


- 1 Module und OOP
- 2 Funktionen
- 3 CGI-Programmierung
- 4 Computerlinguistische Probleme

Module (1)

- F: Wie macht man `xyz` in Perl?
A: Dafür gibt's ein Modul auf CPAN¹.

¹Comprehensive Perl Archive Network, <http://www.cpan.org/> 

Module (1)

- F: Wie macht man *xyz* in Perl?
A: Dafür gibt's ein Modul auf CPAN¹.
- Module verpacken Funktionalität in wiederverwendbarer Form
- Mittlerweile meistens in objektorientiertem Perl geschrieben
- Einbinden eines Moduls mittels „**use** <module>;“

¹Comprehensive Perl Archive Network, <http://www.cpan.org/>

Module (1)

- F: Wie macht man *xyz* in Perl?
A: Dafür gibt's ein Modul auf CPAN¹.
- Module verpacken Funktionalität in wiederverwendbarer Form
- Mittlerweile meistens in objektorientiertem Perl geschrieben
- Einbinden eines Moduls mittels „**use** <module>;“
- Wichtige mitgelieferte Module:
 - **use strict** ; aktiviert strengere Überprüfungen z.B. bezüglich Variablen- und Funktionsdeklaration.
 - **use warnings** ; warnt beim Aufruf undefinierter Funktionen, Verwendung undefinierter Werte etc. (äquivalent zum Interpreteraufwurf mit „-w“)
 - **use diagnostics** ; aktiviert ausführlichere Fehlermeldungen.
 - **use English** ; erlaubt sprechendere Namen für Spezialvariablen wie \$!, \$/ etc.

¹Comprehensive Perl Archive Network, <http://www.cpan.org/>

Module (2)

Externe Module

- Hierarchische Struktur, auf Dateisystemebene in Verzeichnissen, in Perl mit Doppelpunkten getrennt, z.B. Modul installiert als `Lingua/Ident.pm` → Einbinden mit **use** `Lingua::Ident`;

Module (2)

Externe Module

- Hierarchische Struktur, auf Dateisystemebene in Verzeichnissen, in Perl mit Doppelpunkten getrennt, z.B. Modul installiert als `Lingua/Ident.pm` → Einbinden mit **use** `Lingua::Ident`;
- Kleine Auswahl wichtiger Module für die Computerlinguistik
 - `locale` und `utf8`: Mitgeliefert; internationalisierungs- und UTF-8-Funktionen/Klassen.
 - `Lingua::*`, Module zur Sprachverarbeitung, z.B. `Lingua::Stem::Pt`: portugiesischer Stemming-Algorithmus
 - `HTML::*` und `XML::*` alles zur Verarbeitung von HTML- bzw. XML-Dateien, z.B. `XML::Writer`: Ausgabemodul für XML-Dateien.
 - `Encode::*`: Codierungsfunktionen; Umgang mit verschiedenen Zeichensätzen
 - CLUE: `MaLaga`: LA-Parser.

Objektorientierte Programmierung

- Objekte sind aus Benutzerperspektive Referenzen
- Klasse einbinden durch Benutzen des entsprechenden Moduls, z.B. `use IO::File;`
- Konstruktion: `my $obj = KLASSE->new();`, z.B.:
`my $outfile = IO::File->new(">NeueDatei");`
oder (TIMTOWTDI!)
`my $outfile = new IO::File(">NeueDatei");`

Objektorientierte Programmierung

- Objekte sind aus Benutzerperspektive Referenzen
- Klasse einbinden durch Benutzen des entsprechenden Moduls, z.B. `use IO::File;`
- Konstruktion: `my $obj = KLASSE->new();`, z.B.:
`my $outfile = IO::File->new(">NeueDatei");`
oder (TIMTOWTDI!)
`my $outfile = new IO::File(">NeueDatei");`
- Methodenaufruf ebenfalls über die Referenzsyntax:
`$xmlw->startTag("frequenzliste");`

Beispiel: XML-Ausgabe einer Frequenzliste

- `man XML::Writer!`

Beispiel: XML-Ausgabe einer Frequenzliste

- man `XML::Writer!`
- Ausgangspunkt: Frequenzklassen-Programm der letzten Stunde
- Vorgehen:
 - 1 XML-Format überlegen!

Beispiel: XML-Ausgabe einer Frequenzliste

- man `XML::Writer!`
- Ausgangspunkt: Frequenzklassen-Programm der letzten Stunde
- Vorgehen:
 - 1 XML-Format überlegen!
 - 2 Ausgabedatei mittels `IO::File` öffnen.
 - 3 Neues `XML::Writer`-Objekt erzeugen (Konfiguration mittels übergebenem Hash; `DATA_MODE?`)
 - 4 XML-Deklaration für Zeichensatz anlegen
 - 5 Start- und Endtag für Dokument erzeugen

Beispiel: XML-Ausgabe einer Frequenzliste

- man `XML::Writer!`
- Ausgangspunkt: Frequenzklassen-Programm der letzten Stunde
- Vorgehen:
 - 1 XML-Format überlegen!
 - 2 Ausgabedatei mittels `IO::File` öffnen.
 - 3 Neues `XML::Writer`-Objekt erzeugen (Konfiguration mittels übergebenem Hash; `DATA_MODE?`)
 - 4 XML-Deklaration für Zeichensatz anlegen
 - 5 Start- und Endtag für Dokument erzeugen
 - 6 Für jede Frequenzklasse Start- und End-Tag erzeugen; dazwischen einzelne Wortformen ausgeben (Methoden `startTag`, `endTag`, `dateElement`).

Weitere wichtige Funktionen

<code>split</code>	String entsprechend einem regulärem Ausdruck in eine Liste aufspalten: <code>split PATTERN [,EXPR] [,LIMIT]</code>
<code>map</code>	Block oder Ausdruck auf jedes Element einer Liste anwenden: <code>map BLOCK LIST</code> oder <code>map EXPR, LIST</code>
<code>grep</code>	Block oder Ausdruck auf jedes Element einer Liste anwenden und die Liste zurückgeben, für die <code>true</code> zurückgegeben wurde (wie <code>map</code>).
<code>length</code>	Länge eines Strings ermitteln (Achtung: nicht für Listen und Hashes!)
<code>substr</code>	Teilstring ausschneiden: <code>substr EXPR, OFFSET [,LENGTH]</code>
<code>undef</code>	Einer Variablen den Wert „undef“ zuweisen: <code>undef \$var;</code>
<code>delete</code>	Schlüssel in einem Hash löschen: <code>delete \$hash{key}</code>

Übung

- Die Datei `/etc/passwd` enthält Felder, die mit Doppelpunkt getrennt sind, z.B.: `root:x:0:0:root:/root:/bin/bash`. Die Felder bedeuten von links nach rechts:
 - Login
 - Passwort (historisch, unbenutzt)
 - UID
 - GID
 - Name
 - Heimatverzeichnis
 - Shell
- Lesen Sie die Datei zeilenweise ein und spalten Sie sie in ein Array von Feldern auf. kann.
- Speichern Sie alle Zeilen als Array-Referenzen in einem weiteren Array.
- Überlegen Sie sich, wie die Felder nach Namen angesprochen werden können, um z.B. beim Aufruf des Programm mit dem Argument „UID“ alle User-IDs der Datei auszugeben.

CGI – Allgemeines

CGI

Common Gateway Interface – Definition der Schnittstelle zwischen Webservern und externen Programmen, die Web-Inhalte dynamisch produzieren. CGI ist nicht Perl-spezifisch sondern definiert nur die Kommunikation des Webserver mit dem in einer beliebigen Sprache geschriebenen CGI-Programm.

- Die Definition der CGI-Schnittstelle ist ein offiziell abgelaufener „Internet Draft“, aber De-facto-Standard.
- Kommunikation bei Apache und den meisten anderen Servern mittels Umgebungsvariablen (Server→Programm) bzw. Textausgabe auf STDOUT (Programm→Server).
- Wichtige Ressourcen: das Apache 1.3 CGI-Tutorial und die Manpage „CGI“.


CGI in Perl

- Perl als beliebteste Programmiersprache für CGI-Programme² bietet umfangreiche Unterstützung durch das Modul `CGI.pm` und davon abhängiger spezialisierter Module.
- Weitere Möglichkeit: in Apache eingebauter Perl-Interpreter `mod_perl` mit entsprechenden Modulen.

Grundgerüst eines einfachen CGI-Skripts

```
use CGI qw(:standard);
use CGI::Carp qw(fatalsToBrowser);

print header,
      start_html("Der_Seitentitel"),
      "Irgendwelcher_Text",
      end_html;
```

²Alternativen: z.B. Python, C, Ruby, LISP-Dialekte 

Features von CGI.pm

- Modulparameter geben das ins Skript zu importierende Feature-Set an, außer `:standard` z.B. auch `:form` oder `:all`.
- Verschiedene Funktionen geben bestimmte Seitenelemente aus und sind i.d.R. durch Argumente in Hash-Syntax konfigurierbar. Z.B.:
`header()`: Standard Content-Type-Header ausgeben
`header(-expires => '5m')`: Content-Type-Header plus Information zum Gültigkeitszeitraum der Seite ausgeben.
- Es existiert auch eine OO-Schnittstelle für CGI.pm, die aber für einfache Skripte mehr Aufwand als Nutzen bringt.
- Tipp: `lynx -dump <URL>` zum Ansehen der Seite als Text, mit `-head` für HTTP-Header, mit `-source` für Quelltext.
- Übersichtlicheren HTML-Text zur Fehlersuche erzeugt das Modul `CGI::Pretty`.

Wichtige Funktionen

`start_html` Doctype, `<html>` und `<head>`-Bereich (HTML-Header, nicht zu verwechseln mit dem HTTP-Header!) ausgeben. Argumente: `-title`, `-author`, `-meta`, `-lang` u.v.m., s. Manpage!

`hx()` `h1` .. `h6` Heading-Tags mit angegebenem Inhalt

`end_html` HTML-Seite abschließen. erzeugen, s. Beispiel.

`p()` Paragraph mit angegebenem Inhalt erzeugen.

Für die meisten *Container-Tags* (`<div>`, ``, `<blockquote>`, ...) existieren entsprechende Funktionen in `CGI.pm`. Diese können mit beliebig vielen Argumenten aufgerufen werden, die zur Ausgabe einfach konkateniert werden.

Formulare

- Zur Benutzerinteraktion dienen i.d.R. HTML-Formulare. Siehe SelfHTML zu den verfügbaren Elementen.
- Wird im Browser der „submit“-Knopf gedrückt, übermittelt der Browser die eingegebenen Werte an den Server entweder per *POST*- (kodierte in den HTTP-Request-Headern) oder per *GET*-Methode (kodierte im URI).
- Das CGI-Modul erlaubt den Zugriff auf Formularfelder über die `param`-Funktion:
 - `@params = param` ; liefert die Liste aller gesetzten Parameter
 - `$foo = param("foo")` fragt den Wert eines einzelnen Parameters ab. Je nach Art des Parameters kann hier ein Skalar (z.B. für *Textfields*) oder ein Array (z.B. bei *Checkboxes*) zurückgeliefert werden.

Lexikon-Datenbank

Gegeben:

- Sammlung von Lexikoneinträgen als Einzeldateien (~msbethke/FeSh-Extract_A-ZH-x1.tgz, nach /tmp auspacken!)
- Format: Lemma im Dateinamen sowie in erster Zeile, Rest der Datei ist Definition.

Gewünscht:

- BerkeleyDB-Format für gesamtes Lexikon zur einfachen Suche und Darstellung in einer Datei.
- Falls Zeit bleibt: Webinterface zur Lexikonabfrage.

Wichtige Module

BerkeleyDB → BerkeleyDB::Btree

IO::Dir

tie

Praktische Funktion zur Arbeit z.B. mit Datenbank-Files: „bindet“ eine Variable (Array, Hash oder Handle) an ein Objekt, so dass dieses Objekt die eigentliche Implementation dieses Typs bereitstellt.

Z.B.:

```
tie(%HIST, 'NDBM_File', '/usr/lib/news/history', 1, 0);
```

erzeugt ein neues Objekt der Klasse `NDBM_File` entsprechend

```
new NDBM_File('/usr/lib/news/history', 1, 0);
```

und bindet das Hash `\%HIST` daran.

„tie“ wird sowohl von BerkeleyDB als auch `IO::Dir` implementiert.

Vorgehen

- Archiv auspacken

Vorgehen

- Archiv auspacken
- Zwischenschritt: Skript schreiben, das `tie` benutzt und ein Listing eines angebbaren Verzeichnisses liefert. Gleich als Argumente sowohl Datenbankdatei (vorerst unbenutzt) und Verzeichnisnamen vorsehen!

Vorgehen

- Archiv auspacken
- Zwischenschritt: Skript schreiben, das `tie` benutzt und ein Listing eines angebbaren Verzeichnisses liefert. Gleich als Argumente sowohl Datenbankdatei (vorerst unbenutzt) und Verzeichnisnamen vorsehen!
- Datenbank mit `BerkeleyDB::Btree` anlegen (ggf. `BerkeleyDB::Hash`) ausprobieren!

Vorgehen

- Archiv auspacken
- Zwischenschritt: Skript schreiben, das `tie` benutzt und ein Listing eines angebbaren Verzeichnisses liefert. Gleich als Argumente sowohl Datenbankdatei (vorerst unbenutzt) und Verzeichnisnamen vorsehen!
- Datenbank mit `BerkeleyDB::Btree` anlegen (ggf. `BerkeleyDB::Hash`) ausprobieren!
- Dateien einzeln bearbeiten und in Datenbank eintragen.
 - Erste Zeile → Lemma; Hash-Schlüssel (ggf. vorverarbeiten)
 - Rest → Wert im Hash