Skriptsprachenorientierte Programmiertechnik

Perl Teil 2

Prof. Ilse Hartmann

Gliederung

Zufallszahlen

- `rand()` **erzeugt eine (Pseudo-)Zufallszahl**

- **Ohne Parameter: Fließkommawert zwischen 0 und 1**
- **Mit Parameter `n`: Fließkommawert zwischen 0 und `n`.**

```
$zahl1 = rand();
$zahl2 = rand(100);
print "Ihre Zufallszahlen lauten $zahl1 und
$zahl2.\n";
```

- **Häufig werden ganzzahlige Zufallswerte benötigt**
- **Die Konvertierung ist über die `int`-Funktion möglich**
- `int` **schneidet stets den Nachkommaanteil ab (negative Zahlen mit Nachkommaanteilwerden also aufgerundet!)**

```
foreach $i (1..6)
{
        $lotto[$i] = int(rand(48)) + 1;   -→ Vorgriff auf Array
}
print "Ihre Lottozahlen: @lotto\n";
```

Benutzereingaben

- Mit **Standardeingabe** ist meist die **Tastatur** gemeint
- Die **Standardausgabe** ist in der Regel **die Bildschirm-Konsole**
- **Lesen einer Zeile aus der Standardeingabe: <STDIN>**
  (in skalarem Kontext)
- **Zeilen aus der Standardeingabe** enden in der Regel mit einem **Newline-Zeichen**

```
print "Ihre Eingabe: ";
$zeile = <STDIN>;
if ($zeile eq "\n")
{
        print "Sie haben nur die Eingabetaste gedrückt.\n";
}
else
{
        print "Ihre Zeile: '$zeile'\n";
}
```

chomp

- **Entfernt das letzte Zeichen eines Strings**, falls es sich um \n handelt
- Manipuliert den übergebenen String direkt.

```
do
{
        print "Ihre Eingabe: ";
        $zeile = <STDIN>;
        chomp $zeile;
        print "Ihre Eingabe lautet: '$zeile'\n";
} while ($zeile ne '')
```

- Meist wird chomp mit der Zuweisung als Argument zusammen benutzt
- Kompakte Schreibweise, weit verbreitet

```
chomp ($zeile = <STDIN>);
```

chomp und chop

- **chomp** ist eigentlich eine Funktion:

  Gibt die Anzahl der entfernten Zeichen zurück

  Dies kann also nur 0 oder 1 sein

```
$test = "Hallo\n";
$anzahl = chomp($test);
print "Es wurde(n) $anzahl Zeichen entfernt.\n";
print "Die Zeichenfolge lautet nun '$test'.\n";
```

- **chop entfernt hingegen immer das letzte Zeichen**

  Rückgabewert der Funktion: das entfernte Zeichen

```
$test = "Hallo";
$zeichen = chop($test);
print "Es wurde folgendes Zeichen entfernt: '$zeichen'.\n";
print "Die Zeichenfolge lautet nun '$test'.\n";
```

Listen und Arrays

- **Liste: Geordnete Ansammlung von skalaren Werten**
- **Array:** Variable, die eine Liste enthält**. Kennzeichen einer Arrayvariable: @**
- **Zugriff** auf einzelne Elemente **über Indizes (ganzzahlige Werte)**
- **Erstes Element: Index 0**
- **Elemente** können **beliebige,** auch **unterschiedliche Datentypen sein**
- **Beliebige Anzahl von Elementen möglich.**
  Leere Liste = Liste ohne Elemente
- **Array wird nach Bedarf erweitert, bzw. leere Elemente werden erzeugt**
- **Array-Elemente** werden wie **Skalare benutzt**

Listen und Arrays

Index	0	1	2	3	4
Wert	35	12.4	“Hallo“	1.72e30	“Hey\n“

```
$buch[0] = "Faust I";
$buch[1] = "Winnetou";
$buch[2] = "Mord im Orientexpreß";
$buch[4] = "Die Räuber";      # Element 3 wird automatisch erzeugt (undef)
$buch[1] .= " I";
$ix = 2;
print "Mein Lieblingsbuch: '$buch[$ix - 1]'\n";    # $buch[1] = „Winnetou I“
print "Buch Nr. 1234 : '$buch[1234]' ist leer!\n";
print @buch;     # Ohne Leer- oder sonstige Trennzeichen
```

Zugriff auf Array-Elemente

- Indizes sind stets ganzzahlig.
  Bei nicht-ganzzahligen Werte wird der Nachkommaanteil abgeschnitten
- **Höchster Index: $# + Name des Arrays** (z.B. $#buch)
- **Anzahl der Elemente = Höchster Index + 1**
- **Negative Indizes:** Vom Ende der Liste an gezählt (-1 = letztes Element, etc.)
- **Niedrigster zulässiger Index:** -Anzahl

```
$buch[0] = "Faust I";
$buch[1] = "Winnetou I";
$buch[99] = "Mord im Orientexpreß";
print "Mein Lieblingsbuch: '$buch[1.98]'\n";       # 1.98 wird abgerundet
print "Höchster Index : $#buch\n";
print "Anzahl Bücher : " . ($#buch + 1) . "\n";
print "Das letzte Buch : '$buch[-1]'\n";
print "Und das vorletzte : '$buch[-2]'\n";          # $buch[98] = undef
print "Das erste Buch : '$buch[-100]'\n";
```

Listenliterale

- Kommagetrennte Werte in **runden Klammern**
- **Bereichs-Operator (..)** erzeugt eine Liste in Einerschritten vom linken bis zum rechten angegebenen Wert
- Der **Bereichsoperator** funktioniert jedoch nur **„bergauf“**
- Angegebene Werte können auch skalare Variablen sein

```
@zahlen = (42, 4711, 13);
@zahlen = (42, 4711, 13, );          # Dasselbe. Komma am Ende wird ignoriert
@zweiwerte = ("Hallo", 1.23);
@leereliste = ();
@bereich = (3..8);                   # Die Werte 3 bis 8
@bereich = (3.9..8.1);               # Dasselbe. Grenzwerte werden abgerundet
@leereliste = (5..1);                # Linker Wert muß kleiner als rechter sein
$x = 3;
$y = 8;
@bereich = ($x..$y);                 # Wieder 3 bis 8
@dreiwerte = ($x, $y, $x + $y)       # 3, 8 und 11
```

Quoted Words

- **Darstellung einer Wörter-Liste** ohne Anführungszeichen
- **Elemente werden als Strings in einfachen Anführungszeichen behandelt**
- **Whitespaces** (Leerzeichen, Tabulator, Zeilenumbruch) **als Trenner**
- **Listenbegrenzer ist frei wählbar**

```
@namen = ("Fritz", "Elke", "Martin", "Anna");
@namen = qw( Fritz Elke Martin Anna );          # Dasselbe, nur einfacher
@namen = qw( Fritz Elke                         # Zeilenumbrüche trennen auch
             Martin Anna );
@namen = qw! Fritz Elke Martin Anna !;          # „!“ als Listenbegrenzer
@namen = qw{ Fritz Elke Martin Anna };          # Schweifklammern als Begrenzer
@produkte = qw! Fritz\!Box Yahoo\! !;       # Das „!“-Zeichen muß gequotet werden
@verzeichnisse = qw{
                          c:\temp
                          c:\programme
                 };
```

Listenzuweisungen

- **Mehreren Skalarvariablen können per Liste Werte zugewiesen werden**
- Hierdurch **einfacher Variablentausch möglich**
- **Überschüssige Werte** werden bei der **Zuweisung ignoriert**
- **Überschüssigen Variablen** wird **undef** zugewiesen
- Auch **komplette Arrayinhalte** können **zugewiesen werden**
- **Kopieren von Arrays:** Zuweisung eines Arrays an ein anderes Array

Beispiele für Listenzuweisungen

```
($buch, $seitenanzahl, $preis) = ("Faust I", 318, 14.99);
($links, $rechts) = ($rechts, $links);      # Variablenwerte tauschen
($marke, $modell) = qw< Audi Quattro Fiat >;    # „Fiat“ wird ignoriert
($marke, $modell) = qw[Mercedes];      # $modell ist undef
@zahlen = 1..2e4; # Die Zahlen 1 bis 20000
@mehrzahlen = (@zahlen, undef, @zahlen);       # 40001 Elemente
@kram = qw( Buch Fernseher );
@leer = ();
@krempel = (@kram, "PC", @leer, $zahlen[2]); # Buch, Fernseher, PC und 3
# Leere Liste wird ignoriert
@kopie = @krempel;            # @krempel kopieren
```

Interpolation von Arrays

- **Ausgabe eines Arrays in doppelten Anführungszeichen: Elemente werden durch Leerzeichen getrennt ausgegeben**

- **Vorsicht bei E-Mail-Adressen (@-Zeichen)!**

- **Index-Ausdrücke werden außerhalb des Strings ausgewertet**

- **Eckige Klammern nach skalaren Variablen müssen von Arrayelementen abgegrenzt werden**

Beispiele: Interpolation von Arrays

```
@kram = qw( Buch Fernseher );
@leer = ();
print "Ich besitze: @kram. Sonst nix (@leer).\n";
$email = "mustermann@fom.de";       # Versuch @fom zu interpolieren!!
$email = "mustermann\@fom.de";       # Korrekt (gequotet)
$email = 'mustermann@fom.de';        # Ebenfalls korrekt
print "Ich betrachte den $kram[1].\n";
$ix = 1;
print "Und lese ein $kram[$ix-1].\n";    # = $kram[0]
$kram = "PC";                            # gleicher Name, aber Skalar!
print "Vergleiche $kram[1] mit ${kram}[1] und $kram\[1]!\n";
```

foreach-Schleife

- **Eine Kontrollvariable durchläuft alle Listenelemente**

- **Die Variable repräsentiert das Element selbst, ist also keine Kopie!**

- **Nach dem Schleifendurchlauf enthält die Kontrollvariable denselben Wert wie vor Beginn der Schleife**

- Wird die **Kontrollvariable weggelassen**, kann auf die **Standardvariable $_** zugegriffen werden

foreach-Schleife

Beispiel:

```
$item = "nix"; # Kontrollvariable vor Beginn
@kram = qw( Stein Kater Zettel );
foreach $item (@kram)
{
    print "Ich besitze einen $item.\n";
    $item .= "chen"; # Element selbst wird geändert!
}
print "@kram\n";
print "Und jetzt? $item.\n"; # $item ist wie zuvor „nix“
foreach (1..10)
{
    print "Dies ist der $_. Durchlauf.\n";
}
```

push und pop

- Zum Einfügen **(push)** bzw. Entfernen **(pop)** von Werten am **Ende** des Arrays
- Implementiert einen **Stapel (*stack*)**
- **Erstes Argument** ist jeweils ein **Array**
- Mit **pop** kann der entfernte Wert einem **Skalar zugewiesen werden**
- Bei einem **leeren Array** liefert **pop einen leeren Wert (undef) zurück**

```
@kram = qw( Stein Zettel );
push @kram, "Buch";          # Liste ist nun „Stein“, „Zettel“ und „Buch“
push (@kram, "Stift");       # Klammern sind ebenso möglich
push @kram, 3..5;            # Hinzufügen der Zahlen 3, 4 und 5
@zeug = ("Uhu", "Tesa");
push @kram, @zeug;           # Inhalt von @zeug („Uhu“ und „Tesa“) hinzu
$klebeband = pop (@kram); # Entfernt „Tesa“ und weist es dem Skalar zu
$klebstoff = pop @kram;      # Auch hier mit und ohne Klammern möglich
pop @kram;                   # Die 5 wird entfernt, aber nicht genutzt
```

shift und unshift

- Die Pendants zu push und pop
- **Arbeiten jedoch am Anfang eines Arrays**
- **shift** entfernt das erste Element und weist es ggf. einem **Skalar** zu
- **Zum Hinzufügen am Anfang** der Liste wird **unshift verwendet**
- Implementiert eine **Schlange (*queue*)**

```
@kram = qw( Stein Zettel Buch );
$s = shift @kram;          # $s enthält „Stein“, Liste: „Zettel“ und „Buch“
$z = shift (@kram);        # Klammern sind ebenso möglich
shift @kram;               # Liste ist jetzt leer
$nix = shift @kram;        # $nix ist undef, Liste weiterhin leer
unshift @kram, "Stift";    # Liste = „Stift“ (1 Element)
unshift (@kram, "Buch");   # Auch hier mit Klammern möglich
unshift @kram, 3..5;       # Liste: 3, 4, 5, „Buch“, „Stift“
@zeug = ("Uhu", "Tesa");
unshift @kram, @zeug;      # „Uhu“, „Tesa“, 3, 4, 5, „Buch“, „Stift“
```

sort und reverse

- **sort** liefert eine **sortierte Liste zurück**
- **reverse** gibt eine **Liste mit Elementen in umgekehrter Reihenfolge zurück**
- Beide Funktionen können nicht als Prozedur genutzt werden**, d.h. die übergebene Liste selbst wird nicht geändert, sondern nur das Ergebnis der Funktionen kann genutzt werden**
- **Zu beachten: sort sortiert stets nach ASCII!**

```
@kram = qw( Stein Kater Zettel );
sort @kram;                          # FALSCH! @kram selbst wird nicht sortiert!
@kram = sort @kram;                  # Korrekt
@kram = sort (@kram);                # Auch mit Klammern
@zahlen = sort 98..101;              # 100, 101, 98, 99 !!
reverse @kram;                       # Wieder FALSCH
@umgekehrt = reverse @kram;          # So ist es korrekt
@absteigend = reverse 6..10;         # Absteigend trotz Bereichsoperator
```

Skalarer Kontext und Listenkontext

- **Ein und derselbe Ausdruck kann in verschiedenen Situationen unterschiedlich interpretiert werden**

- **Ein Ausdruck steht also stets in einem bestimmten *Kontext.***

- **Erwarteter Kontext wird durch den Perl-Quellcode-Parser festgelegt.**

Häufigste Kontext-Arten:

- **Skalarer Kontext (einzelner Wert wird erwartet) und**

- **Listenkontext (eine Liste von Elementen wird erwartet)**

Sklarer Kontext und Listenkontext

Beispiele:

```
@personen = qw( Fritz Alma Hannes );
@sortiert = sort @personen;              # Listenkontext
$zahl = 42 + @personen;                  # skalarer Kontext: Anzahl der Elemente
@kopie = @personen;                      # Listenkontext
$anzahl = @personen;                     # skalarer Kontext
@rueckwaerts = reverse qw ( abc def ghi );      # Listenkontext: ghi, def, abc
$rueckwaerts = reverse qw ( abc def ghi );      # skalarer Kontext: ihgfedcba
```

Skalarer Kontext und Listenkontext

Beispiele für Listenkontext:

```
$ding = irgendetwas;
$ding[3] = irgendetwas;
123 + irgendetwas
irgendetwas + 456
if (irgendetwas) { ... }
$ding[irgendetwas] = irgendetwas;
$ding = undef;                          # undef ist ein skalarer Wert
```

Beispiele für Listenkontext:

```
@dinge = irgendetwas;
($name, $tel) = irgendetwas;
($name) = irgendetwas;          # Liste mit 1 Element!!
foreach $item (irgendetwas) { ... }
sort irgendetwas;
@dinge = reverse irgendetwas;
@dinge = 6 * 7;                 # Liste mit 1 Element (42)
@dinge = undef;                 # Liste mit 1 Element (undef)
@dinge = ();                    # Leere Liste
```

<STDIN> im Listenkontext

- **Im Listenkontext gibt <STDIN> alle (verbleibenden) Zeilen bis zum Dateiende zurück**
- **Bei Tastatureingaben ist das „Dateiende“ Strg+Z bzw. Strg+D**
- **Die chomp-Funktion** entfernt bei einer Liste aus jedem Element ein etwaiges Newline-Zeichen am Ende

```
@zeilen = <STDIN>;
chomp @zeilen;
```

- Üblich ist hier wiederum die kompaktere Schreibweise:

```
chomp (@zeilen = <STDIN>);
```

- **Da <STDIN> im Listenkontext alles bis zum Dateiende einliest, ist bei umfangreichen Eingabeströmen (z.B. großen Logfiles) Vorsicht geboten!**