SANDATA | IT-Trainingszentrum GmbH
Die IT-Gruppe

Perl - Seminar

Grundkurs

Dr.sc.nat. Michael J.M. Wagner*

Revision 329M

*michael@wagnertech.de

Inhaltsverzeichnis

1 Einführung

Was ist Perl

- Programmiersprache aus dem UNIX-Bereich
- Ursprünglicher Einsatz: Bearbeiten und Auswerten großer Datenmengen (Log-Dateien)
- „Practical Extraction and Reporting Language"
- Nach Erfindung des WWW: Standard für CGI[1]-Programme
- Heute ist Perl für fast alle Betriebssysteme verfügbar
- Syntax basiert auf C, ist aber reicher (komplizierter?)
- Skriptsprache → Portabilität

Entwicklung und Geschichte von Perl

- 1987 von Larry Wall
- Best of C, Basic, UNIX-Shell, awk, sed
- Zitat von Larry Wall

  > Der Entwickler Larry Wall bezeichnet Perl als „... Interpretersprache, die optimiert ist, willkürliche Textdateien zu durchduchen, aus ihnen Informationen zu filtern und Berichte aus diesen Informationen zu erstellen. Außerdem eigent sich Perl sehr gut für fast alle Aufgaben der Systemsteuerung. Die Sprache soll praktischen Nutzen bringen und nicht unbedingt schön sein."[2]

- Open-Source Entwicklergemeinde

Perl installieren

Für diesen Kurs ist eine virtuelle Maschine (LinuxMint) vorbereitet, die Eclipse mit Perl-Addon enthält.

- Eclipse ist eine Standardentwicklungsumgebung
- Perl ist auch als UNIX-Interpreter installiert

Aufgabe:

- Virtuelle Maschine starten
- Eclipse starten

[1] Common Gateway Interface
[2] Teich: S. 7.

- Eclipse-Projekt anlegen
- `HalloWelt.pl` anlegen (Code findet sich am „Schreibtisch", Kapitel 2)
- Start über Eclipse und Kommandozeile

2 Programmieren mit Perl

Anweisungen, Blöcke und Kommentare

- Anweisungen werden mit „;" abgeschlossen
- Tolerante Syntax (z.B. `print` mit und ohne Klammern)
- Anweisungsblöcke mit geschweiften Klammern
- `#` als Kommentarzeichen
- Blockkommentare über `=pdo / =cut`

She-Bang, die erste Zeile

- Eingeschränkte Bedeutung in Windowssystemen
- Optionen für Perl-Interpreter [ST: S. 19]

Regeln für Bezeichner

- Buchstaben, Ziffern, Unterstrich
- Sensibel auf Groß-/Kleinschreibung

Ausgabe von Daten

- Syntax der Print-Anweisung [ST: S. 20]
- „`\n`" muss explizit angegeben werden
- Heredoc [ST: S. 22]

Aufgabe:

Beispieldatei `Kapitel_03/Heredoc.pl` in den *workspace* kopieren und ausführen.

3 Einfache Sprachelemente

Variablen

- Skalare Variablen (Zahlen, Zeichenketten, Referenzen): `$skalar`
- Listen, Arrays: `@array`
- Assoziative Arrays (Hashes): `%hash`
- Wertzuweisung erfolgt über „=" (auch transitiv möglich)
- Perl-Standard: Variablen müssen *nicht* deklariert werden
- Programmierstandard: Variablen müssen deklariert werden → `use strict;`
- Variablendeklaration erfolgt mit `my`. Die Sichtbarkeit der Variable ist dann auf den Block geschränkt (wie bei C, Java, ...).

Datentypen

- Keine Typisierung durch den Programmierer (außer Skalar, Array, Hash)
- Interne Typisierung: Zahlen/Zeichenketten
- Implizite Umwandlung

Operatoren

- Arithmetische Operatoren [ST: S. 35]
- Vergleichsoperatoren (numerisch oder alphanumerisch) [ST: S. 36/37]
- Logische Operatoren [ST: S. 37]
- Zeichenkettenoperatoren (Konkatenation mit „.", Vervielfachung mit „*")
- Zuweisungsoperatoren (+=, ...)

4 Kontrollstrukturen

Anweisungen zur Bedingungsauswahl

- `if / if` (nachgestellt) [ST: S. 44]
- `unless / unless` (nachgestellt) [ST: S. 47]
- Ternärer Vergleichsoperator [ST: S. 47]
- `if - elsif - else` [ST: S. 49]

Bedingte Wiederholungsschleifen

- `while`-Schleife [ST: S. 51]
- `until`-Schleife: wie while, nur umgekehrte Logik
- `do - while`: wie while, nur mit Prüfung am Ende des Anweisungsblock
- `do - until` [ST]

Zählergesteuerte Wiederholung

- `for`-Syntax, wie in C [ST: S. 54]

Aufgabe (Buch, Kapitel 5):

- `AddZahlen`
- `Punkte`

Aufgabe:

Mit dieser Aufgabe beginnt die Implementierung einer „Büchereiverwaltung".

- Legen Sie ein Projekt *Bucherei* an.
- In der Datei `Bucherei.pl` soll abgefragt werden, welche Entität gepflegt werden soll (*Medium, Nutzer, Ausleihe*)
- Danach soll gefragt werden, welche Operation ausgeführt werden soll (*Anlegen, Lesen, Ändern, Löschen, Alle anzeigen*)
- Implementieren Sie eine Schleife um die Eingabe, die erst verlassen wird, wenn die Eingabe gültig ist.
- Fragen Sie je nach dieser Eingabe nach den weiteren benötigten Daten:
  - Anlegen/Ändern von Medien: *Signatur, Autor, Titel, Typ, Seitenzahl, Spieldauer*
  - Lesen/Löschen von Medien: *Signatur*
  - Anlegen von Nutzern: *Nachname, Vorname, Geburtsdatum*
  - Ändern von Nutzern: *Nutzernummer, Nachname, Vorname, Geburtsdatum*
  - Lesen/Löschen von Nutzern: *Nutzernummer*
  - Bei Ausleihen: *Signatur, Nutzernummer*
- Implementieren Sie eine Schleife um die Eingabe, die erst verlassen wird, wenn die benötigten Felder gefüllt sind.
- Implementieren Sie um das Ganze eine weitere Schleife, die erst verlassen wird, wenn bei der Abfrage der Entität „Ende" gewählt wird.

5 Listen und Datenfelder

Daten in Feldern speichern

- Arten von Datenfeldern: Array, Hash [ST: S. 58]

Arrays verwenden

- Definition über runde Klammer: `('Mo', 'Di', 'Mi', 'Do', 'Fr')`
- Alternative: mit `qw`: `qw(Mo Di Mi Do Fr)`
- Variablenname beginnt mit `@`: `@array_var = qw(Mo Di Mi Do Fr);`
- Zugriff auf Einzelelement über `$` und eckige Klammer: `$array_var[3]`
  Das ist der String 'Do'
- Anzahl der Elemente eines Arrays: `$#array_var + 1` (`$#array_var` ist der höchste Index)
- Funktionen für Arrays [ST: S. 64]

Hashes verwenden

- Hashes bestehen aus Schlüssel-Wert-Paaren
- Definition über runde Klammer: `(Mo=>'Montag', Di=>'Dienstag', Mi=>'Mittwoch')`
- Variablenname beginnt mit `%`: `%hash_var`
- Zugriff auf Einzelelement über `$` und geschweifte Klammer: `$hash_var{Mi}`
  Das ist der String `Mittwoch`
- Der Zugriff auf ein Einzelelement kann sowohl lesend als auch schreibend erfolgen.
- Weitere Elemente können durch Zuweisung hinzugefügt werden: `$hash_var{Do} = 'Donnerstag';`
- Array mit Schlüsselwerten extrahieren: `keys(%hash_var)`
- Array mit Wert-Werten extrahieren: `values(%hash_var)`
- Element löschen: `delete $hash_var{key}`

Arrays und Hashes in Schleifen durchlaufen

- `foreach`-Schleife bei Arrays und Hashes [ST: Kap. 6.5]

Aufgabe (Buch, Kapitel 6):

- `Uebung3`
  Hinweis: `defined` prüft, ob eine Variable definiert/ein Eintrag im Hash vorhanden ist.

Aufgabe (Bücherei):

- Nach der Eingabe der gewünschten Operation soll nun in die Ausführung verzweigt werden. Die Implementierung für „Nutzer anlegen" und „Alle Nutzer anzeigen" soll nun vorgesehen werden, für alle anderen Operationen soll erst mal „nicht implementiert" ausgegeben werden.
- Für die Implementierung von „Nutzer anlegen" soll nun:
  - eine Hashmap mit den Daten des Nutzers angelegt werden,
  - diese Map einem Nutzerarray hinzugefügt werden.
- Für die Implementierung von „Alle Nutzer anzeigen" soll:
  - das Nutzerarray Eintrag für Eintrag ausgegeben werden,
  - Zu jedem Nutzer soll die Indexnummer im Array als „Nutzernummer" mit ausgegeben werden.
  - falls leer, „Kein Nutzer im System" ausgegeben werden.

6 Unterprogramme und Funktionen

Unterprogramme in Perl

- Definition mit `sub`: `sub unterprog { ... }`
- Variablenübergabe als Stellungsparameter. Die Stellungsparameter stehen in der gerufenen Routine in einem Array mit Namen `@_` zur Verfügung. Für den Zugriff darauf gibt es verschiedenste Praktiken:
  - Zugriff auf Parameterarray über Index: `my $param = $_[0];`
  - Zugriff über Arrayfunktion `shift`: `my $param = shift;`
  - Abbildung des Arrays auf Einzelvariablen: `(my $a, my $b, my $c) = @_;`

  Achtung: Befindet sich unter den übergebenen Parametern ein Array geht das in `@_` auf.
- Variablenübergabe als benannte Parameter

```
sub sub1 {
  my %params = @_;
  print params{"par1"};
}

sub1(par1 => "Wert1");
```

- Funktionen geben Wert zurück: `return $ret_val;`
- Bei der Verwendung von Unterprogrammen ist ein `use strict`, zusammen mit lokalen Variablen dringend empfohlen. Schlechtes Beispiel: Buch, `Volumen.pl`

Referenzen verwenden

- Referenzieren über `\`: `$var_ref = \$var;`
- Dereferenzieren über `$/@/%`: `$$var_ref, @$array_ref, %$hash_ref`

  Hier erhalte ich die ursprüngliche Variable/Array/Hash
- Dereferenzieren über `->` (nur bei Array, Hash)

  Hier erhalte ich ein Element: `$array_ref->[3], $hash_ref->{Mi}`
- Arrayreferenz definieren: `$arr_ref = ['Mo', 'Di', 'Mi'];`
- Hashreferenz definieren: `$hash_ref = {Mo=>'Montag', Di=>'Dienstag'}`
- Anwendungen
  - Variablenübergabe an Funktionen [ST: Kap. 7.3]
  - Bildung von mehrdimensionalen Arrays (Array von Arrayreferenzen)

Vordefinierte Perl-Funktionen

- → Buch [ST: S. 84]
- Es gibt zahllose Perl-Module, die weitere Funktionen zur Verfügung stellen.
- Die Verwendung eines Perl-Moduls muss mit `use` angekündigt werden.

Funktionen auf Module verteilen

[ST: S. 164]

Aufgabe (Buch, Kapitel 7):

- `Uebung3`

Aufgabe (Bücherei):

Der bisherige Code soll nun in zwei Module aufgeteilt werden:

- Im Hauptprogramm (`Bucherei.pl`) verbleibt die Ein-/Ausgabe.
- In das Modul `Nutzerverwaltung.pm` kommen alle Funktionen, die mit der Nutzerverwaltung zu tun haben.
  - Das Modul enthält das Array für die Nutzer als globale Variable (`our @nutzers`).
  - `anlegen($nutzer)` fügt `$nutzer`, eine Hashreferenz, dem Nutzerarray hinzu.
  - `alle_lesen` liefert eine Referenz auf dieses Nutzerarray zurück, das dann im Hauptprogramm (wie bereits implementiert) ausgegeben wird.

7 Datei und Verzeichnisfunktionen

Mit Dateien arbeiten

- Dateien öffnen [ST: S. 89]
- Dateien schließen: `close($datei_handle);`

Textdateien lesen

- Zeilenweise Verarbeitung [ST: S. 91f.]
- Kompletter Dateiinhalt in einer Arrayvariablen: `@zeilen = <$datei_handle>;`

In Dateien schreiben

- In Datei schreiben: `print $datei_handle 'Inhalt';`

Statusinformationen über Dateien ermitteln

- Dateiprüfoperationen [ST: S. 95f.]

Funktionen des Dateisystems / Mit Verzeichnissen arbeiten

Viele Betriebssystemfunktionen, wie Dateien umbenennen, löschen, etc. sind direkt in Perl verfügbar [ST: S. 100]

Aufgabe (Bücherei):

Das Nutzerarry soll nun in einer Datei `nutzer.csv` gespeichert werden.

- Ergänzen Sie `Nutzerverwaltung` um die Funktionen `laden` und `speichern`, die beim Start und am Ende des Hauptprogramms aufgerufen werden.
- Lesen der Datei:
  - Falls die Datei nicht vorhanden ist, bleibt das Array einfach leer.
  - Die Datei wird Zeile für Zeile gelesen.
  - Jede Zeile wird mit `split` in ihre Elemente zerlegt, daraus eine Hashmap erzeugt und dem Nutzerarray hinzugefügt.
- Speichern: Der Inhalt des Array wird Zeile für Zeile kommasepariert in die Datei geschrieben.

8 Zeichenketten

8.1 Funktionen mit Zeichenketten

Im Kapitel 6 wurden bereits die wichtigsten Funktionen mit Zeichenketten gezeigt.

8.2 Reguläre Ausdrücke

Suchen und Ersetzen mit regulären Ausdrücken

- Zweck [ST: 10.1, erster Absatz]

Mustererkennung mit regulären Ausdrücken

- Einfache Suchmuster [ST: Kap. 10.2]

Einige Zeichen haben innerhalb regulärer Ausdrücke eine spezielle Bedeutung und müssen bei der Verwendung mit einem *backslash* gekennzeichnst werden:

`/`	leitet einen regulären Ausfruck ein
`.`	steht für ein beliebiges Zeichen
`+ * ? {}`	kennzeichnen Kardinalitäten
`^ $`	verankern ein Suchmuster am Anfang/Ende
`|`	kennzeichnet eine Auswahl
`()`	gruppiert Treffer
`[]`	definiert Buchstabenmengen

Optionen für reguläre Ausdrücke

- Alternative Zeichen werden mit eckigen Klammern ausgedrückt
- Zeichenausschluss mit Dach: `[^x]`: kein `x`
- Vordefinierte Zeichenklassen
- Wiederholungen
- Zeilenanfang: `^`, Zeilenende: `$`
- Alternative Ausdrücke werden mit `|` getrennt. [ST: Kap. 10.3]

Speichern von Übereinstimmungen

- Sollen Muster gespeichert werden, so werden sie in runde Klammern geschrieben
- Die Wiederholoperatoren (`+`,`*`) verhalten sich *greedy* (gierig), d.h. sie suchen die maximale Zeichenmenge, die mit dem Suchmuster verträglich ist.
- Nicht-gierige Suche mit `?`
- Leider funktioniert das nicht in allen Regex-Implementierungen. Alternative: `[^x]*` [ST: Kap. 10.4]

Ersetzen von Textteilen mit regulären Ausdrücken

- Syntax: `$var =~ s/Suchmuster/Ersetzung/[option];`
- Falls `/` im Suchmuster vorkommt, kann der Suchausdruck mit einem anderen Zeichen eingeleitet werden: `s!Suchmuster!Ersetzung!`
- Soll eine im Suchmuster gefundene Übereinstimmung im Ersetzungstext wiederverwendet werden:
  - Im Suchmuster mit runden Klammern „speichern"
  - Im Ersetzungstext mit `\1` „abrufen"
  - Beispiel
- → Buch [ST: Kap. 10.6]

Aufgaben:

- Buch Kapitel 10: `Ubung2`
- Lösen Sie das Problem von Kapitel 9, Aufgabe 4 unter Verwendung regulärer Ausdrücke.

Aufgabe (Bücherei):

Ergänzen Sie die Datumseingabe beim Benutzer um eine Regexprüfung.

9 Objektorientierung

- Kaum eigene Sytax (kein `class` o.ä.), nur `new` (wird beim Anlegen eines Objekts aufgerufen)
- Ansatz:
  - Nehme eine Hashreferenz
  - Packe Funktionen in ein Paket (Paketname = Klassenname)
  - Vererbung wird über ein entsprechendes use-Statement realisiert: `use base ('Basisklasse');`
  - Verbinde die Hashreferenz mit dem Modul: `bless($object, 'Klasse');`
- Besonderheiten:
  - Ein Klassenmodul enthält eine Funktion `new` (den Konstuktor), die `bless` enthält und das Objekt (=Hashreferenz) zurückgibt.
  - `$obj->func($var1, $var2)` ruft die Funktion `func` aus dem zur Hashreferenz `$obj` „geblessten" Modul auf. `$obj` wird dabei als erster Parameter (`$_[0]`), `$var1` als zweiter Parameter (`$_[1]`), etc. übergeben.
  - In der Funktion `func` wird der erste Parameter üblicherweise der Variablen `$self` übergeben: `$self = $_[0];`
  - Falls am Ende der Lebenszeit des Objekts Resourcen aufgeräumt werden müssen, kann dies in einem Destruktor namens `DESTROY` implementiert werden.

Die Objektinstanz liegt als als Hashreferenz vor. Damit ist es möglich, auf jedes Element der Instanz direkt zuzugreifen (`obj->{elem}`). Dies widerspricht aber der Idee, dass die Elemente eines Objekts (*members*) dessen Implementierungsgeheimnis sind. In „richtigen" objektorientierten Sprachen wird auf die Elemente eines Objets nur prozedural (über „Methoden") zugegfriffen. Die elementaren Zugriffsmethoden sind dabei die *getter* und *setter*.

[ST: Kap. 13]

Um zu prüfen, ob eine Hashreferenz zu einer bestimmten Klasse gehört, gibt es folgende Möglichkeiten:

- `blessed $obj` liefert den Klassennamen zurück.

- `$obj->isa("Kasse")` liefert `1`, falls `$obj` zur Klassenhierarchie von `Klasse` gehört.

Aufgabe (Bücherei):

Erstellen Sie folgende Klassen/Module:

- `Medium.pm` mit den Klassen `Medium` und davon abgeleitet `Buch` und `CD`.
  - Der Konstruktor von `Buch` wird mit den Parametern `signatur, autor, titel, seitenzahl` aufgerufen.
  - Der Konstruktor von `CD` wird mit den Parametern `signatur, autor, titel, spieldauer` aufgerufen.
  - Beide Konstruktoren verwenden den Konstruktor von `Medium`, der die gemeinsamen Attribute übernimmt.
  - Alle Klassen implementieren eine `format()`-Methode, die eine kommaseparierte Zeichenkette mit allen sechs Parametern (`Signatur,Autor,Titel,Typ,Seitenzahl,Spieldauer`) zurückgibt. Für „nicht-anwendbare" Parameter werden Dummywerte eingesetzt.
- `Medienverwaltung`:
  - Im Konstruktor wird eine leere Hashmap angelegt, dann die Datei `medien.csv` Zeile für Zeile gelesen, in die sechs Parameter zerlegt und `anlegen` aufgerufen.
  - Die Methode `anlegen` nimmt die sechs Parameter, prüft, ob die Signatur nicht bereits in der Map vorhanden ist, instanziert je nach Typ ein `Buch` oder `CD` und fügt die Instanz der Hashmap hinzu. Schlüssel ist dabei die Signatur, der Wert die Instanz.
  - Die Methode `lesen($signaur)` liefert die zu `$signatur` gehörende Medieninstanz zurück. Falls keine vorhanden wird `undef` zurückgegeben.
  - Die Methode `alle_lesen` gibt eine Referenz auf ein Array zurück, das alle Medieninstanzen enthält.
  - Im Destruktor wird die Hashmap in die Datei zurückgeschrieben. Dazu wird die Map durchgegangen und für jede Instanz `format()` aufgerufen, diese Zeichenkette dann der Datei hinzugefügt.
- Im Hauptprogramm wird zu Beginn eine Instanz von `Medienverwaltung` angelegt. Implementieren Sie die Operation *Medium anlegen*, *Medium lesen* und *Alle Medien anzeigen*.

10 Fehlerbehandlung

Zur Fehlerbehandlung gibt es verschiedene Ansätze:

- Rückgabewerte vom Typ `int`
- Rückgabewerte eines speziellen Fehlertyps
- Exceptions
- Eigene Routine zur Ermittlung des Fehlerstatus

Vorteil der Rückgabewerte:

- Einfachere lokale Fehlerbehandlung

Vorteil der Exceptions:

- Fehlerbehandlung stört nicht den logischen Programmfluss.
- Fehlerbehandlung wird nicht vergessen.

Empfehlung:

- Zu erwartende (oft fachliche) Fehler werden auf Rückgabewerte abgebildet.
- Unerwartete Fehler (oft technische Fehler, Logikfehler) werden über Exceptions behandelt.

Perl hat kein echtes Konzept zur Ausnahmebehandlung. Es gibt allerdings den Befehl `die "Fehlergrund";`, der die Programmbearbeitung abbricht.

Umgekeht kann mit der Ausführung eines Programmteils innerhalb eines `eval`-Blockes gerade ein solcher Abbruch verhindert werden. Ein `die` führt dann lediglich zum Verlassen dieses Blockes.[3]

```
sub foo{
  # test something
  if ( ... ){
    die "something happend";
  }
}

eval{
  # do something
  foo();
};
if ($@) {
  # something happend
  print $@;
}
```

Aufgabe (Bücherei):

- Fügen Sie den Dateilesefunktionen in `Nutzerverwaltung` und `Medienverwaltung` das Werfen einer Exception hinzu, falls die gelesenen csv-Zeilen nicht die richtige Anzahl von Parametern haben.

[3] https://www.perl.com/pub/2002/11/14/exception.html/ (13.6.2018)

- Werfen Sie in `Medienverwaltung::anlegen` eine Exception, falls der Typ nicht `B` oder `C` ist.
- Fangen Sie im Verarbeiten-Teil des Hauptprogramms die Ausnahmen.
- Fügen Sie `Medienverwaltung::anlegen` einen `int`-Rückgabewert hinzu. Bilden Sie den Fehlerfall „Signatur bereits vorhandem" entsprechend ab.
- Testen Sie alle Fehlerfälle.

11 CGI-Programmierung

Perl wird gerne auch für Web-Programme benutzt. Dazu müssen die Skripte vom Webserver aufgerufen werden. Ein sehr früher Standard dafür ist das *common gateway interface* CGI.

Perl als CGI-Anwendung einrichten: [ST: S. 146]

Datenübergabe an CGI-Programme: [ST: S. 148]

Aufgabe:

Kapitel 12, Aufgabe 2

- Schreiben Sie eine HTML-Seite `add_medium.html`, die ein Formular mit Eingabefeldern für die sechs Parameter von `Medium` enthält.
- Beim Absenden soll über einen `POST`-Request das CGI-Programm `add_medium.cgi` gerufen werden. Dieses packt die Daten aus, instanziert die `Medienverwaltung` und ruft `anlegen`.
- Je nach Ergebnis von `anlegen` soll eine entsprechende Information („Datensatz angelegt" / „Datensatz konnte nicht angelegt werden: ...") dem Benutzer angezeigt werden.

12 Datenbanken

Interaktive Webprogramme oder betriebliche Anwendungen benötigen für ihre Datenhaltung Datenbanken. Eine im Web-Umfeld weit verbreitete Datenbank ist MySQL.

Datenbanken und Tabellen erstellen: [ST: S. 192]

Verbindung zum Datenbankserver herstellen: [ST: S. 194]

Abfragen erstellen: [ST: S. 197]

Abfrageergebnis ermitteln: [ST: S. 201.]

Aufgabe (Bücherei):

- Legen Sie in der Datenbank eine Tabelle mit folgenden Feldern an:

```
signatur   CHAR(20) NOT NULL,
nutzerid   INT NOT NULL,
PRIMARY KEY (signatur, nutzerid)
```

- Legen Sie ein Modul `Ausleihverwaltung` an, darin die Funktionen `anlegen` und `entfernen` mit den entsprechenden Datenbankoperationen.
- Ergänzen Sie das Hauptprogramm um diese Operationen. Beim Anlegen soll zusätzlich über die entsprechenden `lesen`-Funktionen geprüft werden, ob Signatur und Benutzernummer gültig sind.

Anmerkung: In größeren Programmen werden OR-Mapper (*object-relational mapper*) verwendet, um die Anwendungsklassen frei von SQL-Code zu halten.

13 Socket-Programmierung

Über Sockets wird unter UNIX/im Internet die Kommunikation zwischen Prozessen abgebildet. Für Perl steht unter UNIX das Modul `IO::Socket::UNIX` zur Verfügung, mit der sich die Socketkommunikation einfach implementieren lässt. Als Kommunikationspartner stehen stets Client und Server zur Verfügung. Die Kommunikation kann synchron/verbindungsorientiert oder asynchron/-verbindungslos sein.

Definition des Servers:

```
use IO::Socket::UNIX;
my $server = IO::Socket::UNIX->new(
    Type => SOCK_STREAM,    # synchron
    Local => $SOCK_PATH,    # innerhalb des Servers
    Listen => 1,
);
die "Can't create socket: $!" unless $server;
```

Lesen/Schreiben von Daten am Socket:

```
while (my $conn = $server->accept())
{
        chomp (my $line = <$conn>);
        # Daten stehen in $line zur Verf"ugung
        print $conn "$result\n"; # Antwort
}
```

Definition des Clients:

```
use IO::Socket::UNIX;
my $client = IO::Socket::UNIX->new(
        Type => SOCK_STREAM,
        Peer => $SOCK_PATH,
);
die "Can't create socket: $!" unless $client;
```

Schreiben/Lesen von Daten am Socket:

```
$client->send("$data\n");
chomp (my $ans = <$client>);
```

Für die Kommunikation mit dem Internet steht die Klasse `IO::Socket::INET` zur Verfügung, die in ähnlicher Weise genutzt wird.

Aufgabe:

Ergänzen Sie obiges Beispiel in der Weise, dass

- der Client den Anwender nach seinem Namen fragt,
- der Client den Namen an den Server schickt,
- der Server „Hallo" vor den Namen setzt und an den Client zurück gibt,
- der Client die Serverantwort ausgibt.

14 Quellen

Teich, Peter	Perl 5, Herdt Verlag, 1. Auflage, 2005
ST	Steyer, Ralph; Teich, Peter: Perl 5, Herdt Verlag, 2. Auflage, 2016