

Perl-Praxis

Einführung

Jörn Clausen

`joern@TechFak.Uni-Bielefeld.DE`

Übersicht

- Ursprünge von Perl
- erste Schritte mit Perl
- Datentypen

Warum Perl?

- Warum nicht?
- Perl ...
 - ist leicht zu erlernen
 - erlaubt kompakten Code
 - ermöglicht, unlesbaren Code zu schreiben
 - bietet Lösungen für viele Probleme
 - macht Spaß!

TIMTOWTDI

There Is More Than One Way To Do It!

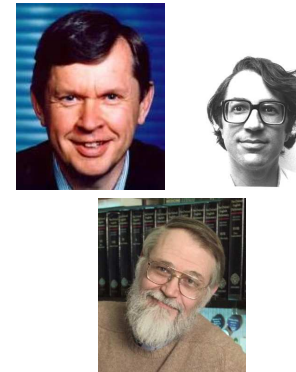
Perls Ahnengalerie



C



Shell



awk



Unix



Perl

Perl verwenden

- Kommandozeile

```
$ perl -e 'print "hello world!\n"'
```

- besser: Perl-Skript: helloworld.pl

- Dateiendung .pl Konvention

```
print "hello world!\n";
```

- Aufruf

```
$ perl helloworld.pl
```

Perl verwenden, cont.

- erste Zeile bestimmt Interpreter

```
#!/vol/perl-5.8/bin/perl
```

- # Kommentarzeichen in Perl

- Datei ausführbar machen

```
$ chmod a+x helloworld.pl
```

- Skript direkt aufrufen

```
$ ./helloworld.pl
```

- Interpreter im Pfad auswählen

```
#!/usr/bin/env perl
```

Dokumentation

- reichhaltige Literatur (vor allem O'Reilly)
- online-Dokumentation
- man-pages:

```
$ man perl
```

```
$ man perlstyle
```

- perldoc:

```
$ perldoc perlstyle
```

```
$ perldoc -f join
```

```
$ perldoc perldoc
```

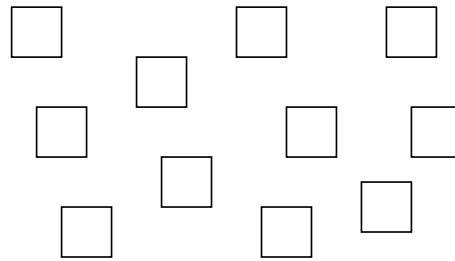
Datentypen

- bestimmen Mächtigkeit einer Programmiersprache
- Fehlerdetektion/-vermeidung durch starke Typisierung
- Deklarationen, mehr „Tipparbeit“
- typische Datentypen:

ganze Zahlen	<code>int, Integer</code>
Fließkommazahlen	<code>float, Real</code>
Zeichen, Strings	<code>char, String</code>
Booleans	<code>boolean</code>
eigene Datentypen	<code>struct, record</code>

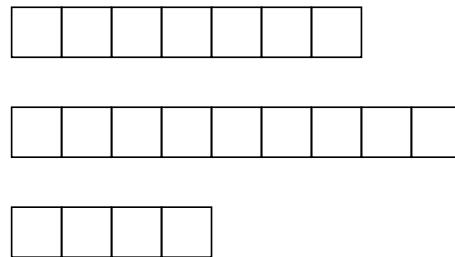
Datentypen in Perl

Skalare:



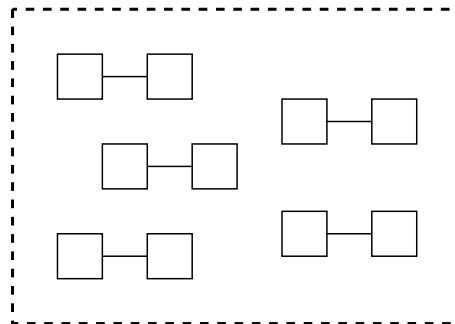
`$a`
`$name`

Arrays:



`@primes`
`@colors`

Hashes:



`%birthday`
`%price`

Skalare

- keine Deklaration notwendig
- Zuweisungen:

```
$num = 123;
$ratio = 0.75;
$name = "Joe User";
$nr = $num;
$result = $num * $ratio;
print "hello $name\n";           # Interpolation
print 'it\'s in the $name', "\n"; # keine Interpolation
print "it's in the \"$name\n";
```

- „Typkonversion“:

```
$mymoney = "give me $num dollar";
$hundred = "100";
$quarter = $hundred / 4;
```

Operatoren

- arithmetische Operatoren:

- Grundrechenarten: +, -, *, /
- Modulo: %, Potenzierung: **
- Inkrement: ++, Dekrement: --

```
$res = $i++ * 10;      # Post-Inkrement  
$res = ++$i * 10;     # Prae-Inkrement
```

- String-Operatoren:

- Konkatination: .
- Multiplikation: x

```
$hi = "hello"; $wd = "world";  
$hw = $hi." ".$wd;  
$hw5 = $hw x 5;
```

Operatoren, cont.

- Zuweisungsoperatoren:
 - einfache Zuweisung: =
 - mit Operator: +=, *=, . =, x=, ...

```
$val = 5;  
$val += 7;  
$val /= 4;  
$hw = "hello world ";  
$hw x= 5;
```

Aufgaben

- Berechne die folgenden Ausdrücke. Wie ist die Präzedenz der Operatoren?

```
$res1 = 4 * 3 + 6;
```

```
$res2 = 4 + 3 * 6;
```

```
$res3 = 100 / 5 % 3;
```

```
$res4 = 5 * 6 ** 7;
```

```
$res5 = 5;
```

```
$n = 7;
```

```
$res5 *= $n++;
```

- Gib den Text

```
Joe User schuldet mir $123!
```

aus, wobei die Zahl „123“ in einer Variablen gespeichert ist.

Strings untersuchen

- Länge eines Strings:

```
$story = "In a hole in the ground there lived a hobbit.";  
$len = length($story);
```

- Textsuche

```
$pos1 = index($story, "ground");
```

- liefert Position (bei 0 beginnend) oder -1

- Startposition der Suche:

```
$pos2 = index($story, "hole", 10);
```

- reiner Text, keine Muster

Strings manipulieren

- in Groß-/Kleinbuchstaben umwandeln:

```
$shout = uc("no smoking, please!");  
$norm = lc("TechFak.Uni-Bielefeld.DE");
```

- letztes Zeichen abschneiden:

```
$bang = chop($shout);
```

- Zeilenumbruch abschneiden:

```
chomp($line);
```

Strings manipulieren, cont.

- Text extrahieren:

```
$cname = "Prof. Dr. Joe User";  
$name = substr($cname, 10);  
$lastname = substr($cname, -4);  
$firstname = substr($cname, 10, 3);
```

- Text manipulieren:

```
$oldname = substr($cname, 10, 3, "Joseph W.");  
substr($cname, -4) = "Smith";
```

```
  0   1   2   3   4   5   6  
  ⋮ a  ⋮ b  ⋮ c  ⋮ d  ⋮ e  ⋮ f  ⋮  
 -6  -5  -4  -3  -2  -1
```


Aufgaben

- Zerlege eine EMail-Adresse der Form

`juser@TechFak.Uni-Bielefeld.DE`

in den *local part* (`juser`) und die *domain* (`TechFak.Uni-Bielefeld.DE`).

Listen

- geordnete Folge von Skalaren

```
(1,2,3,4,5,6);  
("eins", "zwei", "drei");  
(1, 'b', "drei", $vier);
```

- flache Listen

```
( (1,2), 3, (4, (5, 6)) ) # identisch mit erster Liste
```

- Aufzählungsoperator

```
(1..100)
```

- Mehrfachzuweisung

```
($a, $b) = (47, 11);  
($h, $w) = ("hello", "world");
```

Aufgaben

- Gib die Ergebnisse der folgenden Zuweisungen aus:

```
($h, $w) = ("hello", "world");
```

```
($h, $w) = ("hello", "nasty", "world");
```

```
($h, $w) = ("hello");
```

```
$h = ("hello", "dark", "nasty", "cruel", "world");
```

Arrays

- keine Größendeklaration notwendig
- dynamische Speicherallokation
- Definition mit Hilfe von Listen:

```
@primes = (2, 3, 5, 7, 11, 13, 17, 19, 23, 29);  
@nephews = ("Huey", "Dewey", "Louie");  
@stuff = (1, 'b', "drei");
```

- Zuweisung von Arrays:

```
@primes2 = @primes;  
@all = (@primes, @nephews, @stuff);
```

Operationen auf Arrays

- Zugriff auf Elemente:

```
$duck1 = $nephews[0];  
$duck2 = $nephews[1];
```

- \$ statt @, weil Ausdruck zu Skalar evaluiert
- erster Index „0“
- größter Index: \$#

```
$lastidx = $#primes;  
$lastprime = $primes[$lastidx];
```

Aufgaben

- Erzeuge Arrays aus den folgende Listen. Ermittle ihre Größe und gib einige Elemente aus, u.a. das jeweils erste und letzte:

```
(10..99)
```

```
('aa'..'zz')
```

```
((1, 'a'), (2, 'b'), (3, 'c'))
```

- Welcher Wert wird `$a` zugewiesen?

```
@a = ('aa'..'zz');
```

```
$a = @a;
```

Operationen auf Arrays, cont.

- Array in Skalar umwandeln:

```
$list = join(", ", @primes);  
$printme = join("\n", @stuff)."\n";
```

- *slices*:

```
@someprimes = @primes[1,4,5,6];  
@otherprimes = @primes[0..3];
```

- @, denn Ausdruck evaluiert zu Array (bzw. Liste)

Aufgaben

- Sieh Dir mit Hilfe von `join` die Arrays aus der letzten Aufgabe genauer an.
- Was passiert hier?

```
@disney = ("Mickey", "Donald", "Goofy", "Scrooge",  
          "Daisy", "Pluto", "Huey", "Dewey", "Louie");  
@index = (3,0,8,2);  
print join("\n", @disney[@index]), "\n";
```


Arrays verändern

- Überschreiben eines Arrayelements:

```
$stuff[1] = 'B';
```

- Anlegen eines neuen Elements:

```
$stuff[10] = "neun";
```

- *autovivification*:

```
$notusedyet[1023] = "new";
```

- Arrays werden dynamisch erzeugt und vergrößert

Array als Stack

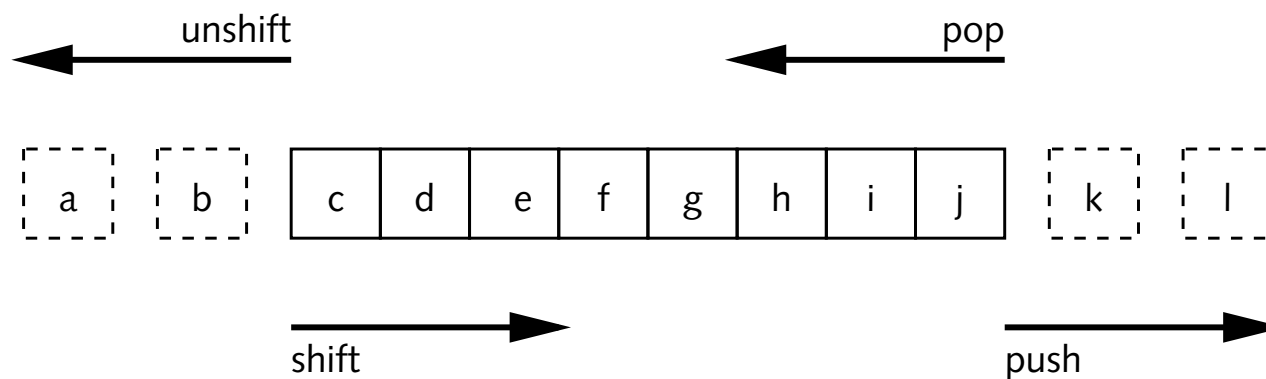
- Elemente am Ende einfügen/entfernen:

```
$newcount = push(@primes, 31,37,41);  
$lastprime = pop(@primes);
```

- Elemente am Anfang entfernen/einfügen:

```
$firstprime = shift(@primes);  
$newcount = unshift(@primes, 1,2);
```

- Indizes werden dabei verschoben



Splicing

- Merriam Webster Collegiate Dictionary:

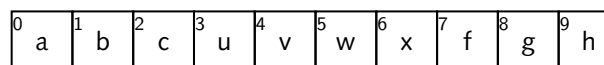
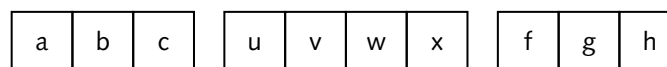
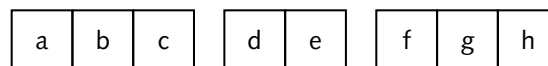
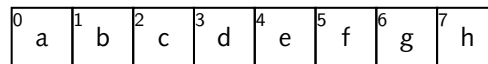
splice: to unite (as two ropes) by interweaving the strands

```
splice(@stuff, 3);
```

```
splice(@stuff, 3, 2);
```

```
splice(@stuff, 3, 2, "foo", "bar", "baz");
```

```
splice(@stuff, 3, 2, @nephews);
```



Aufgaben

- Verwende das Array `@a=(1,2,3,4,5,6,7,8,9)`. Führe die folgenden Operationen nacheinander aus:
 - Entferne das erste Element.
 - Entferne das erste Element und füge es hinten wieder an.
 - Entferne das zweite und dritte Element.
 - Ersetze das dritte Element durch die Liste `(10,11,12)`.

Gib nach jedem Schritt die Liste aus.