Repetition, datatyper

Skalärer:
- Rymmer ett värde (tal/sträng/referens)

Listor:
- En finit mängd av skalära värden
  - Lägga till element: push/unshift
  - Plocka ut element: pop/shift
  - Sortera: sort
  - Lista->sträng: split
  - Sträng->lista: join

Hasher

Perls sista datatyp.

En lista där värdena knyts ihop i par och där första elementet är ”nyckeln till” eller ”namnet med vilken man identifierar” det andra elementet. Första elementen i paren kallas ”keys” och de andra för ”value”. Listan är inte ordnad men paren är det!

Hashvariablers namn börjar med ett ”%”. Samma namngivningsregler som för skalär- och listvariabler. Dvs, de måste börja med en bokstav (a-z).

Exempel

```
%betyg = (’Lisa’,’G’,’Pelle’,’U’,’Anton’,’VG’);

# Detta innebär att Lisa har betyget ’G’,
# Pelle har ’U’ och Anton har ’VG’.


# För att göra dem mera läsbara kan man skriva:
%betyg = (
      ’Lisa’,’G’,
      ’Pelle’,’U’,
      ’Anton’,’VG’);
```


Hasher utgörs av skalär data (precis som listor). Både nyckeln och värdet ska alltså vara skalära!

Därför när vi ska ta fram ett värde från en hash så skriver vi:

```
$betyg{’Lisa’} # Detta uttryck har värdet ’G’
```

Observera:
- $ i början (det är ett skalärt värde!)
- {’Lisa’} (’Lisa’ är en nyckel i en hash!)

Om man vill hämta flera värden från en hash?

Om vi hämtar flera värden får vi en lista, därför skriver vi:

```
@betyg{’Lisa’,’Pelle’}
      # Detta uttryck har värdet (’G’,’U’)
```

Observera:
- @ i början (resultatet är en lista!)
- {’Lisa’,’Pelle’} (två av hashens nycklar!)

Man kan bygga upp en hash genom tilldelning av ett par i taget:

```
$betyg{’Lisa’} =’G’;
$betyg{’Pelle’} =’U’;
$betyg{’Anton’} =’VG’;
```

```
$namn = ’Lisa’;
$bet = ’G’;
$betyg{$namn} = $bet;
```

Eller som vi såg förut, allt på en gång:

```
%betyg = (
      ’Lisa’,’G’,
      ’Pelle’,’U’,
      ’Anton’,’VG’);
```

Eller genom att omvandla en lista till en hash:

```
@klass = (’Lisa’,’G’,’Pelle’,’U’,’Anton’,’VG’);
%betyg = @klass;
```

Eller om man har två listor:

```
@klass = (’Lisa’,’Pelle’,’Anton’);
@resultat = (’G’,’U’,’VG’);
for ($i=0, $i !> $#klass, $i++) {
        $betyg{$klass[$i]} = $resultat[$i];
}
```


Hashfunktioner

Man kan vända på ordningen så att nyckeln blir värdet och värdet blir nyckeln.

OBS! Detta kan bara göras om alla värden är olika!!!

```
%nmn_prsnr = (
        ’Lisa’,’780605-0225’,
        ’Pelle’,’780605-0236’);
%prsnr_namn = reverse %nmn_prsnr;
```

%prsnr_nmn kommer att vara
(’780605-0225’,’Lisa’,’780605-0236’, ’Pelle’)
eller
(’780605-0236’, ’Pelle’,’780605-0225’,’Lisa’)

Hashfunktioner

keys
Syntax:
@nycklar = keys(%hash);
ger: en lista med alla nycklar i hashen %hash!

---

values
Syntax:
@varden = values(%hash);
ger: en lista med alla värden i hashen %hash!

Hashfunktioner

delete - tar bort par från hashen

```
# Om Lisa slutar i klassen:
delete $betyg(’Lisa’);
# %betyg är nu (’Pelle’,’U’,’Anton’,’VG’);
```


Hashiteration

each är en iterationssats för hasher som returnerar alla par i hashen.

```
while( ($elev,$betyg) = each(%betyg) ) {
        print ”$elev har betyget $betyg.\n”;
}
```

ger:

```
        Lisa har betyget G.
        Pelle har betyget U.
        Anton har betyget VG.
```

(dock inte nödvändigtvis i den ordningen!)

Exempel

Frekvensräkning med hasher:

```
open(IN,$ARGV[0]);
while (<IN>) {
        $ord = chomp($_);
        $frekv{$ord} = $frekv{$ord} + 1;
}
while (($ord,$antal) = each(%frekv)) {
        print ”$ord finns $antal g/ggr.\n”;
}
```

Exempel

Frekvensräkning med hasher:

```
open(IN,$ARGV[0]);
while (<IN>) {
      $ord = chomp($_);
      $frekv{$ord}++;
}
while (($ord,$antal) = each(%frekv)) {
      print ”$ord finns $antal g/ggr.\n”;
}
```


litenfil:

```
hej
hur
mår
du
jag
mår
bra
hej
då
tack
ska
du
ha
```

Körning:

```
> mitt_prog litenfil
hej finns 2 g/ggr.
hur finns 1 g/ggr.
mår finns 2 g/ggr.
du finns 2 g/ggr.
jag finns 1 g/ggr.
bra finns 1 g/ggr.
då finns 1 g/ggr.
tack finns 1 g/ggr.
ska finns 1 g/ggr.
ha finns 1 g/ggr.
```


keys och *values* returnerar listor och passar därför att användas med *foreach.*

```
foreach $bet (values(%betyg)) {
      if ($bet eq ’U’) {
            $bet = 1;}
      elsif ($bet eq ’G’) {
            $bet = 3;}
      else {
            $bet = 5;}
}
```



```
foreach $elev (keys(%betyg)) {
      if ($betyg{$elev} eq ’U’) {
            $nr_betyg{$elev} = 1;
      }
      elsif ($betyg{$elev} eq ’G’) {
            $nr_betyg{$elev} = 3;
      }
      else {
            $nr_betyg{$elev} = 5;
      }
}
```


Utskrift av hasher i alfabetisk ordning:

```
open(IN,$ARGV[0]);
while (<IN>) {
      $ord = chomp($_);
      $frekv{$ord}++;
}

foreach $ord ( sort(keys(%frekv))) {
      print ”$ord finns $frekv{$ord} g/ggr.\n”;
}
```


litenfil:

```
hej
hur
mår
du
jag
mår
bra
hej
då
tack
ska
du
ha
```

Körning:

```
> mitt_prog litenfil
bra finns 1 g/ggr.
du finns 2 g/ggr.
då finns 1 g/ggr.
ha finns 1 g/ggr.
hej finns 2 g/ggr.
hur finns 1 g/ggr.
jag finns 1 g/ggr.
mår finns 2 g/ggr.
ska finns 1 g/ggr.
tack finns 1 g/ggr.
```

Funktioner

INDENTERA

- Använd autoindenterande texteditor
  - Unix, t.ex. Emacs
  - PC, t.ex. UltraEdit eller Editeur
- Använd TAB !!!

…eller räkna parenteser och indentera för hand!

Funktioner

- fördefinerade funktioner
  - chomp, print, lc, open, etc

- egna funktioner (subrutiner alias 'sub')
  - lösning av delproblem
  - ett basproblem = en funktion
  - uppdelning och organisation av koden
  - "återanvändning" av kod

Syntax

Hur skrivs subrutiner?

```
sub felmeddelande {
        print "Var nu snäll och ange värde som  uppfyller
                de beskrivna kraven!\n";
}
```

Hur anropas subrutiner?

```
print "Ange en siffran mindre än 10: ";
$a = <STDIN>;
if ($a>=10) {
        felmeddelande();
}
```


Subrutiner och globala variabler

```
sub skriv_ut_rad {
        print "Rad nr $i är: \n $rad\n";
        $utskr_rad = $i + 200;
}
```

```
while (<IN>) {
        $rad = chomp($_);
        if ($i == $utskr_rad) {
                skriv_ut_rad();
        }
        …
}
```


Subrutiner och parametrar

Men om man vill att en funktion ska påverka olika variabler vid varje anrop?

Då måste vi "skicka med" variabeln när subrutinen anropas. Precis som vi gjort när vi använt oss av fördefinierade funktioner.
Med 'chomp($a)' talar vi om att det är just värdet som hålls av variabeln $a som ska 'chompas' (få eventuell radmatning strippad). '$a' är här INPARAMETER till funktionen 'chomp'.

Subrutiner kan också ha inparametrar

```
print "Skriv en sträng så ska jag omvandla den lite åt dig:";
$a = <STDIN>;
$a = gemener($a);
print "Gemener: $a";
$a = versaler($a);
print "Versaler: $a";
$a = storB($a);
print "Stor begynnelse bokstav: $a";
```

Hur ser subrutinen ut då?

```
sub gemener {
        $_[0] =~ tr/A_ZÅÄÖ/a-zääö/;
        return $_[0];
}
```

Vad är $_[0]?? Vad är 'return'??

Inparametrar sätts in i lista @_!

$_[0] är INTE samma sak som $_!! $_[0] är första elementet i listan @_!!
'return' skickar tillbaka ett värde till anropet.

Flera inparametrar, nya return-värde

```
print "Ge mig ett tal: ";
$a = <STDIN>;
print "Ge mig ett tal till: ";
$b = <STDIN>;
chomp($a,$b);
$sum = addera_tva($a,$b);
print "Summan av $a och $b är $sum.\n";
$sum2 = addera_tva($a,$sum);
print "Summan av $sum och $a är $sum2.\n";
$stoerst = stoerst_av_tva($a,$b);
print "$stoerst är störst av $a och $b\n";
```


Vad returneras???

```
sub addera_tva {
        $s = $_[0] + $_[1];
        return $s;
}
```

```
sub stoerst_av_tva {
        if ($_[0] > $_[1]) {
                return $_[0];
        }
        else { return $_[1]; }
}
```


Ännu flera inparametrar

```
print "Ge mig flera tal åtskilda av mellanslag: ";
$tal = <STDIN>;
chomp($tal);
@tal= split(' ',$tal);
$tot_sum = addera_alla(@tal);
print "Summan av alla tal är $tot_sum.\n";
($minst,$stoerst) = min_och_max(@tal);
print "Minsta talet är $minst och
                största talet är $stoerst.\n";
```


Många inparametrar

```
sub addera_alla {
        $s = 0;
        foreach $tal (@_) {
                $s += $tal;
        }
        return $s;
}
```


Returnera flera

```
sub min_och_max {
        $min = shift(@_);
        $max = $min;
        foreach $tal (@_) {
                if($tal < $min) { $min = $tal; }
        }
        foreach $tal (@_) {
                if($tal > $max) { $max = $tal; }
        }
        return($min,$max);
}
```

```
@mini_och_maxi = min_och_max(1,2,3,4,5,6,7,8,9);
($mini,$maxi) = min_och_max(1,2,3,4,5,6,7,8,9);
```