




# Programmeerimine keeles PHP

**Andrei Porõvkin**

**Tartu Ülikool (2009)**

## 1.1 Üldinfo

Alguses oli interneti lehed omavahel seotud staatiliste html dokumentide süsteemina, aga selleks, et mingis dokumendis muutusi teha oli vaja lehti failisüsteemis käsitsi muuta. Kahjuks selline staatiline mudel ei jõua kiirelt muutuva kaasaegse maailma progressile järgi. Seega võeti kasutusele dünaamiline mudel. Dünaamilise mudeli korral ei hoita serveris staatilisi html lehte vaid neid genereeritakse selleks spetsiaalselt välja töötatud programmidega, mis serveril töötavad.

Antud kursuse jooksul tutvume klient-server arhitektuuriga, installeerime enda arvutisse veebiserveri ja php interpretaatori ning saame baasteadmisi serveripoolsest keelest PHP. Kursuse teemad on pühendatud ainult PHP keelele (väljarvatud seitsmes teema), aga see ei tähenda, et sellest piisab suure ja eduka veebilehe loomiseks. Mahuka infosüsteemi ei saa ette kujutada ilma andmebaasideta, cache-süsteemideta, mallimootoriteta, jne.

See baaskursus on mõeldud eelkõige selleks, et õpilasel tekiks huvi PHP vastu ja ta hakkaks ise edaspidi rohkem õppima ja katsetama. Kursuse viimastes peatükkides vaatleme ka MySQL admebaasiga ühendust ja PHP programmide turvalisust.

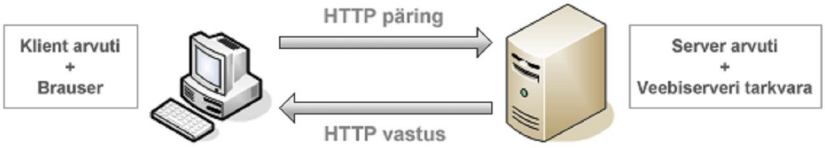
### Veebiserver

Veebiserveriks nimetatakse:

1. Arvutiprogrammi mis saab üle interneti või kohtvõrgu veebikliendilt (brauser ehk veebilehitseja) HTTP päringuid ja saadab tagasi HTTP vastuse, sisaldades peamiselt veebilehti ja faile, mis on veebilehega seotud (pildid, javascriptid, flash-objektid jne.).

2. Arvuti, kus jookseb eelpool punktis defineeritud arvutiprogramm.

Veebiserveri- ja kliendivaheline suhtlus näeb välja järgmiselt:




Antud kursuse jooksul töötame Apache veebiserveriga, sest see on lihtne, töökindel ja tasuta veebiserver ning lisaks tänapäeval ka kõige populaarsem veebiserver maailmas. Apache esimene versioon sai valmis aastal 1995. Tänaseks kasutab Apache veebiserverit umbes 50% kõigist veebisaitidest ja veebiserver on kättesaadav nii Linux, Windows ja Mac opsüsteemidele. Tarkvara lähtekood on avalik, seetõttu on Apache'i jaoks olemas palju teiste arendajate poolt loodud lisateeke.

| Tootja | Nimetus | Saitide arv | Protsent |
|---|---|---|---|
| Apache | Apache | 83 206 564 | 50.22% |
| Microsoft | IIS | 58 540 275 | 35.33% |
| Google | GWS | 10 075 991 | 6.08% |

*Netcraft veebiserverite uuring. Aprill 2008 (värskemad uuringu tulemused - 2009 a.)*

### PHP keel

PHP on serveripoolne skriptikeel, mis võimaldab genereerida dünaamilisi veebilehti. Selle nimi pärineb kõige esimesest versioonist, mida nimetati "Personal Home Page Tools". Täna nimetatakse seda kui "PHP: Hypertext Preprocessor" (hüperteksti-eeltöötleja). PHP skripti sisaldavale HTML lehele omistatakse tavaliselt laiend .php. PHP on tasuta tarkvara ja seda levitatakse Open Source litsentsi all avaliku lähtekoodina. PHP's on võimalik luua ka command-line tarkvara ning visuaalse kasutajaliidesega tarkvara, mis on aga vastuolus selle programmeerimiskeele definitsiooniga. Programmeerimiskeele ajalugu algas 1994. aastast, kui Rasmus Lerdorf tegi väga lihtsa programmi, mis mõistis spetsiaalseid makrosid. Aja jooksul programmi modifitseeriti, laiendati, integreeriti andmebaaside ja uute tehnoloogiate toega ning lisati objekt-orienteeritud kontseptsiooni jne. Tulemuseks on PHP viies versioon.

Kui vaatame programmeerimiskeelte populaarsust, siis näeme, et PHP on praegusel hetkel kolmandal kohal (eelmisel aastal aga viiendal). Arvestada tuleks ka sellega, et PHP on põhimõtteliselt serveripoolne skriptikeel. Java, C, C++ ja Visual Basic – neid kasutatakse rohkem tarkvara loomiseks aga see ei tähenda seda, et neid ei või kautatada ka veebiinfosüsteemide loomisel.

| Koht | Programmeerimiskeel | Reiting |
|---|---|---|
| 1 | Java | 18.718% |
| 2 | C | 16.891% |
| 3 | PHP | 10.390% |
| 4 | C++ | 9.911% |
| 5 | (Visual) Basic | 8.729% |
| 6 | C# | 4.433% |
| 7 | Perl | 3.776% |

*TIOBE Programming Community Index. Oktoober 2009 (vaata lähemalt)*

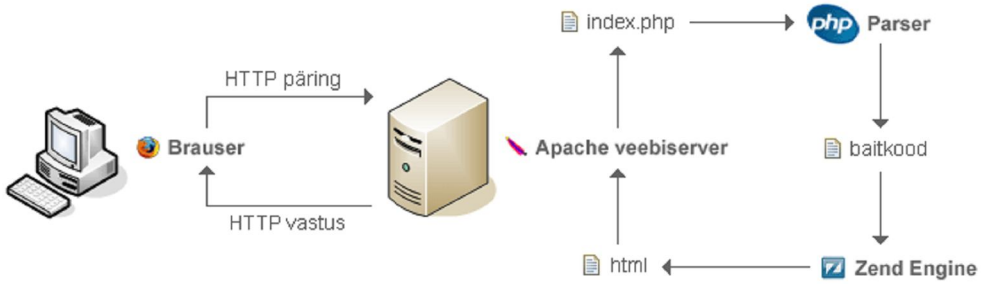
| Koht | Programmeerimiskeel | Reiting |
|---|---|---|

| | | |
|---|---|---|
| 1 | Java | 20.949% |
| 2 | C | 15.565% |
| 3 | C++ | 10.954% |
| 4 | (Visual) Basic | 9.811% |
| 5 | PHP | 8.612% |
| 6 | Python | 4.565% |
| 7 | Perl | 4.419% |

*TIOBE Programming Community Index. Oktoober 2008*

## PHP tööprintsiip

Kõigepealt moodustab kliendi brauser html päringu ja saadab selle vastavale aadressile (näiteks http://www.site.ee/index.php). Sellel aadressil asuv server võtab vastu html päringu, vaatab, et temalt nõutakse .php laiendiga faili (antud juhul index.php) ja veebiserver käivitab PHP parseri, mis loob sellest .php failist baitkoodi. Seejärel interpreteerib Zend Engine baitkoodi ja tagastab veebiserverile html koodi, mis saadetakse veebiserveri poolt omakorda kliendi brauserile http vastusena.




## PHP's tehtud

**Facebook** - http://www.facebook.com
Tuntud sotsiaalvõrk (rohkem kui 90.000.000 aktiivseid kasutajaid)

**Flickr** – http://www.flickr.com
Online piltide haldur (rohkem kui 4.000.000.000 päringuid päevas)

**Digg** – http://digg.com/
Informatsiooni vahetamise portaal (rohkem kui 26.000.000 külastajaid kuus)

**Wikipedia** – http://www.wikipedia.org/
Veebipõhine entsüklopeedia (rohkem kui 251.000.000 külastajaid kuus)

### Lisalugemist

**Veebiserveritest** (www) - inglise keeles
**Apache home page** (www) - inglise keeles
**Apache põhjalik kasutusjuhend** (www) - eesti keeles
**PHP programmeerija piibel** (www) - inglise keeles

### Video

**Programmeerimiskeeled (3:08)** (www) - inglise keeles
**Serveripoolsed skriptikeeled (6:50)** (www) - inglise keeles

---

# 1.2 Installeerimine

Klient-server arhitektuur, millest oli jutt peatükkis 1.1 nõuab vähemalt 3 elementi: kliendi arvutit, võrguühendust ning veebiserveri ja muu vajaliku tarkvaraga serveri arvutit. Arendamise etapil piisab tihti ainult ühest lokaalarvutist, millele on installeeritud brauser (klient), veebiserver (server) ja php interpretaator – selle arhitektuuriga pole vaja internetiühendust ja kogu töö võib teha oma koduarvuti taga.

Loomulikult võib installeerida kõik komponendid eraldi, aga nende ühendamine ehk konfigureerimine võtab rohkem aega ja nõuab rohkem kogemust ja teadmisi. Seega kursuse ülesannete lahendamiseks kasutame distributiivide kogumikku EasyPHP, mis sisaldab endas järgmisi komponente: Apache, PHP, MySQL ja PhpMyAdmin (visuaalne veebipõhine klient MySQL andmebaasile). EasyPHP on olemas ainult Windows operatsioonisüsteemi jaoks. Kui teie arvutis on Linux, MacOS või Solaris süsteem siis võib olla juba vaikimisi Apache veebiserver ja php installeeritud. Kui ei ole siis võite proovida installeerida XAMPP kogumikku. Kui on piisavalt vaba aega ja sihikindlust võib proovida Apache (download) ja PHP (download) eraldi iseseisvalt installeerida ja konfigureerida.

PHP programmide kirjutamiseks on vaja tekstitoimetajat. Tekstitoimetajate valik on puhtalt maitse asi, aga oleks mugavam, kui selles on süntaksi värvimise tugi olemas - siis on lihtsam koodist aru saada ja selles orienteeruda. Mina soovitaks kasutada Notepad++, see on päris võimekas toimetaja ning seda saab endale tasuta.

## EasyPHP (EasyPHP-2.0b1) installeerimine Windows opsüsteemil

Järgmine kasutusjuhend on kirjutatud EasyPHP-2.0b1 installeeriminse jaoks (aasta 2008). Praegune versioon, mida soovitan kasutada on EasyPHP-5.3.0. Selle paigaldus ei erine eelmistest versioonidest. Tegelikult võib vabalt kasutada ka EasyPHP-2.0b1 kuna selle kursuse raames versioonide erinevus pole oluline.

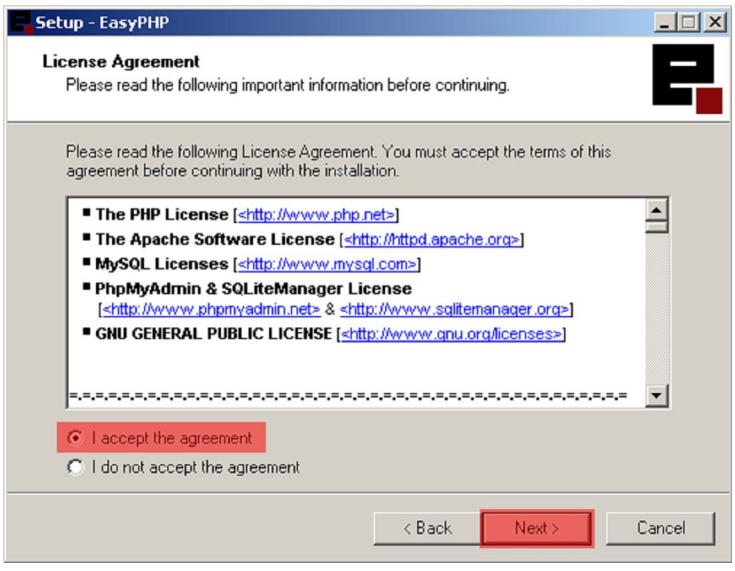
Lae alla EasyPHP installeerimis fail: EasyPHP-2.0b1-setup.exe või EasyPHP-5.3.0-setup.exe
või siis otse sourceforge lehelt: EasyPHP

Juhul kui masinas on Skype käivitatud - lülita see installeerimise ajaks välja.
Käivita allalaetud EasyPHP-xxx-setup.exe fail.

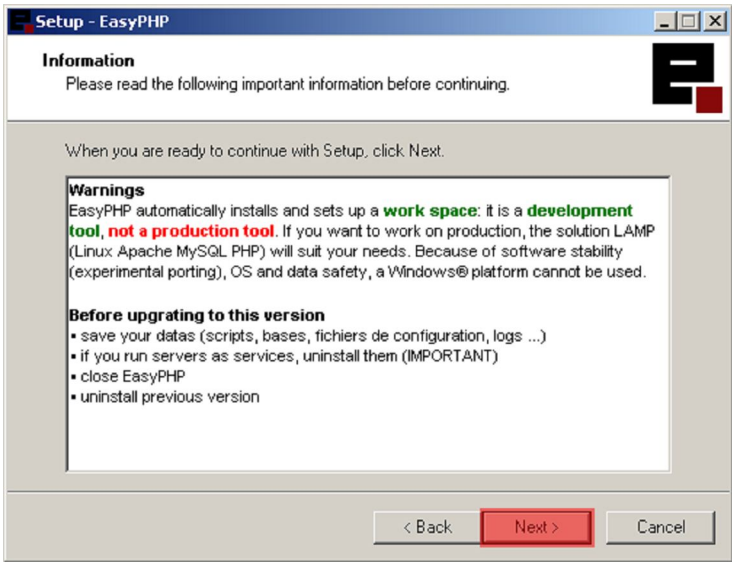
Järgige samme 1-8:

### samm 1

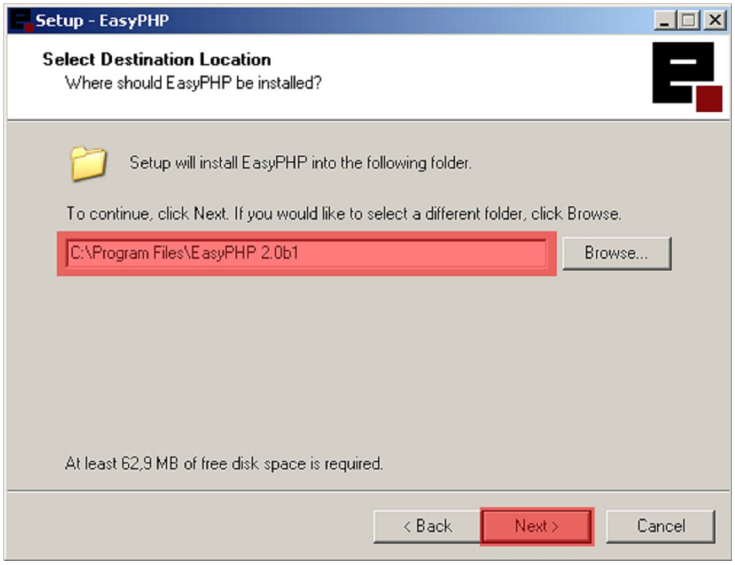

### samm 2



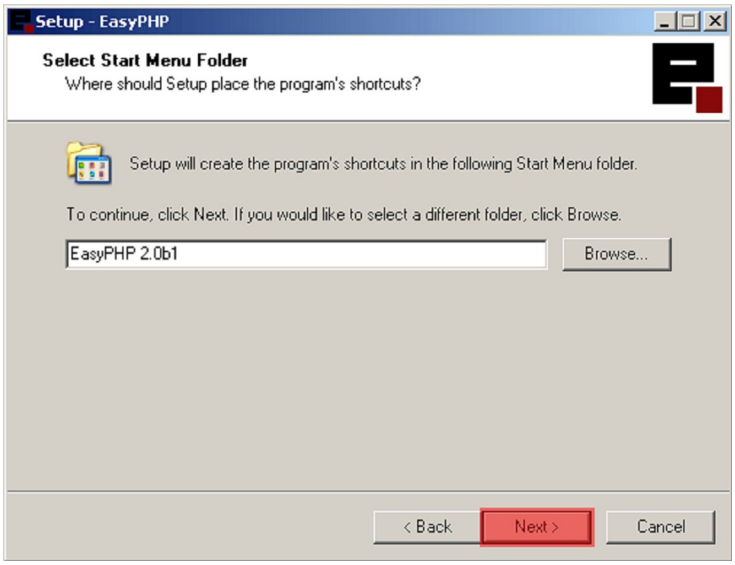

### samm 3



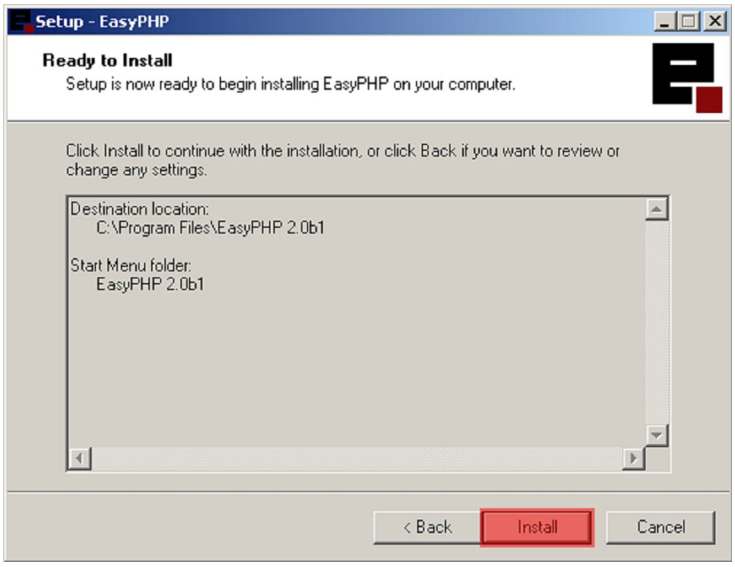

**samm 4**



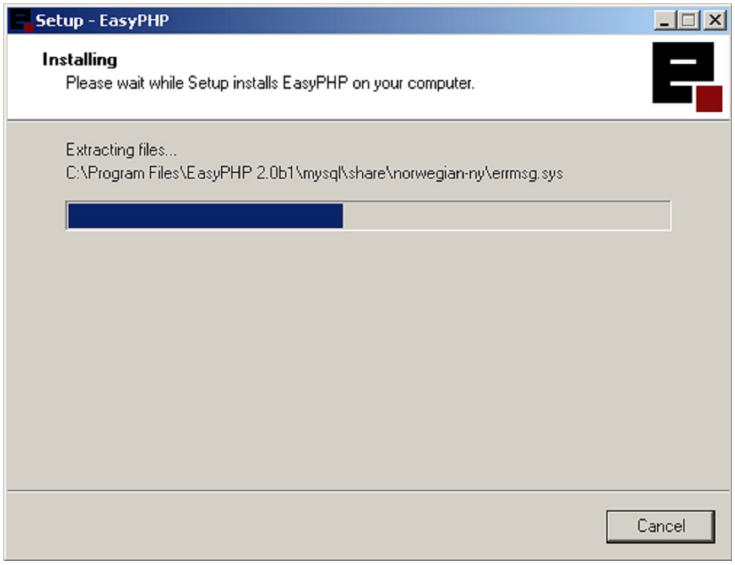

**samm 5**




**samm 6**




### samm 7




### samm 8


## EasyPHP käivitamine

Kui installeerimine on lõppenud peab taskbaris ilmuma EasyPHP ikoon.


Klikkige sellel ikoonil kaks korda, siis ilmub ekraanile EasyPHP kontrollpaneel:


Juhul kui teie arvutis on installeeritud Skype, siis võib juhtuda, et Apache veebiserveri käivitamisel tekib järgmine veateade: **Apache2 port (80) is used by "skNotifyForm" (Skype.exe)!**. Selle põhjuseks on see, et Skype reserveerib endale pordi 80 ja Apache vaikimisi port on alati 80 ning tekib konflikt. Et probleemi lahendada tuleb Skype välja lülitada, käivitada Apache ja seejärel Skype.

Kui Apache teie masinal käib, avage brauser ja pange aadressiks **http://127.0.0.1/** või **http://localhost/**. Nende aadresside tähendus on "lokaalne arvuti" (ehk 127.0.0.1 ja localhost on alati arvuti enda aadressiks).


Peab ilmuma järgmine veebileht:


Nüüd kui veebiserver (Apache), PHP ja MySQL andmebaasi server ning tekstitoimetaja on teie arvutis olemas ja töötavad, võime jätkata esimese PHP programmiga.

### Lisalugemist

**Localhost** (www) - inglise keeles

### Scribd

**Install Apache 2.2.11 on Windows** (www) - inglise keeles

---

## 1.3 Esimene programm

### Hello world!


Leidke oma arvutis EasyPHP kaust ja selles omakorda kaust www - see on meie Apache veebiserveri juurkataloog ehk DocumentRoot. Looge selles kaustas fail nimega hello_world.php. Avage see fail oma teksti redaktoris ja kirjutage sinna järgmised koodi read:

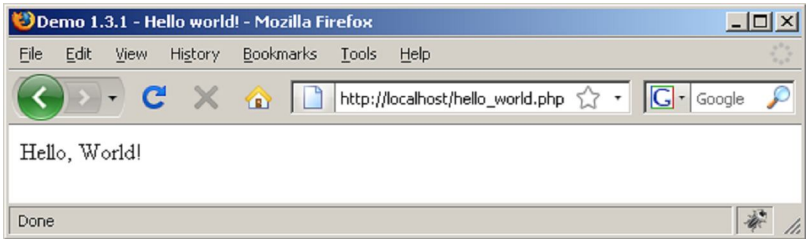
**hello_world.php**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>Demo 1.3.1 - Hello world!</title>
    <META name="author" content="Andrei Poryvkin">
</head>
<body>

    <?php
        # Kirjuta väljundisse "Hello, World!"
        echo "Hello, World!";
    ?>

</body>
</html>
```

**Näide 1.3.1**

Salvestage fail ja käivitage brauser ning aadressiks kirjutage: http://localhost/hello_world.php, tulemusena peab ekraanile ilmuma

järgmine väljund:




Juhul kui näide käivitamine ebaõnnestus, kontrollige kas Apache veebiserver käib ja hello_world.php fail on Apache veebiserveri juurkataloogis olemas (www kaust).

## PHP märgendid

Uurime näide 1.3.1 koodi lähemalt. Esimene asi, mida tuleb meelde jätta on see, kuidas PHP programmi koodi sisestatakse HTML koodi. Selleks kasutatakse PHP märgendeid ja nende vahel kirjutatakse juba PHP kood:

1. **<?php ... ?>**
2. **<? ... ?>**
3. **<script language="php"> ... </script>**
4. **<% ... %>**

Tekst nende vahel interpreteeritakse PHP koodina. Kõike, mis nende sümbolite vahele ei jää, loetakse HTML koodiks ja faili täitmisel väljastatakse selliselt nagu on. Kaks esimest varianti on kõige rohkem levinud stiil ja peamiselt kasutatakse selliseid märgendeid (selle kursuse jooksul kasutame ainult esimest varianti).

## Kommentaarid

Näide esimesel real asub kommentaar **Kirjuta valjundisse "Hello, World!"**. Kommentaare kasutatakse selleks, et teised inimesed saaksid koodist aru (eriti vajavad kommenteerimist teie poolt välja mõeldud algoritmid ja keeruline loogika). Mõnikord juhtub ka nii, et 2-3 kuu pärast ei oska ka koodi autor seletada kuidas tema programm töötab ja temal läheb suhteliselt palju aega selleks, et seda meelde tuletada. Aga kui kommentaarid on olemas piisab selleks ~5 minutist. Kommentaarid peavad kirjeldama loodava koodi eesmärke, kasutatud muutujaid, funktsioone ja algoritme. PHP parser ei loe teksti, mis asub kommentaaride vahele. Ta lihtsalt ignoreerib seda. PHP programmeerimiskeeles on 2 tüüpi kommentaare:

1. **/ voi # - üherealiste märkuste jaoks**
2. **/* ... */ - mitmerealised kommentaarid**

**comments.php**

```php
<?php

$a = 5; // muutuja $a lühikirjeldus

$b = 8; // muutuja $b lühikirjeldus

# kaivitame funktsiooni sum()
# ja kirjutame tulemuse väljundisse
echo $a.' + '.$b.' = '.sum($a, $b);

/*
Funktsioon summa leidmiseks. Sisendiks on
2 argumenti ja tulemuseks on nende summa
*/
function sum ($x, $y) {
    return $x + $y;
}

?>
```

**Näide 1.3.2**

## Eraldajad

PHP programmid on põhimõtteliselt käskude kogumid. Käskude eraldamiseks kasutatakse programmeerimiskeeltes spetsiaalseid sümboleid - eraldajaid. PHP's seda tehakse semikooloniga:

**separators.php**

```php
<?php

// käsk: defineeri muutuja $a ja selle väärtuseks pane 1
$a = 1; // käsu lõpp - semikoolon

// käsk: defineeri muutuja $b ja selle väärtuseks pane 3
$b = 3; // käsu lõpp - semikoolon

// käsk: kirjuta valjundisse muutuja $a väärtus
echo 'a = '.$a; // käsu lõpp - semikoolon

// käsk: kirjuta väljundisse reavahe
echo '<br />'; // käsu lõpp - semikoolon

// käsk: kirjuta väljundisse muutuja $b väärtus
```

```php
echo 'b = '.$b; // käsu lõpp - semikoolon

?>
```

Näide 1.3.3

## Echo

Järgmine on sõna **echo**. See on keelekonstruktsioon, mida kasutatakse sõnede kirjutamiseks väljundisse (meie juhul html faili). Sõned peavad olema kas jutumärkides (" ") või ülakomades (' '). Kui sõne on jutumärkides, siis esimene asi mis PHP parser selliste sõnedega teeb - püüab leida nendes muutujaid ja asendada neid nende väärtustega. Kuna ülakomades sõnedega niimoodi ei tehta, töödeldakse neid kiiremini. Echo käsu jaoks peab olema defineeritud vähemalt üks parameeter, aga neid võib olla ka rohkem - siis neid tuleb eraldada komadega. Testimiseks käivitage järgmine fail:

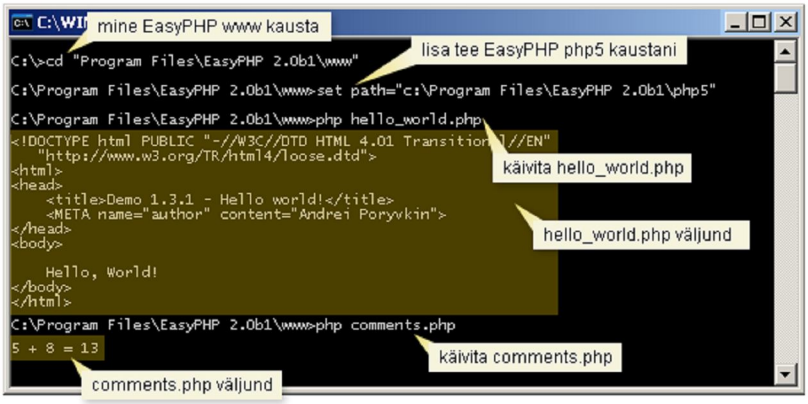
echo.php

```php
<?php

$a = 5;

// ülakomades
echo 'a väärtus on: $a <br />';

// jutumärkides
echo "a väärtus on: $a";

echo '<br /><br />';

// mitu parameetrit
echo 'Hello', ' ', ' World! <br />';
echo 'a väärtus on: ', $a;

echo '<br /><br />';

// sõnade ühendamine (konkatenatsioon)
echo 'a väärtus on: '.$a;

?>
```

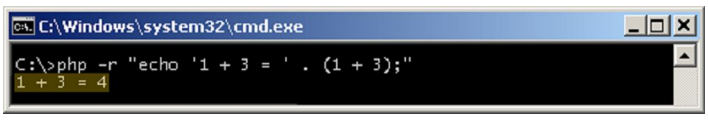
Näide 1.3.4

## PHP käsurealt

Mõnikord võib tekkida vajadus PHP koodi käivitada masinas, kus veebiserverit pole. Siis PHP koodi võib käivitada ka käsurealt (ilma veebiserverita ja brauserita). Selleks ava Windows opsüsteemis käsurea paneel (fail C:\Windows\system32\cmd.exe). Paneeli võib käivitada ka järgmiselt: Start → Run → cmd. Käsurea paneelis mine EasyPHP www kausta (käsk: cd), siis lisa tee php5 kaustani (käsk: set path). Nüüd võid käivitada php programme lihtsalt kirjutades: **php sinu_programm.php**:




Lipp **-r** võimaldab kirjutada koodi otse käsureal:




### Lisalugemist

**Echo** (www) - inglise keeles
**PHP Märgendid** (www) - inglise keeles
**Eraldajad** (www) - inglise keeles
**Kommentaarid** (www) - inglise keeles
**PHP käsurealt** (www) - inglise keeles

### Video

## 2.1 Muutujad

Koodi täitmise ajal paiknevad kõik koodis kasutatavad väärused operatiivmälus. Muutujad on lihtsalt nimelised "mälupesad", millesse võib salvestada mingeid väärtusi ja muutuja nime kaudu viidatakse väärtusele operatiivmälus. Kõik PHP muutujad algavad **$**-märgiga, millele järgneb muutuja nimi. Muutuja nimi algab alati tähe või allkriipsuga, millele järgneb suvaline arv tähti, numbreid ja allkriipse. Muutuja nimed on suur- ja väiketähetundlikud, mis tähendab, et muutuja $name ja $Name on erinevad muutujad. Muutujal on oma andmetüüp, mis näitab, mis liiki väärtusi muutuja esindab. PHP toetab jargmisi andmetüüpe: neli skalaar- ja kaks kompleksandmetüüpi. Lisaks on olemas ka kaks spetsiaalset tüüpi.

### Skalaarsed tüübid

#### Tõeväärtustüüp - boolean

Võib omandada väärtusi kas true (tõene) või false (väär).

**Näide**

```
<?php

     // x on tõene
     $x = true;

     // y on väär
     $y = !$x;

     // z on väär
     $z = $x && $y;

?>
```

#### Täisarvutüüp - integer

Täisarv vahemikus -2147483647 ... 2147483647. Vaikimisi kasutatakse kümnendsüsteemi, kaheksandsüsteemi puhul arv peab algama nulliga ning kuueteistkümnendsüsteemi puhul **0x**-ga.

**Näide**

```
<?php

     // kuueteistkümnendarv 15
     $x = 0xF;

     // kaheksandarv 10
     $y = 012;

     //kümnendarv 25
     $z = 25;

     // väljundiks on kümnendarv 50
     echo $x+$y+$z;

?>
```

#### Ujukomatüüp - double

Reaalarv vahemikus -1.8 x $10^{308}$ ... 1.8 x $10^{308}$. Neid võib esitada tavalises vormingus: **täisosa.murdosa**, alternatiivina võib suuri või väga väikseid arve esitada eksponentsiaalkujul (ujukomaarvuna).

**Näide**

```
<?php

     // tavaline vorming
     $x = 5.8;

     // eksponentsiaalkuju
     $y = 3.45e-16;
     $z = 2.1e13;

?>
```

#### Tekstitüüp - string

Sõned peavad olema kas jutumärkides (" ") või ülakomades (' '). Kui sõne on jutumärkides, siis esimene asi mis PHP parser selliste sõnedega teeb - püüab leida nendes muutujaid ja asendada neid nende väärtustega. Kuna ülakomades sõnedega niimoodi ei tehta, töödeldakse neid kiiremini. Tahtes aga ülakomades teksti sisse samuti ülakomasid kirjutada, tuleb nende ette panna langjoon ****. Samuti kui soovid jutumärkides teksti sisse paigutada jutumärkid, tuleb nende ette panna langjoon ****.

**Näide**

```
<?php
```

```php
    // it's my life
    $x = 'it\'s my life';

    // PHP
    $php = 'PHP';

    // it's "PHP"
    $y = "it's \"$php\"";

?>
```

Kui ei soovi, et pärast muutuja väärtust tuleks tühik või muu sümbol, mille järgi intepretaator saab otsustada, kus lõpeb muutuja nimi ning kust läheb edasi tavaline tekst, sellisel juhul tuleb muutuja nimi panna loogeliste sulgude sisse.

**Näide**

```php
<?php

    $text = 'use';

    // väljund: it was useless
    echo "it was {$text}less...";

?>
```

## Komplekstüübid

### Massiiv - array

Massiiv on andmestruktuur, mis kujutavad ennast elementide hulga. Teisi sõnu, massiivid on omapärased konteinerid, mis võivad hoida samaaegselt mitu väärtust. Vaatleme neid eraldi peatükis 4.

**Näide**

```php
<?php

    // massiiv
    $arr = array(9.4, 0, 'text', true);

    // väljund: 9.4
    echo $arr[0];

    echo '<br />';

    // väljund: text
    echo $arr[2];

?>
```

### Objekt - object

Programmeerimise mõistes objekt on teatud hulk muutujaid, välju koos teatud hulga meetoditega, mis opereerivad muutujate väärtustega. Muutujaid ja meetodeid käsitletakse ühe tervikuna. Analoogias reaalse eluga võib muutujaid mõista näiteks kui mingi eseme kirjeldust ja meetodeid kui tegevusi, mida selle esemega on võimalik teha. Objekte vaatleme lähemalt peatükis 5.

**Näide**

```php
<?php

    // objekt
    $obj = new Test(5, 3);

    // väljund: 8
    echo $obj->sum();

    class Test {

        public $a;
        public $b;

        function __construct ($x, $y) {
            $this->a = $x;
            $this->b = $y;
        }

        function sum() {
            return $this->a + $this->b;
        }

    }

?>
```

## Spetsiaalsed tüübid

### Ressurss - resource

See on mingi ressurss, mida PHP töötleb eri viisil. Näiteks muutuja, mis hoiab avatud faili või andmebaasiga ühenduse deskriptorit. Selle andmetüübiga tutvume peatükis 7.

Näide

```php
<?php

    // avatud faili deskriptor
    $handler = fopen('test.txt', 'w');
    fclose($handler);

?>
```

### NULL

Muutuja on NULL tüüpi siis, kui ei ole määratud mingi väärtus. NULL tüüpi muutujal on üks võimalik väärtus ja see on NULL. NULL tüüpi mutuja tekib siis, kui:

- muutuja väärtuseks oli määratud NULL
- muutuja oli defineeritud aga tal pole veel väärtust
- muutuja oli unset()

Näide

```php
<?php

    $i;
    echo gettype($i); // väljund: NULL
    echo '<br />';

    $j = NULL;
    echo gettype($j); // väljund: NULL
    echo '<br />';

    $k = 5;
    unset($k);
    echo gettype($k); // väljund: NULL
?>
```

## Muutujate funktsioonid

### var_export()

Tulemuseks on muutuja tekstiline kuju. Esimene parameeter on muutuja ise, teine argument on tõeväärtus, mis ütleb kas muutuja string'i on vaja kohe väljundisse saata või string'i tagastada. Vaikimisi teine parameeter on väär (false).

Näide

```php
<?php

    $x = 'text';
    var_export($x); // väljund: 'text'

    $y = !true;
    $varString = var_export($y, true);
    echo $varString; // väljund: false

?>
```

### is_bool(), is_int(), is_double(), is_string(), is_array(), is_object(), is_resource(), is_null()

is_N() tagastab TRUE väärtuse, kui muutuja on N tüüpi

Näide

```php
<?php

    var_export(is_bool(5)); // väljund: false
    var_export(is_int(15)); // väljund: true
    var_export(is_double(3)); // väljund: false
    var_export(is_string('text')); // väljund: true

    $x;
    var_export(is_null($x)); // valjund: true

    $y = NULL;
    var_export(!is_null($y)); // valjund: false

    $z = 1234;
    var_export(is_null($z)); // valjund: false

?>
```

### gettype()

Tagastab muutuja väärtuse tüübi

Näide

```php
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
     <title>Variables</title>
     <style type="text/css">
          body {
               padding: 15px;
               color: black;
               font-family: Arial;
               font-size: 14px;
          }
          .table {
               background: #696969;
          }
          .table td {
               background: white;
               padding: 5px 10px;
               font-size: 12px;
               font-weight: nornal;
               text-align: left;
          }
          .table th {
               width: 120px;
               background: #A5A5A5;
               color: white;
               padding: 5px 10px;
               font-size: 12px;
               font-weight: bold;
               text-align: left;
          }
     </style>
</head>
<body>
     <table class="table" cellspacing="1" cellpadding="0" border="0">
     <tr>
          <th style="width:60px">Variable</th>
          <th style="width:80px">Type</th>
          <th style="width:300px">Value</th>
     </tr>

     <?php

          ################
          #   BOOLEAN    #
          ################

          $a = false;
          echo '<tr>';
          echo '<td>$a</b></td> ';
          echo '<td>'.gettype($a).'</td> ';
          echo '<td>'.var_export($a, true).'</td>';
          echo '</tr>';

          ################
          #   INTEGER    #
          ################

          $b = 4;
          echo '<tr>';
          echo '<td>$b</b></td> ';
          echo '<td>'.gettype($b).'</td> ';
          echo '<td>'.var_export($b, true).'</td>';
          echo '</tr>';

          ################
          #   DOUBLE     #
          ################

          $c = 4.1;
          echo '<tr>';
          echo '<td>$c</b></td> ';
          echo '<td>'.gettype($c).'</td> ';
          echo '<td>'.var_export($c, true).'</td>';
          echo '</tr>';

          ################
          #   STRING     #
          ################

          $d = "Hello World!";
          echo '<tr>';
          echo '<td>$d</b></td> ';
          echo '<td>'.gettype($d).'</td> ';
          echo '<td>'.var_export($d, true).'</td>';
          echo '</tr>';

          ################
          #   ARRAY      #
          ################

          $e = array(-9, true, 'word', 0, -4.2);
          echo '<tr>';
          echo '<td>$e</b></td> ';
          echo '<td>'.gettype($e).'</td> ';
          echo '<td>'.var_export($e, true).'</td>';
          echo '</tr>';

          ################
          #   OBJECT     #
          ################

          $f = new testClass();
          echo '<tr>';
```

```
          echo '<td>$f</b></td> ';
          echo '<td>'.gettype($f).'</td> ';
          echo '<td>'.var_export($f, true).'</td>';
          echo '</tr>';

          ################
          #   RESOURCE   #
          ################

          $g = fopen('text.txt', 'w');
          echo '<tr>';
          echo '<td>$g</b></td> ';
          echo '<td>'.gettype($g).'</td> ';
          echo '<td></td>';
          echo '</tr>';

          ################
          #   NULL       #
          ################

          $h = NULL;
          echo '<tr>';
          echo '<td>$h</b></td> ';
          echo '<td>'.gettype($h).'</td> ';
          echo '<td>'.var_export($h, true).'</td>';
          echo '</tr>';


          class testClass {
               public $var;
               function __construct() {
                    $this->var = 234;
               }
          }

     ?>

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

Väljundiks on järgmine tabel:

| Variable | Type | Value |
|---|---|---|
| $a | boolean | false |
| $b | integer | 4 |
| $c | double | 4.1 |
| $d | string | 'Hello World!' |
| $e | array | array ( 0 => -9, 1 => true, 2 => 'word', 3 => 0, 4 => -4.2, ) |
| $f | object | testClass::__set_state(array( 'var' => 234, )) |
| $g | resource | |
| $h | NULL | NULL |

## isset()

Tagastab TRUE väärtuse, kui muutuja on deklareeritud

**Näide**

```
<?php

     $x;
     var_export(isset($x)); // väljund: false

     $x = 5;
     var_export(isset($x)); // väljund: true

?>
```

## empty()

Tagastab TRUE väärtuse, kui muutuja on deklareerimata või omab tühi-väärtust (NULL, 0, tühi string)

**Näide**

```
<?php

     $x;
     var_export(empty($x)); // väljund: true

     $x = 0;
     var_export(empty($x)); // väljund: true

     $x = '';
     var_export(empty($x)); // väljund: true

     $x = 12;
```

```
    var_export(empty($x)); // väljund: false
?>
```

---

## 2.2 Konstandid

Nagu enamuses programmeerimiskeeltes PHP-s on võimalik kasutada konstante. Konstante on mõtet kasutada muutuja asemel juhul kui tegu on muutujaga mille väärtus ei muutu kogu programmi jooksul. Konstantide kasutamise eelised on:

1. konstantide väärtused on konstantsed - defineerides konstandi ja andes talle mingi väärtuse see ei muutu kunagi
2. Konstandid on globaalsed - kord defineeritud konstante saab kasutada üle terve programmi

Konstandi defineerimisel kasutatakse direktiivi **define**('konstandi_nimi', konstandi_väärtus). Konstanti võib kasutada nagu muutujat, kuid tema ette ei tule $ märki. Konstandi kontrollimiseks võib kasutada funktsiooni **defined**('konstandi_nimi'), mille väärtus on TRUE, kui konstant on defineeritud, vastasel juhul väärtuseks on FALSE. Heaks kombeks on kirjutada konstantide nimed suurte tähtedega.

**Näide**

```
<?php

    define('AUTHOR_NAME', 'John');
    define('ENTRIES_PER_PAGE', 10);

    echo 'Author name: '.AUTHOR_NAME.'<br />';
    echo 'Entries per page: '.ENTRIES_PER_PAGE.'<br />';

    echo 'AUTHOR_NAME defined: '.var_export(defined('AUTHOR_NAME'), true).'<br />';
    echo 'AUTHOR_SURNAME defined: '.var_export(defined('AUTHOR_SURNAME'), true).'<br />';
    echo 'ENTRIES_PER_PAGE defined: '.var_export(defined('ENTRIES_PER_PAGE'), true).'<br />';

    /*

    Väljund:

    Author name: John
    Entries per page: 10
    AUTHOR_NAME defined: true
    AUTHOR_SURNAME defined: false
    ENTRIES_PER_PAGE defined: true

    */

    define('AUTHOR_NAME', 'Bob');
    echo 'Author name: '.AUTHOR_NAME.'<br />';
    // Author name: John

?>
```

PHP-s on olemas ka sisseehitatud konstandid:

- **__FILE__** - hetkel parsitava faili nimi
- **PHP_VERSION** - php versioon
- **PHP_OS** - operatsioonisüsteem
- jne...

### Lisalugemist

**Maagilised konstandid** (www) - inglise keeles
**Sisseehitatud konstandid** (www) - inglise keeles

---

## 2.3 Tehted

### Tehted tõeväärtustega

Põhilised boolean tüübile rakendatavad operatsioonid on järgmised:

- **!** - eitus (loogiline **ei**)
- **&&** - konjunktsioon (loogiline **jah**)
- **||** - disjunktsioon (loogiline **või**)

Tehete prioriteedid (kahanevalt): eitus, konjunktsioon, disjunktsioon.

Tehete tulemusi võib esitatada järgmise tõeväärtustabeliga:

| $a | $b | !$a | $a && $b | $a \|\| $b |
|---|---|---|---|---|
| true | true | false | true | true |

| | | | | |
|---|---|---|---|---|
| true | false | false | false | true |
| false | true | true | false | true |
| false | false | true | false | false |

Loogilistele avaldistele rakendavatel tehetel on järgmised omadused:

- **kommutatiivsus**
  $a && $b == $b && $a
  $a || $b == $b || $a
- **assotsiatiivsus**
  $a && ($b && $c) == ($a && $b) && $c
  $a || ($b || $c) == ($a || $b) || $c
- **distributiivsus**
  $a && ($b || $c) == ($a && $b) || ($a && $c)
  $a || ($b && $c) == ($a || $b) && ($a || $c)
- **de Morgani reegel**
  !($a && $b) == !$a || !$b
  !($a || $b) == !$a && !$b

**Näide**

```php
<?php

    $x = true;
    $y = false;
    $z = !$x;

    var_export(($x && !$y) || $z && !$z); // true
    var_export(!!$x); // true
    var_export($x && $y && $z); // false

?>
```

## Aritmeetilised tehted

- **+** summeerimine
- **-** lahutamine
- ***** korrutamine
- **/** jagamine
- **%** jäägiga jagamine
- **++** inkrement
- **--** dekrement

Tehete tulemusi võib esitatada järgmise tabeliga:

| $a | $b | $a+$b | $a-$b | $a*$b | $a/$b | $a%$b | $a++ | $a-- |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 8 | 2 | 15 | 1.66... | 2 | 6 | 4 |
| -5 | 2 | -3 | -7 | -10 | -2.5 | -1 | -4 | -6 |

**Näide**

```php
<?php

    $x = 5;
    $y = 3.2;
    $z = 10;

    echo ($x + 3) / 2 + $z; // 14
    echo ($z % $x) + $y; // 3.2
    echo -$x + $z * $y / 4; // 3

?>
```

Inkremendi ja dekremendi põhimõte on järgmine:
$a++; on sama mis $a = $a + 1 või $a += 1;
$a--; on sama mis $a = $a - 1; või $a -= 1;

## Ühendatud määramisoperaatorid

Ühendatud määramisoperaatorid koosnevad tavaliselt ühest aritmeetilisest operaatorist, millele järgneb võrdusmärk. Nende operaatorite eesmärk on kiirendada koodi kirjutamist ja muuta kood loetavamaks. Ühendatud määramisoperaatorid on välja toodud allolevas tabelis.

| Operaator | Näide | On sama kui |
|---|---|---|
| += | $a += 2 | $a = $a + 2 |

| | | |
|---|---|---|
| -= | $a -= 2 | $a = $a - 2 |
| *= | $a *= 2 | $a = $a * 2 |
| /= | $a /= 2 | $a = $a / 2 |
| %= | $a %= 2 | $a = $a % 2 |
| .= | $a .= 'text' | $a = $a . 'text' |

### Võrdlemine

- **==** võrdsed (väärtused on võrdsed)
- **===** ekvivalentsed (tüübid ja väärtused on võrdsed)
- **!=** mittevõrdsed
- **!==** mitteekvivalentsed
- **>** suurem
- **>=** suurem või võrdne
- **<** väiksem
- **<=** väiksem või võrdne

**Näide**

```php
<?php

    $w = 2;
    $x = 5;
    $y = '5';
    $z = true;

    var_export($x == $y); // true
    var_export($x === $y); // false
    var_export($x == $z); // true
    var_export($w >= $x); // false
    var_export($w < $x); // true

?>
```

---

## 2.4 Funktsioonid

Ilma funktsioonideta oleks programmeerimine mõtetu. Oleme juba kasutanud tavalisi PHP funktsioone nagu gettype(), is_bool(), var_export() jne. Aga keerulisemateks ülesanneteks jääb nendest väheseks. Tihti sooritatavaid operatsioone on kasulik vormistada iseseisva alamprogrammidena ehk funktsioonidena. Siin on abiks kasutajafunktsioonid - programmeerija poolt kirjutatud funktsioonid.

Kui funktsiion on valmis, siis saab seda kasutada ka muude andmete puhul. Näiteks ruutvõrrandi lahendamisel tuleb iga kord diskriminandi leida. Selle arvutamiseks vajaminevad tegevused võime koondada funktsioonisse. Lisaks saame teha ka funktsioon lahendite leidmiseks.

Ruutvõrrandiks nimetatame võrrandit, mis on esitatud kujul: $\mathbf{ax^2 + bx + c = 0}$

Diskiminant on siis: $\mathbf{D = b^2 - 4ac}$

Ruutvõrrandi lahendamiseks kasutame valemit:

$\mathbf{x_1 = (-b + sqrt(D)) / 2a}$
$\mathbf{x_2 = (-b - sqrt(D)) / 2a}$

kus **sqrt(D)** tähendus on **ruutjuur diskriminandist**

**quadratic_equation.php**

```php
<?php

    $a = 2;
    $b = 7;
    $c = -22;

    solveQuadraticEquation($a, $b, $c);

    $a = -1;
    $b = 0;
    $c = 16;

    solveQuadraticEquation($a, $b, $c);

    // funktsioon lahendite leidmiseks ja väljastamiseks
    function solveQuadraticEquation ($a, $b, $c) {

        // leiame D funktsiooni getDiscriminant abil
        $discriminant = getDiscriminant($a, $b, $c);

        // lahendite leidmine (a != 0)
        $x1 = (-$b + sqrt($discriminant)) / (2 * $a);
        $x2 = (-$b - sqrt($discriminant)) / (2 * $a);

        // väljund
```

```
        echo $a.'x<sup>2</sup> + '.$b.'x + '.$c.' = 0<br />';
        echo 'D = '.$discriminant.'<br />';
        echo 'x<sub>1</sub> = '.$x1.'<br />';
        echo 'x<sub>2</sub> = '.$x2.'<br /><br />';

    }

    // funktsioon diskriminandi leidmiseks
    function getDiscriminant ($a, $b, $c) {
        $D = $b * $b - 4 * $a * $c;
        return $D;
    }

?>
```

**Näide 2.4.1**

Uurime näide 2.4.1 koodi lähemalt.

***function* getDiscriminant ($a, $b, $c)**

***function*** tähendab seda, et tegemist on funktsiooniga.

**getDiscriminant** on funktsiooni nimi (funktsiooni nimi algab alati tähe või allkriipsuga, millele järgneb suvaline arv tähti, numbreid ja allkriipse). Nimi ei tohi kattuda PHP funktsioonide nimedega. Funktsiooni nimed ei ole suur- ja väiketahetundlikud, mis tähendab, et funktsioonid getName() ja GETnAme() on üks ja sama funktsioon.

**$a, $b, $c** on funktsiooni parameetrid, neid kasutatakse selleks et funktsiooni käivitamisel edastada funktsioonile andmeid. Funktsiooni välja kutsudes peavad olema kindlasti antud parameetrid $a, $b ja $c.

Funktsiooni sees asub tavaline PHP kood

**return $D** on naasmisdirektiiv mida kasutatkse väärtuste tagastamiseks funktsioonist (naasmisdirektiiv võib ka puududa - siis tihti nimetatakse sellise funktsiooni protseduuriks).

Funktsiooni väljakutsumine on lihtne ja see ei erine tavaliste PHP funktsioonide väljakutsumisest. Kui kutsume välja funktsiooni ja soovime selle väärtust muutujas salvestada, siis näeb see välja nii:

**$var = funktsiooni_nimi ($param1, $param2);**

Kui kutsume välja protseduuri:

**funktsiooni_nimi ($param1, $param2);**

## Muutujad funktsioonis

Näites edastasime argumentidena kolm muutujat: $a, $b ja $c. Argumentide nimed on kasutusel ainult funktsiooni sees ja funktsiooni väljakutsudes me ei pea kasutama samu nimesid. Võib teha ka nii:

**Näide**

```
<?php

    $x = 2;
    $y = 7;
    $z = -22;

    $discriminant = getDiscriminant($x, $y, $z);

    function getDiscriminant ($a, $b, $c) {
        $D = $b * $b - 4 * $a * $c;
        return $D;
    }

?>
```

Eelmises näides kõik argumendid olid kohustuslikud. Vajadusel PHP-s võib argumente defineerida ka mittekohustuslikeks (juhul, kui vabatahtlik argument puudub, asendatakse see vaikimisi määratud väärtusega, antud juhul siis viiega):

**Näide**

```
<?php

    example(3, 7);
    /*
    1 argument: 3
    2 argument: 7
    */

    example(3);
    /*
    argument: 3
    argument: 5
    */

    function example ($arg1, $arg2 = 5) {
        echo '1 argument: '.$arg1.'<br />';
        echo '2 argument: '.$arg2.'<br />';
    }

?>
```

Funktsiooni argumendid ja funktsiooni sees defineeritud muutujad kehtivad ainult selle funktsiooni sees ning ainult väljakutsumise ajal. Järgmine kord kui kutsume välja funktiooni defineeritakse need muutujad uuesti. See tähendab, et funktsiooni sees olevad muutujad on lokaalsed. Kui soovime funktsiooni sees kasutada muutujaid mis olid defineeritud väljaspool funktsiooni siis tuleb enne kasutada **global** direktiivi:

**Näide**

```
<?php

      $pi = 3.14;

      $area = getCircleArea(5);
      echo $area;

      function getCircleArea ($radius) {
            global $pi;
            return $pi * pow($radius, 2);
      }

?>
```

## Rekursiivne funktsioon

Rekursiivne funktsioon on selline funktsioon, mis kutsub iseennast välja. Rekursiivne funktsiion on hea siis, kui on vaja sooritada korduvaid tegevusi. Hea näide on kaustade puu joonistamine, faktoriaali või suurima ühisteguri leidmine:

**factorial.php**

```
<?php

      echo fact(10);

      function fact($n) {
            if ($n < 0) {
                  return -1;
            }
            if ($n == 0) {
                  return 1;
            }
            return($n * fact($n-1));
      }


?>
```

**Näide 2.4.2**

**greatest_common_divisor.php**

```
<?php

      echo gcd(2520, 3150);

      function gcd ($m, $n) {
            if ($m == 0) {
                  return $n;
            }
            if ($n == 0) {
                  return $m;
            }
            return (gcd($n, $m % $n));
      }

?>
```

**Näide 2.4.3**

### Lisalugemist

**Rekursiivsed funktsioonid** (www) - inglise keeles
**PHP matemaatilised funktsioonid** (www) - inglise keeles
**PHP sõnede funktsioonid** (www) - inglise keeles

### Video

**Funktsioonid** (osa1, osa1, osa1) - inglise keeles

### Scribd

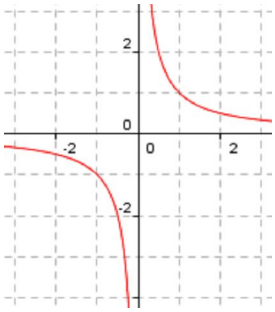
**PHP Funktsioonid** (www) - inglise keeles

---

# 3.1 If-else

Kõik programmid, mis meie oleme vaadelnud olid hästi lihtsad selles mõttes, et käske käivitatakse

algusest lõpuni. Keerukamate programmide kirjutamisel on vaja käske valikuliselt täita või siis isegi täitmata jätta. Näiteks kui meie program arvutab funktsiooni ***f(x) = 1/x*** väärtusi, siis juhul kui x on null – program peab ütlema, et 0 korral funktsioon pole määratud. Programmi loogika peab siis välja nägema järgmiselt:




**Kui x on võrdne nulliga, siis**
**anna veateade**
**vastasel korral**
**arvuta 1/x ja väljasta tulemus**

Sellise loogika implementeerimiseks PHP-s on olemas tingimuslaused (mõnikord neid nimetatakse valikulauseteks) ja nende üldkuju on:

```
if (tingimus) {
   plokk1
} else {
   plokk2
}
```

## Tingimus

Tingimus, mida kasutatakse if-else tingimuslauses on loogiline avaldis (lihtsam näide on võrdlemine, aga vajadusel võib kasutada ka avaldisi mis on ühendatud && või || abil). Programmi täitmisel kõigepealt kontrollitakse tingimuse kehtivust (kehtivus tähendab seda, et loogilise avaldise lõppväärtus on tõene (true)). Juhul kui tingimus kehtib - täidetakse esimene plokk (2 plokki kuuluv kood jääb täitmata), vastasel juhul - teine plokk (1 plokki kuuluv kood jääb täitmata).

**Näide**

```php
<?php

$x1 = 5;
$x2 = 0;
$x3 = -2;

function calculateFunction ($arg) {
     if ($arg == 0) {
          return 'undefined';
     } else {
          return round(1/$arg, 2);
     }
}

?>

<table cellspacing="3" cellpadding="5" border="1">
     <tr>
          <th style="background-color:gray; color:white">x</th>
          <th style="background-color:gray; color:white">f(x)</th>
     </tr>
     <tr>
          <td><?php echo $x1; ?></td>
          <td><?php echo calculateFunction($x1); ?></td>
     </tr>
     <tr>
          <td><?php echo $x2; ?></td>
          <td><?php echo calculateFunction($x2); ?></td>
     </tr>
     <tr>
          <td><?php echo $x3; ?></td>
          <td><?php echo calculateFunction($x3); ?></td>
     </tr>
</table>
```

Tulemuseks on järgmine tabel:

| x | f(x) |
|---|---|
| 5 | 0.2 |
| 0 | undefined |
| -2 | -0.5 |

Juhul kui **else** pole oluline, siis see võib ka puududa:

**Näide**

```php
<?php

$x1 = 5;
$x2 = 0;
$x3 = -2;

function calculateFunction ($arg) {
     if ($arg == 0) {
          return 'undefined';
     } else {
          return round(1/$arg, 2);
     }
}

?>
```

```
<table cellspacing="3" cellpadding="5" border="1">
     <tr>
          <th style="background-color:gray; color:white">x</th>
          <th style="background-color:gray; color:white">f(x)</th>
     </tr>

     <?php

          $result = calculateFunction($x1);
          if (is_numeric($result)) {
               echo "<tr><td>$x1</td><td>$result</td></tr>";
          }
          $result = calculateFunction($x2);
          if (is_numeric($result)) {
               echo "<tr><td>$x2</td><td>$result</td></tr>";
          }
          $result = calculateFunction($x3);
          if (is_numeric($result)) {
               echo "<tr><td>$x3</td><td>$result</td></tr>";
          }

     ?>

</table>
```

Väljund:

| x | f(x) |
|---|---|
| 5 | 0.2 |
| -2 | -0.5 |

Keerulisema tingimuslause hea näide on liigaasta kontroll. Aasta on liigaasta juhul kui aastaarv jagub 4-ga ja pole nii, et aastaarv jagub 100-ga ning ei jagu 400-ga. Seda tingimust võib kirjale panna järgmiselt:

**$aasta % 4 == 0 && !($aasta % 100 == 0 && !($aasta % 400 == 0))**

Võime seda ka lihtsustada kasutades mittevõrdumist:

**$aasta % 4 == 0 && ($aasta % 100 != 0 || $aasta % 400 == 0)**

**Näide**

```
<?php

$aasta = 2009;

if ($aasta % 4 == 0 && ($aasta % 100 != 0 || $aasta % 400 == 0)) {
     echo $aasta.' on liigaasta';
} else {
     echo $aasta.' pole liigaasta';
}

?>
```

Tingimus võib eelnevalt panna ka muutujasse:

**Näide**

```
<?php

$aasta = 2009;
$liigaasta = $aasta % 4 == 0 && ($aasta % 100 != 0 || $aasta % 400 == 0);

if ($liigaasta) {
     echo $aasta.' on liigaasta';
} else {
     echo $aasta.' pole liigaasta';
}

?>
```

## If-else erikujud

Kui plokis on ainult üks käsk, siis loogilisi sulge võib ka ära jätta:

**Näide**

```
<?php

$aasta = date('Y');
$liigaasta = $aasta % 4 == 0 && ($aasta % 100 != 0 || $aasta % 400 == 0);

if ($liigaasta) echo $aasta.' on liigaasta';
else echo $aasta.' pole liigaasta';

?>
```

Lisaks PHP-s on olemas kompaktne if-else tingimuslause kuju:

**tingimus ? true_plokk : false_plokk;**

ja see on absoluutselt sama mis:

**if (tingimus) {**
  **true_plokk**
**} else {**
  **false_plokk**
**}**

**Näide**

```
<?php

$aasta = date('Y') - rand(1, 100);
$liigaasta = $aasta % 4 == 0 && ($aasta % 100 != 0 || $aasta % 400 == 0);

echo $aasta.' '.($liigaasta ? 'on' : 'pole').' liigaasta';

?>
```

## Tingimus tingimuse sees

Mõnikord tuleb kasuks see asjaolu, et tingimuslause on käsk nagu iga teinegi ja seega seda võib kasutada ka teise tingimuslause sees. Kirjutame liigaasta probleemi lahendus üle:

**Näide**

```
<?php

$aasta = 1996;

if ($aasta % 4 == 0) {
     if ($aasta % 100 != 0 || $aasta % 400 == 0) {
          echo $aasta.' on liigaasta';
     } else {
          echo $aasta.' pole liigaasta';
     }
} else {
     echo $aasta.' pole liigaasta';
}

?>
```

## If-elseif-else konstruktsioon

Juhul, kui programmis on vaja mitu tingimust kontrollida, siis PHP-s on olemas **if-elseif-else** tingimuslause konstruktsioon (else võib ka siin puududa):

**if (tingimus1) {**
  **plokk1**
**} elseif (tingimus2) {**
  **plokk2**
**} elseif (tingimus3) {**
  **plokk3**
**} else {**
  **plokk4**
**}**

Kui tingimus1 on väär, siis minnakse järgmise juurde (tingimus2) kui mingi tingimus kehtib siis täidetakse vastav koodi plokk ja järmisi tingimusi arvuti ei kontrolli. Kui kõik tingimused on väärad ja **else** osa on olemas, siis täidetakse seda. **Elseif** võib kirjutada nii palju kui vaja on. Näiteks mängu tulemus on esitatud mingi arvuna (-1 - kaotus, 0 - viik, 1 - võit) ja meil on vaja väljastada teade mängu seisu kohta, siis if-elseif-else abil see on imelihtne:

**Näide**

```
<?php

$result = rand(-1, 1);

echo "Numeric result: $result<br>";

if ($result == -1) {
     echo 'You lost';
} elseif ($result == 0) {
     echo 'Draw';
} elseif ($result == 1) {
     echo 'You win';
} else {
     echo 'Error';
}

?>
```

### Lisalugemist

## 3.2 Switch

PHP-s on olemas veel üks tingimuslause mille abil võib realiseerida hargnemist. See on switch ja case käskudest moodustatud switch-tingimuslause. Selleks, et tulevikus vigu vältida on väga tähtis aru saada kuidas arvuti töödeldab switch lause.

Esialgul jäetakse meelde avaldis (vajadusel arvutatakse seda ka) ja sõltuvalt sellest siirdub täitmisjärg ühte või teise harusse - neid tähistavad avaldise väärtused koos case sõnaga. Lisaks konstruktsiooni lõppu võib lisada ka sõna **default** - seda täidetakse siis, kui üheski case avaldise väärtusevarianti ei leidunud (sarnane else'iga). Switch-tingimuslause üldine struktuur on järgmine:

```
switch (avaldis) {
   case variant1 :
      käsud;
      break;
   case variant2 :
      käsud;
      break;
   ...
   ...
   case variantN :
      käsud;
      break;
   default:
      käsud;
}
```

Iga variandi sisu lõpetab käsk break, mis rangelt võttes pole küll kohustuslik, vaid pigem ülimalt soovitatav. Näiteks, eelmises osas (3.1) if-elseif-else tingimuslause abil kirjutatud programmi võiks kirjutada kasutades ka switch tingimuslause:

**Näide**

```php
<?php

$result = rand(-1, 2);

echo "Numeric result: $result<br>";

switch ($result) {
     case -1:
          echo 'You lost';
          break;
     case 0:
          echo 'Draw';
          break;
     case 1:
          echo 'You win';
          break;
     default:
          echo 'Error';
}

?>
```

Kui me ei kasuta break käsu, leiaks switch küll õige koha, kuid ei lõpetaks tööd ja täidaks ka kõik endast allapoole jäävad tingimused. Kui on vaja, et mitme variandi puhul täidetakse samu lauseid, siis tuleb kirjutada vastavad case-märgendid üksteise järele. Järgmine programm väljastab hinde (0-5) sõnalisel kujul:

**Näide**

```php
<?php

$hinne = 3;

switch ($hinne) {
     case 5:
          echo 'Hinne: vaga hea';
          break;
     case 4:
          echo 'Hinne: hea';
          break;
     case 3:
          echo 'Hinne: rahuldav';
          break;
     case 2:
     case 1:
     case 0:
          echo 'Hinne: mitterahuldav';
          break;
     default:
          echo $hinne.' ei ole hinne';
}

?>
```

Case avaldise väärtusevariandid võivad olla mitte ainult arvulised vaid ka ka string tüüpi väärtused:

**Näide**

```php
<?php

$monthName = date('F');
$maxTemp = getTemperature($monthName, 'max');
$minTemp = getTemperature($monthName, 'min');

echo 'Today is <b>'.$monthName.'</b><br />';
echo 'Max temperature in '.$monthName.' is: '.$maxTemp.'<br />';
echo 'Min temperature in '.$monthName.' is: '.$minTemp.'<br />';
echo 'Avg temperature in '.$monthName.' is: '.getTemperature($monthName).'<br />';

function getTemperature ($month, $type = 'avg') {

     $result = 0;

     switch ($month) {

          case 'December':
          case 'January':
          case 'February':
               if ($type == 'max') $result = 12.7;
               elseif ($type == 'min') $result = -38.1;
               else {
                    $temperature = 12.7 + (-38.1);
                    $result = round($temperature / 2, 2);
               }
               break;

          case 'March':
          case 'April':
          case 'May':
               switch ($type) {
                    case 'max':
                         $result = 22.9;
                         break;
                    case 'min':
                         $result = -14.2;
                         break;
                    default:
                         $result = round((-14.2 + 22.9) / 2);
               }
               break;

          case 'September':
          case 'October':
          case 'November':
               $result = $type == 'max' ? 18.4 : ($type == 'min' ? -11.3 : 3.6);
               break;

          default:
               $result = 'no data';

     }

     return $result;

}

?>
```

## 4. Tsüklid

Programmeerimise põhiline eesmärk on anda korduvad tegevused teha arvutile. Arvutist oleks aga hulga vähem kasu, kui puuduks võimalus panna teda mingit ühesugust tegevust kordama mitu korda järjest. Eelnevates peatükkides vaadeldud programmidel on selline omadus, et iga käsku täidetakse täpselt üks kord (või ei täideta üldse kui tegemist on mõne if- või switch-lause haruga). Sellised programmid jääksid hätta, kui oleks näiteks vaja leida Fibonacci jada liige kohal 14.

| $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ | $F_{11}$ | $F_{12}$ | $F_{13}$ | $F_{14}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 | 233 | 377 |

Loomulikult võime kirjutada koodi, kus iga jada liige jaoks on arvutamise käik eraldi kirja pandud:

**Näide**

```php
<?php

$F0 = 0;
$F1 = 1;
$F2 = $F0 + $F1;
$F3 = $F1 + $F2;
$F4 = $F2 + $F3;
$F5 = $F3 + $F4;
$F6 = $F4 + $F5;
$F7 = $F5 + $F6;
$F8 = $F6 + $F7;
$F9 = $F7 + $F8;
$F10 = $F8 + $F9;
$F11 = $F9 + $F10;
$F12 = $F10 + $F11;
$F13 = $F11 + $F12;
$F14 = $F12 + $F13;

echo 'F<sub>14</sub> = '.$F14;
```

```
?>
```

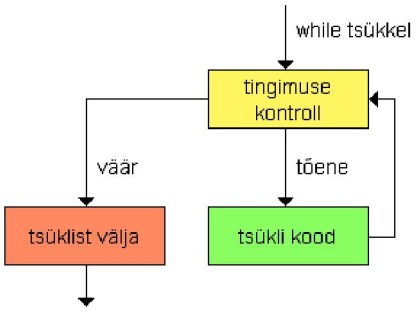
Aga nüüd selleks, et meie programm oskaks Fibonacci jada liige kohal 23 leida - peame koodi veel juurde kirjutama. Selline lähenemine pole eriti hea ja mugav. Seega oleks tore kui saaksime kasutada käske stiilis "täida seda programmilõiku 23 korda". Niisuguse võimaluse annavadki meile tsüklid.

## While - eelkontrolliga tsükkel

Eelkontrolliga tsükli ehk while-tsükli üldkuju on:

```
while (tingimus) {
   käsud;
}
```

Eelkontrolliga tsükli puhul kontrollitakse tingimus iga kord enne tsükli koodi täitmist, kas seal olevaid käske üldse on vaja täita. Pärast ploki läbimist minnakse taas ploki algusesse vaatama, kas tegevust oleks vaja korrata (kas tingimuse tõeväärtus on tõene (true)). Graafiliselt seda võib kujundada järgmiselt:




Kirjutame nüüd getFibonacciNumberAt($n) funktsiooni:

**Näide**

```
<?php

echo 'F<sub>9</sub> = '.getFibonacciNumberAt(9).'<br />';
echo 'F<sub>14</sub> = '.getFibonacciNumberAt(14).'<br />';
echo 'F<sub>17</sub> = '.getFibonacciNumberAt(17).'<br />';
echo 'F<sub>23</sub> = '.getFibonacciNumberAt(23).'<br />';
echo 'F<sub>52</sub> = '.getFibonacciNumberAt(52).'<br />';


/* Funktsioon tagastab Fibonacci jada liige kohal $n */

function getFibonacciNumberAt ($n) {

     $currentIndex = 1; // liige indeks

     $Fn_1 = 0; // eelmise liige väärtus
     $Fn = 1; // liige väärtus kohal $currentIndex



     while ($currentIndex < $n) {
          $tempVar = $Fn;
          $Fn = $Fn + $Fn_1;
          $Fn_1 = $tempVar;
          $currentIndex++; // indeks = indeks + 1
     }

     return $Fn;

}

?>
```

Järgmine näide väljastab Fibonacci jada **n** esimest elementi:

**Näide**

```
<?php

$n = 40;

$currentIndex = 1;

$Fn_1 = 0;
$Fn = 1;

echo $Fn_1.', '.$Fn;

while ($currentIndex < $n) {
     $tempVar = $Fn;
     $Fn = $Fn + $Fn_1;
     $Fn_1 = $tempVar;
     $currentIndex++;
     echo ', '.$Fn;
}

?>
```

On võimalik, et tsüklitingimus osutub kohe esimesel kontrollimisel vääraks, nagu näiteks järgmises programmilõigus. Sellisel juhul ei täideta tsükli sisu ühtegi korda.

**Näide**

```
<?php

      $i = 1;
      while ($i < 0) {
            echo $i*$i;
            $i++;
      }

?>
```

## Break ja continue

Juhul kui on vaja mingil hetkel kohe tsükli lõpetada (näiteks tulemus on käes ja pole mõtet edasi otsida) võib kasutada **break**'i. Juhul kui meie ülesanne on kotrollida kas esimese 50 Fibonacci jada elementide hulgas on vähemalt üks mis jagub 19-ga:

**Näide**

```
<?php

$n = 50;

$currentIndex = 1;

$Fn_1 = 0;
$Fn = 1;

while ($currentIndex < $n) {

      $tempVar = $Fn;
      $Fn = $Fn + $Fn_1;

      if ($Fn % 19 == 0) break;

      $Fn_1 = $tempVar;
      $currentIndex++;
}

echo 'F<sub>'.$currentIndex.'</sub> = '.$Fn.'<br>';
echo 'F<sub>'.$currentIndex.'</sub> % 19 = 0';

?>
```

Ühe tsükli läbimist nimetatakse **iteratsiooniks**. Näiteks järgmise programmi puhul arvuti käivitab tsükli koodi kolm korda (while tsükkel teeb kolm iteratsiooni):

**Näide**

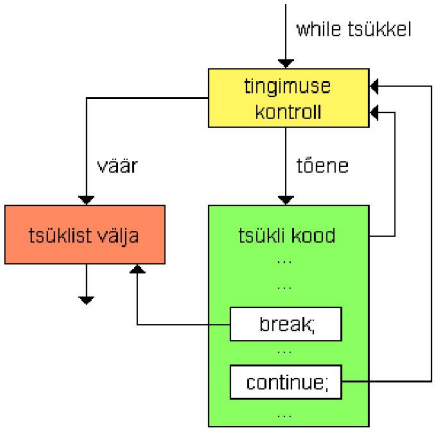
```
<?php

      $a = 0;

      while ($a < 3) {
            $a++;
            echo '<br>itertsioon nr. '.$a;
      }

?>
```

Juhul, kui mingil hetkel me soovime kohe jätkata järgmise iteratsiooniga - **continue** käsk on meile abiks. Näiteks meil on vaja väljastada kõik arvud vahemikus 0...100, mis jaguvad kolmega:

**Näide**

```
<?php

      $a = 0;

      while ($a <= 100) {
            if ($a % 3 != 0) {
                  $a++;
                  continue;
            }
            echo $a.' ';
            $a++;
      }

?>
```

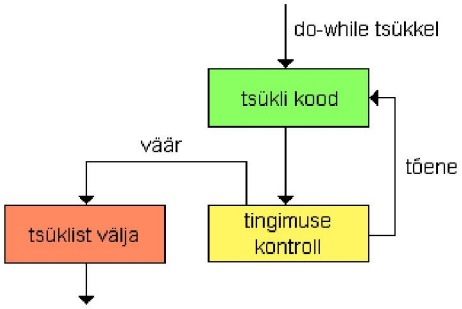
Visuaalselt võib neid käske kujundada nii:




## Do-while - järelkontrolliga tsükkel

Tsükli täitmine algab sisu koodi täitmisest, misjärel kontrollitakse tingimuse kehtivust. Nagu eelkontrolliga tsükli puhulgi, kui tingimus kehtib, siis läheb täitmisjärg uuesti tsükli sisu algusesse, vastasel juhul - väljutakse tsüklist ja võetakse ette tsüklile järgnev programmi käsk:

**do {**
**käsud;**
**} while (***tingimus***);**




**Näide**

```
<?php

      $a = 0;

      do {
            if ($a % 3 == 0) echo $a.' ';
            $a++;
      } while ($a <= 100);

?>
```

Järelkontrolliga tsüklis täidetakse tsükli sisu alati vähemalt üks kord:

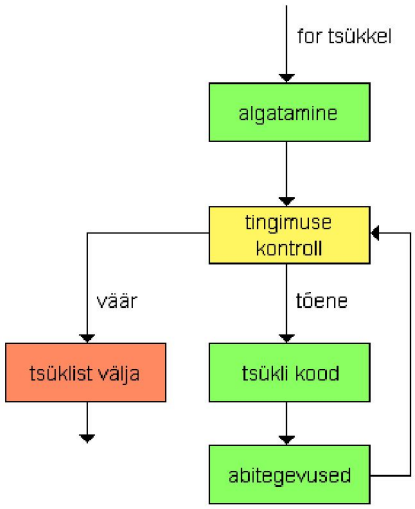
**Näide**

```
<?php

      $i = 1;
      do {
            echo $i*$i;
            $i++;
      } while ($i < 0);

?>
```

## For - kolmikpäisega tsükkel

PHP-s on olemas veel ühte tüüpi tsükkel, kolmikpäisega ehk for-tsükkel, mis küll on samaväärne teatava while-tsükliga, nii et sellest lisandusest keelele uusi väljendusvõimalusi juurde ei tule. Kuid for-tsükli struktuur sobib mõnede programmide koostamiseks eriti hästi ning teda on lihtsam konstrueerida kui vastavat while-tsüklit. For tsükli üldkuju on järgmine:

**for (***algatamine* **;** *tingimus* **;** *abitegevused***) {**
**käsud;**
**}**




ja see on samaväärne eelkontrolliga while-tsükliga:

```
algatamine;
while (tingimus) {
   käsud;
   abitegevused;
}
```

Näiteks selle koodi asemel:

**Näide**

```
<?php

      $a = 0;

      while ($a <= 100) {
            if ($a % 3 != 0) {
                  $a++;
                  continue;
            }
            echo $a.' ';
            $a++;
      }

?>
```

on palju mugavam kirjutada:

**Näide**

```
<?php

      for ($a = 0; $a <= 100; $a++) {
            if ($a % 3 != 0) continue;
            echo $a.' ';
      }

?>
```

Korrutustabel kahe for tsükli abil:

**Näide**

```
<?php

      echo '<table cellspacing="3" cellpadding="5" border="1">';

      for ($i = 0; $i <= 10; $i++) {
            echo '<tr><th>'.($i > 0 ? $i : ' ').'</th>';
            for ($j = 1; $j <= 10; $j++) {
                  if ($i == 0) echo "<th>$j</th>";
                  else echo '<td>'.($j*$i).'</td>';
            }
            echo '</tr>';
      }

      echo '</table>';

?>
```

Tulemus:

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| **2** | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **3** | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| **4** | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| **5** | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| **6** | 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| **7** | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| **8** | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| **9** | 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| **10** | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

### Lisalugemist

**Fibonacci jada** (www) - inglise keeles

---

## 5.1 Massiivid

Üks olulisemaid andmestruktuure programmeerimises on massiivid ehk järjendid. Massiiv on indekseeritud elementide hulk, kus iga elemendil ehk väärtusel on oma unikaalne indeks ehk võti. Võtmeteks sobivad täisarvud ja sõned, väärtusteks - ükskõik mis tüüpi andmed (lihttüübid, massiivid, objektid jne.). PHP-s on olemas tavalised massiivid (võtmed on täisarvud) ja assotsiatiivsed massiivid (võtmed on sõned). Elementide arvu massiivis pole vaja määrata selle loomise ajal (paljudes programmeerimiskeeltes massiivi elementide arv peab olema rangelt defineeritud massiivi loomisel). PHP hoolitseb ise massiivi poolt kasutatava mälu eest ning muudab massiivi suurust dünaamiliselt ning massiivi maksimaalne elementide arv on piiratud ainult masina operatiivmäluga.

### Massiivi loomine ja elementide lisamine

Esimene võimalus on kasutada võtmesõna **array**: `$massivi_nimi = array()` - see kood loob massiivi `$massivi_nimi`, mis on esialgul tühi (ei sisalda ühtegi elementi). Selleks, et lisada elemente massiivi lõppu, võib panna nurksulud massiivi mutuja järgi ja sulgudes elemendi võti, siis võrdusmärk ja elemendi väärtus: `$massiivi_nimi[võti] = element`. Kui võti sulgudes puudub - kasutatakse järgmist vaba indeksit. Pange tähele, et indekseerimine algab nullist!

**Näide 5.1.1**

```php
<?php

    // massiivi $a loomine
    $a = array();

    $a[]  = 'element_1'; // elemendi lisamine (indeks on 0)
    $a[]  = 'element_2'; // elemendi lisamine (indeks on 1)
    $a[4] = 'element_3'; // elemendi lisamine (indeks on 4)
    $a[]  = 'element_4'; // elemendi lisamine (indeks on 5)

    echo '$a size = '.count($a).'<br>';
    echo '<pre>';
    print_r($a);
    echo '</pre>';

?>
```

**Väljund**

```
$a size = 4

Array
(
    [0] => element_1
    [1] => element_2
    [4] => element_3
    [5] => element_4
)
```

Sama tulemust võime saavutada ainult võtmesõna **array** abil kui kasutame seda kujul
`array(võti => väärtus, võti => väärtus, ..., võti => väärtus)`
Kui jätame ära elemendi võti siis asi toimib nagu [] korral:

**Näide 5.1.2**

```php
<?php

    // massiivi $a loomine ja väärtustamine võtmesõna array abil
    $a = array('element_1', 'element_2', 4 => 'element_3', 'element_4');

    echo '$a size = '.count($a).'<br>';
    echo '<pre>';
```

```
    print_r($a);
    echo '</pre>';

?>
```

**Väljund**

```
    $a size = 4

    Array
    (
        [0] => element_1
        [1] => element_2
        [4] => element_3
        [5] => element_4
    )
```

## Assotsiatiivsed massiivid

Assotsiatiivse massiivi saame teha sarnaselt eelmises punktis vaadeldule. Kasutame võtmesõna **array** tühja massiivi moodustamiseks ja pärast lisame elemente:

**Näide 5.1.3**

```
<?php

    // massiivi $personalInfo loomine
    $personalInfo = array();

    $personalInfo['name'] = 'Andrei'; // elemendi lisamine (võti on 'name')
    $personalInfo['gender'] = 'male'; // elemendi lisamine (võti on 'gender')
    $personalInfo['age'] = 25; // elemendi lisamine (võti on 'age')
    $personalInfo['student'] = true; // elemendi lisamine (võti on 'student')

    echo '<pre>';
    print_r($personalInfo);
    echo '</pre>';

?>
```

**Väljund**

```
    Array
    (
        [name] => Andrei
        [gender] => male
        [age] => 25
        [student] => 1
    )
```

Või siis kohe määrame väärtusi massiivi loomisel:

**Näide 5.1.4**

```
<?php

    // massiivi $personalInfo loomine ja väärtustamine võtmesõna array abil
    $personalInfo = array(
                        'name' => 'Andrei',
                        'gender' => 'male',
                        'age' => 25,
                        'student' => true
                    );

    echo '<pre>';
    print_r($personalInfo);
    echo '</pre>';

?>
```

**Väljund**

```
    Array
    (
        [name] => Andrei
        [gender] => male
        [age] => 25
        [student] => 1
    )
```

## Elementide kättesaamine

Massiivi elemendi poole pööratakse tema võtme järgi. Kui teame mis on elemendi võti, siis väärtust saame kätte järgmisel viisil:
`$massiivi_nimi[võti]`

**Näide 5.1.5**

```
<?php

    $numbers = array('one', 'two', 'three', 'four', 'five', 'six');
    echo $numbers[1].' + '.$numbers[2].' = '.$numbers[4];
```

```
?>
```

**Väljund**

```
two + three = five
```

**Näide 5.1.6**

```
<?php

    $personalInfo = array();

    $personalInfo['name'] = 'Andrei';
    $personalInfo['gender'] = 'male';
    $personalInfo['age'] = 25;
    $personalInfo['student'] = true;

    echo $personalInfo['name'].' is '.$personalInfo['age'].' years old.<br>';
    echo $personalInfo['gender'] == 'male' ? 'He' : 'She';
    echo ($personalInfo['student'] ? ' is' : ' is not').' a student.';

?>
```

**Väljund**

```
Andrei is 25 years old.
He is a student.
```

## Massiivi läbimine

Tavalise massiivi puhul (võtmed on täisarvud) kui indeksid (võtmed) on **0...n** - siis võime kasutada **for** ja **while** tsükleid. Sellisel juhul tuleb kasutusele **count()** funktsioon, mille abil saame teada massiivi elementide arvu. Lihtsalt käime kõik indeksid läbi. Alustame esimesest elemendist, mille indeks on 0 ja lõpetame indeksiga mis on võrdne massiivi elementide arvuga miinus üks.

For:

**Näide 5.1.7**

```
<?php

    $numbers = array('one', 'two', 'three', 'four', 'five', 'six');

    for ($i=0; $i<count($numbers); $i++) {
        echo '$i = '.$i.'; $numbers[$i] = $numbers['.$i.'] = '.$numbers[$i].'<br>';
    }

?>
```

**Väljund**

```
$i = 0; $numbers[$i] = $numbers[0] = one
$i = 1; $numbers[$i] = $numbers[1] = two
$i = 2; $numbers[$i] = $numbers[2] = three
$i = 3; $numbers[$i] = $numbers[3] = four
$i = 4; $numbers[$i] = $numbers[4] = five
$i = 5; $numbers[$i] = $numbers[5] = six
```

While:

**Näide 5.1.8**

```
<?php

    $numbers = array('one', 'two', 'three', 'four', 'five', 'six');

    $i = 0;

    while ($i < count($numbers)) {
        echo '$i = '.$i.'; $numbers[$i] = $numbers['.$i.'] = '.$numbers[$i].'<br>';
        $i++;
    }

?>
```

**Väljund**

```
$i = 0; $numbers[$i] = $numbers[0] = one
$i = 1; $numbers[$i] = $numbers[1] = two
$i = 2; $numbers[$i] = $numbers[2] = three
$i = 3; $numbers[$i] = $numbers[3] = four
$i = 4; $numbers[$i] = $numbers[4] = five
$i = 5; $numbers[$i] = $numbers[5] = six
```

Aga for ja while tsüklidega läbimisel peame 100% kindel olema, et massiivi esimese elemendi indeks on 0 ja iga järgmine on eelmisest ühe võrra suurem. Juhul kui see nii ei ole, selline algoritm ei sobi:

**Näide 5.1.9**

```
<?php

    $numbers = array('one', 'two', 6 => 'three', 'four', 'five', 'six');

    echo '<pre>';
```

```
    print_r($numbers);
    echo '</pre>';

    for ($i=0; $i<count($numbers); $i++) {
        echo '$i = '.$i.'; $numbers[$i] = $numbers['.$i.'] = '.$numbers[$i].'<br>';
    }

?>
```

**Väljund**

```
    Array
    (
        [0] => one
        [1] => two
        [6] => three
        [7] => four
        [8] => five
        [9] => six
    )

    $i = 0; $numbers[$i] = $numbers[0] = one
    $i = 1; $numbers[$i] = $numbers[1] = two

    Notice: Undefined offset: 2 in C:\Program Files\EasyPHP 2.0b1\www\tst.php on line 10
    $i = 2; $numbers[$i] = $numbers[2] =

    Notice: Undefined offset: 3 in C:\Program Files\EasyPHP 2.0b1\www\tst.php on line 10
    $i = 3; $numbers[$i] = $numbers[3] =

    Notice: Undefined offset: 4 in C:\Program Files\EasyPHP 2.0b1\www\tst.php on line 10
    $i = 4; $numbers[$i] = $numbers[4] =

    Notice: Undefined offset: 5 in C:\Program Files\EasyPHP 2.0b1\www\tst.php on line 10
    $i = 5; $numbers[$i] = $numbers[5] =
```

Assotsiatiivse massiivi ja massiivi, kus elementide indeksid ei vasta üleval vaadeldud reeglile (0...n) - kasutame massiivi läbimiseks **foreach** tsüklit, mille üldine kuju on:

```
foreach ($massiivi_nimi as $muutuja_elemendi_võtme_jaoks => $muutuja_elemendi_väärtuse_jaoks) {
   käsud;
}
```

Foreach tsükkel käib kõik massiivi elemendid läbi ja igas iteratsioonis paneb $muutuja_elemendi_võtme_jaoks väärtuseks elemendi võti ja $muutuja_elemendi_väärtuse_jaoks elemendi väärtus:

**Näide 5.1.10**

```
<?php

    $personalInfo = array(
                        'name' => 'Andrei',
                        'gender' => 'male',
                        'age' => 25,
                        'student' => true
                    );

    foreach ($personalInfo as $key => $value) {
       echo $key.': '.$value.'<br>';
    }

?>
```

**Väljund**

```
    name: Andrei
    gender: male
    age: 25
    student: 1
```

Kui massiivi läbimisel võtme väärtus pole oluline, võib seda ka ära jätta:

**Näide 5.1.11**

```
<?php

    $numbers = array('one', 'two', 6 => 'three', 'four', 'five', 'six');

    foreach ($numbers as $value) {
       echo $value.' ';
    }

?>
```

**Väljund**

```
    one two three four five six
```

Nagu ka teistes tsüklites võib kasutada **break** ja **continue**:

**Näide 5.1.12**

```
<?php

    $numbers = array('one', 'two', 6 => 'three', 'four', 'five', 'six');
```

```
    foreach ($numbers as $key => $value) {
       if ($key == 1 || $key == 6) continue;
       if ($key == 8) break;
       echo $value.'<br>';
    }

?>
```

**Väljund**

```
one
four
```

## Elementide kustutamine massiivist

Selleks, et massiivi elemendi kustutada, peame kindlasti teadma mis on elemendi võti. Elemente kustutakse võtme järgi käsu **unset** abil:

**Näide 5.1.13**

```
<?php

    $numbers = array('one', 'two', 'three', 'four', 'five', 'six');

    echo '<pre>';
    print_r($numbers);
    echo '</pre>';

    echo 'deleting element with index 1<br>';
    unset($numbers[1]);

    echo 'deleting element with index 4<br>';
    unset($numbers[4]);

    echo '<pre>';
    print_r($numbers);
    echo '</pre>';

?>
```

**Väljund**

```
Array
(
     [0] => one
     [1] => two
     [2] => three
     [3] => four
     [4] => five
     [5] => six
)

deleting element with index 1
deleting element with index 4

Array
(
     [0] => one
     [2] => three
     [3] => four
     [5] => six
)
```

Pange tähele, et elementide võtmed jäid samaks ja nad on nüüd 0, 2, 3 ja 5. Selleks, et need oleksid kujul 0...n võib kasutada **array_values(**$massiiv**)** funktsiooni mille argumendiks on meie massiiv. Antud funktsioon tagastab uue massiivi kus on samad elemendid kui argumendina antud massiivis, aga tagastatava massiivi võtmed on kujul 0...n:

**Näide 5.1.14**

```
<?php

    $numbers = array('one', 'two', 'three', 'four', 'five', 'six');

    echo '<pre>';
    print_r($numbers);
    echo '</pre>';

    unset($numbers[1]);
    unset($numbers[4]);

    echo '<pre>';
    print_r($numbers);
    echo '</pre>';

    $numbers = array_values($numbers);

    echo '<pre>';
    print_r($numbers);
    echo '</pre>';

?>
```

**Väljund**

```
Array
(
     [0] => one
     [1] => two
```

```
        [2] => three
        [3] => four
        [4] => five
        [5] => six
    )

    Array
    (
        [0] => one
        [2] => three
        [3] => four
        [5] => six
    )

    Array
    (
        [0] => one
        [1] => three
        [2] => four
        [3] => six
    )
```

## Mitmemõõtmelised massiivid

Mitmemõõtmeliseks massiiviks nimetatakse massiivi, mille elemendid on ka massiivid. Peamiselt kasutatakse neid andmete loogiliseks struktureerimiseks. Hea näide on inimeste andmete hoidmine:

**Näide 5.1.15**

```php
<?php

    $humanDatabase = array(
                            array(
                                'id' => '38702128612',
                                'data' => array(
                                                'name' => 'John Peterson',
                                                'age' => 38,
                                                'gender' => 'male',
                                                'phone' => array(
                                                            'home' => '6314981',
                                                            'mobile' => '57123912'
                                                            )
                                                ),
                                ),
                            array(
                                'id' => '48112010907',
                                'data' => array(
                                                'name' => 'Jane Anderson',
                                                'age' => 29,
                                                'gender' => 'female',
                                                'phone' => array(
                                                            'home' => null,
                                                            'mobile' => '50929101'
                                                            )
                                                ),
                                ),
                            array(
                                'id' => '34908119132',
                                'data' => array(
                                                'name' => 'Mike Jameson',
                                                'age' => 50,
                                                'gender' => 'male',
                                                'phone' => array(
                                                            'home' => '7430921',
                                                            'mobile' => '58829120'
                                                            )
                                                ),
                                ),

                        );



    showEveryone();


    function showEveryone() {

        global $humanDatabase;

        // creating table header

        echo '<table cellspacing="5" cellpadding="5" border="1">';
        echo '<tr>';
        echo '<th>ID</th>';
        echo '<th>Name</th>';
        echo '<th>Gender</th>';
        echo '<th>Age</th>';
        echo '<th>Contacts</th>';
        echo '</tr>';

        // filling human indormation

        foreach ($humanDatabase as $humanInfo) {
            echo '<tr>';
            echo '<td>'.$humanInfo['id'].'</td>';
            echo '<td>'.$humanInfo['data']['name'].'</td>';
            echo '<td align="center">';
            echo strtoupper(substr($humanInfo['data']['gender'], 0, 1));
            echo '</td>';
```

```
                echo '<td align="center">'.$humanInfo['data']['age'].'</td>';
                echo '<td>';
                foreach ($humanInfo['data']['phone'] as $phoneType => $phoneNumber) {
                    if (is_null($phoneNumber)) continue;
                    echo $phoneType.': '.$phoneNumber.'<br>';
                }
                echo '</td>';
                echo '</tr>';
            }

            echo '</table>';
        }
?>
```

**Valjund**

| ID | Name | Gender | Age | Contacts |
|---|---|---|---|---|
| 38702128612 | John Peterson | M | 38 | home: 6314981<br>mobile: 57123912 |
| 48112010907 | Jane Anderson | F | 29 | mobile: 50929101 |
| 34908119132 | Mike Jameson | M | 50 | home: 7430921<br>mobile: 58829120 |

### Lisalugemist

**Massiivi funktsioonid** (www) - inglise keeles

---

## 6.1 Klassi ja objekti mõisted

Siiani oleme tegelenud programmidega, mis koosnesid ainult funktsioonidest. Nüüd tutvume ka klassidega. Kui soovime koostada süsteemi, mis peab lahendama mitu ülesannet, siis on väga kasulik jagada süsteemi lihtsamateks osadeks ning kirjutada lahendamisalgoritm eraldi iga osa jaoks ja siis pärast ühendada need algoritmid. Mitme väiksema osa programmeerimine on alati lihtsam kui süsteemi kirjutamine tervikuna. Ka edaspidi on sellise süsteemi täiendamine ja parandamine lihtsam ja sellega ka odavam.

Programmeerimise algusaegadel võeti moodulite väljaeraldamisel aluseks tegevused, ülesande lahendamist vaadeldi tegevuste järjendina, millest püüti välja eraldada alamtegevused. Viimaste jaoks oli võimalik koostada omaette lahendamisalgoritm. Väga sageli esinevateks alamülesanneteks on näiteks järjendi elementide summa leidmine, kahe arvu väärtuste vahetamine, lineaarvõrrandisüsteemi lahendamine jms.

Selline lähenemisviis on väga hea siis, kui arvuti lahendab peamiselt arvutuslikke ülesandeid, aga osutub raskepäraseks juhul, kui vaja on töödelda mittearvulist infot. Seepärast on osutunud loomupärasemaks võtta programmi koostamisel aluseks mitte tegevused, vaid objektid, millega tegevusi tehakse. Viimane põhimõte kehtib ka PHP's, seepärast nimetatakse seda objektorienteeritud programmeerimiskeeleks.

### Objekt (object)

Objekt on kompleks andmetüüp, mis on lihtsalt funktsioonide (objekti funktsioone nimetakse tihti objekti meetoditeks) ja muutujate (objekti muutujaid nimetakse tihti objekti väljadeks) kogu, mis on loogiliselt selle objektiga seotud. Analoogias reaalse eluga võib muutujaid mõista kui mingi eseme kirjeldust ja meetodeid kui tegevusi, mida selle esemega on võimalik teha. Näiteks kui meil on tegemist objektiga 'Inimene', siis objekti väljad on nimi, sugu, abikaasa, lapsed, elukoht, töökoht ja objekti meetodid on siis nime muutmine, abiellumine, lahku minemine, töö- ja elukoha muutmine jne. Väljade ja meetodite valik sõltub süsteemi nõuetest ja arhitektuurist. Näiteks pangasüsteemis võib tarvis minna objekti 'Pangakaart', mille väljadeks on kaardi number, omaniku nimi, rahasumma, lukustusindikaator ning meetoditeks näiteks kaardil oleva rahasumma suurendamine mingi väärtuse võrra, rahasumma vähendamine, kaardi lukustamine, lukustuse tühistamine.

Objekti muutuja loomine on sama lihtne kui tavalise muutuja puhul. Esialgul kirjutatakse muutuja nimi ja seejärel luuakse uus klassi objekt kasutades konstruktsiooni **new ClassName(**parameetrid**);**.

**Näide 6.1.1**

```
<?php

$inimene = new Inimene('Mary', 'female');

echo 'Type: '.gettype($inimene).'<br />';
echo 'Class: '.get_class($inimene);

?>
```

**Väljund**

```
Type: object
Class: Inimene
```

## Klass (class)

Klass on lihtsalt objekti andmetüübi kirjeldus. Hea praktika on kirjutada klasse eraldi failidesse nimedega 'ClassName.php'. Klassi kirjeldatakse plokis class ClassName{...}. Klassi arhitekruur on tavaliselt järgmine:

**Näide 6.1.2**

```php
<?php

// klass nimega Inimene

class Inimene {

     // avalikud väljad

     public $name;       // klassi väli, kus hoitakse inimese nimi
     public $gender;     // klassi väli, kus hoitakse inimese sugu

     // privaatne väli

     private $idCode;    // klassi väli, kus hoitakse inimese isikukood


     // klassi konstruktor

     public function __construct ($name, $gender) {
          $this->nimi = $name;
          $this->gender = $gender;
     }

     // klassi meetodid

     public function changeName ($newName) {
          $this->name = $newName;
     }

     public function setIdCode ($idCode) {
          $this->idCode = $idCode;
     }

     public function getIdCode() {
          return $this->idCode;
     }

}

?>
```

Väljade defineerimisel on võimalik määrata ka ligipääsu direktiivi (piirajat). Võimalikud direktiivid:

- **public** - avalik ligipääs, klassi muutujate väärtusi saab vabalt kätte saada ja muuta ning klassi funktsioone käivitada kõikjal
- **private** - privaatne ligipääs, muutujad ja funktsioonid on kättesaadavad vaid selles klassis
- **protected** - kaitstud ligipääs, muutujad ja funktsioonid on kättesaadavad selles klassis ja selle klassi laiendavates klassides

Direktiivide kasutamise näided on olemas osas 6.2.

Igas klassis on vaikimisi olemas konstruktor - meetod, mida käivitatakse siis, kui klassi objekti luuakse, ehk siis iga kord kui kirjutame koodis **new ClassName();** Seda meetodit võib ka käsitsi ümber defineerida, selleks on olemas meetod **__construct(...)**. Näiteks meie näides konstruktoris määratakse kohe inimese nimi ja sugu. Lisaks __construct funktsioonile on olemas ka järgmised "maagilised" funktsioonid: __destruct, __call, __callStatic, __get, __set, __isset, __unset, __sleep, __wakeup, __toString, __invoke, __set_state, __clone. Selle kusrsuse raames kasutame ainult __construct ja __toString.

Konstruktori, destruktori ja __toString näide:

**Näide 6.1.3**

```php
<?php

class Test {

     public function __construct() {
          echo '<br>Konstruktor';
     }

     public function __destruct() {
          echo '<br>Destruktor';
     }

     public function __toString() {
          return '<br>Hello World!';
     }

}


$a = new Test();
echo $a;
unset($a); // muutuja $a kustutamine mälust

?>
```

**Väljund**

```
Konstruktor
Hello World!
Destruktor
```

### Lisalugemist

**Maagilised funktsioonid** (www) - inglise keeles

---

## 6.2 Klassi kirjutamine ja objekti kasutamine

Enne klasside kirjutamist on vaja hoolikalt läbi mõelda, mis klasse on vaja antud süsteemi realiseerimiseks, mis peavad olema nende väljad ja meetodid. Alles siis, kui analüüs on tehtud ja klasside nimekiri on kirjas - siis võib alustada klasside ja nende loogika kodeerimisega.

### Klassi kood

Iga klass algab sõnast **class** ja klassi nimest, millele järgneb klassi keha:

```
class ClassName {

     // klassi keha

}
```

Klassi sees esimesena deklareeritakse klassi muutujad koos ligipääsu direktiividega:

```
class ClassName {

     // klassi muutujad

     public $a;
     public $b;

     private $c;
     private $d;
     private $e;

     protected $f;

}
```

Järgmine on konstruktor (juhul, kui seda on vaja) ja peale seda klassi meetodid:

```
class ClassName {

     // klassi muutujad

     public $a;
     public $b;

     private $c;
     private $d;
     private $e;

     protected $f;

     // klassi konstruktor

     public function __construct() {
          ...
          ...
          ...
     }

     // klassi meetodid

     public function f1 () {
          ...
          ...
          ...
     }

}
```

Selleks, et klassi sees interpritaator saaks aru millal on tegemist klassimuutujaga ja millal mitte - tuleb klassi muutuja ette panna **$this->** ja muutuja nimi. Seega klassis MingiKlass rida **$this->a = 9;** tähendab: leia selles klassis muutuja a ja pane talle väärtuseks 9. Sama moodi on ka klassi meetoditega - kui mingi klassi meetod peab käivitada mingit teist meetodit selles samas klassis - siis lisatakse funktsiooni ette konstruktsioon **$this->**:

**Näide 6.2.1**

```
<?php

class A {

     public $x;
     public $y;
     public $z;

     // konstruktor - seda käivitatakse
```

```
     // automaatselt objekti loomisel

     public function __construct() {

          // klassi muutujate väärtustamine
          $this->x = 1;
          $this->y = 2;
          $this->z = 3;

          // klassi meetodi käivitamine klassi seest
          $this->multiplyAll(-1);

     }

     private function multiplyAll ($multiplier) {
          // klassi muutujate väärtuste muutmine
          $this->x *= $multiplier;
          $this->y *= $multiplier;
          $this->z *= $multiplier;
     }

}


$a = new A();
var_dump($a);

?>
```

**Väljund**

```
     object(A)[1]
        public 'x' => int -1
        public 'y' => int -2
        public 'z' => int -3
```

Klassi muutujatele ja funktsioonidele juurdepääsu väljaspoolt klassi saab ka sarnasel moel, aga siis **$this->** asemel tuleb kirjutada **$objektiNimi->**.

**Näide 6.2.2**

```
<?php

class A {

     public $x;
     public $y;
     public $z;


     public function __construct() {

          $this->x = 1;
          $this->y = 2;
          $this->z = 3;

     }

     public function multiplyAll ($multiplier) {
          $this->x *= $multiplier;
          $this->y *= $multiplier;
          $this->z *= $multiplier;
     }

}


$a = new A();
echo 'variable x from object $a: '.$a->x;
$a->x = 10;
echo '<br />variable x from object $a: '.$a->x;
$a->multiplyAll(5);
echo '<br />variable x from object $a: '.$a->x;

?>
```

## Juurdepääsu direktiivid

Kui klassimuutuja ligipääs on **public** - siis neid võib vabalt muuta ja väärtustada ka väljaspoolt klassi. Miks selline lähenemine võib halb olla? Oletame, et meil on klass B ja selle muutuja x on public ligipääsuga. Kogu süsteemi koodis väärtustakse x muutujat otseselt ($b->x = mingi arv) ja väärtuseks võib olla nii täisarv kui ka murdarv. Nüüd mõni aja pärast otsustame, et klassi B muutuja x väärtus võib olla ainult täisarv - aga siis on vaja vaadata kogu kood üle ja lisada round() funktsiooni seal, kus x väärtuseks pannakse murdarv.

```
class B {

     public $x;

}


$b = new B();
$b->x = 5;
// ...
// ...
$b->x = 4.9;
//...
$b->x = -7.1;
```

Tavaliselt määratakse kõigile klassi muutujatele **private** juurdepääsu ja kogu kommunikatsioon klassi muutujatega toimub läbi klassi funktsioone. Sellel juhul klassi B muutuja x väärtustamiseks tuleb kasutada funktsiooni, näiteks $b->setX(mingi arv). Nüüd kui oleks vaja süsteemi ümber ehitada nii, et x väärtused oleksid ainult täisarvud, võib seda teha ainult lisades round() klassi B meetodi setX juurde. Muutuja x väärtuse saamiseks kirjutame lisaks ka funktsioon getX, sest juhul, kui proovime privaatse muutuja x väärtust saada otseselt - visatakse veateade.

```
class B {

     private $x;

     public function setX ($value) {
          $this->x = round($value);
     }

     public function getX() {
          return $this->x;
     }

}


$b = new B();
$b->setX(5);
// ...
// ...
$b->setX(4.9);
//...
$b->setX(-7.1);
```

## Näide

Kujutame ette, et ühel ettevõttel on vaja kirjutada väike veebiinfosüsteem, mis on seotud töötajate haldamisega. On olemas andmebaas inimeste nimedega ja isikukoodidega. Meie süsteem peab oskama isikukoodi järgi öelda, kas tegemist on mehega või naisega ja lisaks ka isikukoodist võtta isiku sünnipäeva. Kirjutame klassi Human (fail Human.php), mille väljad on vastavalt siis eesnimi, perekonnanimi ja isikukood. Lisame klassi ka funktsioone isikukoodi parsimiseks (http://et.wikipedia.org/wiki/Isikukood):

**Naide 6.2.3 (Human.php)**

```
<?php

/**
 * Class representing human.
 */
class Human {

     private $firstname;
     private $familyname;
     private $idCode;


     /**
      * Human class constuctor.
      * Also sets human firstname and familyname
      *
      * @param string $firstname human firstname
      * @param string $familyname human familyname
      */
     public function __construct($firstname, $familyname) {
          $this->setFirstname($firstname);
          $this->setFamilyname($familyname);
     }


     /**
      * Sets human firstname.
      * Before setting firstname removes empty
      * spaces before and after it and switches
      * first letter to upper case.
      *
      * @param string $name human firstname
      */
     public function setFirstname ($name) {
          $this->firstname = ucfirst(trim($name));
     }


     /**
      * Sets human familyname.
      * Before setting familyname removes empty
      * spaces before and after it and switches
      * first letter to upper case.
      *
      * @param string $name human familyname
      */
     public function setFamilyname ($name) {
          $this->familyname = ucfirst(trim($name));
     }


     /**
      * Returns human firstname.
      * @return string human firstname
      */
     public function getFirstname() {
          return $this->firstname;
     }
```

```php
    /**
     * Returns human familyname.
     * @return string human familyname
     */
    public function getFamilyname() {
        return $this->familyname;
    }


    /**
     * Sets human ID code.
     *
     * @param string $idCode human ID code
     */
    public function setIdCode ($idCode) {
        $this->idCode = $idCode;
    }


    /**
     * Returns human ID code.
     * @return string human ID code
     */
    public function getIdCode() {
        return $this->idCode;
    }


    /**
     * Returns human full name.
     * Full name contains firstname, familyname
     * and space character between them
     *
     * @return string human ID code
     */
    public function getFullname() {
        return $this->firstname.' '.$this->familyname;
    }


    /**
     * Returns human gender.
     * Calculates human gender based on
     * ID code. If no ID code is set -
     * returns null
     *
     * @return null|string human gender (Male or Female)
     */
    public function getGender() {

        if (is_null($this->idCode)) return null;

        $genderNumber = substr($this->idCode, 0, 1);

        if ($genderNumber % 2 == 0) {
            return 'Female';
        } else {
            return 'Male';
        }

    }


    /**
     * Returns human date of birth.
     * Calculates human birthday based on
     * ID code. If no ID code is set -
     * returns null. Date format: DD.MM.YYYY
     *
     * @return null|string human date of birth
     */
    public function getBirthday() {

        if (is_null($this->idCode)) return null;

        $genderNumber = substr($this->idCode, 0, 1);
        $year = '';

        switch ($genderNumber) {

            case '1':
            case '2':
                $year = '18';
            break;

            case '3':
            case '4':
                $year = '19';
            break;

            case '5':
            case '6':
                $year = '20';
            break;

        }

        $year .= substr($this->idCode, 1, 2);
        $month = substr($this->idCode, 3, 2);
        $day = substr($this->idCode, 5, 2);

        return $day.'.'.$month.'.'.$year;

    }
```

```
    /**
     * Magic function returns textual presentation of object.
     * @return string textual presentation of human object
     */
    public function __toString() {
        $result = '<b>Name</b>: '.$this->firstname.' '.$this->familyname.'<br />';
        $result .= '<b>ID code</b>: '.$this->getidCode().'<br />';
        $result .= '<b>Birthday</b>: '.$this->getBirthday().'<br />';
        $result .= '<b>Gender</b>: '.$this->getGender().'<br />';
        return $result;
    }

}

?>
```

Selleks, et klassi Human kasutada mingis skriptis tuleb seda skripti lisada. Failide laadimiseks skripti käigus kasutatakse **include** ja **require** käske. Nende abil võib laadida ükskõik mis failid (*.php, *.html, jne) ning juhul kui laetavas failis on php kood olemas siis seda käivitatakse. Et vältida mitmekordset ühe ja sama faili laadimist kasutatakse **include_once** ja **require_once** - need käsud kontrollivad, kas nõutud fail on juba laetud või mitte ja kui fail on juba laetud - koodi täitmine läheb edasi, vastasel juhul - laetakse faili.

**Naide 6.2.4**

```
<?php

include_once 'Human.php';

$crew = array();

$human1 = new Human('john', 'doe');
$human1->setIdCode('38312280871');

$human2 = new Human('mary', 'james');
$human2->setIdCode('60108131021');

$crew[] = $human1;
$crew[] = $human2;

?>

<table border="1" cellspacing="2" cellpadding="5">
    <tr bgcolor="silver">
        <th>Name</th>
        <th>Gender</th>
        <th>Birthday</th>
    </tr>
    <?php foreach ($crew as $human) { ?>
        <tr>
            <td><?php echo $human->getFullname(); ?></td>
            <td><?php echo $human->getGender(); ?></td>
            <td><?php echo $human->getBirthday(); ?></td>
        </tr>
    <?php } ?>
</table>
```

**Valjund**

| Name | Gender | Birthday |
|---|---|---|
| John Doe | Male | 28.12.1983 |
| Mary James | Female | 13.08.2001 |

## 6.3 Klassi pärilus

Pärilus on väga tähtis objektorienteeritud programmeerimises. Loomulikult toetab seda ka PHP. Päriluse mõte on selles, et on võimalik kijutada üldisemaid klasse, ja nendest juba põlvnevad spetsiifilisemad alamklassid ehk tuletatud (derived) klassid. Ja nendest võib omakorda põlvneda veel mõned teised klassid. Sellel moel tekib hierarhiline struktuur, milles iga klass kasutab ära oma vanem- või ülemklassi ehk baasklassi (base class) omadusi. Tuletatud klass ei pea ise defineerima välju ja funktsioone mis on tema baasklassides juba olemas, ainuke asi mis tuletatud klassis on vaja kodeerida - mingid spetsiifilised meetodid ja ainult sellele klassile vajalikud väljad.

Selleks, et näidata et mingi klass põlvneb mingist klassist kasutatakse võtmesõna **extends** (laiendab) klassi päises. Kui tuletatud klassis on vaja baasklassi funktsiooni üle kirjutada (override) - siis lihtsalt kirjutame tuletatud klassi funktsiooni sama nimega (näides on niimoodi tehtud sayHello() meetodiga).

**Näide 6.3.1**

```
<?php

/**
 * Base class
 */
class A {

    public $a;
    public $b;

    public function getABSum() {
        return $this->a + $this->b;
    }
```

```
     public function sayHello() {
          echo 'class A';
     }

}

/**
 * Derived class
 */
class B extends A {

     public $c;

     public function getMin() {
          return min($this->a, $this->b, $this->c);
     }

     public function getMax() {
          return max($this->a, $this->b, $this->c);
     }

     // klassi A meetodi ülekatmine
     public function sayHello() {
          echo 'class B';
     }

}


$obj = new B();

$obj->a = 10; // päritud muutuja
$obj->b = 15; // päritud muutuja
$obj->c = 20;

$obj->sayHello();

echo '<br /><br />';
echo 'ABSum: '.$obj->getABSum().' (päritud meetod)<br />';
echo 'Min: '.$obj->getMin().'<br />';
echo 'Max: '.$obj->getMax();

?>
```

**Väljund**

```
class B

ABSum: 25 (päritud meetod)
Min: 10
Max: 20
```

## Baasklassi ülekaetud meetodi käivitamine alamklassis

Mõnikord tekib vajadus pöörduda baasklassi meetodi (samuti ka muutuja) poole. Juhul, kui see meetod on päritud baasklassist - siis pole probleemi ja $this->methodName() teeb oma tööd ilusti. Aga mis siis teha kui baasklassi meetod on juba ülekaetud antud klassis, sest $this->methodName() kutsub selles klassis olevat funktsiooni. Selleks on olemas võtmesõna **parent**. Tuleb kirjutada **parent::methodName()** ja sellega pöördume juba kindlasti baasklassi meetodi poole. Pange tähele, et ka maagilisi funktsioone võib niimoodi kätte saada.

**Näide 6.3.2**

```
<?php

/**
 * Base class
 */
class Kolmik {

     public $p1;
     public $p2;
     public $p3;

     public function __construct($a, $b, $c) {
          $this->p1 = $a;
          $this->p2 = $b;
          $this->p3 = $c;
     }

     public function sum() {
          return $this->p1 + $this->p2 + $this->p3;
     }

     public function __toString() {
          return $this->p1.', '.$this->p2.', '.$this->p3;
     }

}

/**
 * Derived class
 */
class Nelik extends Kolmik {

     public $p4;

     public function __construct($a, $b, $c, $d) {
          parent::__construct($a, $b, $c);
          $this->p4 = $d;
```

```
    }

    public function sum() {
        return parent::sum() + $this->p4;
    }

    public function __toString() {
        return parent::__toString().', '.$this->p4;
    }
}


$kolmik = new Kolmik(5, 6, 7);
echo $kolmik;
echo '<br>sum: '.$kolmik->sum();

echo '<br>--------------<br>';

$nelik = new Nelik(1, 2, 3, 4);
echo $nelik;
echo '<br>sum: '.$nelik->sum();

?>
```

**Väljund**

```
5, 6, 7
sum: 18
--------------
1, 2, 3, 4
sum: 10
```

## Pärilus ja ligipääsu direktiivid

Nagu on juba mainitud, **private** teeb muutujat või funktsiooni nähtavaks ainult selles klassis kus nad on defineeritud. Sama lugu on ka pärilusega. Juhul kui baasklassi muutuja või meetod on privaatne - siis alamklassid sellele ligi ei saa. **Public** meetodi ja muutuja poole saab pöörduda ükskõik kus kohast - seega päriluse korral see toimib ka. Lisaks vaatleme nüüd veel ühe direktiivi: **protected**. Protected lubab juurdepääsu muutujatele ja meetoditele ainult klassist kus nad on defineeritud ning lisaks ka kõikidest temast tuletatud alamklassidest.

**Näide 6.3.3**

```
<?php

/**
 * Base class
 */
class A {

    public $a;
    private $b;
    protected $c;

    public function __construct() {
        $this->a = 2;
        $this->b = 4;
        $this->c = 6;
    }

    private function saySomethingPrivate() {
        echo '<br>Private function in class A';
    }

    protected function saySomethingProtected() {
        echo '<br>Protected function in class A';
    }

}

/**
 * Derived class
 */
class B extends A {

    public function __construct() {
        parent::__construct();
    }

    public function getC() {
        return $this->c;
    }


    public function saySomethingProtected() {
        parent::saySomethingProtected();
    }

    public function saySomethingPrivate() {
        parent::saySomethingPrivate();
    }

}


$b = new B();

echo 'a = '.$b->a.'<br>';
//echo 'b = '.$b->b.'<br>' Fatal error: Cannot access private property
//echo 'c = '.$b->c.'<br>' Fatal error: Cannot access protected property
echo 'c = '.$b->getC().'<br>';
```

```
$b->saySomethingProtected();
//$b->saySomethingPrivate(); Fatal error: Call to private method

?>
```

---

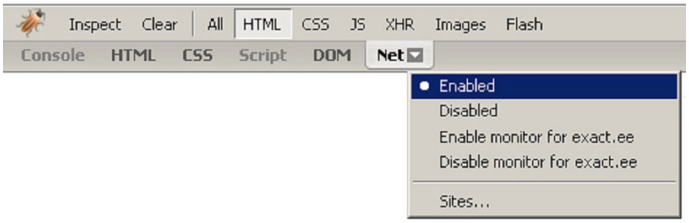
## 7.1 Ülevaade

HTTP on suhteliselt lihtne tekstipõhine protokoll informatsiooni edastamiseks arvutivõrkudes. Nagu me juba teame, brauser ja veebiserver suhtlevad omavahel HTTP (Hyper-Text Transfer Protocol) protokolli abil. Teiste sõnadega HTTP on keel, mida räägivad omavahel kliendi brauser ja veebiserver. HTTP on päring-vastus protokoll, kliendi brauser moodustab ja esitab serverile päringu ja vastusena saab serverilt päritud faili.

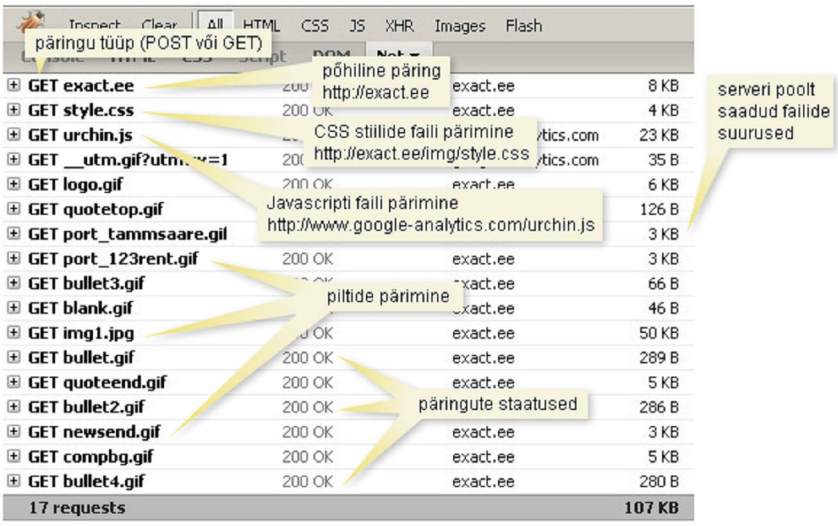
Selleks, et vaadata täpsemalt, kuidas HTTP suhtlus toimub kasutame Firebug plugini Firefox brauseri jaoks. Juhul, kui teie arvutis veel puudub Firefox brauser, siis on õige aeg seda installeerida. Nüüd lisame brauserisse Firebug plugini (kasutame versiooni 1.3.2, sest versioonis 1.3.3 on mingi viga post parameetrite näitamisel). Juhul, kui installeerimine õnnestus näete paremas alumises nurgas Firebug ikooni. Firebugi kasutatakse tihti javascripti ja css käitumise debugimiseks ning DOM struktuuri õiguse kontrollimiseks. Meie aga hakkame seda praegu kasutama HTTP GET ja POST meetodite uurimiseks.


Firebug plugini ikoon Firefox brauseri paremas alumises nurgas

Klikkige Firebug ikooni peale ja menüüs valige **Net** → **Enabled**.

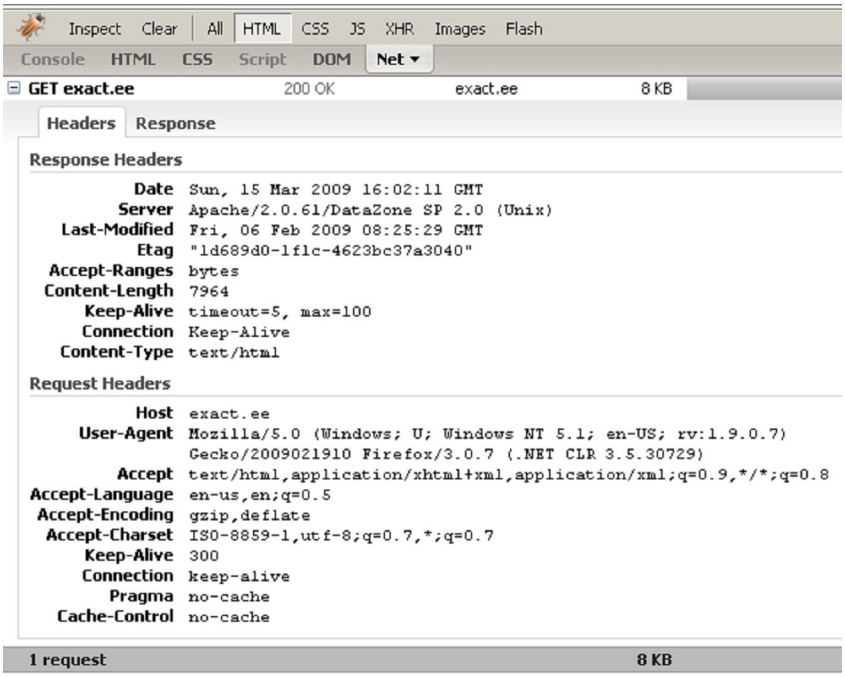



Avage näiteks http://exact.ee lehe ja kui Firebug ei ole avatud klikkige selle ikooni peale ja valige **Net**. See, mis teie praegu näete on kõik päringud mis teie brauser on teinud selleks, et laadida ja peale seda nädata exact.ee lehe:




Päringuid võib ka sorteerida:


Valige **HTML**. Nüüd nimekirjas on ainuke päring, uurime seda põhjalikumalt. HTTP päring koosneb päringust, päistest ja andmetest. Päringurida koosneb päringu tüübist, dokumendi URL-ist ja protokolli versioonist. Näiteks praegu päringureaks on **GET http://exact.ee HTTP/1.1**, päringureale järgenavad päised, mis lõpevad tühja reaga ja peale seda päringu andmed (neid saadetakse serverile ainult POST päringu puhul, GET puhul parameetreid kirjutakse otse dokumendi URL-i):



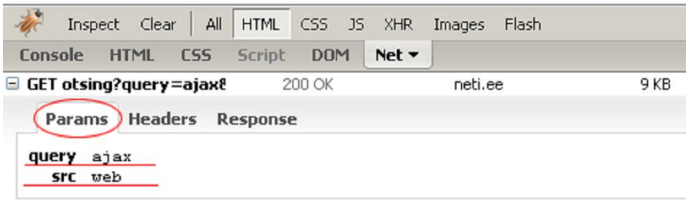

Mõnede tähtsamate päisete selgitused (klient → server):

**Host** - hosti nimi (exact.ee, google.com, neti.ee ...)
**User-Agent** - info brauseri ja opsüsteemi kohta
**Accept** - meedia tüübid mis on aktsepteeritavad brauseri poolt
**Accept-Language** - keeled, mis brauser eelistab
**Accept-Charset** - lehe charsetid, mis on eelistatud

HTTP vastus koosneb vastusest, päistest ja andmetest (dokumendist). Vastuses (struktuur: protokoll kood tekst) sisalduv kood näitab kas päring õnnestus või mitte: **HTTP/1.1 200 OK**. Vastusele järgnevad päised, neid lõpetab samuti tühi rida ja peale seda lähevad vastuse andmed. Mõnede tähtsamate päisete selgitused (server → klient):

**Date** - aeg millal vastus oli saadetud
**Server** - info serveri kohta
**Last-Modified** - aeg millal päritud resurss oli viimati muudetud
**Content-Length** - saadetud ressurssi suurus
**Content-Type** - ressurssi meedia tüüp

Kuna GET parameetreid kirjutatakse otse URL-i - parameetrite sümbolite arv on piiratud (erinevate brauserite ja serverite puhul maksimaalne URL-i pikkus on erinev). GET parameetreid kirjutatakse URL-is peale küsimärki kujul parameetri_nimi=parameetri_väärtus ning erinevaid parameetreid eraldatakse & märgi abil. GET parameetrite näide: http://www.neti.ee/cgi-bin/otsing?query=ajax&src=web. Siin parameetriteks on **query** ja **src** ning parameetrite väärtusteks vastavalt **ajax** ja **web**.




Nüüd avage http://www.aleixcortadellas.com/main/ ja sisestage paremas ülemises nurgas otsingu aknas mingi sõna ja vajutage Enter. Firebug aknas tuleb üks tab nimega **POST** juurde. Selles tabis näidetakse, mis parameetreid edastatakse serverile koos päistega päringu andmete osas. Parameetrid on alati paar: parmaeetri nimi - väärtus. POST-iga saab edastada nii palju informatsiooni kui soovite (seda piiratakse tavaliselt serveri konfiguratsiooni failis ja paljudel juhtumitel POST andmete suuruse limiit on suurem, kui GET puhul URL-i pikkuse limiit) ning andmed pole nähtavad URL-is.


## Lisalugemist

**HTTP 1.1 spetsifikatsioon** (www) - inglise keeles

**Firebug dokumentatsioon** (www) - inglise keeles

## 7.2 GET

GET parameetrid on kättesaadavad PHP skriptis läbi globaalse **$_GET** massiivi. Massivi elementide võtmeteks on parameetrite nimed ja väärtusteks vastavalt parameetrite väärtused. Lihtne näide: skript võtab URL-ist parameetrid param1 ja param2 ning väljastab nende väärtused ekraanile.

**Näide 7.2.1**

```
<?php

echo 'param1: '.$_GET['param1'].'<br />';
echo 'param2: '.$_GET['param2'];

?>
```

**http://localhost/index.php?param1=Hello&param2=world**

```
param1: Hello
param2: world
```

**http://localhost/index.php?param1=yahoo**

```
param1: yahoo
param2:
```

Hästi tihti kasutakse GET parameetreid mingi info näitamiseks parameetri järgi. Näiteks uudiseid:

**Näide 7.2.2**

```
<?php

$news = array(
     14 => array(
          'title' => 'Systems go for Sunday Shuttle Discovery launch',
          'text' => 'Commander Lee Archambault is set to lead a crew of seven
                     on the STS-119 mission, which is aimed at installing the
                     fourth and final set of power-generating solar arrays to
                     the ISS. The arrays will provide electricity to fully power
                     science experiments and support the station\'s expanded
                     crew in May.'
     ),
     5 => array(
          'title' => 'Facebook Announces Facebook Connect for iPhone',
          'text' => 'Facebook has announced Facebook Connect for the iPhone,
                     which enables users to take their Facebook friends,
                     identity and privacy with them wherever they go.'
     ),
     9 => array(
          'title' => 'Forget the Fail Whale: Twitter Jumps the Shark',
          'text' => 'Twitter has jumped the shark for the digerati attending
                     South by Southwest here in Austin. Daniel Terdiman at
                     C|Net points out what everyone trying to follow the #sxsw
                     tweets have discovered -there are just too many of them.'
     ),
     23 => array(
          'title' => 'iPhone firmware followup: the numbers',
          'text' => 'Which SDK should you compile for? 2.2.1? 2.2?
                     Choosing an active SDK version can affect both your target
                     audience and possible App Store sales. We recently wrote about
                     SDK choices, noting that building for 2.2.1 (the latest version,
                     as of this writing) might cut out users who had yet to upgrade
                     from the 2.2 firmware.'
     )
);

if (isset($_GET['id']) && isset($news[intval($_GET['id'])])) {

     echo '<a href="/index.php">Back to the list</a><br /><br />';
     echo '<h1>'.$news[intval($_GET['id'])]['title'].'</h1>';
     echo $news[intval($_GET['id'])]['text'];

} else {

     foreach ($news as $id => $data) {
          echo '<a href="/index.php?id='.$id.'">'.$data['title'].'</a><br />';
     }

}

?>
```

**http://localhost/index.php**

```
Systems go for Sunday Shuttle Discovery launch
Facebook Announces Facebook Connect for iPhone
Forget the Fail Whale: Twitter Jumps the Shark
iPhone firmware followup: the numbers
```

**http://localhost/index.php?id=14**

```
    Back to the list

    Systems go for Sunday Shuttle Discovery launch

    Commander Lee Archambault is set to lead a crew of seven on the STS-119
    mission, which is aimed at installing the fourth and final set of power-
    generating solar arrays to the ISS. The arrays will provide electricity
    to fully power science experiments and support the station's expanded
    crew in May.
```

### Lisalugemist

**HTTP 1.1 spetsifikatsioon** (www) - inglise keeles
**Firebug dokumentatsioon** (www) - inglise keeles

---

## 7.3 Vormid

Juhul, kui soovite saada kasutaja käest mingi info saada - siis abiks on html vormid. Kasutaja saab sisestada informatsiooni teksti lahtritesse (text, textarea), valida erinevaid väärtusi kontrollkastide (checkbox), raadionuppude (radio) ja menüüde (select) abil. Kõik need elementid pannakse vormi sisse, mida on võimalik saata serverile kas GET või POST parameetritena. Vaatleme vormi elemendid eraldi ja pärast kasutame neid vormi loomiseks.

### Form

Vormi element ise, mis koondab elemendid, mis vormi sisse käivad. Vormi tähtsamateks atribuutideks on method ja action. Meetodiks võib olla kas GET või POST - selle abil võib defineerida kuidas kasutaja poolt sisestatud info serverini jõuab (kas GET või POST abil). Action defineerib URL-i millele saadetakse GET või POST päring. Näiteks järgmises vormis:

**Näide 7.3.1**

```
<form method="POST" action="http://localhost/comment.php">
   ...
   ...
</form>
```

päringu saatmiseks kasutatakse POST meetodit ja andmeid saadetske skripti http://localhost/comment.php. Vaikimisi meetod on GET ja action on sama URL, kus asub vorm ise. Näiteks lehel http://localhost/form.php asub vorm:

**Näide 7.3.2**

```
<form>
   ...
   ...
</form>
```

Vormi saatmisel kasutatakse GET päring ja skript, kuhu andmeid saadetakse on see sama http://localhost/form.php.

### Text

Teksti lahter, mida kasutatakse peamiselt vaba teksti sisestamiseks. Peaaegu kõikide vormi elementide ühistägiks on <input> ning elemendi tüüpi määrab **type** atribuut. Teksti lahtri atribuut on **text**. Teised olulised atribuudid on **name** (sama atribuut on kõikidel vormi elementidel olemas, sest see on GET või POST parameetri nimi), **value** (väärtus, mis asub lahtris, päringu saatmisel parameetri väärtus), **maxlength** (maksimaalne sümbolite arv lahtris). All on toodud text input'i näited:

**Näide 7.3.3**

```
<form>
     <input type="text" name="param1" maxlength="5" />
</form>
```

**Tulemus**

**Näide 7.3.4**

```
<form>
     <input type="text" name="param1" value="Eeldefineeritud väärtus" />
</form>
```

**Tulemus**


### Password

Teksti lahter, mille sümbolid on peidetud kasutaja eest. Tihti kasutatakse parooli sisestamiseks:

**Näide 7.3.5**

```
<form>
     <input type="password" name="user_password" maxlength="5" />
</form>
```

**Tulemus**

## Button

Tavaline nupp, mis ei mängi vormis erilist rolli, vaid seda saab edukalt kasutada javascripti käivitamiseks. Atribuut **value** määrab nupu teksti.

**Näide 7.3.6**

```
<form>
     <input type="button" value="click me" onClick="alert('javascript')" />
</form>
```

**Tulemus**

click me

**Näide 7.3.7**

```
<form>
     <input type="button" value="google.com" onClick="window.location='http://www.google.com'" />
</form>
```

**Tulemus**

google.com

## Checkbox

Konrollkast. Kui elemendi sees on olemas atribuut **checked** - siis kontrollkast on linnutatud, vastasel juhul kast on tühi. Name-value paari edastatakse serverile GET või POST päringus vaid siis, kui kontrollkast on linnutatud, vastasel juhul seda ei tehta. Võib defineerida ka atribuuti **id** (unikaalne identifikaator) selleks, et pärast panna <label for="id"> tägi, mis viitab vastavale input elemendile.

**Näide 7.3.8**

```
<form>

     <input id="season1" type="checkbox" name="winter" value="1" />
     <label for="season1">Winter</label><br />

     <input id="season2" type="checkbox" name="spring" value="1" />
     <label for="season2">Spring</label><br />

     <input id="season3" type="checkbox" checked name="summer" value="1" />
     <label for="season3">Summer</label><br />

     <input id="season4" type="checkbox" checked name="autumn" value="1" />
     <label for="season4">Autumn</label>

</form>
```

**Tulemus**

- [ ] Winter
- [ ] Spring
- [x] Summer
- [x] Autumn

## Radio

Raadionupp. Atribuut **checked** töötab nagu checkbox korral. Name-value paari edastatakse serverile GET või POST päringus vaid siis, kui Raadionupp on valitud, vastasel juhul seda ei tehta. Atribuuti **id** võib ka kasulik olla. Selleks, et luua raadionuppude grupi - peab gruppi kuuluvate raadionuppude nimed olema samad.

**Näide 7.3.9**

```
<form>

     <input id="season5" type="radio" name="season" value="winter" />
     <label for="season5">Winter</label><br />

     <input id="season6" type="radio" checked name="season" value="spring" />
     <label for="season6">Spring</label><br />

     <input id="season7" type="radio" name="season" value=summer" />
```

```
    <label for="season7">Summer</label><br />

    <input id="season8" type="radio" name="season" value="autumn" />
    <label for="season8">Autumn</label>

</form>
```

**Tulemus**


## Submit

Vormi saatmise nupp. Peale vajutamist saadetakse vormi andmed kas GET või POST-iga. Atribuutid on samad, mis tavalise nupu korral.

**Näide 7.3.10**

```
<form>
    <input type="submit" value="Saada" />
</form>
```

**Tulemus**


## Select

Menüü valik. Elemendiks on <select>. Menüü punktid on esitatud <option> elementide kaudu. Select elemendil on **name** atribuut ning igal **option** elemendil on value atribuut. Päringus edastatakse select elemendi nimi parameetri nimeks ja valitud option elemendi value väärtust. Selleks, et määrata mingi option element valituks kasutage **selected** atribuuti.

**Näide 7.3.11**

```
<form>
    <select name="season">
        <option value="winter">Winter</option>
        <option value="spring">Spring</option>
        <option selected value="summer">Summer</option>
        <option value="autumn">Autumn</option>
     </select>
</form>
```

**Tulemus**


## Textarea

Tekstiväli suuremate tekstide sisestamiseks. Atribuutidena võib määrata **rows**, ehk mitu rida on tekstiväljal, **cols** - mitu veergu on tekstiväljal. Mõlemate pikkus on tähemärkides.

**Näide 7.3.12**

```
<form>
    <textarea name="description" rows="6" cols="50"></textarea>
</form>
```

**Tulemus**

**Näide 7.3.13**

```
<form>
    <textarea name="description" rows="6" cols="50">Sisestatud tekst</textarea>
</form>
```

**Tulemus**

## Hidden input

Mõnikord tekib vajadus saata vormiga koos mõned andmed, mis on teie poolt defineeritud ja mida kasutaja ei pea ise sisestama ja isegi ei pea nendest muretsema. Näiteks uudise ID, mille jaoks kirjutatakse kommentaar. Selleks on olemas hidden input, mille jaoks piisab kui on defineeritud nimi ja väärtus.

**Näide 7.3.14**

```
<form>
     <input type="hidden" value="41" name="news_id" />
</form>
```

## Näide 1

Oletame, et meil on olemas infosüsteem, kus kasutaja võib lugeda erinevaid artikleid (uudistest, blogidest ja ajakirjadest). Artikleid on hästi palju ja selleks, et oleks mugavam neid lugeda me näitame kasutajale järgmise vormi, kus ta saab valida mis kujul, kui palju ja mis allikatest artikleid ta soovib saada:

**Näide 7.3.15**

```
<form method="GET" action="http://www.example.com/display_information.php">
     <input type="hidden" name="action" value="show_info" />
     <table class="exampleForm" cellpadding="0" cellspacing="0" border="0">
          <tr>
               <td class="title" colspan="2">Setup information display</td>
          </tr>
          <tr>
               <th>Category:</th>
               <td>
                    <input type="checkbox" checked id="news_box" name="news" value="1" />
                    <label for="news_box">Online news</label><br />
                    <input type="checkbox" id="mag_articles" name="articles" value="1" />
                    <label for="mag_articles">Magazine articles</label><br />
                    <input type="checkbox" checked id="blog_emtries" name="blog" value="1" />
                    <label for="blog_emtries">Blog entries</label>
               </td>
          </tr>
          <tr class="even">
               <th>Items:</th>
               <td>
                    <select name="items">
                         <option value="10">10 per page</option>
                         <option value="20">20 per page</option>
                         <option selected value="30">30 per page</option>
                         <option value="40">40 per page</option>
                    </select>
               </td>
          </tr>
          <tr>
               <th>Show:</th>
               <td>
                    <input type="radio" id="full_texts" name="show" value="full_texts" />
                    <label for="full_texts">Full texts</label><br />
                    <input type="radio" checked id="abstracts" name="show" value="abstracts" />
                    <label for="abstracts">Abstracts</label><br />
               </td>
          </tr>
          <tr>
               <td class="button" colspan="2">
                    <input type="submit" value="Continue" />
               </td>
          </tr>
     </table>
</form>
```

Tulemuseks on järgmine vorm. Pange tähele, et teiste HTML elementide kasutamine vormi ülesehitamisel teeb vormi kujundust palju ilusam (CSS stiilid).


Juhul, kui kasutaja vajutab nuppu "Continue" - saadetakse GET päring skripti http://www.example.com/display_information.php, ehk siis koos GET parameetritega URL on järgmine: http://www.example.com/display_information.php?action=show_info&news=1&blog=1&items=30&show=abstracts ja selles skriptis või järgmine kood vastutada loogika eest:

**Näide 7.3.16**

```
<?php
```

```php
if (isset($_GET['action'])) {

     switch ($_GET['action']) {

          case 'show_info':

               $dataArray = getInfoFromSources($_GET['news'], $_GET['articles'], $_GET['blog'], $_GET['show']);

               if ($_GET['show'] == 'abstracts') {
                    showAbstracts($dataArray);
               } else {
                    showFulltext($dataArray);
               }

          break;

     }

}

function getInfoFromSources ($fromNews, $fromArticles, $fromBlog, $limit) {

     // sooritakse andmebaasi päring ja tagastatakse tulemuse

}

function showAbstracts ($dataArray) {

     // kuvatakse artiklite abstraktid

}

function showFulltext ($dataArray) {

     // kuvatakse artiklite tekstid

}

?>
```

### Lisalugemist

**Vormi ja vormi elementide spetsifikatsioon** (www) - inglise keeles
**NiceForms projekt** (näide, koduleht) - inglise keeles

---

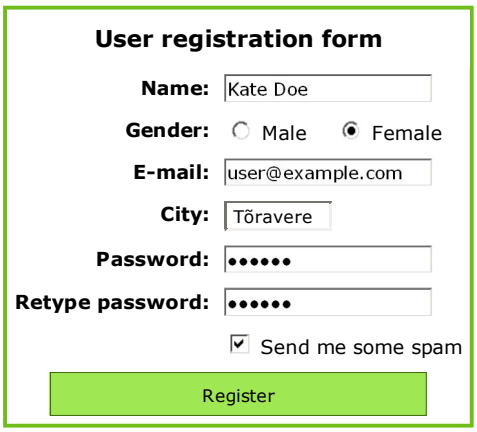
## 7.4 POST

Ainuke erinevus GET ja POST vahel PHP skriptides on see, et $_GET massiivi asemel POST parameetreid kirjutatakse massiivi **$_POST**. Järgmises näides kasutaja registreerimise vormi väärtusi edastatakse POST parameetritena:

**Näide 7.4.1**

```php
<form method="POST" action="http://www.example.com/register_user.php">
     <input type="hidden" name="action" value="new_user" />
     <table class="exampleForm" cellpadding="0" cellspacing="0" border="0">
          <tr>
               <td class="title" colspan="2">User registration form</td>
          </tr>
          <tr>
               <th>Name:</th>
               <td>
                    <input type="text" name="name" value="Kate Doe" />
               </td>
          </tr>
          <tr class="even">
               <th>Gender:</th>
               <td>
                    <input type="radio" checked id="male" name="gender" value="male" />
                    <label for="male">Male</label>   
                    <input type="radio" checked id="female" name="gender" value="female" />
                    <label for="female">Female</label><br />
               </td>
          </tr>
          <tr>
               <th>E-mail:</th>
               <td>
                    <input type="text" name="email" value="user@example.com" />
               </td>
          </tr>
          <tr>
               <th>City:</th>
               <td>
                    <select name="city_id">
                         <optgroup label="Harjumaa">
                              <option value=""32>Tallinn</option>
                              <option value=""4>Keila</option>
                              <option value=""19>Tabasalu</option>
                         </optgroup>
                         <optgroup label="Tartumaa">
                              <option value=""2>Tartu</option>
                              <option value=""11>Elva</option>
                              <option selected value=""8>Tõravere</option>
```

```
                    </optgroup>
                </select>
            </td>
        </tr>
        <tr>
            <th>Password:</th>
            <td>
                <input type="password" name="pass" value="qwerty" />
            </td>
        </tr>
        <tr>
            <th>Retype password:</th>
            <td>
                <input type="password" name="pass_confirmation" value="qwerty" />
            </td>
        </tr>
        <tr>
            <th></th>
            <td>
                <input type="checkbox" checked id="newsletters" name="newsletters" value="1" />
                <label for="newsletters">Send me some spam</label><br />
            </td>
        </tr>
        <tr>
            <td class="button" colspan="2">
                <input type="submit" value="Register" />
            </td>
        </tr>
    </table>
</form>
```




**Näide 7.4.2**

```
<?php

if (isset($_POST['action'])) {

    switch ($_POST['action']) {

        case 'new_user':

            if (userDataIsValid($_POST)) {

                saveNewUser($_POST);

                echo 'Dear '.$_POST['name'].'<br><br>';
                echo 'Your data was saved<br><br>';
                echo 'Username: '.$_POST['email'].'<br>';
                echo 'Password: '.$_POST['pass'];

            } else {

                displayRegistrationErrors();

            }

        break;

    }

}

function saveNewUser ($dataArray) {

    // uue kasutaja salvestamise loogika

}

function userDataIsValid ($dataArray) {
    return true;
    // andmete valideerimise loogika

}

function displayRegistrationErrors() {

    // valideerimise veade näitamine

}

?>
```

## 8.1 MySQL sissejuhatus

MySQL on populaarseim vabatarkvaraline SQL (Structured Query Language) andmebaasiserver maailmas. MySQL on tuntud eelkõige oma kiiruse,stabiilsuse ja töökindluse poolest. MySQL'i kasutavad näiteks Google, Yahoo, NASA ja teised tuntud tegelased. MySQL andmebaase kasutatakse peamiselt veebiserverites (aga samuti keegi ei keela MySQL andmebaaside kasutamist teise tüüpi tarkvaras). MySQL server on kirjutatud keeltes C ja C++ ning sellel on olemas API erinevatele programmeerimiskeeltele: C, C++, Java, Perl, PHP, Python, Ruby jne.

Andmebaas on peaaegu alati infosüsteemi tuum, milles organiseeritakse ja kus hoitakse infosüsteemi andmeid. On olemas erinevad andmebaaside tüübid:

- Lameandmebaasid (andmebaasid, mis koosnevad ainult ühest tabelist)
- Tekstiandmebaasid (nagu contacts.txt fail kodutöös 5 ja 6)
- Hierarhilised (andmebaasis leiavad kajastamist omaniku ja alluva suhted)
- Võrkmudel (sarnased hierarhiliste andmebaasidega, aga siin alluval võib olla mitu omanikku ja mitu alluvat)
- Relatsioonilised (objektid andmebaasis ja nendevahelised seosed esitatud tabelite kujul)
- Objekt-orienteeritud (võimaldab säilitada objekt-orienteeritud programmis loodud objekte)

MySQL on relatsiooniline andmebaas. Relatsioonilises andmebaasis hoitakse infot mingite objektide kohta (töötaja, kasutaja, auto, raamat jne.) Sellistest infoobjektidest saavad ühe baasi andmeobjektid ehk tabelid. Tabel esitab infot mingi ühetüübiliste objektide kohta - "raamatud", "isikud" oleksid tüüpilised andmebaasi tabelite nimed. Näiteks tabelis **raamatud** hoitakse informatsiooni raamatute kohta (autor, lehekülgede arv, köide jne.). Tabelis **isikud** asub informatsioon isikute kohta (nimi, sugu, sünniaasta, telefon jne.). Tabelid koosnevad ridadest ja veergudest.

| nimi | sugu | sünniaasta | telefon | |
|---|---|---|---|---|
| | | | | ← päis |
| Andres | mees | 1981 | 5341791 | |
| Reet | naine | 1967 | 5126781 | ← rida (üks kirje, üks konkreetne isik) |
| Ilmar | mees | 1994 | 5456172 | ← atomaarne andmeelement |

↑ veerg (atribuut)

Tabeli igas veerus asuvad sama tüüpi andmed (samad objekti omadused, näiteks nimi, sugu, sünniaasta ja telefoni number). Igas reas asub unikaalne objekt (väärtuste poolest) - meie näides on 3 unikaalset isikut. Veerguda ja ridade järjekord pole oluline, sest tabelis paiknev informatsioon ei sõltu järjekorrast. Iga lahtri sisu nimetakse atomaarseks andmeelemendiks - objekti andmete osa, mida ei saa enam alamandmeteks jagada.

### Võtmed

Nüüd vaatame, kuidas erinevaid tabeleid omavahel seotakse. Oletame, et meil on vaja luua raamatukogu andmebaasi. Teeme eraldi tabelid isikute, raamatute ja laenutamiste ajade jaoks:

| nimi | telefon |
|---|---|
| Andres | 5341791 |
| Reet | 5126781 |
| Ilmar | 5456172 |

isikud

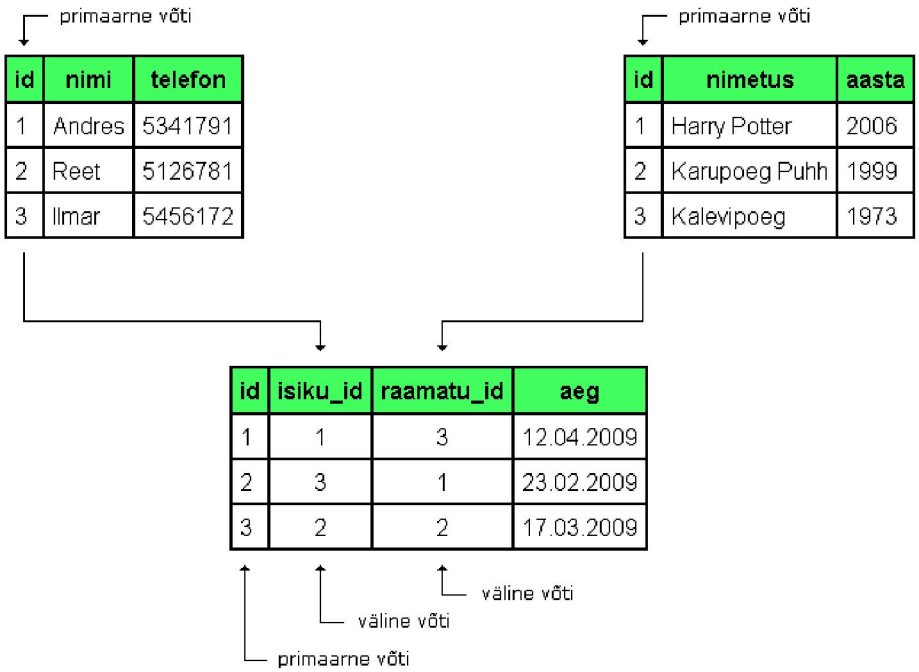
| nimetus | aasta |
|---|---|
| Harry Potter | 2006 |
| Karupoeg Puhh | 1999 |
| Kalevipoeg | 1973 |

raamatud

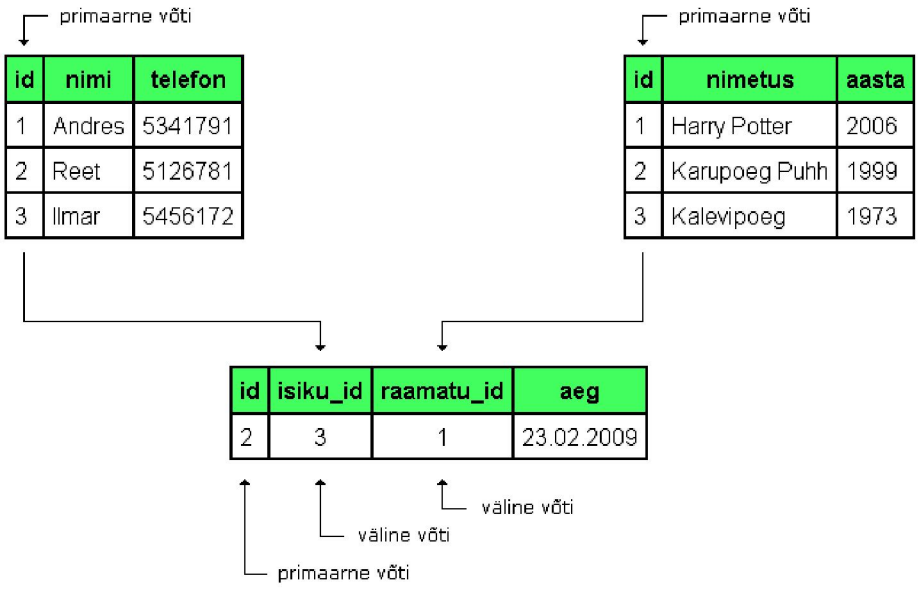
| aeg |
|---|
| 12.04.2009 |
| 23.02.2009 |
| 17.03.2009 |

laenutuse_ajad

Selleks, et teha kindlaks kes mida ja millal laenutas tuleb kuidagi seada need 3 tabelit omavahel. Selleks tehakse iga tabeli kirje jaoks unikaalset identifikaatorit. Sellist identifikaatorit nimetakse primaarseks võtmeks (primary key) - tavaliselt see on INT tüüpi väärtus. Tihti seadistakse tabeli primaarse võtme atribuuti nii, et iga lisatava kirje jaoks pannakse identifikaatoriks eelmise identifikaatori väärtust + 1 (sellist seadistust nimetakse auto_increment'iks). Mõnede tabelite puhul primaarne võti ei ole kohustuslik, aga selle lisamine on alati soovitatav. On olemas ka üks teine identifikaator, mis seob teise tabeli rea antud tabeliga - seda nimetakse välisvõtmeks (foreign key). Lisame tabelitesse vajalikud võtmed:




Nüüd on seosed paigas ja saame aru, et Andres laenutas raamatu "Kalevipoeg" 12.04.2009, Reet - raamatu "Karupoeg Puhh" 17.03.2009 ning Ilmaril on "Harry Potter" 23.02.2009. Juhul, kui Reet ja Andres toovad oma raamatud tagasi ning meil pole vaja laenutuste ajalugu hoida - kustutame vastavad read tabelist laenutuse_ajad:




## Andmetüübid

Iga tabeli atribuutidel on kindel andmetüüp, mille valik sõltub veeru andmetest, ehk kas hoitakse seal teksti, numbri või hoopis kuupäeva. Kõikide admetüüpidega võib huvi korral tutvuda MySQL veebilehel. Põhilised ja tihti MySQL-is kasutatavad andmetüübid on järgmised:

### INT

Täisarvtüüpi andmete hoidmiseks (võib lisaks defineerida kas atribuut on alati positiivne (unsigned) või mitte)

### DOUBLE

Ujukomaarvtüüpi andmete hoidmiseks (võib lisaks defineerida kas atribuut on alati positiivne (unsigned) või mitte)

### VARCHAR

Tekstiliste andmete hoidmiseks (tavaliselt mitte pikem, kui 255 sümbolit)

### TEXT

Tekstiliste andmete hoidmiseks (tavaliselt suured tekstid nagu uudised, kommentaarid jne)

### DATE

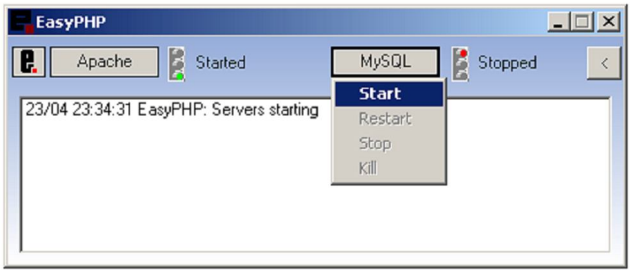
Kuupäevade hoidmiseks. Andmeid hoitakse kujul AAAA-KK-PP. Näiteks esimene detsember 2006 on "2006-12-01"

**DATETIME**

Kuupäevade hoidmiseks koos kellaaega. Andmeid hoitakse kujul AAAA-KK-PP TT:MM:SS. Näiteks esimene detsember 2006 kell 14:15 on "2006-12-01 14:15:00"

### Andmebaasi server

Vaikimisi serveri port on 3306. Serveri käivitamine EasyPHP abil on väga lihtne:



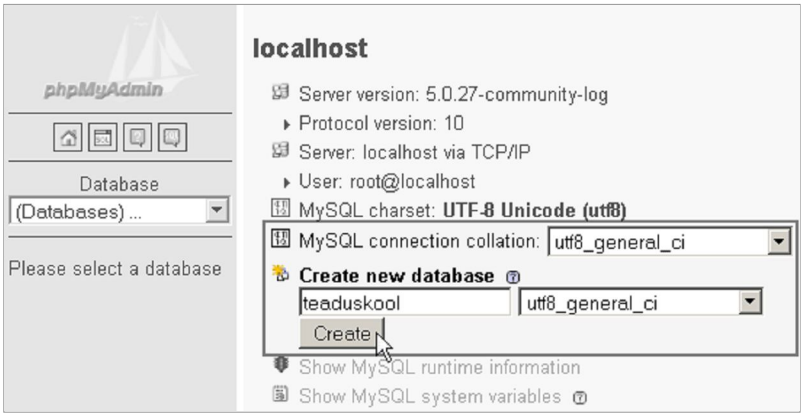

---

## 8.2 phpMyAdmin

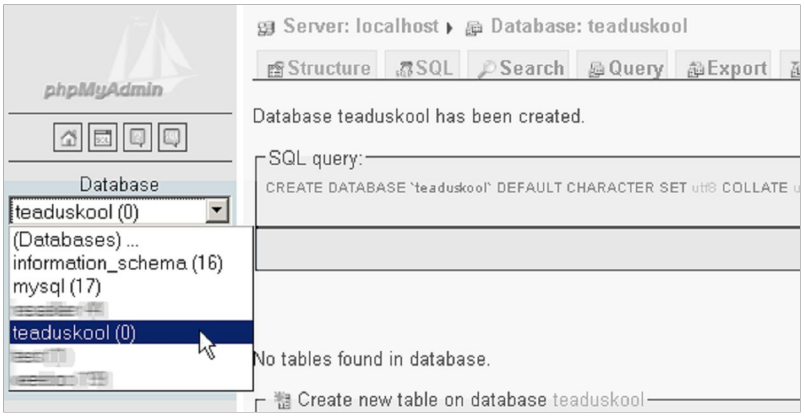
Andmebaasi serveri võib hallata käsurealt või spetsiaalselt selleks loodud tarkvara abil. Antud kursuse raames kasutame PHP's kirjutatud open source projekti PhpMyAdmin. See käib koos EasyPHP tarkvaraga ja saate sellele liigipääsu aadressil http://127.0.0.1/home/mysql. See on väga mugav ja populaarne tarkvara MySQL andmebaaside administreerimiseks. Vaatame, kuidas võib raamatute laenutuste andmebaasi luua PhpMyAdmin'is. Lisaks pange tähele, et seal alati näidetakse SQL päringut, mis sai käivitatud peale teie tegevust - see võib aidata SQL keele õppimisel.

### Admebaasi loomine

Sisesage uue andmebaasi nimi (meie juhul **teaduskool**), valige collation'iks **utf8_general_ci** ning vajutage **Create**:



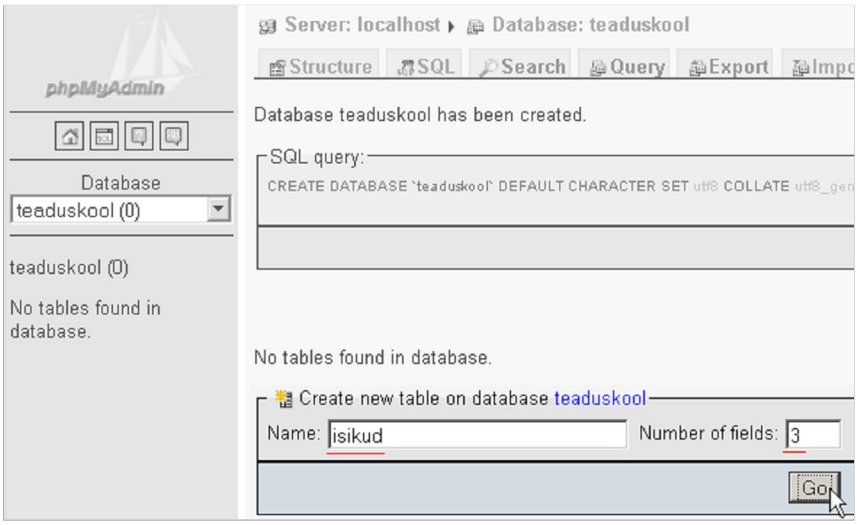

Tööks vajaliku andmebaasi valimine käib vasakul paikneva valikmenüü abil:



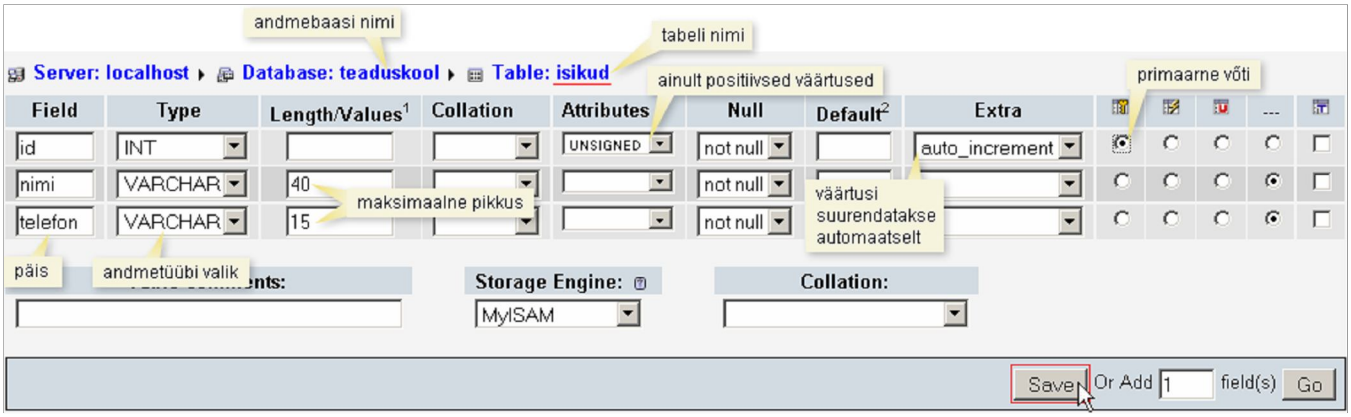

Nüüd meil on olemas tühi andmebaas **teaduskool**, kus praegu pole ühtegi tebelit.

### Tabelite loomine

Valige admebaasi **teaduskool**, leidke see koht, kus on kirjutatud "Create new table on database teaduskool" ning pange selle nimeks **isikud** ja veergude arvuks **3**:




Nüüd tuleb sisestada tabeli veergude nimed, andmetüübid jms:

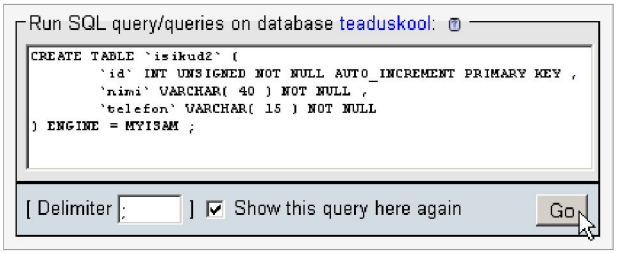



Pange tähele, et peale **Save** vajutamist ekraanil ilmus SQL päring:

```
CREATE TABLE `isikud` (
   `id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
   `nimi` VARCHAR( 40 ) NOT NULL,
   `telefon` VARCHAR( 15 ) NOT NULL
) ENGINE = MYISAM ;
```

Seda päringut võib edaspidi kasutada, kui peame sellist andmebaasi uuesti looma kas PhpMyAdmin'is, käsureaalt või PHP koodis. Kopeerige see päring ja avage PhpMyAdmin'i SQL päringute aken:


Sisestage kopeeritud päring (aga `isikud` asemel pange nimeks `isikud2`) ning vajutage **Go**:




Nüüd vasakul võib näha, et meie andmebaasis on juba 2 tabelit (isikud ja isikud2).

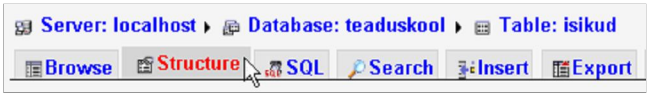
## Tabelite kustutamine

Kustutame tabel **iskud2** maha. Selleks valige vasakul tabel **isikud2** ning vajutage **Drop**


```
DROP TABLE `isikud2`;
```

## Tabelite struktuuri muutmine

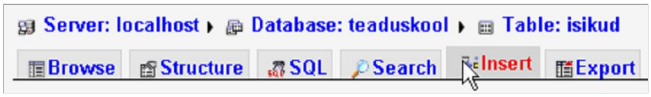
Tabeli struktuuri muutmiseks valige tabel **iskud** ja vajutage **Structure**:



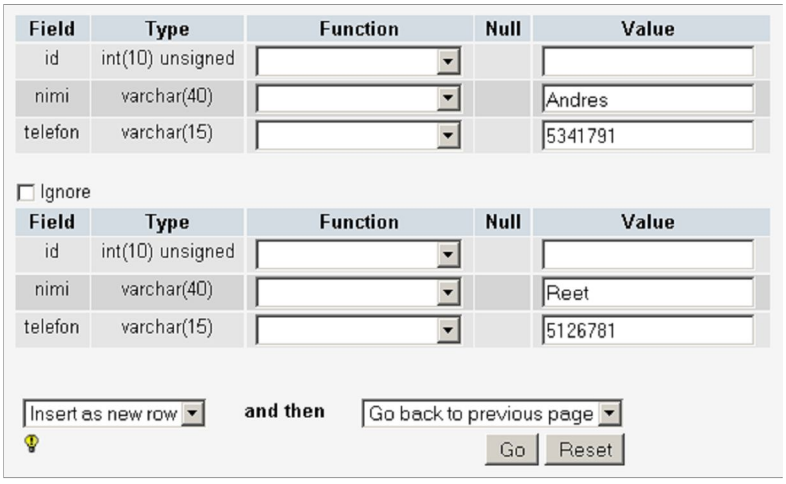

Nüüd saate tabelis **iskud** igasuguseid struktuuri muutuseid teha:


## Andmete sisestamine

Valige tabel **isikud** ning vajutage **Insert**:



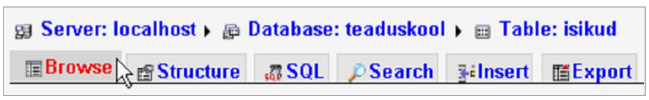

Nüüd võime sisestada andmeid. **id** väärtus jääb sisestamata, sest me oleme selleks määranud **auto_increment** ja seda sisestatakse automaatselt (1, 2, 3, 4, ...). Juhul, kui soovite lisada kaks kirjet korraga, siis selleks on mõeldud alumine vorm (ainuke märkus - ärge unustage Ignore checkbox'ist linnukest eemaldada). Kui admed on sisestatud - vajutage **Go**:



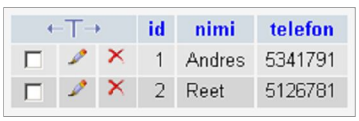

```
INSERT INTO `isikud` (`id`, `nimi`, `telefon`)
VALUES
    (NULL , 'Andres', '5341791'),
    (NULL , 'Reet', '5126781');
```

## Andmete vaatamine

Valige tabel **isikud** ning vajutage **Browse**:




```
SELECT * FROM `isikud` LIMIT 0, 30;
```




Näeme, et baasis on praegu 2 kirjet, mille **id** on 1 ja 2. Lisaks igal real on olemas 2 ikooni - muutmine ja kustutamine - nende abil saab

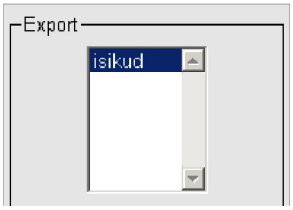
infot manipuleerida.

## Andmete eksport

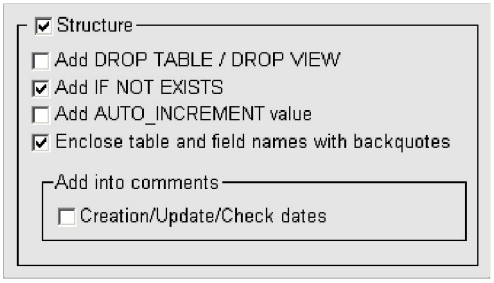
Valige **teaduskool** admebaas:


Vajutage **Export**:

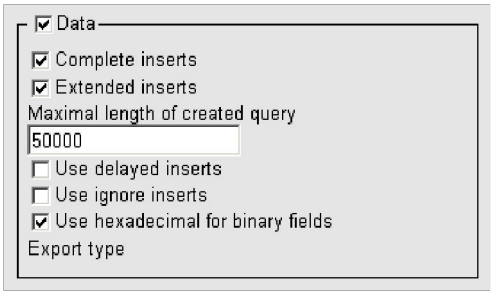

Vasakus nurgas valige tabeleid, mida soovite exportida (hoidke Ctrl, et valida mitu tabelit korraga):



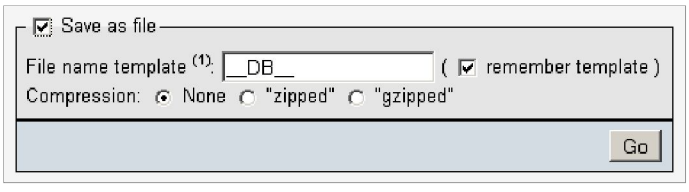

Kuna meil on tegemist väikeste admebaasidega, seadistused on järgmised:




Kui soovite ka tabeli kirju exportida:




Valige faili tüüp ja vajutage **Go**:




Eksport faili sees on lihtsalt SQL päringud, millega saab valitud tabeleid luua ja andmeid sisestada:

```
-- phpMyAdmin SQL Dump
-- version 2.9.1.1
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generation Time: Apr 25, 2009 at 09:41 AM
-- Server version: 5.0.27
-- PHP Version: 5.2.0
--
-- Database: `teaduskool`
--

-- --------------------------------------------------------

--
-- Table structure for table `isikud`
--

CREATE TABLE IF NOT EXISTS `isikud` (
   `id` int(10) unsigned NOT NULL auto_increment,
```

```
   `nimi` varchar(40) NOT NULL,
   `telefon` varchar(15) NOT NULL,
   PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

--
-- Dumping data for table `isikud`
--

INSERT INTO `isikud` (`id`, `nimi`, `telefon`) VALUES
   (1, 'Andres', '5341791'),
   (2, 'Reet', '5126781');
```

### Andmete import

Valige admebaas, millesse soovite andmeid importida. Vajutage **Import**, valige fail ja **Go**.

---

## 8.3 SQL päringud

SQL keeles on palju päringu tüüpe ja erinevaid konstruktioone. Vaatleme aga põhilised päringud.

- **INSERT** - andmete lisamine tabelisse
- **UPDATE** - andmete muutmine tabelis
- **DELETE** - andmete kustutamine tabelist
- **SELECT** - andmete pärimine tabelist

### INSERT

Insert lausega saab andmeid lisada. Elementaarne SQL insert lause näeb välja järgmiselt:

```
INSERT INTO tabeli_nimi (veeru_1_nimi, veeru_2_nimi, veeru_3_nimi, ...)
VALUES (veeru_1_väärtus, veeru_2_väärtus, veeru_3_väärtus, ...);
```

Teksti stringidele tuleb lisada ühe- või kahekordsed jutumärgid. Juhul, kui kirjutate arvulise tüübi väärtust jutumärgides - ei juhtu ka midagi.

**Näide:**

```
INSERT INTO isikud (nimi, telefon)
VALUES ('Priit', '6134515');
```

### UPDATE

Insert lausega saab andmeid uuendada. Selleks, et määrata mis kirjet või kirjede hulka on vaja uuendada, kasutatakse **where** süntaksit. Update üldine süntaks on järgmine:

```
UPDATE tabeli_nimi
SET
   veeru_1_nimi = veeru_1_uus_väärtus,
   veeru_2_nimi = veeru_2_uus_väärtus,
   veeru_3_nimi = veeru_3_uus_väärtus,
   ...
WHERE ...
```

Where lauses võib kasutada MySQL operaatoreid ja funktsioone, neid võib lähemalt uurida siin.

**Näide:**

Esialgne tabel:

| id | nimi | telefon |
|---|---|---|
| 1 | Andres | 5341791 |
| 2 | Reet | 5126781 |
| 3 | Ilmar | 5456172 |
| 4 | Anton | 5409182 |
| 5 | Ivo | 5871001 |

```
UPDATE isikud
SET nimi = 'Kadi'
WHERE id = 5;
```

Uuendatud tabel:

| id | nimi | telefon |
|---|---|---|
| 1 | Andres | 5341791 |
| 2 | Reet | 5126781 |
| 3 | Ilmar | 5456172 |
| 4 | Anton | 5409182 |
| 5 | Kadi | 5871001 |

```
UPDATE isikud
SET
   nimi = 'Margo',
   telefon = '6732190',
WHERE nimi = 'Andres' AND telefon = '5341791';
```

Uuendatud tabel:

| id | nimi | telefon |
|---|---|---|
| 1 | Margo | 6732190 |
| 2 | Reet | 5126781 |
| 3 | Ilmar | 5456172 |
| 4 | Anton | 5409182 |
| 5 | Kadi | 5871001 |

```
UPDATE isikud
SET telefon = '6010100',
WHERE nimi = 'Reet' OR telefon = '5456172';
```

Uuendatud tabel:

| id | nimi | telefon |
|---|---|---|
| 1 | Margo | 6732190 |
| 2 | Reet | 6010100 |
| 3 | Ilmar | 6010100 |
| 4 | Anton | 5409182 |
| 5 | Kadi | 5871001 |

```
UPDATE isikud
SET telefon = '55555555',
WHERE nimi IN ('Reet', 'Anton', 'Margo');
```

Uuendatud tabel:

| id | nimi | telefon |
|---|---|---|
| 1 | Margo | 55555555 |
| 2 | Reet | 55555555 |
| 3 | Ilmar | 6010100 |
| 4 | Anton | 55555555 |
| 5 | Kadi | 5871001 |

## DELETE

Delete lausega saab andmeid kustutada. Selleks, et määrata mis kirjet või kirjede hulka on vaja kustutada, kasutatakse samuti **where** süntaksit. Delete üldine süntaks on järgmine:

```
DELETE FROM tabeli_nimi
WHERE ...
```

**Näide:**

Esialgne tabel:

| id | nimi | telefon |
|---|---|---|
| 1 | Andres | 5341791 |
| 2 | Reet | 5126781 |
| 3 | Ilmar | 5456172 |
| 4 | Anton | 5409182 |
| 5 | Ivo | 5871001 |

```
DELETE FROM isikud
WHERE id = 3;
```

Uuendatud tabel:

| id | nimi | telefon |
|---|---|---|
| 1 | Andres | 5341791 |
| 2 | Reet | 5126781 |
| 4 | Anton | 5409182 |
| 5 | Ivo | 5871001 |

```
DELETE FROM isikud
WHERE id > 2;
```

Uuendatud tabel:

| id | nimi | telefon |
|---|---|---|
| 1 | Margo | 6732190 |
| 2 | Reet | 5126781 |

## SELECT

Andmete päringu tehakse Select lause abil. Elementaarne Select lause on järgmise süntaksiga:

```
SELECT veeru_1_nimi, veeru_2_nimi, ...
FROM tabeli_nimi
WHERE ...
```

Juhul, kui kirjutame

```
SELECT * FROM tabeli_nimi
WHERE ...
```

siis võetakse kõik veerud.

Kui on vaja tulemust sorteerida - võib kasutada ORDER BY süntaksit:

```
SELECT veeru_1_nimi, veeru_2_nimi, ...
FROM tabeli_nimi
WHERE ...
ORDER BY veeru_2_nimi sorteerimistüüp, veeru_4_nimi sorteerimistüüp, ...
```

Sorteerimistüüpideks on kas ASC või DESC.

**Näide:**

Tabel:

| id | nimi | telefon |
|---|---|---|
| 1 | Andres | 5341791 |
| 2 | Reet | 5126781 |
| 3 | Ilmar | 5456172 |
| 4 | Anton | 5409182 |

| | | |
|---|---|---|
| 5 | Ivo | 5871001 |

```
SELECT nimi FROM isikud
ORDER BY nimi ASC;
```

Tulemus:

| nimi |
|---|
| Andres |
| Anton |
| Ilmar |
| Ivo |
| Reet |

```
SELECT nimi, id FROM isikud
WHERE id >= 2 AND id <= 4
ORDER BY id DESC;
```

Tulemus:

| nimi | id |
|---|---|
| Anton | 4 |
| Ilmar | 3 |
| Reet | 2 |

```
SELECT * FROM isikud
WHERE id NOT IN (1, 3, 5)
ORDER BY name ASC;
```

Tulemus:

| id | nimi | telefon |
|---|---|---|
| 4 | Anton | 5409182 |
| 2 | Reet | 5126781 |

---

## 8.4 PHP ja MySQL

Failid: raamatukogu.zip, raamatukogu.sql

Kuidas PHP skript ja MySQL server omavahel suhtlevad vaatame raamatute laenutuste infosüsteemi näitel. Loome süsteemi, kus saab raamatuid lisada ja kustutada ning inimesi lisada ja kustutada ja lisaks ka määrata kes mida laenutas. Esialgul teeme oma andmebaas valmis. Loome uus andmebaas nimega **raamatukogu**. Meil läheb vaja 3 tabelit: **isikud**, **raamatud** ja **laenutused**. Importige neid PhpMyAdmin'is failist raamatukogu.sql.

Esialgul importige tabelid ja pakkige arhiveeritud failid lahti. Seejärel käivitage index.php, raamatud.php ja isikud.php ning katsetage nende tööd. Ärge unustage vaadata PhpMyAdmin'is kuidas muutub tabelite sisu vastavalt skripti päringutele. Peale seda uurige koodi. Edasi käib uue koodi seletamine.

Teeme eraldi fail db.php, kus asub ainuke funktsioon admebaasi ühenduse loomiseks:

db.php

```php
<?php

$link;

function connect_db() {

     global $link;

     $link = mysql_connect('localhost', 'root', '');
     if (!$link) {
    die('Could not connect: ' . mysql_error());
     }

     $db_selected = mysql_select_db('raamatukogu', $link);
     if (!$db_selected) {
         die ('Can\'t use database raamatukogu: ' . mysql_error());
```

```
    }

}

?>
```

**mysql_connect()** loob ühendust MySQL serveriga (esimene parameeter on serveri aadress, teine - kasutaja nimi ja kolmas - salasõna). Esialgse seadete kasutajatunnus on 'root' ja salasõna on tühi. Ühendus püsib kuni kogu skript lõpetab oma tööd või kutsutakse **mysql_close** funktsiooni. Laadime **db.php** faili skriptis kus läheb vaja andmebaasi ühendust ning seejärel käivitame **connect_db** funktsiooni.

Lisaks meil läheb vaja 3 klassi: Isik, Raamat ja Laenutus. Vaatleme Isik klassi funktsioone mis saadavad päringuid MySQL serverisse:

**Isik.php**

```
class Isik {

    private $id; // isiku id andmebaasis
    private $nimi;
    private $telefon;

    ...
    ...
    ...

    public function loadFromDatabase() {
        $result = mysql_query('
            SELECT `nimi`, `telefon`
            FROM `isikud`
            WHERE id = '.$this->id
        );
        $this->setData(mysql_fetch_assoc($result));
    }

    public function saveToDatabase() {
        mysql_query('
            INSERT INTO `isikud`
            (`nimi`, `telefon`)
            VALUES ("'.$this->nimi.'", "'.$this->telefon.'")
        ');
    }

    public function deleteFromDatabase(){
        mysql_query('
            DELETE FROM `isikud`
            WHERE id = '.$this->id.'
        ');
    }

}
```

Funktsioon **loadFromDatabase** pärib andmebaasist isiku andmed (nimi ja telefon) isiku id järgi. Seal kasutame **mysql_query** funktsiooni, mis tagastab viide (resurss) andmebaasist saadud tulemusele. Selleks, et neid admeid kätte saada kasutame **mysql_fetch_assoc**, mis tagastab tulemusest üks rida assotsiatiivse massivina (kujul ['veeru_nimi'] => 'veeru_väärtus') ja liigub oma pointerit edasi teisele reale jne kuni tulemuse ridade lõpuni. Kuna isiku andmete laadimisel id järgi tulemuseks peab tulema ainult üks rida (id on primaarne võti ja seega unikaalne) - pole mõtet **mysql_fetch_assoc** mitu korda käivitada.

Funktsioonid **saveToDatabase** ja **deleteFromDatabase** lisavad andmebaasi ja kustutavad andmebaasist isiku andmed samuti id järgi. Kuna antud juhul nende funktsioonide tulemused ei paku meile huvi - seega pole ka vaja neid töödelda.

Klassid Raamat ja Laenutus on andmebaaside päringute poolt peaaegu samad, mis Isik.

Nüüd vaatleme skripti isikud.php, mis võimaldab isikuid hallata:

**isikud.php**

```
...
...
...

if (isset($_POST['action']) && $_POST['action'] == 'add') {

    $isik = new Isik();
    $isik->setData($_POST);
    $isik->saveToDatabase();
    header('Location: isikud.php');
    exit;

} elseif (isset($_GET['delete']) && intval($_GET['delete']) > 0) {

    $isik = new Isik();
    $isik->setID(intval($_GET['delete']));
    $isik->deleteFromDatabase();
    header('Location: isikud.php');
    exit;

} else {

    $isikud = array();

    $result = mysql_query('
        SELECT *
        FROM `isikud`
        ORDER BY nimi ASC'
    );

    while ($rowData = mysql_fetch_assoc($result)) {
        $isik = new Isik();
```

```
        $isik->setData($rowData);
        $isikud[] = $isik;
    }

}

...
...
...
```

Juhul, kui on olemas $_POST andmed - see tähendab, et kasutaja lisas isiku läbi vormi, seega luuakse uus Isik objekt, sellel käivitakse meetod **setData** kuhu edastatakse $_POST andmed ja seejärel käivitatakse **saveToDatabase** meetod, mis salvestab isiku andmeid andmebaasi.

Juhul, kui on olemas $_GET delete parameeter - see tähendab, et kasutaja vajutas 'kustuta' - seega luuakse Isik objekt, tema id saab väärtust $_GET['delete'] ja seejärel käivitatakse **deleteFromDatabase** mis kustutab andmebaasist rea kus id võrdub Isik objekti id'ga.

Juhul, kui mingit tegevust ei toimunud - pärime andmebaasist kõik andmed kõikide iskute kohta ning sorteerime neid nimi järgi kasvavas järjekorras. Kuna tulemuses võib olla rohkem kui üks rida - kasutame while tsükli selleks, et käia läbi kõik tulemuse read ja luua iga rea andmete põhjal unikaalne Isik klassi objekt. Objekte omakorda salvestatakse massiivi $isikud, et pärast kasutajale näidata.

**raamatud.php** on sarnane **isikud.php** skriptiga.

Meil on jäänud viimane skript **index.php**, mille kõige huvitavam osa on:

index.php

```
...
...
...

} else {

    $laenutused = array();
    $laenutatudRaamatud = array();

    $result = mysql_query('
        SELECT *, TIMESTAMPDIFF(DAY, laenutuse_aeg, NOW()) AS duration
        FROM `laenutused`
        ORDER BY `laenutuse_aeg` DESC
    ');

    while ($rowData = mysql_fetch_assoc($result)) {

        $laenutatudRaamatud[] = $rowData['raamatu_id'];

        $isik = new Isik();
        $isik->setID($rowData['isiku_id']);
        $isik->loadFromDatabase();

        $raamat = new Raamat();
        $raamat->setID($rowData['raamatu_id']);
        $raamat->loadFromDatabase();

        $laenutus = new Laenutus();
        $laenutus->setIsik($isik);
        $laenutus->setRaamat($raamat);
        $laenutus->setDate($rowData['laenutuse_aeg']);
        $laenutus->setID($rowData['id']);
        $laenutus->setDuration(intval($rowData['duration']));

        $laenutused[] = $laenutus;
    }

    $isikud = array();

    $result = mysql_query('
        SELECT `id`, `nimi`
        FROM `isikud`
        ORDER BY `nimi` ASC'
    );

    while ($rowData = mysql_fetch_assoc($result)) {
        $isikud[$rowData['id']] = $rowData['nimi'];
    }

    $vabadRaamatud = array();

    if (count($laenutatudRaamatud) > 0) {
        $result = mysql_query('
            SELECT `id`, `nimetus`
            FROM `raamatud`
            WHERE `id` NOT IN ('.implode(',', $laenutatudRaamatud).')
            ORDER BY `nimetus` ASC'
        );
    } else {
        $result = mysql_query('
            SELECT `id`, `nimetus`
            FROM `raamatud`
            ORDER BY `nimetus` ASC'
        );
    }

    while ($rowData = mysql_fetch_assoc($result)) {
        $vabadRaamatud[$rowData['id']] = $rowData['nimetus'];
    }

}

...
...
```

```
...
```

Esialgul loome 2 massiivi: $laenutused - sellesse kirjutame Laenutus klassi objekte, ja $laenutatudRaamatud - siia salvestame kõik raamatute identifikaatoreid (id), mis on laenutatud, et hiljem kasutata neid päringu NOT IN (...) konstruktsioonis.

Pärime kõik andmed tabelist **laenutused** ning lisaks ka üks rida, kuhu kirjutatakse palju päevi on möödunud laenutuse kuupäevast tänani (tänase aja saamiseks on olemas MySQL sisseehitatud funktsioon **NOW()**) ja paneme sellele uuele reale nimeks **duration**. Nüüd while tsüklis käime kõik tulemuse read läbi ning moodustame Laenutus klassi objektide massiivi. Peale seda pärime isikuid tabelist **isikud**, et neid pärast select elemendis näidata. Nüüd pärime ka raamatuid, mille identifikaatorid ei leidu $laenutatudRaamatud - sest selectis näitame ainult neid raamatuid, mis pole veel laenutatud.

---

## 9.1 Turvaline php.ini konfigureerimine

PHP konfiguratsioon asub failis php.ini, mida igaüks saab oma arvutis välja otsida. Selle sees on hulk seadistusi, aga vaatame ainult kaks, mis võivad kaasa tuua tõsiseid probleeme.

**1. Register_Globals**: väga oluline seadistus (võib olla sisse või välja lülitatud (on/off)). Juhul kui register_globals on sisse lülitatud - siis PHP muudab $_ENV, $_GET, $_POST, $_COOKIE ja $_SERVER massivide elemente globaalseteks muutujateks, kus massiivi võtmed saavad muutujate nimedeks ja massiivi elementide väärtused - muutuja väärtusteks. See on väga kriitiline turvalisuse probleem. Oletame, et meie skript saab vormist POST päringu ja kontrollib, kas vormis täidetud kasutajanimi ja parool on õiged. Kui need on õiged, siis initsialiseeritakse muutujat $admin ning pannakse talle väärtuseks true. Edaspidi kontrollitakse kas muutuja $admin väärtus on true ning antakse administratiivsed privileegid:

```
if ($_POST['username'] == 'admin' && $_POST['password'] == 'abc123')
{
     $admin = true;
}

...
...
...

if ($admin)
{
     // give privilege to manage system users
}
```

Juhul, kui register_globals on sisse lülitatud ja me teame (või eeldame) muutuja $admin tähtsust - proovime seda sisestada (inject) meie programmis $_GET parameetri abil: http://www.example.com/script.php?admin=1, mis on ekvivaletne sellele koodile:

```
$admin = 1;

...
...
...

if ($_POST['username'] == 'admin' && $_POST['password'] == 'abc123')
{
     $admin = true;
}

...
...
...

if ($admin)
{
     // give privilege to manage system users
}
```

Nüüd on ilmselt selege, miks PHP turvalisuse eksperdid soovitavad lülitada register_globals alati välja.

**2. Display_errors**: seadistab, kas veateateid väljastatakse või mitte. Väga mugav arenduse ja testimise etapil, kuigi toob turvalisuse probleeme avalikus veebis. Veateade järgi saab teada palju infot failisüsteemist ja arhitektuurist ning samuti ka programmi loogikast, mis ei ole just see info, mida te sooviksite potentsiaalsele ründajale edastada. Seega soovitatakse display_errors live-saitidel välja lülitada ning suunata kõik veateateid logisse (log_errors = on).

## 9.2 Skriptide laadimine $_GET parameetrite põhjal

Oletame, et meie programm laadib PHP skripte vastavalt sellele, mis on $_GET parameetri X väärtus:

```
include_once($_GET['X'].'.php');
```

**http://www.example.com/index.php?X=product_details** - laaditakse faili product_details.php
**http://www.example.com/index.php?X=products_list** - laaditakse faili products_list.php
**http://www.example.com/index.php?X=http://www.bad.com/hack** - laaditakse faili hack.php domeenist bad.com

Viimase punkti käitumine ei ole soovituslik ning on tõsine turvaauk ja kes teab mis hack.php sees võib olla ja mis tagajärjed selle käivitamisega kaasnevad. Turvaaugu vältimiseks on mitu võimalust. Üks on kontrollida, kas fail, mida tuleb laadida asub eelnevalt defineeritud kataloogis ja laadida seda ainult positiivse vastuse juhul.

```
$includesPath = 'includes/products/';

...
...
...

if (file_exists($includesPath.$_GET['X'].'.php'))
{
     include_once($_GET['X'].'.php');
}
```

Teine variant on defineerida massiiv faili nimedega, mida võib $_GET parameetriga laadida.

```
$includesList = array('product_details', 'products_list', 'product_images');

...
...
...

if (in_array($_GET['X'], $includesList))
{
     include_once($_GET['X'].'.php');
}
```

## 9.3 Kasutaja sisendi kontrollimine

Mitte kunagi ei saa loota kasutaja sisendi peale. Tuleb alati valideerida informatsiooni, mis on sisestatud kasutaja poolt. Näiteks meil on olemas skript:

```
$a = $_GET['a'];
$b = $_GET['b'];
echo $a / $b;
```

Kõik töötab nii nagu me planeerime juhul kui a ja b on arvud ja b ei võrdu nulliga. Aga kui urliks on http://www.example.com /script.php?a=test&b=fail - siis enne jagamist argumente muudetakse automaatselt int tüübideks, ehk siis $a / $b saab 0 / 0 ning ekraanile ilmub hoiatuse veateade, mis on seotud sellega, et nulliga jagamine on keelatud. Nüüd oletame, et oleme kirjutanud külaliste raamatu, kus iga veebilehe külastaja saab oma tagasisidet kirja panna ning peale seda tema postitus saab nähtavaks külaliste raamatus:

```
if (isset($_POST['action']) && $_POST['action'] == 'add_guestbook_entry')
{
     mysql_query("INSERT INTO guestbook_entries VALUES ('{$_POST['entry_text']}')");
}

$result = mysql_query('SELECT entry_text FROM guestbook_entries');

while ($entry = mysql_fetch_row($result))
{
     echo $entry['entry_text'].'<br />';
}
```

Lisame külaliste raamatu järgmise teksti: <script>alert('hello world!');</script>. Nüüd lehel, mis näitab teateid visatakse javascripti alert teadet, mis on tegelikult kahjutu, aga kui ma kirjutan <script>window.location="http://www.google.com;"</script> - siin on see juba rohkem kahjulik kuna redirektib minu külalisi teisele lehele. Samamoodi võib kirjutada ka kahjulikumaid skripte. Selleks, et seda vältida võiks kasutada PHP funktsiooni strip_tags(), mis kustutab sõnumist kõik html tägid ära.

```
if (isset($_POST['action']) && $_POST['action'] == 'add_guestbook_entry')
{
     mysql_query("INSERT INTO guestbook_entries VALUES ('".strip_tags($_POST['entry_text'])."')");
}

$result = mysql_query('SELECT entry_text FROM guestbook_entries');

while ($entry = mysql_fetch_row($result))
{
     echo $entry['entry_text'].'<br />';
}
```

## 9.4 Faili laiendi valimine

Alati pange oma PHP skriptidele laiend **.php**. Vanasti mõnedele abifunktsioonidega failidele anti laiendi .inc, mis tegi selgeks, et neid faile otse ei käivita vaid laaditakse skriptide sees. Aga juhul, kui veebiserver pole seadistatud nii, et brauser ei saa ligi kaustale nende failidega - siis ma saan minna lehele http://www.example.com/includes/db.inc ja näen brauseris skripti sisu, mille sees v]ib vabalt leida näiteks MySQL serveri parooli ja kasutajanime:

```
$mysqlData['host'] = '84.12.3.165';
$mysqlData['user'] = 'root';
$mysqlData['password'] = 'tghwj132';

...
...
...
```

```
$db = mysql_connect($mysqlData['host'], $mysqlData['user'], $mysqlData['password']);

...
...
...
```

See on võimalik sest veebiserver on vaikimisi seadistatud nii, et interpreteerib php skriptidena ainult faile .php laiendiga aga teisi faile lihtsalt saadab tagasi kliendi brauserile. Sama jutt käib ka tihti konfigureerimiseks kasutatavale failile config.ini. Juhul, kui te pole kindel, et internetist ei saa ligi kaustadele .ini, .inc jne failidega - ärge neid laiendeid kasutage!

## 9.5 Salasõna hoidmine

Enamuses veebi projektides tuleb luua kasutajate süsteemi selleks, et külastajad saaksid veebilehel registreerida ja pärast ka oma kontosse sisse logida. Isiku autentsuse kontrollimiseks on harilikult vaja kasutajanime ja parooli. Kasutaja andmeid on mõitlik hoida andmebaasi tabelis. Tihti noored arendajad kirjutavad paroole avalikul kujul nii, et juhul kui keegi saab juurdepääsu andmebaasile - siis kohe saab kätte ka kasutajanimede ja salasõnade nimekirja, mis ei ole eriti hea.

Selleks, et salasõnade varastamist vältida - tuleb neid enne andmebaasi kirjutamist krüpteerida. Mis algoritmi kasutada krüpteerimiseks - see on juba projekti tehniliste nõuete kirjutajal, aga kõige populaarsemad on MD5 ja SHA1. Mõlemad on ühesuunalised - see tähendab, et krüpteeritute sõnumite taastamine (dekrüpteerimine) pole võimalik. md5() funktsioon genereerib hash'i pikkusega 32 sümbolit ning sha1() hash on 40 sümbolit pikk.

Isegi siis, kui salasõnad süsteemis on krüpteerimisega kaitstud - kurjategijal jääb võimalus leida kasutaja salasõna lihtsalt proovimise teel. Sellise olukorda vältimiseks kasutaja salasõna peab ka turvaline olema ja meie eesmärk on teda aidata. Esiteks võime piirata salasõna minimaalset pikkust, näiteks keelata registreerimisel salasõnu, mis on lühem, kui 8 sümbolit pikk, lisaks keelata kasutada salasõna mis on võrdne kasutajanimega või sisaldub kasutajanimes. Kui salasõna võib leida sõnaraamatust - siis parooli murdmine on mitu suurusjärku lihtsam - seega võime nõuda seda, et salasõnas peab olema vähemalt üks number.

**Näide X.1 - uue kasutaja lisamine andmebaasi**

```
// user registration form

$passwordHash = md5($_POST['password']);

mysql_query("INSERT INTO users (username, password) VALUES ('{$_POST['username']}', '$passwordHash')");
```

**Näide X.2 - autentimine**

```
// user login form

$passwordHash = md5($_POST['password']);

$result = mysql_query("SELECT * FROM users WHERE username = '"{$_POST['username']}"' AND password = '$passwordHash'");

if (mysql_num_rows($result) == 0)
{
     // return login error
}
else
{
     // implement login logic
}
```

Samuti MySQL'is on olemas funktsioonid ND5 ja SHA1 ning PHP asemel võib hash'i arvutamist jätta MySQL serverile:

- `mysql_query("INSERT INTO users (username, password) VALUES ('{$_POST['username']}', MD5('{$_POST['username']}'))");`
- `mysql_query("INSERT INTO users (username, password) VALUES ('{$_POST['username']}', SHA1('{$_POST['username']}'))");`