```
#!/usr/bin/perl -w

# muuttujat.pl by Tero Toivonen <tero.toivonen@cc.jyu.fi>
# Esimerkki perlin muuttujista. Tulostelee epämääräisesti muuttujien
# arvoja. Lisää aiheesta: perldoc perlvar

# use disclaimer # eli käyttäminen yms vapaasti omalla vastuulla.

# käyttö: muuttujat.pl

# strict-moduli käyttöön. Se "sitoo" muuttujat paikallisiksi, eikä eri
# ohjelmalohkoissa esitellyt muuttujat ole muualla
# käytössä. Muuttujien esittelu my- tai local-funktioilla.
use strict;

# Muuttujan esittely... ilman alustuksia.
my $muuttuja;

# Muuttujien esittelyitä... alustuksilla.
my $eka = 1;
my $toka = 2;

# Laskee muuttujien osamäärän, tulostaa tuloksen ja rivinvaihdon.
print $eka / $toka . "\n";

# Muuttujaan tekstiä...
$muuttuja = "tekstiä";
print "muuttuja: $muuttuja\n";

# Konkatenointi... toinen tapa: $muuttuja = $muuttuja . "plaaplaa"
$muuttuja .= " lisää tekstiä perään\n";
print("konkatenointi ja muuttuja nyt: $muuttuja");

# Muuttujaan numeroita.. "muuttujan tyyppi vaihtuu lennosta".
$muuttuja = 1234567890;
print "toinen tyyppi muuttujaan: $muuttuja\n";

# Taulukon esitteleminen ja tietoja alustuksessa.. Huom. @muuttuja
# viittaa koko taulukkoon $muuttuja[n] sen alkioon.
my @array = ("eka","toka");
print "array: @array\n"; # Taulukon alkiot yritetään konkatenoida,
                        # kun se on lainausmerkkien sisällä.
print "array: ".@array."\n"; # Vrt. tämä.. alkioiden lkm.

# Push-funktiolla taulukkoon lisää alkioita. Ne lisätään taulukon
# loppuun.
push(@array,"kolmas");
print "\"kolmas\" array-muuttujaan: @array ja taulukon toinen alkio: ".
  "\"$array[2]\"\n";

# Shift-funktiolla taulukon ensimmäinen alkio "pullautetaan" pois
# taulukosta. Vrt. aliohjelmakutsuissa shift, jolloin käytetään
# oletustaulukkoa @_.
print "shift arraysta (ensimmäinen): \"".shift(@array)."\" ja loput \"@array\" \n";

# $#taulukko kertoo viimeisen indeksin numeron..
$muuttuja = $#array;
print("arrayn viimeinen indeksi (lukumäärä -1): ", "$muuttuja\n");

$muuttuja = @array;
print "arrayn alkioiden lukumäärä: $muuttuja\n";
```



```
# Useamman dimensioiset taulukot vastaavasti kuin muissa
# ohjelmointikielissä, paitsi ettei tyypeillä ole väliä
# perlissä. Taulukon alkio voi olla mitä tyyppiä vaan. Esimerkiksi
# tässä $array2[1][1] on eo taulukko.
my @array2;
$array2[0][0]="jotain";
$array2[0][1]="juttua";
$array2[1][0]=234;
$array2[1][1]=\@array;

print "print \@{\$array2[0]}: @{$array2[0]}\n";
print "\$array2[1][0]: $array2[1][0]\n";
print "\@{\$array2[1][1]}: @{$array2[1][1]}\n";


# Hajautustauluun (hashtable) viitataan %hajautustaulu ja sen
# "alkioon" $hajautustaulu{avain}.

# Esitellään hajautustaulu ja asetetaan kolme avain-arvo -paria.
my %hash = (
            '1' => 'yksi',
            '2' => 'kaksi',
            '3' => 'kolme',
           );

# Asetetaan muuttujaan hash:n arvo, kun avain on "1" ja tulostetaan
# se. Kyseinen arvo voitaisiin tulostaa myös suoraan eli muuttujaan
# sitä ei käyttämistä varten tarvitse erikseen laittaa.
$muuttuja = $hash{1};
print "\$hash{1}: $muuttuja\n";

# Asetetaan muuttujaan viittaus hash:iin. Tästä tulostuksesta ei
# seuraa mitään kaunista...
$muuttuja = \%hash;
print "\\%hash: $muuttuja\n";

# ... vaan se pitäisi tehdä näin, jos halutaan tulostaa hajautustaulun
# arvo. Lisää aiheesta: perlref ja perlreftut.
print "\$muuttuja->{'2'}: ".$muuttuja->{'2'}."\n";

# Hajautustaulun avaimet (vast. arvot) saa taulukoksi keys-funktiolla
# (vast. arvot values-funktiolla), jolloin niitä voi käsitellä kuin
# mitä tahansa taulukon alkioita.

# Seuraavan voisi tehdä myös: while(my ($avain,$arvo) = each
# %$muuttuja) { print "$avain => $arvo\n"; }

foreach my $avain (keys %$muuttuja) {
  $muuttuja->{$avain}="neljä" if $muuttuja->{$avain} eq "yksi";
  print "avain: $avain => arvo: ".$muuttuja->{$avain}."\n";
}
```

```
#!/usr/bin/perl -w

# aliohjelma.pl by Tero Toivonen <tero.toivonen@cc.jyu.fi>
# Esimerkki aliohjelmien käytöstä perlissä.

# use disclaimer # eli käyttäminen yms vapaasti omalla vastuulla.

# Käyttö: aliohjelma.pl <numero> <+-*/> <numero>
#         aliohjelma.pl <numero><+-*/><numero>

use strict;

# Aliohjelma laske
sub laske {
  my ($eka, $oper, $toka) = @_;

  # Hajautustauluun aliohjelmaviittaukset suoritettavan
  # laskutoimituksen mukaan.

  my %lasku=( "-" => \&vahennys,
              "+" => \&yhteen,
              "*" => \&kerto,
              "/" => \&jako
            );

  # Haetaan viittaus oikeaan aliohjelmaan, jos haluttua operaatiota ei
  # löydy hajautustaulusta, se palauttaa 'undef' eli $oikea_aliohjelma
  # ei ole tämän jälkeen määritelty.

  my $oikea_aliohjelma = $lasku{$oper};

  if(defined $oikea_aliohjelma) {

    # Jos aliohjelma löytyy, suoritetaan se ja parametreiksi
    # ensimmäinen ja kolmas komentoriviltä saatu (tai pilkottu)
    # parametri.

    my $tulos = (&{$oikea_aliohjelma}($eka,$toka));
    return $tulos;
  } else {
    # Pieleen meni.
    # Tämä vastaa esimerkiksi javan "return null;" -lausetta.
    return undef;
  }
}


# Komentorivin parametrit saadaan @ARGV-muuttujasta. Jos halutaan
# tehdä ohjelma, joka käyttää laajemmin parametreja niin kannattanee
# käyttää parametrien hallintaan Getopt::* -moduleja, koska muuten
# parametrilistojen hallinta on kohtuullisen työlästä. Esimerkiksi:
# kuinka parsitaan *nix:ien ls -komennon parametrit: ls -la == ls -al
# == ls -a -l == ls -l -a == ls --all -l ja niin edellen.

# Tämän ohjelman "main"-funktio ...

# Testataan parametrien lukumäärä
if(@ARGV == 3 or @ARGV == 1) {
  my ($lasku,$tulos);

  # Jos parametreja yksi, niin testataan onko se muotoa
  # <numero><merkki><numero>, missä <numero> on mikä vaan numero (\d
```



```
  # regexpeissä) ja niitä numeroita on vähintään yksi (+ regexpissä)
  # ja <merkki> on joku näistä: "+-*/". Lisäksi ne on ryhmitelty
  # suluilla jolloin "osumat" saadaan suoraan oletusmuuttujiin $1...$9
  # löytymis- järjestyksessä. Kenoviivalla "suojataan" erikoismerkit
  # tai "merkitään" tietyntyyppisiä merkkejä. Perldoc:t perlre ja
  # perlretut kertovat lisää..

  if(@ARGV == 1 and $ARGV[0] =~ /(\d+)([\-\+\*\/])(\d+)/ ) {

    # Jos ennettu parametri on oikeaa muotoa, niin kutsutaan
    # aliohjelmaa "laske" ja sen palauttama tulos laitetaan
    # muuttujaan. Esim. kirjaimet parametreina eivät nyt kelpaa.

    $tulos = &laske(($1,$2,$3));

    # Vain ohjelman tulosteen siistimistä..
    $lasku = "$1 $2 $3";

    # Tapaus 3 parametria. Tässä jokainen parametri testataan
    # erikseen.
  } elsif($ARGV[0] =~ /\d+/ and
          $ARGV[1] =~ /[-+\*\/]/ and
          $ARGV[2] =~ /\d+/) {

    # Parametrit ovat jo oikeassa järjestyksessä, ne ovat
    # oikeanmuotoisia ja niitä on oikea määrä, jolloin
    # laske-aliohjelmaa voidaan kutsua suoraan parametrit välittävällä
    # taulukolla.
    $tulos = &laske(@ARGV);
    $lasku = "@ARGV";
  } else {
    # Jotain väärää parametreina..
    print "väärä syöte\n";
    exit 1;
  }

  # Tulostellaan lasku ja sen tulos, jos tulos-muuttuja on määritelty.
  print "$lasku = $tulos\n" if defined $tulos;
} else {
  print "plaah, syötit @ARGV ja ".@ARGV." parametria\n";
  exit 1;
}
exit 0;

# Laskurutiinit sisältävät aliohjelmat:

sub yhteen { my ($eka, $toka) = (shift,shift); return $eka + $toka; }
sub vahennys { my ($eka, $toka) = (shift,shift); return $eka - $toka; }
sub kerto { my ($eka, $toka) = (shift,shift); return $eka * $toka; }
sub jako { my ($eka, $toka) = (shift,shift); return $eka / $toka; }

# Parametrit välitetään aliohjelmalle oletustaulukossa @_, josta
# shift-funktiolla otetaan 2 ensimmäistä muuttujiin.
```

```
#!/usr/bin/perl -w

# regexpeja.pl by Tero Toivonen <tero.toivonen@cc.jyu.fi>

# Esimerkki perlin säännöllisistä lauseista. Tulostelee epämääräisesti
# muuttujien arvoja. Lisää aiheesta: perlre ja perlretut.

# use disclaimer # eli käyttäminen yms vapaasti omalla vastuulla.

# käyttö: regexpeja.pl

use strict;

my $muuttuja = "jotain tekstiä ja muuta sellasta";
my $lkm=0;
my ($alku,@loppu);

# Ehtolauseessa, kun säännöllisen lauseen ehto täyttyy
if ($muuttuja =~ /teksti/ ) { #tulostaa löytyi.
#if ($muuttuja =~ /foo/ ) {   #tulostaa ei löytynyt.
#if ($muuttuja =~/^\S+/ ) {   #tulostaa löytyi \S ==non-whitespace.
#if ($muuttuja =~ /^\s+$/ ) { #tulostaa ei löytynyt \s == whitespace,
                              #^ == rivin alku ja $ == rivin loppu.
#if ($muuttuja =~/^(.*\s+.*){3,}/g ) { # tulostaa löytyi.
#if ($muuttuja =~ /^(.*\s+.*){5,}/g ) { #tulostaa ei löytynyt.
  print "löytyi\n";
}else{
  print "ei löytyny\n";
}

# tr -funktiolla voi vaihtaa merkkien paikkaa.
# Tämä vaihtaa kaikki j-kirjaimet X-kirjaimiksi ja a:t Y:ksi.
$lkm = $muuttuja =~ tr/ja/XY/;
print "muutoksia: $lkm\nmuuttuja nyt: \"$muuttuja\"\n";

$lkm = $muuttuja =~ tr/XY/ja/;
print "muutoksia: $lkm\nmuuttuja nyt: \"$muuttuja\"\n";

# rot13-"kryptaus" tr:llä.

$lkm = $muuttuja =~ tr/[a-zA-Z]/[n-za-m][N-ZA-M]/;
print "muutoksia: $lkm\nmuuttuja nyt: \"$muuttuja\"\n";

# ajetaan sama uudestaan muuttujalle -> pitäisi olla alkuperäinen...
$lkm = $muuttuja =~ tr/[a-zA-Z]/[n-za-m][N-ZA-M]/;
print "muutoksia: $lkm\nmuuttuja nyt: \"$muuttuja\"\n";

# split -funktiolla katkaistaan muuttuja annetun säännöllisen lauseen
# kohdalta. Jos "katkaisukohtaa" halutaan mukaan, niin sen ympärille
# on laitettava kaarisulut. Tässä katkaistaan $muuttuja sanan "ja"
# kohdalta.

$lkm = ($alku,@loppu) = split(/ja/, $muuttuja);
print "$lkm split \"ja\" alku: $alku + loppu: @loppu\n";

# Nyt katkaistaan $muuttuja kirjaimien "t" kohdalta.
$lkm = ($alku,@loppu) = split(/t/, $muuttuja);
print "$lkm split \"t\" alku: $alku + loppu @loppu\n";

# Ja samat regexpin m-funktiolla (match).. Nämä eivät ole yleispäteviä
# split-funktion korvaajia.
$lkm = ($alku,@loppu) = $muuttuja =~ m/^(.*)ja(.*)/g;
```



```
print "$lkm regex \"ja\" alku: $alku + loppu @loppu\n";
$lkm = ($alku,@loppu) = $muuttuja =~ m/([^t]+)/g;
print "$lkm regex \"t\" alku: $alku + loppu @loppu\n";

# s-funktiolla (substitute) vaihdetaan muuttujan säännöllinen lause
# toiseen. Huom. etsimisessä ryhmitellyt "osumat" oletusmuuttujiin
# $1..$9, joita voi käyttää sujuvasti.

# Esim $x="foo"; $x=~s/foo/bar/; Niin $x=="bar".
# Tässä vaihdetaan lauseen osat ja-sanan kohdalta päikseen..

$muuttuja =~ s/^(.*)( ja )(.*)$/$3$2$1/;
print "$muuttuja\n";

# sed-tyyppinen edelliseen oletusmuuttujien osalta.. toimii
# perlissäkin, mutta jos on varoitus-optio päällä, niin tulkki
# valittaa:
# $muuttuja =~ s/^(.*)( ja )(.*)$/\3\2\1/;
```

```
#!/usr/bin/perl -w

# tiedosto.pl by Tero Toivonen <tero.toivonen@cc.jyu.fi>

# Esimerkki tiedostojen käyttämisestä perlillä. Lisää aiheesta:
# perldoc perlfunc, IO ja IO::*. Kannattanee käyttää noita
# IO-moduleja..

# use disclaimer # eli käyttäminen yms vapaasti omalla vastuulla.

# käyttö: tiedosto.pl
# Tämä ohjelma lukee syötteitä STDIN:stä (näppäimistö, putki, jne)
# Poistuminen ctrl-c ...

use strict;

# Autoflush päälle.. varmuuden vuoksi. perlfunc, perlipc ja perlport
# perldoc:it kertovat lisää..

$|=1;

# Luetaan STDIN:stä rivi kerrallaan oletusmuuttujaan $_. Tämän voisi
# tehdä myös: while(defined (my $rivi = <STDIN>)) { ja käsitellään
# $rivi -muuttujaa... }
while(<STDIN>) {

  # Palataan silmukan alkuun paitsi, jos syötteessä on syötteessä on
  # jotain muitakin kuin "whitespace" -merkkejä.
  next unless /\S+/;
  my @rivit;

  # Poistetaan mahdolliset rivinvaihdot rivin lopusta. chop -funktio
  # poistaa oletuksena viimeisen merkin ...
  chomp;

  # Jaetaan syöte komentoon ja parametreihin.
  my($komento,@params) = split(/\ /,$_);

  # Vastaava kuin $komento =~ /^dir$/i ..
  if(uc($komento) eq "DIR") {
    # Avataan hakemisto DIR-nimiseen tiedostokahvaan. Tiedostokahvan
    # nimi ei ole merkitsevä.
    opendir(DIR,$params[0]) or die "$params[0] ei voinu aukasta: $!\n";

    # Luetaan hakemiston sisältö (tiedostojen nimet) muuttujaan.
    @rivit = readdir(DIR);

    # Suljetaan tiedostokahva.
    closedir(DIR);
  }elsif(uc($komento) eq "CP") {

    # Avataan ensimmäisenä parametrina annettu tiedosto lukemista
    # varten FH-nimiseen tiedostokahvaan.
    open(FH,"<$params[0]") or die "$params[0] ei voinu aukasta: $!\n";

    # Avataan toisena parametrina annettu tiedosto kirjoittamista
    # varten FD-nimiseen tiedostokahvaan.
    open(FD,">$params[1]") or die "$params[1] ei voinu aukasta: $!\n";

    # Luetaan FH-tiedostokahvasta (tiedostosta)...
    while(<FH>) {
      #... ja kirjoitetaan se FD-tiedostokahvaan (tiedostoon)
```



```
      print FD $_;
    }

    # Suljetaan tiedostokahvat.
    close(FD);
    close(FH);
    $rivit[0]="$params[0] kopioitu tiedostoon $params[1]\n";
  }elsif(uc($komento) eq "CAT") {
    open(FH,"<$params[0]") or die "$params[0] ei voinu aukasta: $!\n";
    @rivit = <FH>;
    close(FH);

# Toinen tapa tehdä eo lukeminen... silmukan sisällä voi tietoa
# käsitellä niinkuin haluaa.
#   while(defined (my $rivi = <FH>)) { chomp($rivi);
#
#     print "$rivi\n";
#    }
  }

# Tulostuksia..
  if(defined $rivit[0]) {
    foreach my $rivi (@rivit) {
      chomp $rivi;
      print "$rivi\n";
    }
    undef @rivit;
  }
}
```

```
#!/usr/bin/perl -w

# testi.cgi by Tero Toivonen <tero.toivonen@cc.jyu.fi>
# Esimerkki perl-modulien käyttämisestä ja cgi-skriptistä.

# use disclaimer # eli käyttäminen yms vapaasti omalla vastuulla.

# käyttö: cgi-ohjelmana www-palvelimelta. http://osoite/polku/testi.cgi

use strict;
# moduli, joka huolehtii siita ettei kayteta maarittelemattomia
# muuttujia, vaatii 'my' funktion, jolla maaritellaan muuttujat lokaaleiksi
# lohkon tai nimiavaruuden sisalla. Mm. nopeuttaa tulkkaamista.
use Tulosta;
# Oma moduli, ja vaatii etta samasta alihakemistosta loytyy tiedosto
# 'Tulosta.pm'

# Esimerkkina 2 tapaa luoda uusi 'tulosta-tyyppinen' muuttuja.

my $t1 = new Tulosta("asia 1");
my $t2 = Tulosta->new("toinen juttu");

print "Content-type: text/html\n\n";

# tulostetaan mita on muuttujissa ennen muutosta
$t1->tulosta(*STDOUT); print "<br>";
$t2->tulosta(*STDOUT); print "<br>";

# suoritetaan katenointi
$t1->katenoi(" plaaah hiihoo");
$t2->katenoi(" joo ja nii");

# tulostetaan lopputulos
print "<br>katenoinnin jälkeen:<br><br>\n";
$t1->tulosta(*STDOUT); print "<br>";
$t2->tulosta(*STDOUT); print "<br>";

exit (0);
```

```
# Tulosta.pm by Tero Toivonen <tero.toivonen@cc.jyu.fi>
# Esimerkki omien modulien tekemisestä perlissä.

# use disclaimer # eli käyttäminen yms vapaasti omalla vastuulla.

# package maaraa mihin 'pakettiin' kuuluu vrt. use Tulosta
package Tulosta;

use strict;

# konstruktori jota kutsutaan kun luodaan
# $uusitallainen = new Tulosta("alkuarvo");
# voi olla eriniminenkin,mutta new on vakiintunut
sub new {
    my $this = shift;
    my $self = {};

    if(@_) { $self->{TIETO} = $_[0] }

# bless:lla saadaan 'olio' tietamaan missa paketissa se on.
    bless ($self, $this);
    return $self;
}


# tulostaa itsensa sinne minka saa parametrinaan, aliohjelmakutsu
# pitaa sisallaan sen itsensa, joka otetaan parametreista $self:iin.

sub tulosta {
    my ($self, $fh) = @_;
    print $fh "$self->{TIETO}" if ($fh and exists $self->{TIETO});
}

# katenoi aliohjelma lisaa merkkijonon 'olion' tiedon loppuun.
sub katenoi {
    my ($self, $lisays) = @_;
    $self->{TIETO} .= $lisays if ($lisays and exists $self->{TIETO});
}

# HUOM! Moduleissa totuusarvo palautettava != 0
1;
```

```
#!/usr/bin/perl -w

# dbtesti.cgi by Tero Toivonen <tero.toivonen@cc.jyu.fi>
# Esimerkki perl-modulien käyttämisestä ja cgi-skriptistä.

# use disclaimer # eli käyttäminen yms vapaasti omalla vastuulla.

# käyttö: cgi-ohjelmana www-palvelimelta. http://osoite/polku/dbtesti.cgi

use strict;

# Otetaan CGI-moduli käyttöön. Lisää ko. modulista perldoc CGI
use CGI;

# Otetaan pari omaa modulia käyttöön. Näistä ei ole sen kummempia
# dokumentaatioita :) Yleensä perldoc:it kirjoitetaan itse skriptiin
# ja aiheesta lisää perldoc perlpod ja perlpodspec.

use luettelodb;
use User;

# Uusi CGI-"olio". Moduleista ja niiden käytöstä lisää: perldoc
# perltoot.
my $q=new CGI;
my %users;

# Uusi tietokantamoduli-"olio".
my $sql = new luettelodb();

# Haetaan tietokanta-modulilta sinne varastoidut tiedot. Jokainen
# sivun lataaminen hakee tietokannasta tiedot uudelleen.
my @tyypit = $sql->hae_tyyppi();

my @uids;
my %nimet=("-" => "---Valitse---");

# Käydään tiedot läpi.
foreach my $tyyppi (@tyypit) {
  # $tyyppi on User-modulin esiintymä. Perldoc UNIVERSAL kertoo
  # erimodulien tunnistamisesta ajon aikana yms. Nyt pistetään
  # jokainen $tyyppi hajautustauluun myöhempää (nopeaa) käyttöä
  # varten.
  $users{"$tyyppi->{'uid'}"}=$tyyppi;

  # Tiedot pariin apumuuttujaan valikon rakentamista varten.
  push(@uids, "$tyyppi->{'uid'}");
  $nimet{"$tyyppi->{'uid'}"}="$tyyppi->{'etunimi'} $tyyppi->{'sukunimi'}";
}

# Tulostetaan (se näkyvä) sivu CGI-modulin avulla.
print
  $q->header,
  $q->start_html("Luettelo"),
  $q->h3("Otsikko tähän"),
  $q->start_form,
  $q->p("Luettelo"),
  $q->popup_menu(-name=>'valinta',
                 -values=>['-',@uids],
                 -labels=>\%nimet,
                 -default=>'---Valitse---'),
  $q->p($q->submit('Kattotaas!')),
  $q->end_form(),$q->hr;
```



```
my $haettu;

# Jos mukana on jotain parametreja (nyt: valittu jotain valikosta).
if($q->param) {
  # haetaan arvo muuttujaan.
  $haettu = $users{$q->param('valinta')};

  # Tulostetaan tulos html-tauluun. CGI-modulista olisi varmasti
  # löytynyt myös taulujen muodostamiseen apuja, mutta voi ne tehdä
  # itsekin kirjottamalla haluamansa STDOUT:iin.
  print "
<table border=1>
 <tr><td>
   $haettu->{'etunimi'} $haettu->{'sukunimi'}
 </td>";

  foreach my $numero (keys %{$haettu->{'numerot'}}) {
    print "<td>$haettu->{'numerot'}->{$numero}</td>\n";
  }
  print "
</td></tr>
</table>\n";
}

# Loppuun kiitokset ja kumarrukset.
print $q->end_html; print "\n";

# Cgi-ohjelman varsinkin on hyvä palauttaa jotain (0 == kaikki meni
# ok). Palautusarvoja on muutenkin hyvä käyttää, niin niitä skriptejä
# voi käyttää osana muita skriptejä ja niillä voi tehdä tarkistuksia
# ko. ohjelman onnistumisesta.
exit 0;
```

```
# Tulosta.pm by Tero Toivonen <tero.toivonen@cc.jyu.fi>
# Esimerkki omien modulien tekemisestä perlissä.

# use disclaimer # eli käyttäminen yms vapaasti omalla vastuulla.

# Tämä moduli on puhelinluettelo -esimerkin "User-luokka".

use strict;

package User;

my $uid;
my $etunimi;
my $sukunimi;
my %numerot;

sub new {
  my $proto = shift;
  my $class = ref($proto) || $proto;
  my $this = {};
  bless($this,$proto);
  return $this;
}

sub set_tiedot {
  my($this,$id,$etu,$suku) = @_;
  $this->{'uid'}=$id;
  $this->{'etunimi'}=$etu;
  $this->{'sukunimi'}=$suku;
}

1;
```

```
#!/usr/bin/perl -w

# luettelodb.pm by Tero Toivonen <tero.toivonen@cc.jyu.fi>
# Esimerkki perl-modulien tekemisestä.

# use disclaimer # eli käyttäminen yms vapaasti omalla vastuulla.

# käyttö: perl-ohjelman osana. Vrt. use luettelodb.pm

# Tämä ei ole lähellekään valmis saatika hyvä esimerkki, mutta näin
# tietokantaan saa yhteyden :-) Esimerkiksi perl-modulit kannattaisi
# nimetä isolla kirjaimella alkavaksi, tietokantahakuja suorittavat
# osuudet voisi ehkä kirjoittaa järkevämmin (yleiskäyttöisemmin) jne.

# Kerrotaan modulille mihin "pakettiin" se kuuluu.
package luettelodb;
use strict;

use User;

# Tietokantaliittymän ja tietokanta-ajuri -modulit käyttöön.
use DBI;
use DBD::Pg;

my $driver = "DBI:Pg";
my $dbname = "luettelo";
my $hostname = "localhost";
my $port=5432;

my $user="tietokantakäyttäjä";
my $pass="tietokantakäyttäjän_salasana";

# Luodaan tietokantayhteys...
my $conn = DBI->connect("$driver:dbname=$dbname;
            host=$hostname;
            port=$port",
                          "$user","$pass");

# "Konstruktori"
sub new {
  my $proto = shift;
  my $class = ref($proto) || $proto;
  my $this = {};
  bless($this,$proto);
  return $this;
}

# Aliohjelma henkilötietojen hakemiseen.
sub hae_henkilo {
  my $lause=shift;

  # Valmistellaan kysely... vrt. jdbc ja java.sql.PreparedStatement.
  # Jos tulee monta samanlaista kyselyä peräkkäin, voidaan valmistella
  # kyselyn runko erikseen ja kertoa arvot erikseen. Lisää infoa
  # perldoc DBI. Tässä esimerkissä tätä ominaisuutta ei oikeasti edes
  # käytetä ennenkuin päivitetään johonkin useampia samanmoisia
  # tietoja (käyttäjän lisäyksessä useampi numero yhdellä
  # kertaa). Numeroiden haut voisi myös toteuttaa tätä ominaisuutta
  # hyväksikäyttäen.

  my $kasittelija = $conn->prepare($lause)
    || die "pieleen meni .. suorita prepare .. ".
```



```
      $conn->errstr."\nSQL: $lause\n";

  # Suoritetaan kysely.
  $kasittelija->execute()
    || die "pieleen meni .. suorita execute .. ".
      $kasittelija->errstr."\nSQL: $lause\n";

  my (@userit, $user);

  # Haetaan kyselyn tulokset rivi kerrallaan taulukkoon, jossa
  # jokainen tieto on erillinen taulukon alkio.
  while(my @rivi = $kasittelija->fetchrow_array()) {

    # Luodaan uusi User-esiintymä ja laitetaan sille juuri haetut
    # tiedot. Tämän jälkeen laitetaan uusi käyttäjä users-taulukkoon.
    $user = User->new;
    $user->set_tiedot($rivi[0],$rivi[1], $rivi[2]);
    push(@userit,$user);
  }

  # Palautetaan taulukko, jossa alkioina on haetut käyttäjät.
  return @userit;
}


# Aliohjelma numeroiden hakuun.. Pääasiassa samanlainen kuin
# henkilötietojen hakuun käytetty aliohjelma. Tulos vain laitetaan
# hajautustauluun.

sub hae_numerot {
  my $lause=shift;
  my $kasittelija = $conn->prepare($lause)
    || die "pieleen meni .. suorita prepare .. ".$conn->errstr."\nSQL: $lause\n";
  $kasittelija->execute()
    || die "pieleen meni .. suorita execute .. ".$kasittelija->errstr."\nSQL: $lause\n";
  my($user, @rivi, %nrot);
  while(@rivi = $kasittelija->fetchrow_array) {
    $nrot{$rivi[0]} = $rivi[1];
  }
  return %nrot;
}

# Aliohjelma käyttäjien lisäämiseen... Vastaava kuin eo. aliohjelmat.

sub lisaa_henkilo {
  my $lause=shift;
  my $kasittelija = $conn->prepare($lause)
    || die "pieleen meni .. suorita prepare .. ".$conn->errstr."\nSQL: $lause\n";
  $kasittelija->execute()
    || die "pieleen meni .. suorita execute .. ".$kasittelija->errstr."\nSQL: $lause\n";
  my ($rivi, @rivit);
  while(my @rivit = $kasittelija->fetchrow_array) {
    $rivi = "@rivit";
  }
  return $rivi;
}

# Aliohjelma yhden henkilön hakuun... Tämä konsultoi hae_tyypit
# -aliohjelmaa sujuvasti, jos on aihetta.

sub hae_tyyppi {
  my ($this,$uid)=(shift, shift);
```

```
  if($uid !~ /\d+/) { return undef; }
  return (&hae_tyypit($this,$uid));
}

# Aliohjelma yhden tai useamman henkilön hakuun...

sub hae_tyypit {
  my ($this,$uid,@tyypit)=(shift, shift,undef);

  # Generoidaan sql-kyselylause ... HUOM! Puolipiste sql-lauseen
  # loppuun!
  my $sql="SELECT uid,etunimi,sukunimi FROM henkilo";
  if(defined $uid and $uid =~ /\d+/) {
    $sql .= " WHERE uid=$uid;";
  }else {
    $sql .= ";";
  }

  # Haetaan henkilöt tietokannasta ...
  @tyypit = (&hae_henkilo($sql));

  # Haetaan jokaiselle tyypille erikseen puhelinnumerot. Tämän voisi
  # varmaan tehdä myös tuossa edellisen haun aliohjelmassa..

  for(my $i=0;$i<@tyypit;$i++) {
    $sql="SELECT pid,numero FROM puhelin WHERE uid=$tyypit[$i]->{'uid'};";
    %{$tyypit[$i]->{'numerot'}} = &hae_numerot($sql);
  }
  return @tyypit;
}

# Aliohjelma uuden henkilön lisäämiseen tietokantaan.
sub lisaa_tyyppi {
  my $this = shift;
  my ($etunimi,$sukunimi) = (shift,shift);

  my $sql = "INSERT INTO henkilo (etunimi,sukunimi) VALUES ('$etunimi','$sukunimi'); SELECT CURRV
AL('henkilo_seq');";

  # lisaa -aliohjelma palauttaa uuden henkilön uid:n.
  my $uid = &lisaa("$sql");

  return $uid;
}

sub lisaa_numero {
  # Toteuttamatta ...
  ;
}

1;
```

Table of Contents