათგან დღეს ყველაზე უფრო პოპულარულია:

- Apache სისტემა Unix ოპერაციული სისტემისათვის;
- Microsoft Internet Internation Server (IIS) Windows-ების ოჯახისათვის.

მათი მეშვეობით ხდება ბროუზერიდან მოთხოვნების მიღება-დამუშავება, HTML-ფურცლების და CGI-პროგრამების მოძებნა (უკანასკნელთა შესრულებაზე გაშვებაც) და მიღებული შედეგების ბროუზერისათვის გადაგზავნა.

Web-სერვერები HTML-ფაილების და CGI-პროგრამების შესანახად იყენებენ სტანდარტულ სტრუქტურებს:



დასაშვებია ამ სტრუქტურის იმგვარად გადაწყობაც, რომ Web-სერვერმა cgi-პროგრამები htdocs საქაღალდეში ეძებოს. ასეთ მიდგომას ერთი ღირსებაც გააჩნია – შესაძლებელი ხდება კვანძის საწყისი ფურცლის გენერირება სწორედ cgi-პროგრამის მეშვეობით განხორციელდეს.

შეიძლება, აგრეთვე, ფსევდონიმების (aliases) შექმნით ნებისმიერი საქაღალდე ვაქციოთ htdocs საქაღალდის ქვესაქაღალდედ.

ხაზი გავუსვათ შემდეგ გარემოებას - Web-სერვერ-პროგრამა მხოლოდ განსაზღვრავს, რომელი cgi-პროგრამა უნდა შესრულდეს (ან რომელი სტატიკური html-ფაილი უნდა გადაეგზავნოს ბროუზერს), პროგრამის შესრულება კი ოპერაციული სისტემის პრეროგატივაა. უმეტესად გამოიყენება შემდეგი გზა:

cgi-პროგრამები განთავსდება cgi-bin საქაღალდეში და აქ მყოფი რომელიმე პროგრამის შესრულებაზე გასაშვებად ბროუზერიდან ხდება Web-სერვერისადმი მიმართვა. შევნიშნოთ, რომ აქ განთავსებულ ფაილებს სერვერი პროგრამად თვლის.

მეორე გზა კი, როგორც ზემოთ აღვნიშნეთ, გახლავთ პროგრამების სტატიკურ ფაილებთან ერთად განთავსება. ასეთ შემთხვევაში Web-სერვერ-პროგრამას მიეთითება, რომ ყურადღება უნდა მიაქციოს ფაილის გაფართოებას.

დასასრულ, მოვიყვანოთ Perl ენაზე დაწერილი მარტივი პროგრამის მაგალითი. შევინახოთ ის test.pl სახელით cgi-bin საქაღალდეში.

```
#! /usr/local/bin/perl
use CGI;
$query=new CGI;
print $query–>header;

print “<html><head><title>შემოწმება</title></head>\n”;
print “<body>შემოწმებამ წარმატებით ჩაიარა.</body></html>”;
```

პირველი სტრიქონი მიუთითებს, თუ სად მდებარეობს ჩვენ კომპიუტერზე Perl ინტერპრეტატორი. Apache სერვერისაგან განსხვავებით, Windows-ის ეგიდის ქვეშ მომუშავე სხვა Web-სერვერები ამ სტრიქონის იგნორირებას ახდენს, რადგანაც მათ საჭირო ინფორმაციას ფაილის გაფართოება აწვდის.

ახლა უკვე შესაძლებელია cgi-პროგრამის გაშვება ბროუზერის სამისამართო სტრიქონიდან შემდეგი ბრძანების აკრეფვით:

http:// localhost/cgi-bin/test.pl

თუ ყველაფერი სწორად გვაქვს გაკეთებული, კლიენტი-კომპიუტერის ეკრანზე აისახება ასეთი შეტყობინება:

The test was successful

სხვა შემთხვევებში კი ჯერ უნდა შევამოწმოთ, შესაძლებელია თუ არა პროგრამის გაშვება Web-სერვერისთვის გვერდის ავლით საბრძანებო სტრიქონიდან:

c:\\perl\bin\perl.exe c:\\apache\cgi-bin\test.pl

აღვნიშნოთ, რომ ბრძანების თავში მითითებულია ინტერ-პრეტატორის ადგილსამყოფელი.

თუ პროგრამის აკრეფისას შეცდომა არ მოგვსვლია, ეკრანზე უნდა გამოვიდეს ასეთი შედეგი:

Content–type: text/html

<html><head><title>შემოწმება</title></head>

<body>შემოწმებამ წარმატებით ჩაიარა.</body></html>

იმ შემთხვევაში, თუ პროგრამაში რაიმე შეცდომა გვაქვს დაშვებული, ინტერპრეტატორი მოგვაწვდის შესაბამის ინფორმაციას.

დანართი_3

პრეპროცესორი

პრეპროცესორი არის კომპიუტერული პროგრამა, რომელიც შესასვლელზე მიღებულ მონაცემებს (მათ როლში, მეტწილად , გვევლინება ამა თუ იმ ენაზე დაწერილი პროგრამა) გარდაქმნის გამოსასვლელ მონაცემებად, რომლებიც უკვე სხვა პროგრამისათვის, მაგალითად, კომპილატორისათვის, წარმოადგენს შემავალ მონაცემებს.

გარდაქმნის სახე და შედეგები დამოკიდებულია პრეპროცესორზე - ზოგიერთი მათგანი მხოლოდ მარტივ ჩანაცვლებებს ახდენს ამოსავალ ტექსტში, ნაწილის შესაძლებლობები კი დიდად არ ჩამოუვარდება დაპროგ-რამების ენებისას.

ყველაზე ხშირად პრეპროცესორებს იყენებენ ასეთი დანიშნულებით - მათ ევალებათ, ამა თუ იმ წესით დაამუშაონ საწყისი პროგრამული კოდი კომპი-ლაციის მომდევნო ნაბიჯის წინ.

რაც შეეხება PHP ენას, უმეტეს შემთხვევაში ის გამოიყენება ვებ-ფურცლების დასამუშავებლად, რომლებშიც არსებული ტექსტი უცვლელად გამოდის <?php ?> უბნამდე. ამ უკანასკნელში კი განთავსებულია ენის ის

ინსტრუქციები, რომლებიც აფორმირებენ ეკრანზე გამოსატანი დამატებითი ინფორმაციისათვის შესაბამის HTML კოდს.

მოგვყავს მიმდინარე დროის ამსახველი ვებ-ფურცლის მაგალითი:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
  <title> მიმდინარე დრო </title>
<head>
<body>
<h1>მიმდინარე დრო</h1>
 <?php
   print strftime('თბილისის დროით %H სთ, %M წთ და %S
წამია.');
 ?>
</body>
</html>
```

PHP-ის პრეპროცესორი მოახდენს შემდეგ ჩანაცვლებას:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>მიმდინარე დრო</</title>
<head>
<body>
 <h1>მიმდინარე დრო</</h1>
 თბილისის დროით 10 სთ, 15 წთ და 20 წამია.
</body>
</html>
```

დანართი_4

ბიზნესლოგიკა

ბიზნესლოგიკა, ზოგადად, საინფორმაციო სისტემებისათვის განიმარტება, როგორც ადამიანის საქმიანობის ამა თუ იმ სფეროს ობიექტებისთვის დამახასიათებელი ქმედებების მარეგულირებელი წესების, პრინციპების, შეზღუდვების ერთობლიობა.

ზემოთ თქმულიდან გამომდინარე, თვლიან, რომ ბიზნესლოგიკა და სიტყვათშეთანხმება „საგნობრივი სფეროს ლოგიკა“ (ინგლ.*domain logic*) სინონიმებია. ბიზნესლოგიკის მეშვეობით ხდება საგნობრივი სფეროსათვის ორგანული გარემოს რეალიზება საინფორმაციო სისტემაში. კონკრეტულად მისი გამოხატულება შესაძლებელია იყოს:

ექსელის ფაილებში ჩატარებული გათვლები თანამშრომელთა ხელფასების გამოსაანგარიშებლად, პროექტის ხელმძღვანელის ინიცირებით თანამშრომლებისათვის ელექტრონული ფოსტით შეტყობინებების ავტომატიზებული გაგზავნა, ავიარეისების გადადებისას ოტელში ადგილების შეკვეთაზე უარის თქმის შეტყობინება და სხვ.

ბიზნესლოგიკის ფიქსირებისათვის გამოიყენება:

- ტექსტი;
- საგნობრივი სფეროს კონცეფტუალური ანალიტიკური მოდელები (ე.წ. ონტოლოგია);
- სხვადასხვა სახის გრაფები (მაგალითად, მდგომარეობიდან მდგომარეობაში გადასვლების ამსახველი), დიაგრამები, ბლოკ-სქემები, ალგორითმები;
- ბიზნესპროცესების მოდელები.

კომპიუტერული სისტემის დაპროექტების საწყის ფაზაში ბიზნესლოგიკის ანალიზი და ასახვა, როგორც წესი, ხდება UML ენის (ან მისი მსგავსი) ენების დიაგრამებში.

შემდგომ ეტაპზე - პროგრამების დაწერისას, თუ პროცედურულ ენებს ვიყენებთ, მაშინ ბიზნესლოგიკა რეალიზდება ფუნქციებსა და პროცედურებში, ხოლო ენებში ობიექტზე ორიენტირებული შესაძლებლობების გამოყენებისას - კლასებისა და მეთოდების მეშვეობით.

დანართი_5

სერვერული გლობალური ცვლადები

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)

$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server

$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

დავალება: *დაწერეთ პროგრამა, რომელიც ეკრანზე გამოიტანს ამ გლობალური ცვლადების მნიშვნელობებს; გაანალიზეთ მიღებული შედეგები და შეადარეთ ცვლადების ზემოთ აღწერილ დანიშნულებებს. თუ ინგლისურ ენას ცუდად ან საერთოდ ვერ ფლობთ, მიაკითხეთ მთარგმნელ პროგრამებს, მაგალითად,* google*-ს და იზრუნეთ, აითვისოთ ინგლისური ენა თუნდაც მხოლოდ ისეთ დონეზე, რომელიც საკმარისი იქნება ინფორმატიკის დარგის საგნების შესასწავლად. ამ ენის ცოდნის გარეშე სპეციალობაში წარმატებების მიღწევა იქნება სათუო.*

დანართი_6

სტრიქონების დამმუშავებელი ფუნქციები

PHP-ში

chr	აბრუნებს სიმბოლოს ASCII-ში კოდის სახით.
chunk_split	სტრიქონს დაყოფს მოცემული სიგრძის ქვესტრიქო-ნებად.
crypt	ახდენს სტრიქონის შიფრაციას.
echo	ეკრანზე გამოჰყავს ერთი ან რამდენიმე სტრიქონი.
explode	სტრიქონს ყოფს ქვესტრიქონებად, აქცევს მათ მასივად და ელემენტებს განაცალკევებს მოცემული სიმბოლოთი.
html_entity_decode	HTML-კოდი გადაჰყავს შესაბამის სპეციალურ სიმბოლოებში ანუ htmlentites ფუნქციის საპირისპიროდ იქცევა.
htmlentites	სპეციალური სიმბოლოები გადაჰყავს HTML-წარმოდგენაში.
htmlspecialchars	ყველა სიმბოლოს აძლევს HTML-სახეს.
implode	მასივის ელემენტებიდან ქმნის სტრიქონს.
ltrim	სტრიქონიდან შლის წინა შუალედებს.
rtrim	სტრიქონიდან შლის ბოლოში მდებარე შუალედებს.
number_format	რიცხვს წარმოადგენს სტრიქონის სახით.
ord	აბრუნებს სიმბოლოს ASCII-კოდს.
parse_str	ახდენს URL სტრიქონის დაყოფას და ცვლადებს ანიჭებს მნიშვნელობას.
print	გამოჰყავს სტრიქონი.

printf	გამოჰყავს დაფორმატებული სტრიქონი.
sprintf	გვიბრუნებს დაფორმატებულ სტრიქონს.
setlocale	იძლევა ინფორმაციას კოდური ფურცლის შესახებ.
similar_text	ადგენს ორი სტრიქონის მსგავსების ხარისხს.
sscanf	სტრიქონს ყოფს შაბლონის მიხედვით და მიღებულ მნიშვნელობებს ანიჭებს ცვლადებს.
str_ireplace	იქცევა როგორც str_replace, მაგრამ მისგან განსხვავებით, ყურაღებას არ აქცევს სიმბოლოების რეგისტრს.
str_pad	სტრიქონს ავსებს მოცემულ სიგრძემდე სხვა სტრიქონით.
str_repeat	სტრიქონს იმეორებს მოცემულ რიცხვჯერ.
str_replace	სტრიქონში ყველა ქვესტრიქონს ცვლის მოცემული სტრიქონით.
str_shuffle	სტრიქონში შემთხვევითობის წესით ცვლის სიმბოლოების თანმიმდევრულობას.
str_split	სტრიქონის სიმბოლოებისაგან ქმნის მასივს.
str_word_count	სტრიქონში ითვლის სიტყვების რაოდენობას.
strcasecmp	სტრიქონებს ერთმანეთს ადარებს ბაიტების დონეზე, რეგისტრში განსხვავებას ყურადღებას არ აქცევს.
strcht	strstr-ის მსგავსად ფუნქციონირებს.
strcmp	იქცევა strcasecmp-ის მსგავსად, მაგრამ, მისგან განსხვავებით, ითვალისწინებს რეგისტრში სხვაობასაც.
strip_tags	სტრიქონიდან ამოაგდებს ყველა HTML და PHP ტეგს.
stripos	გამოავლენს სტრიქონში ქვესტრიქონის შესვლის

	პირველ ადგილს. ამასთან, ყურადღებას არ აქცევს რეგისტრში განსხვავებას.
stristr	იქცევა strstr-ის მსგავსად, მაგრამ არ ითვალისწინებს სიმბოლოების განსხვავებას რეგისტრში.
strlen	გამოთვლის სტრიქონის სიგრძეს.
strnatcasecmp	Strnatcmp-სგან განსხვავებით, ყურადღებას არ აქცევს სიმბოლოების რეგისტრში სხვაობას.
strncmp	ახდენს სტრიქონების პირველი n სიმბოლოს შედარებას.
strpos	ეძებს სტრიქონში ქვესტრიქონის პირველ შესვლას.
strrchr	ეძებს სტრიქონში სიმბოლოს ბოლო შესვლას.
strrev	სტრიქონის მიმართულებას ცვლის საპირისპიროზე.
strripos	ეძებს სტრიქონში ქვესტრიქონის ბოლო შესვლას რეგისტრის გათვალისწინების გარეშე.
strrpos	ეძებს სტრიქონში ქვესტრიქონის ბოლო შესვლას.
strspn	გვიბრუნებს მოცემული სიმბოლოებისაგან შედგენილი სტრიქონის მონაკვეთის სიგრძეს.
strstr	გვიბრუნებს ქვესტრიქონის შესვლის პირველი ადგილი-დან ბოლომდე სტრიქონის ნაწილს.
strtolower	დიდი ასოები გადაჰყავს სტრიქონულ რეგისტრში.
strtoupper	მოქმედებს strtolower-ის საპირისპიროდ.
strtr	ცვლის სტრიქონში მოცემულ სიმბოლოებს.
substr_compare	მოცემული პოზიციიდან ერთმანერთს ადარებს ორ სტრიქონს.
substr_count	ითვლის, თუ რამდენჯერ შედის მოცემული ქვესტრი-

	ქონი სტრიქონში.
substr_replace	სტრიქონის მოცემულ უბანში არსებულ ქვესტრიქონებს ცვლის სხვა სტრიქონით.
substr	სტრიქონიდან ამოჭრის ქვესტრიქონს.
trim	სტრიქონიდან ამოაგდებს დასაწყისსა და ბოლოში არსებულ შუალედებს.
ucfirst	სტრიქონის პირველი ასო გადაჰყავს ასომთავრულში.

ლიტერატურა:

1. http://www.w3schools.com/php/php
2. http://php720.com/
3. Максим Кузнецов. PHP5 практика разработки web-сайтов
4. http://gtu.ge/View/index.html#http://gtu.ge/book/g_RvinefaZe_PHP.pdf
5. Освой самостоятельно PHP4 за 24 часа. Мэт Зандстра, «Вильямс», 2001.
6. http://www.w3schools.com
7. http://metanit.com/web/angular/
8. http://www.intuit.ru/
9. გ. ღვინეფაძე. Javascript და მისი შესაძლებლობების განმავითარებელი თანამედროვე ტექნოლოგიები. "ტექნიკური უნივერსიტეტი". 2016 წ. ISBN 99940-14-80-3.

სარჩევი