Skriptsprachenorientierte Programmiertechnik

Perl Teil 8

Prof. Ilse Hartmann

Gliederung

grep {Block} Array;

grep (Ausdruck, Array);

- Zur Suche nach bestimmten Einträgen in einem Array
- Gibt im Listenkontext eine Teilliste all jener Listenelemente Zurück, für die die Bedingung (oft ein Pattern) zutrifft.
- Anweisungen im Block oder Ausdruck werden für jedes Element im Array durchlaufen. Für jedes Element des Array wird $_ verwendet
- Im skalaren Kontext liefert grep die Anzahl der gefundenen Elemente im Array

```
my @array = (1..15);

my @result = grep {$_ > 5 && $_ <12} @array; # liefert alle Zahlen > 5 und < 12
```

```
my @ungerade = grep { $_ % 2 } (1..1000);
                                    # liefert alle ungeraden Zahlen zwischen1 und 1000
```

Bearbeitung von Dateien mit grep

```
open (FILE, „laber.txt“) ||  die „$!“;

my @zeilen = grep { /Perl/ } <FILE>;
     # sucht nach allen Zeilen die das Wort Perl enthalten

close (FILE);
```

```
opendir (DIR, “/opt/documents/projects“);
my @dateien = readdir (DIR);    # Verzeichnis wird in Array gelesen
closedir (DIR);

my @result = grep (/\.html$\.php/,@dateien);
     # alle Dateien mit Endung php und html werden herausgesucht
```

Listen modifizieren mit map

map {Block} Array;

map (Ausdruck, Array);

- Array wird Element für Element durchlaufen und Kopien der Elemente werden bearbeitet
- Gibt im Listenkontext ein neues Array zurück, das die modifizierten Elemente enthält. Die ursprünglichen Elemente bleiben unverändert.
- Aktuelles Element wird mit $_ angesprochen
- Ausdruck bzw. Block wird auf jedes Element angewendet

Beispiele map

```
@array = (5..8),

@neu = map $_*2,@array;          #10 12 14 16

@neuer = map $_=sqrt, @neu;     #3.16 3.46 3.74 4.0
```

```
@neu = map {split /-/,$_} qw (Perl-ist-spitze);

print “@neu“;      #Perlistspitze

@array = qw(Perl ist Spitze);

@ergebnis = map m/[aeiou]/g,@array;              # e i i e
```

```
@data =(‘a42‘,‘b‘,‘c0‘,123,‘d‘,‘1a‘,‘xyz‘,‘u2‘);

@result = map (m/(\d)/ && m/(a)/, @data);

print join(‘,‘,@result),“\n“;    # a,,,a
```

Splice() fügt irgendwo im Array Elemente ein

Parameter:

- Das zu ändernde Array
- Die Position im Array, ab der Elemente hinzugefügt oder entfernt werden
- Anzahl zu entfernender oder zu ersetzender Elemente (optional)
- Listenelemente, die dem Array hinzugefügt werden sollen

Beispiel: Array-Elemente entfernen

```
@zahlen = (0..9);

splice (@zahlen,5,3);        # (0,1,2,3,4,8,9)

@zahlen = (0..9);

splice (@zahlen,5);          #(0,1,2,3,4)
```

Beispiel: Array Elemente ersetzen

```
@zahlen = (0..9);

splice (@zahlen,5,3, qw(55 66 77));          # (0,1,2,3,4,55,66,77,8,9)

@zahlen = (0..9);

splice (@zahlen, 4, 3, “void“);              # (0,1,2,3,“void“,7,8,9)
```

Beispiel: Array-Elemente hinzufügen

```
@zahlen = (0..9);

splice (@zahlen,5,0,(5.10, 5.20, 5.30, 5.40));

@neuer = map $_=sqrt, @neu;     #(0,1, 2, 3, 4, 5, 5.10, 5.20, 5.30, 5.40, 6, 7, 8, 9)
```

Slice: Teilmenge eines Array oder eines Hash

Array-Kontext: Präfix @ + [ ]

```
my @array = (1,2,3,4,5,10,12,14);

my $wert = $array[0];            # Skalarzugriff: $wert ist 1

my ($wert) = @array;             # Skalarzugriff $wert ist 1

my @slice = @array;              # @slice ist (1,2,3,4,5,10,11,12,14)

my @sclice = @array[0,1,2];  # @slice ist (1,2,3)

my @slice = @array[1,4];     # @sclice ist  (3,5)

@array[1,0] = @array[0,1];   # Array-Elemente tauschen

@array[6,7] = (17,18);         # @array ist (2,1,3,4,5,10,17,18)
```

Verwendung des Bereichsoperators

```
my @monate = qw (januar, februar, maerz, april, mai, juni, juli, august, september,
      oktober, november, dezember);

my @halbjahr = @monate[0..6];
```

Hash-Slices

```
my %hash = (‘januar‘=>1, ‘februar‘=>2, ‘maerz‘=>3);

%hashsclice = @hash {‘januar‘, ‘februar‘};

# Liste bildet ein Array aus Schlüssel-Wert-Paaren.

# Durch Zuweisung an eine Hash-Variable wird aus dem slice wieder ein Hash
```

Beispiel:

Doppelte Elemente eines Array werden entfernt, ohne dass sich die Reihenfolge ändert

```
my @arrary = (7,1,4,7,7,6,5,1,9,10,11,14,12);

my %hash;

my @uniq  = grep { !$hash{$_}++;} @array;

print join(“, “,@uniq),“\n“;
```

Weitere nützliche Listenfunktionen:

use List::Util qw (…)

die gewünschten Funktionen in der Klammer angeben

max (Array) : liefert das Array-Element mit dem größten numerischen Wert oder undef, wenn die Liste leer ist

min (Array) : liefert das Array-Element mit dem kleinsten numerischen Wert oder undef, wenn die Liste leer ist

```
my @list = (7,1,4,7,7,6,5,1,9,10,11,14,12);

my $result = max @list;   # $result = 12

my $result = min (1..10);   # 1

my $result = max (@arr1, @arr2);
```

Modul List::Util

maxstr (Array) : liefert das Array-Element mit dem größten Stringwert (basierend auf dem gt-Operator) oder undef, wenn die Liste leer ist

minstr (Array) : liefert das Array-Element mit dem kleinsten Stringwert (basierend auf dem gt-Operator) oder undef, wenn die Liste leer ist

```
my @list = ('Hans','Peter','Hilde');

my $result = minstr @list;   # $result = 'Hans'

my $result = maxstr ('A'..'Z');   # 'A'

my $result = maxstr (@arr1, @arr2);
```

first (Block Array): die Anweisungen des Blocks werden auf $_ angewendt und die Liste durchlaufen. Das erste Array-Element wird zurückgeliefert, bei dem der Block wahr ergibt. Triftt es nie zu oder ist die Liste leer, wird undef zurückgeliefert

reduce (Block Array) : reduziert das Array, indem der Block im skalaren Kontext für jedes Array-Element ausgewertet wird. Dabei werden $a und $b immer neu besetzt. Beim ersten Durchlauf ist $a das erste Element und $b das zweite. Bei nachfolgenden Durchläufen wird $a mit dem aktuellen Ergebnis besetzt und $b mit dem nächsten Element. Ergebniswert ist das letzte Ergebnis der Auswertung des Blocks oder undef bei leerer Liste

```
my @list = (7,1,4,7,7,6,5,1,9,10,11,14,12);

my $result = first { $_>10} @list;   # liefert erste Element >10: 11

my $result = reduce { $a < $b ? $a: $b} @list;       # min

my $result = reduce {$a + $b} @list;     # Summe
```

shuffle (Array) : liefert die Elemente im Array in einer zufälligen Reihenfolge

sum (Array) : liefert die Summe aller Elemente im Array oder undef, wenn die Liste leer ist

```
my @arrary = (7,1,4,7,7,6,5,1,9,10,11,14,12);

my $result = shuffle @array;

my $result = sum @array;   # 96

my $result = sum (@arr1, @arr2);
```