Eine Einführung

von Serap Tekke & Elif Öner

Inhalt

1. Allgemeines

2. Datentypen

3. Algorithmische Elemente

4. Module

5. Objektorientiertes Programmieren

6. Pattern Matching

Was ist Perl ...

Perl – **P**ractical **E**xtraction and **R**eport **L**anguage
(Praktische Sprache für Daten-Extraktion und -Ausgabe)

Stärken:

- Extraktion - Anschauen von Dateien und das *Herausziehen* wichtiger Teile
- Report - Generieren von Ausgaben und *Berichte* über die gefundenen Informationen
- praktisch - weil es viel leichter ist, diese Art von Programmen in Perl zu schreiben

Skriptsprache, die zum Ausführen eines Programms einen Interpreter benötigt => *Interpretersprache*

… und wo kommt es her?

1987 von **Larry Wall** entwickelt

Ziel: Entwicklung einer einfachen Sprache mit Schleifen, Such- und Ersetzungsroutinen

Synthese aus diversen Unixtools und –sprachen, Shell-Skripte und C

seit Perl 5 übernehmen freiwillige Perl-begeisterte Programmierer (Perl 5 Porter) die Entwicklung

aktuelle Version: 5.8.7

Hello World

```
#! /usr/bin/perl -w

print "Hallo Welt!\n";
```

- Shebang/Hashbang
- Pfad zum Perl-Interpreter
- Parameter, welcher Fehlermeldungen im Kommandofenster/Browserfenster ausgibt
- Anweisung für Textausgabe
- Kürzel für neue Zeile
- Ende einer Anweisung

bei Windows: #! C:\Programme\Perl\bin\perl.exe

Datentypen

Perl unterscheidet lediglich zw. 2 generellen Sorten von Daten

- nach Singular oder Plural

Skalare:

- einzelner Wert
- String (Zeichenfolge) oder Ziffer

```
$skalar = wert;
```

- Bezeichner von Skalaren
- Zuweisungsoperator

Listen (Arrays):

- Satz von skalaren Daten

```
@liste = (wert1, wert2, wert3,
          … , wertN);
```

- Bezeichner von Listen
- skalare Daten durch Kommata getrennt

Hashes:

- spezielle Form von Listen, auch bekannt als assoziative Arrays
- Ansammlung skalarer Daten, deren individuelle Elemente über einen Index angesprochen werden können
- beliebige Skalare (auch Schlüssel genannt) werden später verwendet, um die Werte aus dem Array wieder herauszuholen

```
%hash = (Schlüssel1=> Wert1,
Schlüssel2 => Wert2, … );
```

- Bezeichner von Hashes
- Operator zur Spezifikation der Paare (Schlüssel links, Wert rechts)

Beispiel: Wörterbuch mit Hash

```
#!C:\Programme\Perl\bin\perl.exe

%deutschwort = ("go“ =>"gehen",
                "job“ =>"Arbeit",
                "pearl" => "Perle",
                "francais" => "französisch");

$deutschwort{"computer"} = "Computer";
$deutschwort{"job"} = "Stelle";
delete $deutschwort{"francais"};                    # Entfernung Schlüssel-Wert-Paar

while (1) {                                         # Endlosschleife
      print "Enter english word: ";
      chomp ($englishword = <STDIN>);               # <STDIN> liest Benutzereingaben;
                                                      chomp schneidet Zeilenvorschubszeichen ab

      if(exists $deutschwort{$englishword}) {       # Prüfung, ob ein Wert als Schlüssel vorkommt
            print "$englishword = $deutschwort{$englishword}\n";
      }
      else {
            print "$englishword not in dictionary.\n";
       }
}
```

Bedingungen & Schleifen

- **If … elsif … else,**
- **Bedingungsoperator**

- **unless** (if not)**:**

```
unless (Bedingungsausdruck) {
# mache das hier
}
```

- **while, for, do, until**

- **foreach:**

```
foreach ([Liste])
{
[Anweisungsblock]
}
```

- each (while-Schleife für Hashes)**:**

Im skalaren Kontext:

```
while ($key = each %hash) {
print “$key = $hash{$key}“;  }
```

Im Listenkontext:

```
while (($key, $value) = each %hash) {
print “$key = $value \n“;
}
```

Schleifensteuerung

- **redo** *(Wiederholen der Schleife)*:

```
$i = 0;
while ($i < 10)
{
$i++;
If ($i == 3) {redo}
print “$i /n“;
}
```

next *(Überspringen der Schleife)*:

```
$i = 0;
for ($i=0; $i < 10; $i++)
{
If ($i == 3) {next}
print “$i /n“;
}
```

- last *(Abbruch der Schleife)*:

```
$i = 0;
while (1)
{
$i++;
print “$i /n“;
If ($i == 5) {last};
}
```

Labels (Ausstieg aus mehreren Schleifen):

```
LABEL: while (Bedingung1) {
    # …
    while (Bedingung2) {
        # …
        if (noch_eine_Bedingung) {
       last LABEL; }
 }}
```

Subroutinen

(Funktionen, Methoden)

Subroutine definieren:

```
sub guten_tag
{
print “Guten Tag!“
}
```

Subroutine aufrufen:


```
&guten_tag();
```

Aus Subroutinen Werte zurückgeben:

```
$sum = &sumnums();
print "Summe: $sum\n";

sub sumnums {                                   # Zahlen addieren
        print 'Geben Sie eine Zahl ein: ';
        chomp($zahl1 = <STDIN>);
        print 'Geben Sie eine weitere Zahl ein: ';
        chomp($zahl2 = <STDIN>);
        return $zahl1 + $zahl2;
}
```

In Subroutinen Parameter/Argumente übergeben:

Bei skalaren Argumenten:

```
&meine_subroutine (1, 2, 3);
```

Bei Listenargumenten:

```
&meine_subroutine (@liste, @andere_liste);
```

Parameter in der Subroutine entgegennehmen

Die an eine Subroutine übergebene Parameterliste wird im lokalen Spezialarray @_ gespeichert.

```
&addiere(2,3);          #Aufruf der Subroutine
sub addiere {
      return $_[0] + $_[1];
}
```

Module

- eine Art Funktion, die nicht direkt in den Perl-Interpreter eingebaut sind, sondern in Form von Perl-Skripts zur Verfügung stehen
- sie stellen Funktionen (aber auch Variablen) für best. Routine-Aufgaben zur Verfügung, die man mittels dem use-Operator in sein eigenes Skript einbauen kann

Frei verfügbare Perl-Module:
Standardmodule, die zs. mit dem Perl-Interpreter ausgeliefert werden
CPAN-Module, welche im Internet zum Download zur Verfügung stehen

Import eines Moduls:

```
use strict;
```

Import eines Teils von einem Modul:

```
use Math::BigInt;
```

Objektorientiertes Programmieren in Perl

- unterstützt seit der Version 5.0

- Sammlung von Objekten, die in einer vordefinierten Art&Weise miteinander interagieren

- bei Erstellung eines oo-Skripts, erzeugt man eine oder mehrere eigene Klassen, die K. und Obj. von anderen Quellen (i.d.R. Module) importieren und verwenden

- bei Ausführung des Perl-Skripts werden aus versch. Klassen Laufzeitobjekte erstellt, die untereinander ihre Variablen ändern und ihre jew. Subroutinen aufrufen, um am Ende eine Art von Ergebnis zu liefern

- um eine Klasse zu programmieren, also das, wovon sich eine Objektinstanz ableiten lässt, braucht man in Perl ein eigenes Package, also einen eigenen Namensraum

- unterstützt Polymorphie

Klasse = Paket (*Package*)

- der Namensbereich, der vom Paket definiert wird, definiert die Kapselung und den Gültigkeitsbereich für die Klasse.
- wird kein Namensraum bzw. Package angegeben, dann befindet man sich im Default-Package *main*

```
package serelif;
```

- Unterroutinen ruft man mit dem Namen des Pakets auf gefolgt von ::

```
&main::name_subroutine
```

- ein Package ist allerdings noch nicht automatisch eine Klasse für Objekte
- sie muss erst als solche def. und erstellt werden

```
$obj = serelif->new();
```

Vererbung

- Basisklassen werden in Perl über den speziellen Array @ISA (is a …) angegeben

- @ISA legt fest, in welchen Klassen nach Methodendefinitionen gesucht wird, wenn die Definiton einer aufgerufenen Methode nicht in der aktuellen Klasse existiert
- qw = quoted words, nimmt eine Liste von durch Leerzeichen getrennten Strings entgegen und gibt eine Liste der einzelnen Stringelementen zurück..

```
# Auto.pm
use Fortbewegungsmittel;
package Auto;

@ISA = qw( Fortbewegungsmittel );
```

Pattern Matching (Mustererkennung)

Muster *(pattern)***:**

beschreibt eine Menge von Wörtern
beschrieben durch regulären Ausdruck

Einsatzgebiete:

- Eingabe-Validierung
- Prüfung, ob die Eingabe in dem korrekten spezifischen Format erfolgt ist
- Herausziehen bestimmter Teile einer Datei, die einem bestimmten Kriterium entsprechen
- Zerlegung eines Strings auf der Basis bestimmter Trennzeichen-Felder
- Feststellen von Unregelmäßigkeiten in einer Datenmenge
- Das Zählen der Vorkommen eines Musters in einem String.
- Suchen&Ersetzen-Operationen
- uvm

Reguläre Ausdrücke

Code	**Äquivalente Zeichenklasse**	**Bedeutung**
/ /	( )	Anfang und Ende des regulären Ausdrucks
^		Anfang einer Zeile
$		Ende einer Zeile
\d	[0-9]	Alle Ziffern
\D	[^0-9]	Alle Zeichen außer Ziffern
\w	[0-9a-zA-z_]	Alle »Wortzeichen« (alphanumerische Zeichen und _)
\W	[^0-9a-zA-z_]	Alle Zeichen außer »Wortzeichen«
\s	[\t\n\r\f]	Whitespace-Zeichen
\S	[^ \t\n\r\f]	Alle Zeichen außer Whitespace-Zeichen
\.		sucht nach einem Punkt

Beispiel: Eingabenprüfung mit PM

```
while () {
  print 'Geben Sie eine Zahl ein (0-9): ';
  chomp($_ = <STDIN>);
  if (/^\d$/) {                                    # korrekte Eingabe
     print "Danke!\n";
     last;
  } elsif (/^$/) {
     print "Sie haben nichts eingegeben.\n";
  } elsif (/\D/) {                                 # keine Zahlen
     if (/[a-zA-z]/) {                             # Buchstaben
       print "Sie koennen mich nicht taeuschen.  Die Eingabe enthaelt Buchstaben.\n";
     } elsif (/^-\d/) {                            # negative Zahlen
        print "Das ist eine negative Zahl. Bitte nur positive Zahlen!\n";
     } elsif (/\./) {                              # Dezimalzahlen
        print "Das sieht sehr nach einer Dezimalzahl aus.\n";
        print "Ich kann Dezimalzahlen nicht in Worten ausgeben. Versuchen Sie eine neue Zahl.\n";
     } elsif (/[\W_]/) {                           # andere Zeichen
        print "huh?  Das sieht *wirklich* nicht nach einer Zahl aus\n";
     }
  } elsif ($_ > 9) {
     print "Zu gross!  0 bis 9, bitte.\n";
  }
}
```

Quellen

Literatur

PERL in 21 Tagen – Laura Lemay
Einführung in Perl - Randal L. Schwartz, Tom Christiansen
Programmieren lernen mit Perl – Joachim Ziegler
Webseiten programmieren und gestalten – Mark Lubkowitz

Internet

http://de.selfhtml.org/perl/
http://de.wikibooks.org/wiki/Perl-Programmierung
http://www.mathe2.uni-bayreuth.de/perl/

Vielen Dank für eure Aufmerksamkeit!

>>There is more than one way to do it<<

Larry Wall