Einführung in Perl 5

“Perl is the Swiss Army chainsaw of scripting languages.”
Henry Spencer

"exceedingly powerful but ugly and noisy and prone to belch noxious fumes"

"highly versatile but distressingly inelegant"

Adaptiert nach:
Siever et al., **Perl in a Nutshell**
Teil1 (Getting started) und 2 (Language Basics)

Was ist Perl? – Folie 1/2

- im Prinzip eine interpretierte Skript-Sprache
- aber: Quellcode wird vor Abarbeitung komplett in einen Bytecode übersetzt, daher recht effizient
- viele Module (CPAN)
- typische Anwendungsbereiche:
  - Automatisierung der Systemadministration
  - “glue code” (Kopplung anderer Software)
  - populäre Sprache für CGI (eine Technik zum Erzeugen dynamischer Webseiten)
  - insgesamt eher für kurze Skripte

Was ist Perl? – Folie 2/2

- unterstützt imperative Programmierung, kommt aber auch mit objektorientierter Erweiterung
- recht einfach zu erlernen, wenn man von den manchmal kryptischen Spezialvariablen absieht
- erlaubt kompakte Programme
- je nach Stil hoher Wartungsaufwand
- Syntax an vielen Stellen an C angelehnt
- frei erhältlich für alle Plattformen -> perl.com
- Mott: *"There's more than one way to do it."*

Hello World!

Datei `helloworld.pl`:

```
#! /usr/bin/perl
print "Hello world!\n";
```

erlaubt direkten Aufruf unter UNIX:

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

Immer an das Semikolon denken!

Aufruf:

```
$ perl helloworld.pl
Hello world!
$
```

Programmstruktur

- Deklaration von Variablen und Subroutinen
  - Subroutinen müssen explizit deklariert werden
  - Variablen nicht (Abhilfe: `use strict 'vars';` oder `perl -w`), sie werden sonst automatisch instanziiert und ein Nullwert zugewiesen
- Kommentare
  - eingeleitet mit #, dann gültig bis Zeilenende
  - Zeilen, die mit = beginnen, sind eingebettete Dokumentation für die eingebette Dokumentationsmechanismus "pod"
- Semikolon
  - jede einfache Anweisung muss mit einem Semikolon beendet werden
  - nach Blöcken { } braucht man kein Semikolon
  - eine einzelne Anweisung in einem Block braucht kein Semikolon

Datentypen

Drei Datentypen:

- scalar, $variable
  - Zahlen: 123, -123, 0, 0.45, 0xFFFF
  - Zeichenketten: "Hallo Welt.", 'text1' (versch. Quoting-Techniken)
  - Referenzen
- array – ab Null indiziertes Feld, auch als Liste
  - @lines
- hash – ungeordnete Parameter->Wert-Paare (dictionary)
  - %phonebook

- geliebt und gehasst: automatische Typkonvertierung zwischen Zeichenketten und Zahlen

Variablen

```
$age = 26;
@date=(8,24,70);

%fruit = ('apples',3,'oranges',6);
# alternativ
%fruit = (
        apples  => 3,
        oranges => 6
);
```

Ausdrücke zum Zugriff auf Elemente:

```
$date[0]       # is 8
$date[-1]      # is last element, so 70

$fruit{'apples'}      # is 3
```

Bedingte Ausführung und Schleifen

```
if (expression) {block} else {block}

unless (expression) {block} else {block}

if (expression1) {block}
elsif (expression2) {block}
  ...
elsif (lastexpression) {block}
else {block}

while (<STDIN>) {
    print STDOUT "$_\n";
}

for ($i = 1; $i < 10; $i++) {
    ...
}
```

$_ ist eine implizite Schleifenvariable, "default input and pattern searching space"

Spezialvariablen

```
$_ default input and pattern-searching space

$. input line number
$/ input record separator
$, output field separator
$\ output record separator

$$ Prozess-ID

@ARGV Zugriff auf Kommandozeilen-Argumente

%ENV Zugriff auf Umgebungsvariablen

plus viele weitere …
```

Reguläre Ausdrücke

- werden in Perl an vielen Stellen genutzt
  - Pattern-Matching (passt ein String auf einen Pattern)
  - Suchen und Ersetzen von Text
- Ausdruck /pattern/ liefert "1" (matcht) oder "" (matcht nicht)
- wenn kein Argument angegeben ist, wird $_ verwendet
- andere Variablen, Aucdruck: $text =~ /sampo/
- Ersetzen: s/Apfel/Birne/ : "Apfelmus" -> "Birnemus"
- statt / können auch andere Zeichen verwendet werden, z.B. :
- es gibt zusätzlich noch Optionen
  - Groß-Klein-Schreibung -> i
  - ersetze alle -> g
  - erweiterte reguläre Ausdrücke -> x

- Beispiele
  - /you|me|him|her/
  - /And(y|rew)/
  - s/([^c])e/$1ie/g; # insert "i before e, except after c":

Beispiel-Programm `accounts.pl`

```
use strict 'vars';      # Variablen müssen deklariert werden
my %total;              # globale Variablendeklaration

while (<STDIN>) {               # für alle Zeilen der Eingabe
  my ($who, $action, $how_much) = split; # trennen von $_ und zuweisen
   unless ($how_much =~ /^\d+\.\d\d$/) {  # Regexp-Matching
     die "Malformed amount field";         # Programm-Ende
  }

  if ($action eq 'deposit')     { $total{$who} += $how_much }
  elsif ($action eq 'withdraw') { $total{$who} -= $how_much }
  else { die "Bizarre action field `$action'" }
}

for my $customer (sort keys %total) {       # Kontostände ausgeben
  print "$customer:\t$total{$customer}\n";
}
```

Beispiel-Lauf

Datei transactions.log :

```
larry   deposit        155.87
larry   deposit        200.34
larry   withdraw        254.27
donald  withdraw        96.75
larry   withdraw        253.73
harry   deposit        123.34
```

$ perl accounts.pl < transactions.log

```
donald: -96.75
harry:  123.34
larry:  -151.79
```

Literatur

```
$ man perlintro
```

http://perldoc.perl.org/

Siever et al., *Perl in a Nutshell*, O´Reilly

Wall,*Programming in Perl*, O´Reilly