

Webcode - Übungsdateien

C30F-A647-8E52

ipso Bildung AG

In Kooperation mit dem HERDT-Verlag stellen wir Ihnen eine PDF inkl. Zusatzmedien für Ihre persönliche Weiterbildung zur Verfügung. In Verbindung mit dem Programm HERDT-Campus ALL YOU CAN READ stehen diese PDFs für Forschung und Lehre nur Mitarbeiterinnen und Mitarbeitern sowie Studierenden der oben genannten Hochschule zur Verfügung. Eine Nutzung oder Weitergabe für andere Zwecke ist ausdrücklich verboten und unterliegt dem Urheberrecht. Jeglicher Verstoß kann zivil- und strafrechtliche Konsequenzen nach sich ziehen.

Ralph Steyer, Peter Teich

2. Ausgabe, Juli 2016

ISBN 978-3-86249-587-0

Perl 5

Grundlagen und
CGI-Programmierung

PRLS



1 Informationen zu diesem Buch	4	7 Unterprogramme und Funktionen	76
1.1 Voraussetzungen und Ziele	4	7.1 Unterprogramme in Perl	76
1.2 Aufbau und Konventionen	5	7.2 Unterprogramme erstellen und verwenden	77
2 Einführung in Perl	8	7.3 Übergabe von Argumenten und Rückgabe von Werten	79
2.1 Was ist Perl?	8	7.4 Referenzen verwenden	82
2.2 Entwicklung von Perl	9	7.5 Vordefinierte Perl-Funktionen	84
2.3 Arbeitsweise von Perl	9	7.6 Schnellübersicht	86
2.4 Existenz von Perl testen und gegebenenfalls installieren	10	7.7 Übungen	86
2.5 Hilfe und Dokumentation verwenden	12	8 Datei- und Verzeichnisfunktionen	88
2.6 Das erste Perl-Programm	14	8.1 Mit Dateien arbeiten	88
2.7 Übung	15	8.2 Textdateien lesen	91
3 Programmieren mit Perl	16	8.3 In Dateien schreiben und Dateien sperren	92
3.1 Editoren für Perl-Programme	16	8.4 Statusinformationen über Dateien ermitteln	95
3.2 Anweisungen, Blöcke und Kommentare	17	8.5 Dateien zeichenweise bearbeiten	96
3.3 She-Bang, die erste Zeile	18	8.6 Funktionen des Dateisystems	100
3.4 Regeln für Bezeichner	19	8.7 Mit Verzeichnissen arbeiten	103
3.5 Daten ausgeben	20	8.8 Der Diamond-Operator (<>)	105
3.6 Schnellübersicht	23	8.9 Schnellübersicht	107
3.7 Übung	23	8.10 Übungen	108
4 Einfache Sprachelemente	24	9 Funktionen für Zeichenketten	110
4.1 Variablen	24	9.1 Zeichenketten formatiert ausgeben	110
4.2 Datentypen	32	9.2 Teile einer Zeichenkette suchen und ausschneiden	113
4.3 Operatoren	35	9.3 Zeichenketten modifizieren und umwandeln	115
4.4 Rangfolge der Operatoren	39	9.4 Weitere Funktionen für Zeichenketten	116
4.5 Übung	40	9.5 Schnellübersicht	116
		9.6 Übungen	117
5 Kontrollstrukturen	42	10 Reguläre Ausdrücke	118
5.1 Kontrollstrukturen verwenden	42	10.1 Suchen und Ersetzen mit regulären Ausdrücken	118
5.2 Anweisungen zur Bedingungsauswahl	43	10.2 Mustererkennung mit regulären Ausdrücken	119
5.3 Bedingte Wiederholungsschleifen	50	10.3 Optionen für reguläre Ausdrücke	120
5.4 Zählergesteuerte Wiederholung	54	10.4 Übereinstimmungen speichern	125
5.5 Anweisung zur Schleifen- und Programmsteuerung	55	10.5 Regeln für das Verwenden regulärer Ausdrücke	128
5.6 Schnellübersicht	56	10.6 Ersetzen von Textteilen mit regulären Ausdrücken	128
5.7 Übungen	57	10.7 Schnellübersicht	132
		10.8 Übungen	133
6 Listen und Datenfelder	58	11 Datum und Zeit ausgeben	134
6.1 Daten in Feldern speichern	58	11.1 Datums- und Zeitfunktionen in Perl	134
6.2 Arrays verwenden	59	11.2 Datum und Zeit ausgeben	135
6.3 Mehrdimensionale Arrays	65	11.3 Datum in Unix-Zeitangabe umwandeln	138
6.4 Hashes verwenden	67	11.4 Übungen	140
6.5 Arrays und Hashes in Schleifen durchlaufen	71		
6.6 Schnellübersichten	73		
6.7 Übungen	74		

12 CGI-Programme mit Perl	142
12.1 Interaktive Webseiten mit CGI-Programmen	142
12.2 Installation des Webservers Apache	143
12.3 Perl als CGI-Anwendung einrichten	146
12.4 Datenübergabe an CGI-Programme	148
12.5 Schnellübersichten	156
12.6 Übungen	157
13 Pakete, Module und OOP in Perl	158
13.1 Pakete und Module	158
13.2 Objektorientierte Programmierung (OOP) in Perl	167
13.3 Schnellübersichten	170
13.4 Übungen	171
14 Erweiterte CGI-Funktionalität	172
14.1 Das Modul <i>CGI.pm</i> verwenden	172
14.2 Header ausgeben	173
14.3 Formulardaten und Umgebungsdaten ermitteln	174
14.4 HTML-Elemente erzeugen	177
14.5 Cookies verwenden	181
14.6 Schnellübersicht	183
14.7 Übungen	184
15 Datenbanken mit MySQL	186
15.1 Datenbankmanagementsysteme und Perl	186
15.2 Installation von MySQL	187
15.3 DBI-Modul installieren	190
15.4 Datenbanken und Tabellen erstellen	192
15.5 Verbindung zum Datenbankserver herstellen	194
15.6 Abfragen erstellen	197
15.7 Abfrageergebnis ermitteln	201
15.8 Weitere Methoden des DBI-Objekts	205
15.9 Schnellübersichten	205
15.10 Übungen	206
16 E-Mails senden	208
16.1 E-Mail senden	208
16.2 Mailversand unter Unix & Co	209
16.3 Mailversand mit dem Modul <code>Mail::Sendmail</code>	211
16.4 E-Mail-Adresse prüfen	213
16.5 Übung	214
Anhang	216
A.1 Perl installieren	216
A.2 XAMPP	221
A.3 Editoren und Entwicklungstools für Perl	225
Stichwortverzeichnis	234

1 Informationen zu diesem Buch

In diesem Kapitel erfahren Sie

- ✓ an wen sich dieses Buch richtet
- ✓ welche Vorkenntnisse Sie mitbringen sollten
- ✓ welche Hard- und Software Sie für die Arbeit mit diesem Buch benötigen
- ✓ wie dieses Buch aufgebaut ist
- ✓ welche Konventionen verwendet werden

1.1 Voraussetzungen und Ziele

Zielgruppe

- ✓ Auszubildende/Studenten in IT-Berufen
- ✓ Web-Programmierer
- ✓ Administratoren in Unix/Linux-Systemen

Empfohlene Vorkenntnisse

Eine Besonderheit von Perl ist, dass Sie bereits am Anfang des Lernprozesses einfache, funktionierende Programme erstellen können. Der Einstieg in Perl ist dennoch einfacher, wenn Sie bereits Erfahrungen in einer Programmiersprache besitzen. Da die Syntax und viele Befehle von der Sprache C abstammen, sind gerade C-Kenntnisse sehr nützlich.

Folgende Kenntnisse werden für eine erfolgreiche Benutzung dieses Buchs vorausgesetzt:

- ✓ Kenntnisse im Umgang mit Windows, MacOS X oder Unix/Linux
- ✓ Kenntnisse zum Internet und World Wide Web (WWW)
- ✓ Grundkenntnisse in Programmierung
- ✓ Grundkenntnisse in der Bedienung von Anwendungsprogrammen – insbesondere Editoren und Browsern
- ✓ Unter Unix/Linux: Arbeiten mit der Shell und Installieren von Anwendungen

Lernziele

Zu Beginn erhalten Sie einen Überblick über die Historie und die Einsatzmöglichkeiten von Perl und lernen Methoden und ganz kurz bereits Werkzeuge für die Entwicklung sowie Ausführung von Perl-Skripten kennen.

Im weiteren Verlauf werden grundlegende Elemente von Perl und deren Verwendung sowie die Syntax von Perl vorgestellt. Dabei kommen auch zahlreiche vorgefertigte Funktionen aus dem API (Application Programming Interface) von Perl zum Einsatz.

Da Perl sehr oft im Internet eingesetzt wird, lernen Sie abschließend Perl im Einsatz als CGI-Skript kennen. Dabei werden unter anderem Daten auf dem Webserver entgegengenommen und auf eine MySQL-Datenbank zugegriffen.

Hinweise zur Software

- ✓ Im Buch wird von einer Erstinstallation von Perl in der Version 5.20 bzw. 5.22 ausgegangen. Die meisten Beispiele lassen sich auch anhand einer früheren Perl-Version umsetzen. Da es verschiedene Distributionen von Perl gibt und auch das Betriebssystem von Bedeutung ist, wird im Anhang auf die Installation von Perl eingegangen. Als Perl-Distributionen kommen im Buch Strawberry Perl 5.22 für Windows und ActivePerl zum Einsatz. Ebenso stellt XAMPP (s. u.) eine eigene Version von Perl zur Verfügung, auf die bei CGI-Skripten zurückgegriffen wird. Unter Linux/Unix und Mac OS X ist Perl in der Regel automatisch integriert, kann aber auch nachinstalliert werden.
- ✓ Für die Ausführung von CGI-Skripten mit Perl benötigen Sie einen Webserver. Im Buch wird der Webserver Apache eingesetzt. Apache ist Bestandteil des XAMPP-Pakets. Dieses Paket erleichtert sowohl die Installation, die Konfiguration und den Betrieb des Servers als auch die Ausführung der CGI-Perl-Skripte erheblich (vgl. Anhang).
- ✓ Zu den verwendeten Editoren bzw. Entwicklungsumgebungen und deren Installation und Konfiguration finden Sie im nächsten Kapitel sowie im Anhang wichtige Informationen.
- ✓ Bei Bedarf werden verschiedene weitere Programme und Tools vorgeschlagen.
- ✓ Die verwendeten Betriebssysteme im Buch sind Windows 10 und Ubuntu 15 als Linux-Distribution, denn gerade im WWW kommt Linux sehr oft zum Einsatz. Mac OS X wird ebenfalls berücksichtigt.
- ✓ Als Referenzbrowser werden im Buch Chrome 44, Firefox 40 und Microsoft Edge verwendet. Die Beispiele, in denen Perl für CGI eingesetzt wird, funktionieren aber auch in den meisten älteren, neueren oder anderen Browsern.

1.2 Aufbau und Konventionen

Aufbau und inhaltliche Konventionen des Buchs

- ✓ Am Anfang jedes Kapitels finden Sie die Lernziele und am Ende einiger Kapitel eine Schnellübersicht mit den wichtigsten Funktionen im Überblick.
- ✓ Die meisten Kapitel enthalten Übungen, mit deren Hilfe Sie die erlernten Kapitelinhalte einüben können.
- ✓ Die Notizseiten im Buch geben Ihnen die Möglichkeit, eigene Anmerkungen und Erkenntnisse sowie praktische Arbeitstechniken einzutragen.

Hervorhebungen im Text

Im Text erkennen Sie bestimmte Programmelemente an der Formatierung. So werden z. B. Bezeichnungen für Programmelemente wie Register immer *kursiv* geschrieben und wichtige Begriffe **fett** hervorgehoben.

Kursivschrift	kennzeichnet alle von Programmen vorgegebenen Bezeichnungen für Schaltflächen, Dialogfenster, Symbolleisten, Menüs bzw. Menüpunkte (z. B. <i>Datei - Schließen</i>) sowie alle vom Anwender zugewiesenen Namen wie Dateinamen, Ordnernamen, eigene Symbolleisten, Hyperlinks und Pfadnamen.
Courier New	kennzeichnet Programmtext, Programmnamen, Funktionsnamen, Variablennamen, Datentypen, Operatoren etc.
<i>Courier New kursiv</i>	kennzeichnet Zeichenfolgen, die vom Anwendungsprogramm ausgegeben oder in das Programm eingegeben werden.
[]	Bei Darstellungen der Syntax einer Programmiersprache kennzeichnen eckige Klammern optionale Angaben.
	Bei Darstellungen der Syntax einer Programmiersprache werden alternative Elemente durch einen senkrechten Strich voneinander getrennt.

Was bedeuten die Symbole im Buch?



Hilfreiche Zusatzinformation



Praxistipp



Warnhinweis

HERDT BuchPlus: Beispieldateien und Ergebnisdateien im Download

Nutzen Sie unsere maßgeschneiderten, im Internet frei verfügbaren Medien:



- Rufen Sie im Browser die Internetadresse www.herdt.com auf.

The screenshot shows the **HERDT** website with a search bar containing "Alles". A red box highlights the "Codes" button in the dropdown menu. A green arrow points from a callout box labeled "1 Wählen Sie Codes." to the "Codes" button. Another green arrow points from a second callout box labeled "2 Geben Sie den folgenden Matchcode ein: PRL5." to the search input field.

Weitere Medien von HERDT nutzen

Hat Ihnen das vorliegende Buch gefallen, besuchen Sie doch einmal unseren Webshop unter www.herdt.com. Sie möchten beispielsweise ...

- ✓ mehr über SQL erfahren? Hierzu empfehlen wir Ihnen das HERDT-Buch *SQL – Grundlagen und Datenbankdesign*.

Wir wünschen Ihnen viel Spaß und Erfolg mit diesem Buch.

Ihr Redaktionsteam des HERDT-Verlags

2 Einführung in Perl

In diesem Kapitel erfahren Sie

- ✓ Grundlegendes über die Programmiersprache Perl
- ✓ Informationen zur Geschichte und Entwicklung von Perl
- ✓ wie Sie Perl installieren und testen

Voraussetzungen

- ✓ Installation von Programmen unter Unix/Linux, Mac OS X oder Windows
- ✓ Umgang mit einem Editor

2.1 Was ist Perl?

Perl und Unix

Perl ist eine Programmiersprache aus dem Unix-Bereich, die bereits vor der Verbreitung des Internets Verwendung fand. Dabei wurde Perl anfangs vor allem von Systemadministratoren verwendet, um die komplizierten Verwaltungsabläufe und die alltäglichen Arbeiten unter Unix zu automatisieren.

Der häufigste Einsatzzweck von Perl lag in der Vergangenheit im Bearbeiten und Auswerten großer Datenmengen, wie sie z. B. in Log-Dateien oder Quelltextdateien vorkommen. Daher kommt auch der Name Perl als Abkürzung für **Practical Extraction and Reporting Language**, der auf Deutsch so viel wie „praktische Sprache für Auswertung und Ausfiltern (von Daten)“ bedeutet.

Perl und das WWW

Die ersten Webserver wurden vor allem unter Unix-Systemen betrieben. Im Laufe der Entwicklung des WWWs zu einem interaktiven Medium wurden bald Programme bzw. Skripte auf der Seite des Webservers benötigt, die beispielsweise Formulareingaben der Benutzer auswerten konnten, anfallende Daten verarbeiten und dem Webserver wieder zur Verfügung stellen. Die standardmäßige Verfügbarkeit von Perl unter Unix und dessen einfache Anwendung gaben den Ausschlag, dass sich Perl als Standard für so genannte **CGI-Programme** (Common Gateway Interface) entwickelt hat.

Aktuelle Einsatzgebiete von Perl

Heute ist Perl für fast alle Betriebssysteme erhältlich und wird in verschiedenen Umgebungen eingesetzt. Dazu gehört neben der CGI-Programmierung auch die Verarbeitung von Dateien, Skripterstellung unter Microsoft Windows, der Zugriff auf Datenbanken und sogar die Erstellung grafischer Benutzeroberflächen.

Perl lernen

Es gibt sicher einfachere Programmiersprachen als Perl, denn die Syntax basiert auf der Sprache C und verfügt über viele Besonderheiten. Gerade bei größeren Programmen kann Perl-Quelltext für Einsteiger gelegentlich kompliziert und undurchschaubar erscheinen. Für fortgeschrittene Perl-Programmierer liegen die Stärken der Sprache jedoch genau in diesen Besonderheiten, für Einsteiger bedeuten sie eine gewisse Einarbeitungszeit. Grundsätzlich können Sie mit Perl aber auch ohne viel Programmiererfahrung recht schnell Ihre ersten Programme schreiben und damit für Erfolgsergebnisse sorgen.



Perl ist eine sogenannte **Skriptsprache**. Skripte sind Programme, die als ASCII-Datei im Quelltext vorliegen und erst zur Ausführungszeit interpretiert werden. Im Gegensatz dazu stehen Programme, die meist aus umfangreicheren Quelltexten und mehreren Dateien bestehen und vor der Ausführung kompiliert werden müssen. Unter Perl können die Begriffe „Programm“ und „Skript“ synonym verwendet werden und das wird im Buch auch so gehalten. In der Betriebssystemumgebung von Microsoft wird für Skripte meistens die Bezeichnung Makro verwendet.

2.2 Entwicklung von Perl

Die Programmiersprache Perl wurde 1987 von Larry Wall erschaffen. Basierend auf C und unter Einfluss von Basic, der Unix-Shell und den Unix-Utilities *awk* und *sed* schuf er eine neue Sprache, die er kostenlos im Internet veröffentlichte. Perl verwendet aus vielen anderen Programmiersprachen das Beste und verbindet es zu einer einzigen funktionellen Sprache.

Perl wird von einer Gruppe freiwilliger Entwickler auf der ganzen Welt weitergeführt. Ähnlich wie bei Mozilla oder Linux handelt es sich um eine OpenSource-Programmiersprache, die kostenlos und frei im Quelltext zur Verfügung steht.

Die aktuelle Versionsreihe ist immer noch Perl 5. Perl 5.0 wurde bereits 1994 veröffentlicht und war der bis dahin größte Fortschritt für die Sprache. Seit der Zeit entwickelt sich die Sprache kontinuierlich weiter, obwohl die Hauptversionsnummer 5 erhalten blieb. Dazu erscheinen immer wieder neue, von Fehlern bereinigte und teilweise um neue Features erweiterte Versionen, die mit einer zusätzlichen Ziffer gekennzeichnet werden, die wiederum mit einer dritten Ziffer weitere Unterversionen angeben kann. So wurde etwa Perl 5.6 im Januar 2002, veröffentlicht, 5.16 im Mai 2012, 5.20 im Mai 2014 und 5.22 im Juni 2015, wobei derzeit die Versionsreihe 5.20.x weiterhin gepflegt wird (Stand 9. Mai 2016: Version 5.24.0).

Die Versionen 5.18.2, 5.16.3, 5.14.4, 5.12.5, 5.10.1 und 5.8.9 sind Schlusspunkte ihrer jeweiligen Reihe. Dabei werden jedoch immer noch sicherheitsrelevante Verbesserungen bis zu 3 Jahre nach Veröffentlichung einer Version nachgereicht.

Sehr wichtig ist, dass bei Kernmodulen auch bei einer Weiterentwicklung meist auf eine Kompatibilität bis zu 5.6 geachtet wird. Bei wichtigen sogenannten **CPAN-Modulen** wird meist auf eine Kompatibilität zu der Version 5.8.3 geachtet. CPAN (Comprehensive Perl Archive Network) ist ein weltweit gespiegeltes Online-Repository für Perl-Module, -Anwendungen und -Dokumentationen.

Die Versionsreihe 5 soll auch in Zukunft noch weiter entwickelt werden, obwohl es bereits seit dem Jahr 2000 eine Version 6 von Perl gibt. Die nächsten Perlversionen werden also unter 5.x und nicht Perl 6 veröffentlicht, denn Perl 6 ist ein Langzeitprojekt, in dem die Sprache, der Interpreter und die umgebende Infrastruktur vollständig neu gestaltet werden sollen. Perl 6 wird als „Schwester sprache“ propagiert, ohne jegliche Absicht, Perl 5 mittelfristig zu ersetzen.

Die jeweils neueste Version erhalten Sie auf der Webseite www.perl.com im Bereich *Downloads*. Eine andere sehr gute und wichtige Referenzseite im Internet ist www.perl.org.

2.3 Arbeitsweise von Perl

So funktioniert Perl

Perl ist eine interpretierte Skriptsprache, d. h., der Quelltext wird erst zur Ausführungszeit in vom Computer ausführbaren Code umgewandelt. Der Perl-Interpreter besitzt dabei eine Besonderheit: Der Quellcode wird zuerst vollständig in einen perlinternen Zwischencode (so genannten **Bytecode**) umgewandelt und optimiert. Erst danach wird der Zwischencode ausgeführt. Dies führt zu der, für eine Interpretersprache, sehr hohen Verarbeitungsgeschwindigkeit.

Programme in einer kompilierten Sprache wie C oder C++ müssen vor ihrem Einsatz für jedes Betriebssystem neu übersetzt werden. Perl-Programme können dagegen meist sofort auf jeder Plattform ausgeführt werden, auf der ein Perl-Interpreter vorhanden ist.

Perl verwenden

Perl besitzt keine grafische Benutzeroberfläche wie andere Programmiersprachen. Der Aufruf von Perl-Programmen erfolgt aus einer Befehlszeile. Unter Windows erfolgt der Aufruf in der Konsole und unter Unix/Linux/MacOS X in der Shell bzw. Konsole (oft auch Terminal genannt).



Um einheitlich für jedes Betriebssystem agieren zu können, soll im Buch in der Regel von einer Konsole gesprochen werden, wenn nicht ausdrücklich eine Besonderheit betont werden soll.

Perl-Skripte können aber auch von einem Webserver aufgerufen werden. Perl kann dabei als CGI-Modul in einen Webserver, z. B. Apache, eingebunden werden. Die Perl-Programme werden dann innerhalb der CGI-Umgebung des Webservers ausgeführt.

Programme erstellen

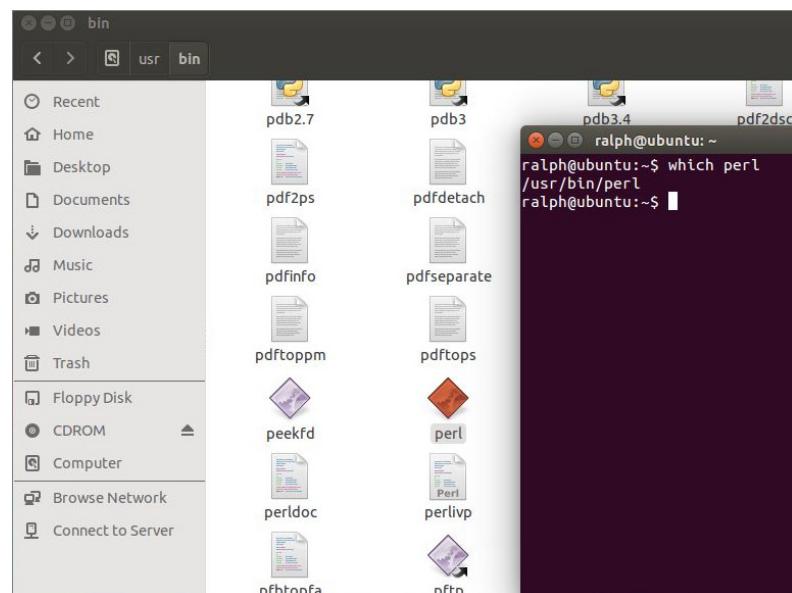
Perl-Programme können in einem beliebigen Texteditor erstellt werden. Dazu ist z. B. der unter Windows standardmäßig vorhandene Texteditor *Notepad* ausreichend. Unter Unix würde ein ganz einfacher Editor wie *vi* genügen.

Mehr Komfort bieten jedoch Texteditoren, die eine Syntaxhervorhebung für Perl-Programme besitzen oder spezielle integrierte Entwicklungsumgebungen (IDE) für Perl. Im Anhang werden einige Editoren und IDEs vorgestellt.

2.4 Existenz von Perl testen und gegebenenfalls installieren

Um mit Perl programmieren zu können, muss Perl installiert sein. Während Perl unter Windows in der Regel nachinstalliert werden muss, ist Perl unter Linux/Unix, aber auch Mac OS X (was die gleiche technische Basis hat) in der Regel bereits automatisch vorhanden. Das bedeutet, dass Sie bei einer aktuellen Linux/Unix/Mac OS X-Version sehr wahrscheinlich auch eine aktuelle Perl-Version installiert haben.

Ein vorhandenes Perl befindet sich dann meist in den Verzeichnissen */usr/bin/perl* oder */usr/local/bin/perl*. Mit der Anweisung `which perl` in einer Konsole können Sie nachsehen, wo sich Ihre Perl-Standardinstallation befindet. Der Befehl funktioniert auch in neuen Windows-Versionen.



Die Standardinstallation von Perl in einem Linux-System



Im Anhang wird ausführlich darauf eingegangen, wie Sie verschiedene Versionen bzw. Distributionen von Perl unter Windows als auch Linux/Unix/Mac OS X installieren können. Dabei können Sie auch mehrere Versionen von Perl auf Ihrem Rechner installieren bzw. verwenden.

Installation testen

Um zu testen, ob Perl auf Ihrem Rechner installiert ist, können Sie in einer Konsole den Perl-Interpreter mit dem Parameter `-v` aufrufen. Als Ergebnis erhalten Sie die Versionsbezeichnung von Perl. Dabei müssen Sie ggf. den Pfad zum Installationsverzeichnis von Perl angeben.

- ▶ Rufen Sie unter Windows die Eingabeaufforderung/Konsole auf.
Je nach Version von Windows müssen Sie dazu unterschiedlich vorgehen. Unter Windows 10 können Sie beispielsweise mit der rechten Maustaste auf den Start-Knopf und im daraufhin eingeblendeten Kontextmenü auf *Eingabeaufforderung* klicken.
Alternativ können Sie die Eingabeaufforderung mit dem Befehl `cmd` öffnen. Dazu können Sie auch drücken.
- ▶ Unter Linux/Unix/Mac OS X öffnen Sie für den Aufruf eine Konsole. Dabei gibt es diverse Wege, wie Sie eine solche Befehlszeile erhalten und die verschiedenen Distributionen stellen zudem oft eigenständige Wege zur Befehlszeile zur Verfügung. Sehr oft finden Sie in dem Startmenü Ihrer Linux-Distribution einen Eintrag *Terminal*, *Console*, *Konsole*, *Shell* oder *Bash*. Ebenso gibt es so gut wie immer ein passendes Icon auf dem Desktop. Damit können Sie eine Konsole öffnen.

```
C:\WINDOWS\system32>perl -v
This is perl, v5.8.8 built for msys
Copyright 1987-2006, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl". If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.

C:\WINDOWS\system32>
```

Versionsanzeige unter Windows 10 ohne Pfadangabe zu einer speziellen Perl-Version

```
C:\WINDOWS\system32>F:\Strawberry\perl\bin\perl -v
This is perl 5, version 20, subversion 2 (v5.20.2) built for MSWin32-x64-multi-thread
Copyright 1987-2015, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl". If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.

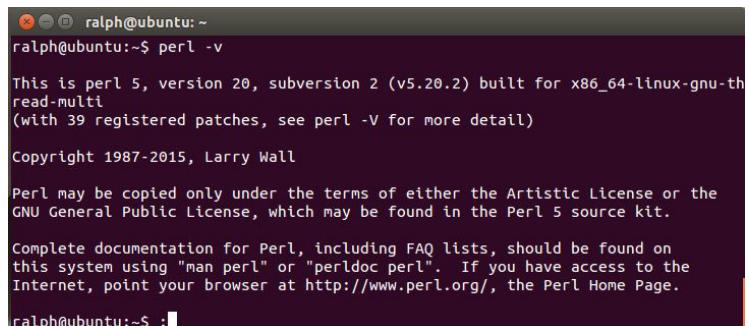
C:\WINDOWS\system32>
```

Versionsanzeige unter Windows 10 mit einer Pfadangabe zu einer speziellen Perl-Version

- Geben Sie den Befehl `perl -v` ein.

Das Ergebnis sollte einer der Anzeigen in den drei Abbildungen entsprechen.

Sollte eine Fehlermeldung auftauchen, müssen Sie entweder gezielt den Pfad zum Installationsverzeichnis von Perl angeben oder Perl nachinstallieren (vgl. Anhang).



```
ralph@ubuntu:~$ perl -v
This is perl 5, version 20, subversion 2 (v5.20.2) built for x86_64-linux-gnu-thread-multi
(with 39 registered patches, see perl -V for more detail)

Copyright 1987-2015, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl". If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.

ralph@ubuntu:~$ ;
```

Versionsanzeige in einem Terminal unter Linux

2.5 Hilfe und Dokumentation verwenden

Bei der Installation von Perl wird gleichzeitig eine umfangreiche Online-Hilfe und Dokumentation installiert, die auch einige Beispieldateien enthält.

Als erstes sollen die sogenannten **Man pages** genannt werden, die schon lange im Umfeld von Unix Hilfe-informationen bereitstellen. Bei der Installation von Perl können Sie angeben, ob die Man pages mit installiert werden sollen. Damit erhalten Sie lokalen Zugriff auf diese Hilfe. Andernfalls können Sie die Man pages als Onlinehilfe nutzen. In den Man pages finden Sie alle notwendigen Informationen zu Perl.

Die einzelnen Perl-Hilfedateien werden mit Kurzwörtern bezeichnet, die ihren Ursprung in den Unix-Manual-Seiten haben. Die folgende Tabelle listet einige wichtige Kurzwörter für entsprechende Hilfeseiten:

<code>perl</code>	Allgemeine Hauptseite
<code>perlsyn</code>	Informationen zur Syntax von Perl, Kontrollelemente, Schleifen
<code>perlop</code>	Aufführung und Verwendung der Operatoren
<code>perlfunc</code>	Erklärung der eingebauten Funktionen
<code>perlre</code>	Detaillierte Informationen zu regulären Ausdrücken
<code>perldata</code>	Hilfe zu den verschiedenen Datentypen
<code>perlsub</code>	Erklärung von benutzerdefinierten Funktionen und Prozeduren

Hilfe unter Linux/Unix/Mac OS X

Der Aufruf der Man pages erfolgt in Linux/Unix/Mac OS X über die Konsole.

```
man kurzwort
```

Nach dem Befehl `man` fügen Sie die Bezeichnung der Hilfedatei ein. Eine Übersichtsseite erhalten Sie mit `man perl`.

```
ralph@ubuntu:~
```

PERL(1) Perl Programmers Reference Guide PERL(1)

NAME

perl - The Perl 5 language interpreter

SYNOPSIS

```
perl [ -sTtuUWX ] [ -hv ] [ -V[:configvar] ]
[ -cw ] [ -d[t][:debugger] ] [ -D[number/list] ]
[ -pna ] [ -Fpattern ] [ -l[octal] ] [ -0{octal/hexadecimal} ]
[ -Idir ] [ -m[-]module ] [ -M[-]module...' ] [ -f ]
[ -C [number/list] ] [ -S ] [ -x[dir] ]
[ -i[extension] ]
[ [-e|-E] 'command' ] [ -- ] [ programfile ] [ argument ]...
```

For more information on these options, you can run "perldoc perlrun".

GETTING HELP

The `perldoc` program gives you access to all the documentation that comes with Perl. You can get more documentation, tutorials and community support online at <<http://www.perl.org/>>.

If you're new to Perl, you should start by running "perldoc perlintro", which is a general intro for beginners and provides some background to Manual page perl(1) line 1 (press h for help or q to quit).

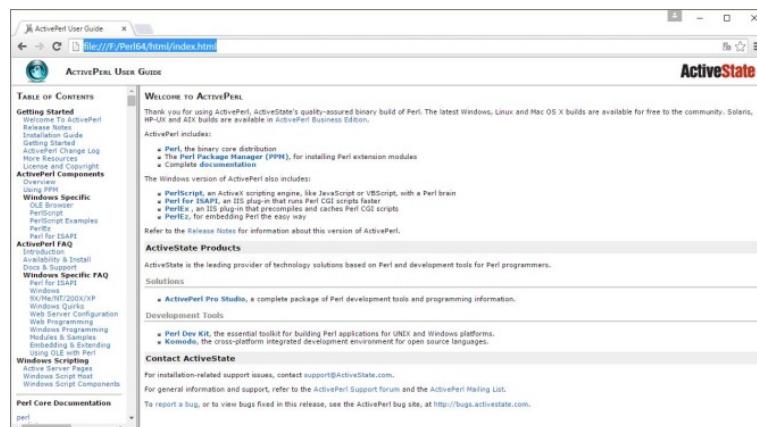
Die Hilfe in einem Terminal unter Linux

Hilfe unter Windows bzw. Perl-Distributionen

Unter Windows werden in der Regel keine Man pages genutzt. Die verschiedenen modernen Distributionen stellen eigene Hilfen in Form von Webseiten bereit. Die Hilfe zum ActiveState-Perl liegt beispielsweise als HTML-Dateien im Unterverzeichnis `html` des Perl-Installationsverzeichnisses vor, z. B. `C:\Perl64\html`.

Rufen Sie die Datei `index.html` in Ihrem Webbrowser auf, und folgen Sie den Links auf der Seite. Unter dem Stichwort *Perl Core Documentation* finden Sie auch hier die Kurzwörter für die Dokumente der Perl-Hilfe.

Unter <http://perldoc.perl.org/perl.html> finden Sie online ebenfalls die Informationen.



Die Hilfeseiten von ActiveState-Perl unter Windows

Weitere Hilfe im Internet

Im Internet finden Sie eine Vielzahl an Dokumentationen, Hilfedateien und Beispielen zu Perl:

Seite	Erklärung
http://perldoc.perl.org/	Umfangreiche Dokumentation zu allen Themen der Programmierung mit Perl, auch archivierte Artikel mit Programmertipps.
http://strawberryperl.com/	Die Strawberry-Distribution von Perl für Windows.
http://wiki.selfhtml.org/wiki/Perl	Ein Wiki zu Perl mit wichtigen Informationen und Diskussionen.
http://www.activestate.com/	Informationen zur Perl-Programmierung unter Windows. Hier erhalten Sie auch die ActivePerl-Distribution.

Seite	Erklärung
http://www.cpan.org/	CPAN steht für Comprehensive Perl Archive Network und ist eine Sammlung vieler Perl-Module und -Erweiterungen. Unter http://www.cpan.org/ports/ finden Sie Verweise auf Binärdistributionen für eine Vielzahl von Betriebssystemen. Diese enthalten in der Regel auch eine Installationsanleitung.
http://www.cygwin.com/	Wenn Sie Windows verwenden, können Sie neben speziellen Perl-Distributionen für Windows auch die Perl-Distributionen für Linux/Unix nutzen, wenn Sie die freie Cygwin-Umgebung als virtuelle Emulation verwenden. Diese enthält viele Linux/Unix-Programme – u. a. auch Perl.
http://www.perl.de/	Diskussionsforen zum Thema Perl in deutscher Sprache.
http://www.perl.org/	Sammlung von Links zu vielen Perl-spezifischen Themen.
http://www.pm.org/	Perl-Mongers, Gemeinschaft der Perl-Programmierer mit Links zu User Groups
https://de.wikipedia.org/wiki/Perl_(Programmiersprache)	Informationen zu Perl in Wikipedia

2.6 Das erste Perl-Programm

Das folgende Perl-Programm zeigt Ihnen als erstes kleines Beispiel den Aufbau und die Ausführung eines Perl-Programms. Die Bedeutung der Syntax der verwendeten Anweisungen werden Sie in den folgenden Kapiteln kennenlernen.

Beispiel: *HelloWelt.pl*

```

① #!/usr/bin/perl
② print "Bitte geben Sie Ihren Namen ein: ";
③ $name=<STDIN>;
④ chomp ($name);
⑤ print "Guten Tag $name!\n";

```

- ① Hier wird der Pfad zum Perl-Interpreter angegeben. Mit dieser Zeile sollte jedes Perl-Programm beginnen, wenn Sie das Perl-Programm unter Linux/Unix starten oder über einen Webserver ausführen, der auf einen Perl-Interpreter unter Linux/Unix zugreift. In allen anderen Fällen wird die Zeile ignoriert. Sie schadet aber auch nicht.
- ② Mit der Anweisung `print` erfolgt eine Ausgabe auf dem Bildschirm. In diesem Fall wird eine Aufforderung zur Eingabe eines Namens angezeigt.
- ③ Die Eingabe des Benutzers wird von der Standardeingabe, normalerweise der Tastatur, eingelesen und in der Variablen `$name` gespeichert. Die Eingabe muss mit `\n` abgeschlossen werden.
- ④ Die Perl-Funktion `chomp` entfernt alle führenden oder abschließenden Leerzeichen.
- ⑤ Mit der `print`-Anweisung erfolgt wieder die Ausgabe auf dem Bildschirm. Dabei wird auch der eingebaute Name durch die Variable `$name` angezeigt. Die Angabe `\n` bewirkt einen Zeilenvorschub.

Die Vorgehensweise

- ▶ Starten Sie einen beliebigen Texteditor.
- ▶ Öffnen Sie die Datei *HelloWelt.pl* aus dem vorhergehenden Beispiel, oder geben Sie den Quelltext des Programms wie angegeben ein.
- ▶ Öffnen Sie eine Konsole.
- ▶ Wechseln Sie zum Verzeichnis, in dem das Perl-Programm gespeichert ist, beispielsweise mit dem folgenden Befehl:
`cd f:\PerlProgramme\Beispiele\Kapitel_02.`
- ▶ Starten Sie das Programm mit dem Befehl `perl HelloWelt.pl`. Unter Umständen müssen Sie den vollständigen Pfad zum Perl-Interpreter angeben, wenn der Perl-Interpreter nicht gefunden wird.

```
F:>>cd F:\PerlProgramme\Kapitel_02
F:\PerlProgramme\Kapitel_02>perl HelloWelt.pl
Bitte geben Sie Ihren Namen ein: Ralph
Guten Tag Ralph!
F:\PerlProgramme\Kapitel_02>
```

Das Beispielprogramm in Aktion

Wenn Sie Perl unter Windows korrekt installiert haben, können Sie die Programme auch durch einen Doppelklick auf die jeweilige Datei im Explorer starten. Die Konsole wird jedoch nach dem Ende des Programms sofort wieder geschlossen. Fügen Sie Ihrem Perl-Programm am Ende die Anweisung `<STDIN>;` hinzu, um dies zu verhindern.



Unter Windows können Sie Perl-Programme auch mit einem Doppelklick starten, wenn das konfiguriert wurde.

2.7 Übung

Basiswissen zu Perl

Übungsdatei: --

Ergebnisdatei: --

1. Perl ist eine interpretierende Programmiersprache. Was bedeutet dies?
2. Geben Sie an, wie ein Perl-Programm gestartet wird.
3. Rufen Sie die Onlinehilfe oder die lokal installierte Hilfe von Perl auf, und suchen Sie nach den vordefinierten Funktionen in der Hilfdatei *perfunc*.

3 Programmieren mit Perl

In diesem Kapitel erfahren Sie

- ✓ welche Editoren Sie verwenden sollten
- ✓ wie Sie Anweisungen und Kommentare angeben
- ✓ was in der ersten Zeile eines Perl-Programms steht
- ✓ wie Sie Daten ausgeben

Voraussetzungen

- ✓ Starten von Perl-Programmen

3.1 Editoren für Perl-Programme

Zum Erstellen und Bearbeiten von Perl-Programmen benötigen Sie nur einen Editor. Bei der Arbeit unter Windows reicht für einfache Programmierprojekte der zum Betriebssystem gehörige Editor *Notepad*. Bei größeren Programmierprojekten mit komplexem Programmcode sollten Sie auf einen Editor zurückgreifen, der folgende Funktionen besitzt:

- ✓ Syntax-Hervorhebung, auch **Syntax-Highlighting** genannt, um Schlüsselwörter, Variablen und Perl-Funktionen automatisch farbig hervorzuheben.
- ✓ Automatische Einrückung des Quelltexts, um die Struktur des Programmes schneller erfassen zu können.
- ✓ Einfalten (engl. Folding) zum Verstecken von Programmblöcken, die momentan nicht bearbeitet werden.
- ✓ Suche nach zusammengehörigen Klammernpaaren, damit bei verschachtelten Programmblöcken nach möglichen Fehlern gesucht werden kann.

Im Internet finden Sie verschiedene kostenlos verfügbare Editoren mit Syntax-Hervorhebung für Perl und vielen anderen Funktionen, wie beispielsweise Notepad++ (<https://notepad-plus-plus.org/>).

Unter allen auf Unix basierenden Betriebssystemen können Sie den Editor *emacs* (<https://www.gnu.org/software/emacs/>) verwenden, der die oben genannten Funktionalitäten übertrifft und auf vielen Unix-Systemen (darunter Linux und Mac OS X) bereits installiert ist. Dieser Editor steht mittlerweile aber auch für Windows zur Verfügung. Die Bedienung erfordert allerdings etwas Lernaufwand.

Noch mehr Unterstützung bieten integrierte Entwicklungsumgebungen (IDEs), die allerdings eine gewisse Einarbeitungszeit benötigen.



- ✓ Im Anhang finden Sie mehr Informationen zu verschiedenen Editoren und IDEs und zu deren Installation.
- ✓ Sie speichern in jedem Fall Perl-Programme als reinen ASCII-Text. Dabei sollten Sie die Dateierweiterung .pl verwenden.

3.2 Anweisungen, Blöcke und Kommentare

Anweisungen

Perl ähnelt in seiner Syntax den Programmiersprachen C, C++, PHP und Java. Wie in diesen Sprachen muss in Perl jede Anweisung mit einem **Semikolon** ; abgeschlossen werden.

Syntax von Anweisungen

```
print "Hallo Welt!";
print("Hallo Welt");
```

Funktionsaufrufe wie beispielsweise print "Hallo Welt!" können im Gegensatz zu vielen anderen Sprachen mit oder ohne runde Klammern geschrieben werden.



Anweisungsblöcke

Zusammengehörende Anweisungen, z. B. in einer Schleife, bilden einen Anweisungsblock. In Perl werden solche Blöcke in geschweifte Klammern { und } eingeschlossen – genauso wie in C, C++, Java oder JavaScript. Der gesamte Block gilt als eine Anweisung.

Syntax von Blöcken

```
{
Anweisung1;
Anweisung2;
...
}
```

Damit ein Perl-Programm syntaktisch korrekt ist und ausgeführt werden kann, müssen alle geöffneten Blöcke auch wieder geschlossen werden. In umfangreichen Programmen kann die Vielzahl von Klammern schnell unübersichtlich werden. Ein Editor, der zusammengehörige Klammerpaare suchen kann, ist dabei von Nutzen.



Kommentare

Ein Kommentar ist eine Zeile in Ihrem Quelltext, die nicht als Code interpretiert und ausgeführt wird. Sie können Kommentare zur **Dokumentation** Ihres Programms verwenden. Dies hilft Ihnen, einen Quelltext auch nach längerer Zeit zu verstehen. Kommentare enthalten z. B. Informationen zur Verwendung einer eigenen Funktion, beschreiben den Zweck von Variablen oder erklären einen Algorithmus. In Perl wird eine Kommentarzeile mit einer Raute # eingeleitet. Der Text bis zum Ende der Zeile wird als Kommentar aufgefasst.

Im Gegensatz zu anderen Sprachen gibt es in Perl keine mehrzeiligen Kommentare. Sie müssen jede Zeile erneut mit dem Zeichen # beginnen. Manche Editoren erleichtern das Auskommentieren und Beseitigen von Kommentaren, indem einfach alle zu bearbeitenden Zeilen markiert werden und dann ein Befehl des Editors das für alle Zeilen automatisch erledigt.



Syntax von Kommentaren

```
# Dies ist ein Kommentar
```



Sie können Kommentare auch verwenden, um zu Testzwecken eine bestimmte Anweisung von der Verarbeitung auszuschließen. Setzen Sie dazu vor die betreffende Anweisung das Kommentarzeichen `#`.

Allgemeine Layoutregeln

Perl-Programme besitzen kein vorgegebenes Format. Sie können daher theoretisch alle Anweisungen in eine Zeile schreiben. Für eine bessere Übersichtlichkeit sollten Sie Ihr Programm jedoch lesbar strukturieren. Dies gilt besonders dann, wenn Sie den Quelltext an andere Programmierer weitergeben möchten.

Eine gute Struktur, Kommentare und geeignete Variablennamen verbessern die Lesbarkeit Ihrer Programme:

- ✓ Schreiben Sie möglichst jede Anweisung in eine eigene Zeile.
- ✓ Verwenden Sie Leerzeichen und Tabulatoren, um zusammengehörende Anweisungen in Blöcken einzurücken.
- ✓ Nutzen Sie Kommentare, um Ihren Quelltext zu dokumentieren.
- ✓ Brechen Sie lange Zeilen sinnvoll um.
- ✓ Verwenden Sie verständliche Variablennamen. Eine sprechende Variable mit dem Namen `$alter` ist besser als `$a`.

Weitere Hinweise zum Layout von Perl-Programmen finden Sie in der Hilfdatei `perlstyle`.

3.3 She-Bang, die erste Zeile

She-Bang (`#!`)

Unter Unix und einigen anderen Plattformen sollte ein Perl-Programm mit der so genannten She-Bang in der ersten Zeile beginnen. She-Bang ist die englische Abkürzung für die Zeichen `#` (Sharp) und `!` (Bang).

Die She-Bang wird benötigt, um den **Pfad zum Perl-Interpreter** festzulegen.

Syntax der She-Bang

```
#!/Pfad/zu/perl Optionen
```

- ✓ Die She-Bang beginnt mit den Zeichen `#!`. Danach folgt der Pfad zum Perl-Interpreter. Der Standardpfad unter Unix lautet dabei `/usr/bin/perl`.
- ✓ Nach dem Pfad können zusätzliche Optionen für den Interpreter übergeben werden, beispielsweise bewirkt die Option `-w` die Ausführung im Warning-Modus.



Perl-Interpreter unter Windows und Mac OS benötigen die She-Bang nicht. Ist sie vorhanden, wird der angegebene Pfad ignoriert. Die Optionen werden jedoch an den Interpreter übergeben.

Optionen des Perl-Interpreters

In der She-Bang können Optionen zur Steuerung des Perl-Interpreters übergeben werden. Die gleichen Optionen können Sie auch beim direkten Aufruf des Interpreters über die Eingabeaufforderung bzw. Shell übergeben. Einige Optionen, z. B. -c zur Syntaxprüfung, sind nur beim direkten Aufruf sinnvoll.

Die folgende Tabelle zeigt eine Auswahl wichtiger Optionen:

Option	Erklärung
-c	Der Interpreter prüft die Syntax des Programms und zeigt Fehlermeldungen an, ohne es auszuführen.
-d	Das Programm wird mit dem Perl-Debugger ausgeführt.
-e Anweisungszeile	Die Anweisungszeile mit Perl-Anweisungen wird direkt ausgeführt. Es wird kein Perl-Programm geöffnet.
-h	Es wird eine Hilfe zu den Perl-Optionen angezeigt.
-P	Das Programm wird vor der Ausführung durch einen C-Präprozessor bearbeitet.
-S	Bei der Suche nach Programmen wird die UmgebungsvARIABLE PATH von Perl verwendet.
-T	Führt so genannte Taint-Kontrollen aus Sicherheitsgründen durch. Besonders in CGI-Programmen wird dadurch das Ausführen unerlaubter Systembefehle unterbunden.
-v	Die exakte Perl-Version wird angezeigt.
-V	Eine Zusammenfassung der wichtigsten Konfigurationswerte wird angezeigt.
-w	Es werden Warnungen zur Syntax ausgegeben, auch wenn sie keine Fehler erzeugen.
-W	Es werden alle Warnungen angezeigt.
-X	Alle Warnungen werden deaktiviert.

3.4 Regeln für Bezeichner

Alle Konstanten, Variablen, Prozeduren und Funktionen müssen einen Namen erhalten, über den sie im Programm angesprochen werden können. Diese Namen werden **Bezeichner** genannt und müssen folgende Bedingungen erfüllen:

- ✓ Es dürfen nur Buchstaben, Ziffern und der Unterstrich verwendet werden. Umlaute sind nicht erlaubt.
- ✓ Sie müssen mit einem Buchstaben oder einem Unterstrich beginnen.
- ✓ Es dürfen keine Leerzeichen im Namen verwendet werden.
- ✓ Es sollten aussagekräftige Namen verwendet werden, die den Verwendungszweck widerspiegeln.

Perl unterscheidet zwischen Groß- und Kleinschreibung bei den Bezeichnern. Die Variable \$alter ist daher nicht identisch mit einer Variablen \$Alter.



3.5 Daten ausgeben

Zeichenketten und Variablen ausgeben

Für die Ausgabe von Daten wird in Perl die Anweisung `print` verwendet. Die Daten werden danach am Bildschirm angezeigt.

Syntax der `print`-Anweisung

```
print "Daten";
print("Daten");
print "Daten1", 12345, $variable;
print "Daten $variable";
```

- ✓ Die `print`-Anweisung kann mit oder ohne runde Klammern verwendet werden.
- ✓ Nach der Anweisung folgt eine Zeichenkette, eine Zahl oder ein Variablenname.
- ✓ Mehrere auszugebende Daten werden durch Kommata getrennt. Die gesamte Anweisung endet mit einem Semikolon.



Auszugebende Texte werden in Anführungszeichen `"` oder Hochkommata `'` als sogenannte Zeichenkette eingeschlossen. Bei der Verwendung von Anführungszeichen werden alle in der Zeichenkette angegebenen Variablen durch ihren jeweiligen Wert ersetzt. Möchten Sie innerhalb eines Textes die Zeichen " oder ' verwenden, müssen Sie vor das entsprechende Zeichen einen Backslash `\` setzen.

Beispiel: *DatenAusgabe.pl*

Das Beispiel definiert zwei Variablen und verwendet verschiedene Möglichkeiten für die Datenausgabe mit der `print`-Anweisung.

```
#!/usr/bin/perl

① $datum=localtime;
$name="Herb Kater";

② print $datum;
③ print "\n";
④ print "Der Name lautet \"$name\".\n";
⑤ print 'Der Name lautet ', $name, '.', "\n";
⑥ print 'Der Name lautet \"$name\".\n';
```

- ① Der Variablen `$datum` wird über die Perl-Funktion `localtime` das aktuelle Datum zugewiesen. Danach wird die Variable `$name` mit einer Zeichenkette definiert.
- ② Der Inhalt der Variablen `$datum` wird am Bildschirm angezeigt.
- ③ Der Backslash `\` wird zur Kennzeichnung so genannter Escape-Zeichen verwendet. Das Escape-Zeichen `\n` bewirkt einen Zeilenvorschub.
- ④ An dieser Stelle erfolgt die Ausgabe von Text in Anführungszeichen. Die Variable `$name` wird durch ihren Wert ersetzt. Damit die Anführungszeichen innerhalb des Textes angezeigt werden, wird ihnen ein Backslash vorangestellt.

```
F:\PerlProgramme\Kapitel_03>perl DatenAusgabe.pl
Thu Sep 3 07:54:05 2015
Der Name lautet "Herb Kater".
Der Name lautet "Herb Kater".
Der Name lautet \"$name\".
F:\PerlProgramme\Kapitel_03>
```

Datenausgabe mit der `print`-Anweisung

- ⑤ Bei der Verwendung von Hochkommata werden Variablen und Escape-Zeichen nicht durch ihren Wert ersetzt. Die Ausgabe erfolgt daher in Teilen, die durch Kommata getrennt werden.
- ⑥ In dieser Zeile erfolgt keine korrekte Ausgabe der Variablen und des Escape-Zeichens.

Escape-Zeichen verwenden

Mit Escape-Zeichen können Sie bei der Ausgabe von Zeichenketten bestimmte Funktionen ausführen, z. B. einen Zeilenvorschub erzeugen. Eine Escape-Zeichenfolge wird stets mit einem Backslash \ eingeleitet.

Escape-Zeichen können nur in Zeichenketten verwendet werden, die in Anführungszeichen eingeschlossen sind.

Escape-Zeichen	Erklärung	Beispiel	Ergebnis
\a	Ausgabe eines kurzen Signaltons	print "\a";	
\b	Zurücksetzen um ein Zeichen	print "Hallo\bWelt";	HallWelt
\f	Seitenvorschub	print "Hallo Welt\f";	
\l	Nächsten Buchstaben als Kleinbuchstaben ausgeben	print "Hallo \lWelt";	Hallo welt
\n	Zeilenvorschub	print "Hallo\nWelt";	Hallo Welt
\r	Wagenrücklauf, vorhandener Text wird überschrieben	print "Hallo\rWelt";	Welto
\t	Tabulator setzen	print "Hallo\tWelt";	Hallo Welt
\u	Nächsten Buchstaben als Großbuchstaben ausgeben	print "H\uallo Welt";	HALlo Welt
\\\	Backslash ausgeben	print "Hallo Welt\\\";	Hallo Welt\\"
\\"	Doppelte Anführungszeichen ausgeben	print "Hallo \"Welt\"";	Hallo "Welt"
\L	Alle Buchstaben werden bis zum Zeichen /E klein ausgegeben.	print "\LHALLO\E Welt";	hallo Welt
\U	Alle Buchstaben werden bis zum Zeichen /E groß ausgegeben.	print "\Uhallo\E Welt";	HALLO Welt

Lange Texte ausgeben

Besonders beim Einsatz von Perl für CGI-Programme kann es notwendig sein, längere Texte auszugeben, die keine Anweisungen enthalten. In diesem Fall können Sie die so genannte **heredoc-Syntax** der print-Anweisung verwenden.

Die angegeben Daten werden so ausgegeben, als wären Sie in Anführungszeichen eingeschlossen. Daher können Sie auch Variablen und Escape-Zeichen einschließen.

Syntax der `print`-Anweisung für längere Texte

```
print <<"Endekennzeichen";
...
Text
...
Endekennzeichen
```

- ✓ Nach dem Schlüsselwort `print` wird mit den Zeichen `<<` ein heredoc-Block eingeleitet. Danach folgt das gewünschte Endekennzeichen. Dies kann ein beliebiger Text sein, der optional in Anführungszeichen `"` eingeschlossen wird.
- ✓ Nach der mit einem Semikolon abgeschlossenen Anweisung wird der gewünschte Text angegeben. Er kann Variablen und Escape-Zeichen enthalten.
- ✓ Die Ausgabe stoppt, wenn das Endekennzeichen gefunden wird. Es wird nicht mit ausgegeben.



Das Endekennzeichen muss am Beginn der Zeile und als einzige Anweisung in der Zeile stehen. Es dürfen keine Leerzeichen oder Tabulatoren davor oder dahinter gesetzt werden.

Beispiel: `Heredoc.pl`

Das Beispiel gibt einen längeren Text mit der heredoc-Syntax aus. Zusätzlich wird die aktuelle Perl-Version ermittelt und ebenfalls angezeigt.

```
#!/usr/bin/perl

① $version=$] ;

② print << "[ENDE]" ;
<h3>Heredoc-Syntax</h3>
<p>Die heredoc-Syntax ist besonders beim Einsatz von Perl
als CGI-Sprache interessant, da so auch Teile einer Webseite
mit HTML-Befehlen ohne umständliche print-Befehle ausgegeben
werden können.</p>
<p><i>Verwendete Perl-Version: $version</i></p>
[ENDE]
```

- | | |
|---|---|
| <p>① Die von Perl vordefinierte Variable <code>\$]</code> liefert die Version des verwendeten Perl-Interpreters.</p> <p>② Diese Zeile leitet den heredoc-Bereich ein. Alle folgenden Zeilen im Quelltext werden bis zum Endekennzeichen <code>[ENDE]</code> als Text interpretiert und ausgegeben.</p> <p>③ Das Endekennzeichen muss als einzige Anweisung in der Zeile stehen.</p> | <pre>F:\PerlProgramme\Kapitel_03>F:\Strawberry\perl\bin\perl Heredoc.pl Heredoc-Syntax Die heredoc-Syntax ist besonders beim Einsatz von Perl als CGI-Sprache interessant, da so auch Teile einer Webseite mit HTML-Befehlen ohne umständliche print-Befehle ausgegeben werden können. Verwendete Perl-Version: 5.020002 F:\PerlProgramme\Kapitel_03>_</pre> <p style="text-align: center;"><i>Ausgabe eines längeren Textes in Perl</i></p> |
|---|---|

3.6 Schnellübersicht

Sie möchten ...	
Text ausgeben	<code>print "Text";</code>
den Wert einer Variablen ausgeben	<code>print \$variable;</code>
Text mit eingebetteten Variablen ausgeben	<code>print "Text \$variable";</code>
bei der Ausgabe eine neue Zeile beginnen	<code>print "\n";</code>
längerer Text ausgeben	<code>print << [ENDE] ; Text ... [ENDE]</code>

3.7 Übung

Erste Schritte in der Perl-Programmierung

Übungsdatei: --

Ergebnisdatei: **Übung1-E.pl**

1. Welchen Unterschied gibt es bei der Verwendung von Anführungszeichen und Hochkomma bei der Datenausgabe?
2. Womit wird in Perl ein Kommentar eingeleitet?
3. Was ist die She-Bang, und wozu wird sie benötigt?
4. Informieren Sie sich mit der Option `-h` über die verschiedenen Optionen des Perl-Interpreters und ihre Verwendung.
5. Welche Regeln müssen Sie bei der Vergabe von Bezeichnern beachten?
6. Erstellen Sie ein neues Perl-Programm. Weisen Sie einer Variablen das aktuelle Datum als Text zu.
7. Geben Sie das Datum mithilfe der Variablen einzeln und in Verbindung mit dem Text *Heute ist der ...* aus.
8. Wie können Sie längere Texte in Perl ausgeben? Was müssen Sie dabei beachten?
9. Nennen Sie Gründe für die Verwendung von Kommentaren. Erstellen Sie Kommentare für Ihr Programm.

4 Einfache Sprachelemente

In diesem Kapitel erfahren Sie

- ✓ wie Sie Variablen verwenden und ihnen Werten zuweisen
- ✓ welche Datentypen Perl unterstützt
- ✓ welche Operatoren Sie verwenden können
- ✓ welche Rangfolge die Operatoren besitzen

Voraussetzungen

- ✓ Erstellen und Starten von Perl-Programmen

4.1 Variablen

Variablen dienen zum Speichern von Werten bzw. Daten in einem Programm, damit sie bei einer späteren Verwendung im Programmablauf wieder zur Verfügung stehen. Um auf die gespeicherten Werte zugreifen zu können, erhalten Variablen einen eindeutigen Bezeichner bzw. Namen. Dabei erhalten Sie, je nach Variablentyp, eines der Zeichen `$`, `@` oder `%` als Kennzeichen vorangestellt, z. B. `$name`, `@werte`.

Der Inhalt von Variablen kann im Programmablauf über eine Wertzuweisung beliebig oft geändert werden.

Typen von Variablen

Perl unterscheidet drei verschiedene Arten von Variablen, die für unterschiedliche Zwecke eingesetzt werden können.

Skalare Variablen	Skalare Variablen können Zeichenketten und Zahlen speichern. Jede Variable speichert dabei genau einen Wert. Dies ist der am meisten genutzte Variablentyp. Gekennzeichnet werden skalare Variablen mit einem Dollarzeichen <code>\$</code> . Beispiel: <code>\$name</code> , <code>\$anzahl</code> , <code>\$datum</code>
Listen, Arrays	Arrays können Zeichenketten und Zahlen als eindimensionale Felder (Liste) oder mehrdimensionale Felder (Array) speichern. Die einzelnen Werte des Feldes werden über Indizes angesprochen. Arrays und Listen werden mit einem AT-Zeichen <code>@</code> gekennzeichnet. Beispiel: <code>@namen</code> , <code>@werte_liste</code>
Assoziative Arrays (Hashes)	Ähnlich den Feldern können in Hashes mehrere Zeichenketten und Zahlen gespeichert werden. Der Zugriff erfolgt hier jedoch über zugeordnete Zeichenketten als Schlüssel . Hashes werden mit einem Prozentzeichen <code>%</code> gekennzeichnet. Beispiel: <code>%wortzahl</code> , <code>%tage_pro_monat</code>

Beispiel: Variablen.pl

Im Beispiel werden einigen Variablen Werte zugewiesen. Danach erfolgt eine Ausgabe der Werte.

```
#!/usr/bin/perl

① $datum=localtime();
② $datum="Das aktuelle Datum ist: $datum";
③ $preis=15.99;
$anzahl=10;
④ $gesamtpreis=$preis*$anzahl;

print "\n";
print "$datum\n";
⑤ print "Der Gesamtpreis beträgt $anzahl * $preis = $gesamtpreis\n";
⑥ print 'Die Variable $preis enthält den Wert ', $preis, "\n";

⑦ $anzahl="10 Stück";
print $anzahl, "\n";
```

- ① Mit der Perl-Funktion `localtime` wird das aktuelle Datum ermittelt und der Variablen `$datum` zugewiesen.
- ② Hier erhält `$datum` einen neuen Wert. Wie bei der Datenausgabe mit `print` werden auch bei der Wertzuweisung alle Variablen in der Zeichenkette durch den jeweiligen Wert ersetzt. Beim Auswerten der Zeichenkette enthält `$datum` noch den alten Wert.
- ③ Den Variablen `$preis` und `$anzahl` wird eine Zahl zugewiesen.
- ④ Mit Variablen können Sie auch Berechnungen durchführen. Hier wird der Gesamtpreis berechnet und einer neuen Variablen zugewiesen.
- ⑤ Da die Zeichenkette in Anführungszeichen eingeschlossen ist, werden alle Variablen durch ihren Wert ersetzt.
- ⑥ Soll der Name einer Variablen ausgegeben werden, müssen Hochkommata verwendet werden.
- ⑦ Die Variable `$anzahl` wurde zuerst zum Speichern einer Zahl verwendet. Nun wird der Inhalt mit einer Zeichenkette überschrieben und wieder ausgegeben.

```
F:\PerlProgramme\Kapitel_04>perl Variablen.pl
Das aktuelle Datum ist: Thu Sep 3 10:16:32 2015
Der Gesamtpreis beträgt 10 * 15.99 = 159.9
Die Variable $preis enthält den Wert 15.99
10 Stück
F:\PerlProgramme\Kapitel_04>
```

Das Beispielprogramm zeigt auch die Ausgabe von Variablen

Die Ausgabe von deutschen Umlauten und Sonderzeichen kann in der Konsole Probleme bereiten. Sollte das der Fall sein, können Sie mit der Zeile `binmode (STDOUT , ":encoding(cp437)");` vor der ersten Ausgabe in die Konsole dafür sorgen, dass das richtige Encoding verwendet wird und die deutschen Umlaute und Sonderzeichen korrekt dargestellt werden.



Wertzuweisung bei skalaren Variablen

Mithilfe des Operators `=` werden Variablen Werte zugewiesen. Der Zuweisungsoperator weist den Wert auf seiner rechten Seite der Variablen auf der linken Seite zu.

Falls die Variable bereits einen Wert enthielt, wird dieser überschrieben. Sie können einer Variablen einen konstanten Wert, das Ergebnis einer Funktion oder einer Operation, aber auch den Inhalt einer anderen Variablen zuweisen.

Beispiel

```
$datum=localtime();
$preis=15.99;
$anzahl=10;
$anzahl="10 Stück";
```

 In Perl können ohne entsprechend Gegenmaßnahmen Variablen auch Werte mit **unterschiedlichen Datentypen** zugewiesen werden. Die Typkonvertierung wird dabei automatisch durchgeführt.

Wollen Sie mehreren Variablen den gleichen Wert zuweisen, können Sie eine Kurzform der Wertzuweisung verwenden.

Beispiel

Die drei Variablen \$datum_alt, \$datum_neu und \$datum_heute sollen den gleichen Wert erhalten.

```
$datum_alt = $datum_neu = $datum_heute = "27.02.2016";
```

Variablen deklarieren

Im Gegensatz zu anderen Programmiersprachen müssen Variablen in Perl nicht deklariert werden. Eine Variable wird automatisch angelegt, wenn die erste Wertzuweisung erfolgt.

 Beachten Sie, dass Sie durch den fehlenden Zwang zur Deklaration den Überblick über die verwendeten Variablen verlieren. Ebenso wird ein Programm schwer wartbar und sehr fehleranfällig. Durch die einfache Verwendung werden auch häufig mehr Variablen verwendet, als notwendig wären.

Deklaration mit `use strict` erzwingen

Mit der Anweisung `use strict;` am Beginn eines Programms können Sie in Perl eine Variablen-Deklaration erzwingen. In diesem Fall muss jede Variable vor ihrem ersten Gebrauch mit der Anweisung `my` – beispielsweise `my $variable;` – bekannt gegeben werden. Wenn Sie im Programmablauf eine nicht deklarierte Variable verwenden, erhalten Sie eine Fehlermeldung. Damit können Sie Programmfehler durch falsch verwendete Variablen vermeiden.

Beispiel: *Variablen2.pl*

Im folgenden Programm wird die Deklaration von Variablen erzwungen. Die benötigten Variablen werden am Beginn des Programms deklariert.

```
#!/usr/bin/perl
① use strict;

② my $preis;
   my $anzahl;
   my $gesamtpreis;

   $preis=15.99;
   $anzahl=10;
   $gesamtpreis=$preis*$anzahl;

   print "Der Gesamtpreis beträgt $anzahl * $preis = $gesamtpreis\n";
```

- ① Die Angabe `use strict;` steht am Beginn des Programms.
- ② Die drei im Programm verwendeten Variablen werden mit der `my`-Anweisung deklariert. Danach können sie wie üblich verwendet werden.

Um Perl-Programme wartbar und stabil zu entwickeln, sollten Sie immer mit der Anweisung `use strict;` arbeiten. Das macht die Erstellung von Perl-Programmen zwar am Anfang etwas aufwändiger, aber Sie vermeiden in der Folge sehr viele Probleme, deren Beseitigung meist ungleich mehr Aufwand bedeutet.



Beispiel: *Variablen2_Fehler.pl*

Im folgenden Programm wird erneut die Deklaration von Variablen erzwungen, aber die benötigten Variablen werden am Beginn des Programms nicht deklariert. Auf die Deklaration der Variablen mit `my` wird verzichtet, um die Wirkung der Anweisung `use strict` zu verdeutlichen.

```
#!/usr/bin/perl
① use strict;

② $preis=15.99;
$anzahl=10;
$gesamtpreis=$preis*$anzahl;

print "Der Gesamtpreis beträgt $anzahl * $preis = $gesamtpreis\n";
```

- ① Die Angabe `use strict;` steht wieder am Beginn des Programms.
- ② Die Variablen `$preis`, `$anzahl` und `$gesamtpreis` werden verwendet, wurden aber zuvor nicht deklariert.

Wenn Sie das Programm ausführen, erhalten Sie eine Reihe an Fehlermeldungen und das Programm wird abgebrochen.

```
F:\PerlProgramme\Kapitel_04>perl Variablen2_Fehler.pl
Global symbol "$preis" requires explicit package name at Variablen2_Fehler.pl line 4.
Global symbol "$anzahl" requires explicit package name at Variablen2_Fehler.pl line 5.
Global symbol "$gesamtpreis" requires explicit package name at Variablen2_Fehler.pl line 6.
Global symbol "$preis" requires explicit package name at Variablen2_Fehler.pl line 6.
Global symbol "$anzahl" requires explicit package name at Variablen2_Fehler.pl line 6.
Global symbol "$anzahl" requires explicit package name at Variablen2_Fehler.pl line 8.
Global symbol "$preis" requires explicit package name at Variablen2_Fehler.pl line 8.
Global symbol "$gesamtpreis" requires explicit package name at Variablen2_Fehler.pl line 8.
Execution of Variablen2_Fehler.pl aborted due to compilation errors.

F:\PerlProgramme\Kapitel_04>
```

Programmabbruch wegen nicht deklarerter Variablen

Beachten Sie, dass diese Fehlermeldungen an sich kein Problem sind, sondern eine gewünschte Situation bzw. Unterstützung durch den Perl-Interpreter darstellen.

Undefinierte Variablen

Variablen enthalten vor ihrer ersten Verwendung noch keinen Wert und sind undefiniert. Mit der Anweisung `defined` können Sie mit einer Kontrollstruktur prüfen, ob eine Variable definiert ist. Die Funktion liefert in diesem Fall den Wert `true` (wahr).

Wollen Sie eine Variable löschen bzw. auf undefined setzen, dann können Sie die `undefined`-Anweisung verwenden.

Beispiel

Der Variablen \$preis wird zuerst ein Wert zugewiesen. Mit der `undef`-Anweisung wird sie danach auf undefined gesetzt.

```
$preis=15.99;  
undef $preis;
```



Wird der Perl-Interpreter im Warning-Modus aufgerufen (Option -w), wird eine Meldung ausgegeben, wenn Sie eine undefinierte Variable ausgeben oder in einer Berechnung oder Funktion verwenden. Auf diese Weise können Sie Programmfehler finden.

Geltungsbereich von Variablen

Alle Variablen, die Sie in einem Perl-Programm verwenden, sind **standardmäßig global**, auch wenn sie innerhalb von Funktionen oder Anweisungsblöcken initialisiert werden. Dies ist ein wesentlicher Unterschied zu mehreren anderen verwandten Sprachen. Allerdings können Sie mit dem Einsatz von drei Schlüsselworten den Geltungsbereich bzw. Gültigkeitsbereich beeinflussen.

Dabei muss zwischen dem Bereich, in dem der Variablenname gültig ist, und dem Bereich, in dem der Variableninhalt verfügbar ist, unterschieden werden. Diese beiden Bereiche sind für Variablen nur identisch, wenn keine weitere Referenz auf den Wert der Variable angelegt wurde.

Perl kennt für die Verwaltung seiner Variablen zwei verschiedene Bereiche:

- ✓ **statisch gebundene** Variablen bzw. **lexikalische** Variablen:
Es gibt die mit dem Schlüsselwort `my` deklarierten **lexikalischen** Variablen. Ein anderer Bezeichner ist **statisch gebundene** Variablen, da der Gültigkeitsbereich nach der Deklaration nicht mehr geändert werden kann. Ein solcher Bereich kommt in Perl in verschiedenen Formen vor. Es kann ein Codeblock, eine Funktion, ein `eval()`-Aufruf oder eine Datei sein. Ein direkter Zugriff von außerhalb des Bereichs auf diese Variablen ist nicht möglich.
- ✓ **dynamisch gebundene** Variablen:
Das Gegenstück dazu sind die **dynamisch gebundenen** Variablen. Diese werden innerhalb von sogenannten Namensräumen verwaltet. Die Lebensdauer wird von Perl nicht beschränkt, so dass man hier auch von **globalen** Variablen spricht. Die unbeschränkte Lebensdauer macht den Umgang mit den Gültigkeitsbereichen in Perl etwas schwierig, weil das nicht exakt dem gleichen Verständnis von globalen Variablen in vielen anderen Programmiersprachen entspricht.



Bei lexikalischen Variablen wird ein Zugriff schneller erfolgen als bei dynamisch gebundenen Variablen. Wenn es keinen ausdrücklichen Grund für eine dynamische Bindung gibt, sollten Sie statisch gebundene Variablen verwenden.



Mit Perl 5.6 wurde das Schlüsselwort `our` eingeführt, das einen **lexikalischen Alias** für eine Variablen im sogenannten Paket (package) anlegt. Paket ist im Allgemeinen der globale Gültigkeitsbereich. Sie können damit `our` im Wesentlichen wie `my` verwenden und globale oder blockspezifische lexikalische Variablen anlegen. Allerdings legt `our` in dem Fall **keine** neuen Variablen an (wie `my`), sondern eben nur einen Alias für eine bereits existierende Variable. Der Sinn ist, dass man bei der Verwendung von `use strict` eine package-Variable ohne explizite Angabe des package-Namens verwenden kann – aber nur innerhalb des lexikalischen Gültigkeitsbereichs der `our`-Deklaration.

Lokale Variablen

Mit den Anweisungen `my` und `local` können Sie in Perl auch lokale Variablen deklarieren (und im Prinzip auch mit `our` einen lokalen Alias anlegen). Allerdings gibt es einen gravierenden Unterschied zwischen `my` und `local`:

- ✓ Mit `my` wird eine neue Variable **angelegt**.
- ✓ Mit `local` wird der Wert einer **bereits vorhandenen** Variablen temporär **verdeckt**.

<code>local \$variable;</code>	Für die globalen Variablen bietet Perl mit dem Schlüsselwort <code>local</code> die Möglichkeit den Wert einer vorhandenen Variable auf einen bestimmten Bereich des Programms begrenzt zu verändern. Die Änderung geht beim Verlassen des Bereichs verloren. Das ist der gewünschte Effekt, denn so sind globale Variablen effektiver einsetzbar, da sichergestellt ist, dass der ursprüngliche Wert für alle anderen Teile des Programms, wieder hergestellt wird. Beachten Sie - die Variable ist weiter global und im aktuellen Anweisungsblock und in allen aufgerufenen Funktionen und Prozeduren gültig – nur der Wert wird lokal verdeckt. Beachten Sie, dass die Verdeckung des Werts der globalen Variablen auch in tieferen Ebenen erfolgt. Das bedeutet, dass etwa die Verdeckung in einem Unterprogramm sich auf ein weiteres Unterprogramm auswirkt, das von dieser Stelle aus aufgerufen wird.
<code>my \$variable;</code>	Die Variable ist nur im aktuellen Anweisungsblock gültig. Wird die Deklaration außerhalb von einer Funktion oder Block vorgenommen, ist die Variable global, ansonsten lokal.

Als Anweisungsblock gelten alle von geschweiften Klammern `{ }` eingeschlossenen Anweisungen, z. B. der Programmteil eines Unterprogramms.



Beispiel: *Variablen3.pl*

Das nachfolgende Perl-Programm verwendet ein Unterprogramm zur Berechnung des Gesamtpreises. Dazu wird die Variable `$gesamtpreis_global` außerhalb des Unterprogramms und die Variable `$gesamtpreis_lokal` als lokale Variable innerhalb des Unterprogramms definiert.

<code>#!/usr/bin/perl</code>	
<code>use strict;</code>	
① <code>my \$preis=15.99;</code>	
<code>my \$anzahl=10;</code>	
<code>my \$gesamtpreis_global;</code>	
② <code>&Gesamtpreis;</code>	
③ <code>print "Globale Variable \\$gesamtpreis_global=\$gesamtpreis_global\n";</code>	
<code>sub Gesamtpreis {</code>	
④ <code>my \$gesamtpreis_lokal=\$preis*\$anzahl;</code>	
<code>print "Lokale Variable \\$gesamtpreis_lokal=\$gesamtpreis_lokal\n";</code>	
<code>\$gesamtpreis_global=\$gesamtpreis_lokal;</code>	
<code>}</code>	

- ① Alle Variablen, die mit der Anweisung `my` im Hauptteil des Programms definiert werden, sind auch in allen Unterprogrammen verfügbar.
- ② Hier erfolgt der Aufruf des Unterprogramms `Gesamtpreis`.

- ③ Nach dem Abarbeiten des Unterprogramms enthält die Variable `$gesamtpreis_global` den berechneten Wert. Da die Variable `$gesamtpreis_lokal` erst im Unterprogramm definiert wurde, ist sie an dieser Stelle nicht verfügbar.
- ④ Innerhalb des Unterprogramms wird die Variable `$gesamtpreis_lokal` definiert und berechnet sowie ausgegeben.

Beispiel: *Variablen3_Fehler.pl*

Dieses Beispiel enthält eine kleine Abwandlung gegenüber dem vorherigen Beispiel, bei dem aus dem globalen Gültigkeitsbereich auf die lokale Variable zugegriffen werden soll. Durch diesen unerlaubten Zugriff wird das Programm mit einem Fehler abgebrochen.

```
...  
&Gesamtpreis;  
print "Globale Variable \$gesamtpreis_global=$gesamtpreis_global\n";  
  
① print "Lokale Variable \$gesamtpreis_lokal=$gesamtpreis_lokal\n";  
  
..."
```

- ① Hier wird, im Unterschied zum Beispiel *Variablen3.pl* versucht, auf eine lokal in dem Unterprogramm nur gültige Variable zuzugreifen. Da im Beispiel die Anweisung `use strict;` verwendet wurde, wird das Programm mit einer Fehlermeldung beendet. Der Grund ist, dass der Zugriff auf nicht definierte Variablen, in diesem Fall auf die lokale Variable `$gesamtpreis_lokal`, unterbunden wird.

Wenn Sie die Anweisung `use strict;` weglassen, wird das Programm nicht beendet und der Wert der nicht deklarierten Variable wird als Nullwert angenommen. Das kann im weiteren Verlauf zu logischen Problemen führen.

Beispiel: *Variablen4.pl*

Das Zusammenspiel bzw. die Unterschiede zwischen lokalen und globalen Variablen und Bereichen sowie die Unterschiede zwischen Verdecken und Neu anlegen von Variablen soll noch ein weiteres Perl-Programm verdeutlichen. Das Beispiel verwendet bewusst „unsauberen“ Programmierstil, um das Verhalten von Perl in solchen „Grenzsituationen“ zeigen zu können.

```
#!/usr/bin/perl  
use strict;  
  
① my $a = 1;  
$b = 2;  
our $c = 3;  
  
print "Hauptebene:\t\t\$a = $a,\t\$b = $b,\t\t\$c = $c\n";  
② sub sub1 {  
    $a = -1;  
    local $b = -2;  
    my $c = -3;  
    print "Unterprogramm 1:\t\$a = $a,\t\$b = $b,\t\$c = $c\n";  
    sub2();  
}
```

```
(4) sub sub2 {
    print "Unterprogramm 2:\t\$a = $a, \$b = $b, \$c = $c\n";
}
print "Hauptebene:\t\$a = $a, \$b = $b, \$c = $c\n";
```

- ① Es werden drei Variablen angelegt, die in allen Unterprogrammen verfügbar sind. Dabei wurde für die Variable `$b` explizit auf die Anweisung `my` verzichtet. Auch wenn mit `use strict;` eine explizite Deklaration von Variablen gefordert wird, erzwingt das nicht in jedem Fall die Verwendung von `my`. Allerdings müssen Sie für Variablen, die einfach mit einer Wertzuweisung im Hauptteil des Programms definiert werden, „irgendwo“ eine explizite Deklaration vornehmen. Im Beispiel erfolgt die Deklaration innerhalb des Unterprogramms `sub1` mithilfe des Schlüsselwortes `local`. Damit sind global einfach per Wertzuweisung eingeführte Variablen in allen Unterprogrammen global verfügbar. Die Variable `$c` wurde hingegen global mit `our` global deklariert. Auch das ist trotz `use strict;` möglich – sofern in einem Unterprogramm eine „saubere“ Deklaration über `local`, `my` oder auch `our` selbst erfolgt. Deshalb ist auch das kein Anlegen eines Alias, sondern die Deklaration einer neuen globalen Variablen. Die nachfolgende `print`-Anweisung gibt die anfänglichen Werte der Variablen (Initialisierungswerte) aus. Sie erhalten die Werte 1, 2 und 3 in der Ausgabe.
- ② Hier erfolgt der Aufruf des Unterprogramms `sub1`.
- ③ Im Unterprogramm `sub1` erhält die Variable `$a` einen neuen Wert -1. Da die Variable global ist, wird der Wert damit auch in der Hauptebene geändert. Die Variable `$b` wird hingegen mit `local` nur lokal verdeckt. Global wird sie nicht geändert, aber diese Anweisung erlaubt es, dass Sie bei der globalen Deklaration trotz der `use strict`-Anweisung auf das Voranstellen von `my` verzichten können. Beachten Sie, dass dies kein sauberer Programmierstil ist und nur das Verhalten von Perl zeigen soll. Auch die Variable `$c` bekommt lokal einen neuen Wert. Aber mit `my` wird temporär eine **neue** Variable angelegt, die nur innerhalb des Gültigkeitsbereichs des Unterprogramms existiert. Aber auch in dem Fall wird die Variable in der Hauptebene nicht geändert und auch diese hätte bei einer saubereren Programmierung explizit mit `my` statt `our` deklariert werden müssen. Die Ausgabe in dem Unterprogramm `sub1` zeigt die neuen Werte -1, -2 und -3. Danach wird ein weiteres Unterprogramm `sub2` aufgerufen.
- ④ Innerhalb des Unterprogramms `sub2` wird auf die Variable `$a`, `$b` und `$c` zugegriffen, um deren Werte auszugeben. Die Variable `$a` war global und wurde in dem Unterprogramm `sub1` geändert. Also erhalten Sie den geänderten Wert -1. Der Wert von `$b` wurde in dem Unterprogramm `sub1` lokal verdeckt und da das Unterprogramm `sub2` von `sub1` aus aufgerufen wurde, ist der Wert von `$b` immer noch -2 (der verdeckende Wert aus `sub1`). Der Wert von `$c` ist jedoch 3, denn in `sub1` wurde `$c` mit `my` nur in dem lokalen Gültigkeitsbereich von `sub1` definiert. Diese lokale Variable ist in dem neuen Unterprogramm nicht mehr definiert und in `sub2` greifen Sie auf die globale Variable zu. Die abschließende Ausgabe zeigt, wie sich die globalen Variablen am Ende des Programms geändert haben. Nur der Wert von `$a` wurde global geändert.

```
F:\PerlProgramme\Kapitel_04>perl Variablen4.pl
Hauptebene:      $a = 1,      $b = 2,      $c = 3
Unterprogramm 1:  $a = -1,     $b = -2,     $c = -3
Unterprogramm 2:  $a = -1,     $b = -2,     $c = 3
Hauptebene:      $a = -1,     $b = 2,      $c = 3

F:\PerlProgramme\Kapitel_04>
```

Die Werte der Variablen in den verschiedenen Gültigkeitsbereichen

4.2 Datentypen

Perl ist eine so genannte typisierte Sprache. Das bedeutet, dass es mehrere Datentypen gibt, die eine Variable annehmen kann. Der Datentyp gibt an, welche Werte eine Variable aufnehmen kann und welche Operationen mit der Variablen ausgeführt werden können. Beispielsweise können Variablen vom Typ „Zahl“ miteinander multipliziert werden. Bei Zeichenketten würde dies keinen Sinn ergeben.

In Perl wird grundsätzlich zwischen Zahlen bzw. numerischen Daten und Zeichenketten unterschieden. Dabei kann Perl jedoch, wenn dies erforderlich ist, automatisch und transparent Zeichenketten in numerische Daten umwandeln oder umgekehrt.

Zahlen

Numerische Datentypen (engl. Numbers) dienen zum Speichern von Zahlen. Sie werden dabei in Ganzzahl- und Fließkomma-Datentypen unterteilt. Variablen mit diesem Datentyp können für Berechnungen, Aufzählungen oder Nummerierungen eingesetzt werden. Dabei kann Perl neben den üblichen Dezimalzahlen auch mit Hexadezimal- oder Oktalzahlen arbeiten.

 Im Gegensatz zu vielen anderen Programmiersprachen sind in Perl ganze Zahlen und Fließkommazahlen austauschbar. Bei der Angabe von Werten können Sie zwar ganze Zahlen angeben, intern arbeitet Perl jedoch stets mit Fließkommazahlen.

Beispiel	Erklärung
5 oder 1050	Ganze Zahl (wird intern als Fließkommazahl interpretiert)
23.5 oder .045	Fließkommazahlen
-17 oder -12.5	Negative ganze bzw. Fließkommazahlen
3e20 oder -2.5e4	Zahlen in Exponentialschreibweise
5e-8 oder -4e-12	Zahlen in Exponentialschreibweise mit negativer Zehnerpotenz
0x55FE	Hexadezimalzahlen
0512	Oktalzahlen

Haxedezimal- und Oktalzahlen

Hexadezimalzahlen haben die Basis 16 und werden in Perl durch ein vorangestelltes `0x` gekennzeichnet. Oktalzahlen mit der Basis 8 müssen mit der Zahl 0 beginnen. Beide Zahlenformate erleichtern den Zugriff auf den Speicher eines Computers und sind nur bei besonderen Programmieraufgaben notwendig.

Bei der Eingabe von Werten über die Tastatur geht Perl stets von **Dezimalzahlen** aus. Eine Erkennung von Hexadezimal- und Oktalzahlen können Sie jedoch mit den Funktionen `hex()` bzw. `oct()` erzwingen.

Interne Verwendung von ganzen Zahlen erzwingen

Bei der standardmäßigen Verwendung von Fließkommazahlen können bei verschiedenen Berechnungen Rundungsfehler auftreten. Mit der Anweisung `use integer;` am Beginn Ihres Programms können Sie die interne Verwendung von ganzen Zahlen (engl. Integers) bei Berechnungen erzwingen (Ganzzahlarithmetik). Mit der Anweisung `no integer;` wird wieder in den normalen Modus gewechselt (Gleitkommaarithmetik).

Das Beispiel *Variablen5.pl* zeigt die Unterschiede zwischen Ganzzahl- und Gleitkommaarithmetik.

```
#!/usr/bin/perl
use strict;
① use integer;
my $a = 5;
my $b = 2;
② print $a / $b . "\n";
③ no integer;
④ print $a / $b;
```

- ① Die Festlegung, dass in der Folge Ganzzahlarithmetik betrieben werden sollen.
- ② Die Ausgabe der Division von 5 geteilt durch 2. Das müsste rein mathematisch 2,5 ergeben, aber die Festlegung auf Ganzzahlarithmetik bewirkt, dass sich der Wert 2 ergibt.
- ③ Die Ganzzahlarithmetik wird für die folgenden Anweisungen abgeschaltet.
- ④ Die Ausgabe der Division von 5 geteilt durch 2 ergibt 2,5. Beachten Sie, dass das in Perl als 2.5 dargestellt wird (amerikanische Notation – Floatingpoint).

Mit `use`-Anweisungen können Sie bestimmte Verhaltensweisen im Perl-Programm festlegen und diese auch auf bestimmte Bereiche im Programm begrenzen.



```
F:\PerlProgramme\Kapitel_04>perl Variablen5.pl
2
2.5
F:\PerlProgramme\Kapitel_04>_
```

Ganzzahl- und Gleitkommaarithmetik

Eine Geschwindigkeitssteigerung bei der Verwendung ganzer Zahlen gibt es in Perl nur bei der Verwendung auf älteren Prozessoren ohne Fließkommaeinheit.

Zeichenketten

Eine Zeichenkette (engl. String) ist eine Aneinanderreihung von beliebigen Zeichen des ASCII-Zeichensatzes. Dazu gehören Ziffern, Symbole und Buchstaben. Zeichenketten werden in Anführungszeichen oder Hochkommata eingeschlossen.

In Perl gibt es, im Gegensatz zu anderen Programmiersprachen, keine Beschränkung der Länge von Zeichenketten. Sie könnten im Prinzip den gesamten verfügbaren Arbeitsspeicher ausnutzen.



Wollen Sie Variableninhalte in einer Zeichenkette verwenden, muss diese in Anführungszeichen eingeschlossen werden. Es gelten dabei die gleichen Regeln wie bei der Ausgabe von Zeichenketten mit der `print`-Anweisung.

Die heredoc-Syntax, die Sie bereits von der Ausgabe von Zeichenketten kennen, können Sie auch bei der Wertzuweisung verwenden.



Beispiel

Der Variablen `$text` wird ein mehrzeiliger Text in der heredoc-Syntax zugewiesen. Der Text enthält Variablen, deren Wert ausgegeben wird.

```
$text = << [ENDE] ;
Heute ist der $datum.\n
Sie verwenden gerade die Perl-Version $]\n
Viel Spaß bei der Programmierung.
[ENDE]
```

Datentypkonvertierung

Automatische Typisierung

Perl erleichtert Ihnen die Arbeit mit Variablen, da die Variablen bei einer Wertzuweisung automatisch den richtigen Datentyp erhalten. Der Perl-Interpreter versucht dabei, den übergebenen Wert zu erkennen, in dem er das erste Zeichen des Wertes überprüft. Kann er daran den Datentyp nicht eindeutig ermitteln, vergleicht er, ob bestimmte Zeichen innerhalb des Wertes vorhanden sind.

Die folgende Tabelle zeigt die Regeln, die Perl verwendet, um Datentypen zu erkennen:

Zahlen	Das erste Zeichen ist eine Ziffer, und der Wert enthält außer den Zeichen <code>-</code> , <code>.</code> oder dem Dezimalpunkt <code>.</code> keine anderen Buchstaben oder Sonderzeichen. Beginnt der Wert mit den Zeichen <code>0x</code> , dann sind auch die Buchstaben <code>A</code> bis <code>F</code> erlaubt, die in Hexadezimalzahlen auftreten.
Zeichenkette	Ist der Wert in Anführungszeichen bzw. Hochkommata eingeschlossen oder enthält der Wert Buchstaben bzw. Sonderzeichen, wird er als Zeichenkette interpretiert.

Typkonvertierung

Beim Verwenden von Variablen führt Perl eine automatische transparente Typkonvertierung durch, wenn dies nötig ist. Wenn Sie beispielsweise eine Zeichenkette in einer Berechnung verwenden, versucht Perl von links beginnend alle Ziffern auszulesen, bis ein Buchstabe oder Sonderzeichen auftritt.

Beispiel: Typkonvertierung.pl

Es werden zwei Variablen definiert. Der Variablen `$preis` wird eine Zahl zugewiesen. Der Variablen `$anzahl` dagegen eine Zeichenkette. Bei der Multiplikation führt Perl eine automatische Typkonvertierung durch.

```
#!/usr/bin/perl
use strict;
① my $preis=15.99;
my $anzahl="10 Stück";
② $gesamtpreis=$preis*$anzahl;
③ print "Der Gesamtpreis beträgt $gesamtpreis\n";
```

- ① Die zwei Variablen werden mit unterschiedlichen Datentypen initialisiert.
- ② Bei der Multiplikation erfolgt eine automatische Konvertierung der Variablen `$anzahl` in eine Zahl. Von links beginnend wird dabei nur die 10 beachtet. Der Rest der Zeichenkette fällt weg.
- ③ Zur Überprüfung erfolgt eine Ausgabe des Ergebnisses.



Kann der Interpreter am Anfang der Zeichenkette keine Zahl finden, dann wird mit dem Wert 0 gerechnet. Bei logischen Vergleichen wie `<` (kleiner als) oder `>` (größer als), erfolgt **keine** Umwandlung von Zeichenketten in Zahlen. Sie können daher unerwartete Ergebnisse erhalten.

Beispiel

Wenn Sie Zahlen als Zeichenketten definieren, dann aber numerische Operationen anwenden, werden die Variablen automatisch umgewandelt.

```
my $a="10";
my $b="5";
my $c=$a+$b;
print $c; # Ergebnis ist 15
```

4.3 Operatoren

Operatoren und Ausdrücke

Um die in Variablen gespeicherten Werte zu verarbeiten, werden Operatoren benötigt. Ein Operator erwartet einen oder mehrere Werte als so genannte Operanden und liefert ein Ergebnis zurück. Mit Operatoren ist es möglich, Berechnungen durchzuführen, Zeichenketten zu verknüpfen oder logische Bedingungen zu formulieren. Die Verbindung von Operanden, Operator und Ergebnis wird als **Ausdruck** bezeichnet. Ausdrücke gehören zu den kleinsten ausführbaren Einheiten eines Programms.

Folgende Typen von Operatoren werden unterschieden:

Arithmetische Operatoren	Mit ihnen werden Berechnungen mit numerischen Werten durchgeführt.
Vergleichsoperatoren	Sie dienen zum Vergleichen von Werten und liefern einen Wahrheitswert.
Logische Operatoren	Mehrere Vergleichsausdrücke können mit logischen Operatoren verknüpft werden.
Zuweisungsoperatoren	Dienen zum Zuweisen von Werten zu einer Variablen. Teilweise sind gleichzeitige Berechnungen möglich.
Bitbearbeitungsoperatoren	Bitweise Bearbeitung oder Vergleich von Werten
Zeichenkettenoperatoren	Dienen zum Verknüpfen und Erstellen von Zeichenketten

Arithmetische Operatoren

Mit diesen Operatoren können Sie mathematische Berechnungen durchführen. Falls die übergebenen Werte keine Zahlen sind, werden sie automatisch konvertiert. Das Ergebnis ist stets eine Zahl.

Operator	Funktion	Beispiel (\$a=20, \$b=3)	Ergebnis	Erklärung
+	Addition	\$a + \$b	23	Summe von \$a und \$b
-	Subtraktion	\$a - \$b	17	Differenz zwischen \$a und \$b
*	Multiplikation	\$a * \$b	60	Produkt aus \$a und \$b
/	Division	\$a / \$b	6.6666...	Quotient aus \$a durch \$b
%	Modulo	\$a % \$b	2	Rest der ganzzahligen Division von \$a durch \$b
**	Exponential	\$a ** \$b	8000	Potenzfunktion: \$a hoch \$b

Operator	Funktion	Beispiel (\$a=20, \$b=3)	Ergebnis	Erklärung
++	Präinkrement	<code>++\$a + \$b</code>	24	Erhöhung um 1 vor der Verwendung
	Postinkrement	<code>\$a++ + \$b</code>	23 (\$a=21)	Erhöhung um 1 nach der Verwendung
--	Prädekrement	<code>--\$a + \$b</code>	22	Verringerung um 1 vor der Verwendung
	Postdekrement	<code>\$a-- + \$b</code>	23 (\$a=19)	Verringerung um 1 nach der Verwendung

Die Inkrement- und Dekrementoperatoren sind gleichbedeutend mit dem Ausdruck `$a=$a+1` bzw. `$a=$a-1`.

Vergleichsoperatoren

Vergleichsoperatoren werden zur Entscheidungsfindung verwendet und liefern ein logisches Ergebnis, entweder `true` (wahr) oder `false` (falsch). Solche Vergleichsausdrücke werden beispielsweise bei allen Kontrollstrukturen benötigt.



Perl interpretiert alle numerischen Werte, die größer oder gleich 1 sind, als wahr. Alle Werte kleiner oder gleich 0 sind falsch.



Es gibt unterschiedliche Vergleichsoperatoren für Zahlen und Zeichenketten. Zwar können Sie auch bei Zeichenketten einen numerischen Vergleich durchführen, dann wird jedoch nicht der Text selbst verglichen, sondern nur der ASCII-Wert des ersten Zeichens. Dies kann vor allem dann zu Fehlern führen, wenn der Text Zahlen enthält.

Numerischer Vergleich

Operator	Funktion	Beispiel (\$a=20, \$b=3)	Ergebnis	Erklärung
<code>==</code>	Gleichheit	<code>\$a == \$b</code>	<code>false</code>	wahr, wenn \$a und \$b gleich sind
<code>!=</code>	Ungleichheit	<code>\$a != \$b</code>	<code>true</code>	wahr, wenn \$a und \$b ungleich sind
<code><</code>	Kleiner als	<code>\$a < \$b</code>	<code>false</code>	wahr, wenn \$a kleiner ist als \$b
<code>></code>	Größer als	<code>\$a > \$b</code>	<code>true</code>	wahr, wenn \$a größer ist als \$b
<code><=</code>	Kleiner oder gleich	<code>\$a <= \$b</code>	<code>false</code>	wahr, wenn \$a größer oder gleich \$b ist
<code>>=</code>	Größer oder gleich	<code>\$a >= \$b</code>	<code>true</code>	wahr, wenn \$a größer oder gleich \$b ist
<code><=></code>	Vergleich	<code>\$a <=> \$b</code>	1 0 -1	1, wenn \$a größer ist als \$b 0, wenn \$a gleich \$b ist -1, wenn \$a kleiner ist als \$b

Vergleich von Zeichenketten

In der folgenden Tabelle gilt \$a="Katze" und \$b="Hund".

Operator	Funktion	Beispiel	Ergebnis	Erklärung
eq	Gleichheit	\$a eq \$b	false	wahr, wenn \$a und \$b in allen Zeichen identisch sind
ne	Ungleichheit	\$a ne \$b	true	wahr, wenn \$a und \$b in mindestens einem Zeichen nicht identisch sind
lt	Kleiner als	\$a lt \$b	false	wahr, wenn von links beginnend mindestens ein ASCII-Wert von \$a kleiner ist als der von \$b an gleicher Stelle
gt	Größer als	\$a gt \$b	true	wahr, wenn von links beginnend mindestens ein ASCII-Wert von \$a größer ist als der von \$b an gleicher Stelle
le	Kleiner oder gleich	\$a le \$b	false	wahr, wenn von links beginnend mindestens ein ASCII-Wert von \$a kleiner oder gleich ist als der von \$b an gleicher Stelle
ge	Größer oder gleich	\$a ge \$b	true	wahr, wenn von links beginnend mindestens ein ASCII-Wert von \$a größer oder gleich ist als der von \$b an gleicher Stelle
cmp	Vergleich	\$a cmp \$b	1 0 -1	1, wenn ein ASCII-Wert in \$a größer ist als in \$b 0, wenn alle ASCII-Werte in \$a gleich denen in \$b sind -1, wenn ein ASCII-Wert in \$a kleiner ist als in \$b

Logische Operatoren

Logische Operatoren werden zum Vergleich von Wahrheitswerten oder zum Verknüpfen von Vergleichsausdrücken verwendet.

In der folgenden Tabelle gilt \$a=1 und \$b=0.

Operator	Funktion	Beispiel	Ergebnis	Erklärung
!	Nicht	!\$a	false	Umkehrung des Wahrheitswertes
&& and	Und	\$a && \$b	false	wahr, wenn \$a und \$b wahr sind
 or	Oder	\$a \$b	true	wahr, wenn \$a oder \$b wahr ist

Bitbearbeitungsoperatoren

Bei der Bitbearbeitung werden Zahlen als geordnete Folge von Bits ausgewertet. Bits können nur den Wert 1 oder 0 annehmen. Ihnen liegt das Binärsystem (Basis 2) zugrunde.

In der folgenden Tabelle gilt `$a=10` (binäre Darstellung 1010) und `$b=7` (binäre Darstellung 0111).

Operator	Funktion	Beispiel	Ergebnis	Erklärung
<code>~</code>	Bitweises Nicht	<code>~\$a</code>	5 (0101)	alle Bits werden invertiert
<code> </code>	Bitweises Oder	<code>\$a \$b</code>	15 (1111)	alle Bits werden nach der Oder-Logik verknüpft
<code>&</code>	Bitweises Und	<code>\$a & \$b</code>	2 (0010)	alle Bits werden nach der Und-Logik verknüpft
<code>^</code>	Bitweises exklusives Oder	<code>\$a ^ \$b</code>	13 (1101)	alle Bits werden nach der Xoder-Logik verknüpft
<code><<</code>	Bitweise Linksverschiebung	<code>\$a << 1</code>	20 (10100)	Verschieben der Bits nach links, Auffüllen mit 0
<code>>></code>	Bitweise Rechtsverschiebung	<code>\$a >> 1</code>	5 (0101)	Verschieben der Bits nach rechts, Auffüllen mit 0



Bei der bitweisen Linksverschiebung entspricht jede Verschiebung um ein Bit einer Division durch 2. Die Rechtsverschiebung um ein Bit entspricht einer Multiplikation mit 2.

Zeichenkettenoperatoren

Mit dem Zeichenkettenoperator `□` können Sie mehrere Zeichenketten miteinander verbinden. Als Resultat erhalten Sie eine neue Zeichenkette. Der Operator `☒` dient dem Wiederholen eines Textes.

In der folgenden Tabelle gilt `$a="Katze"` und `$b="Hund"`.

Operator	Funktion	Beispiel	Ergebnis	Erklärung
<code>.</code>	Verketten	<code>\$a . " und " . \$b</code>	"Katze und Hund"	alle beteiligten Zeichenketten werden zu einer einzigen verbunden
<code>x</code>	Wiederholen	<code>\$a x 3</code>	"KatzeKatzeKatze"	Wiederholung der Zeichenkette um einen bestimmten Faktor

Zuweisungsoperatoren

Der wichtigste Zuweisungsoperator ist das Gleichheitszeichen `=`. Es weist den rechts davon stehenden Wert der Variablen auf der linken Seite zu.

Zusätzlich können auch alle arithmetischen Operatoren und die Bitbearbeitungsoperatoren mit dem Gleichheitszeichen verknüpft werden. In diesen Fällen ist die Zuweisung keine Anweisung, sondern ein Ausdruck, der ein Ergebnis liefert. Die gewünschte Operation wird mit den links und rechts stehenden Werten durchgeführt und das Ergebnis der auf der linken Seite stehenden Variablen zugewiesen.

Folgende Zuweisungsoperatoren sind möglich:

```
=, +=, -=, %=, *=, /=, .=, &=, |=, ^=, ~=, <<=, >>=
```

Beispiel

Die Variable \$a soll um den Wert 5 erhöht werden, und die Variable \$b soll mit 10 multipliziert werden.

```
my $a=15;
my $b=8;
$a+=5; # entspricht $a = $a + 5 (20)
$b*=10; # entspricht $b = $b * 10 (80)
```

4.4 Rangfolge der Operatoren

Bevor Sie ein Programm schreiben, sollten Sie die Rangfolge der Operatoren kennen. Viele Fehler lassen sich vermeiden, wenn Sie genau wissen, welche Operationen in welcher Reihenfolge abgearbeitet werden.

Die Operatorenrangfolge bezieht sich auf die Reihenfolge, in der die Operationen in einem Ausdruck abgearbeitet werden:

- ✓ Operationen mit höherem Rang werden vor denen mit niedrigerem Rang ausgeführt.
- ✓ Operatoren mit gleichem Rang werden von links nach rechts abgearbeitet.
- ✓ Die Abarbeitung von Operationen kann durch das Setzen von runden Klammern beeinflusst werden.
Die Ausdrücke in runden Klammern werden stets zuerst und von innen nach außen abgearbeitet.

Die folgende Tabelle listet die Operatoren nach absteigender Rangfolge geordnet auf:

Operator	Erklärung
(...)	Runde Klammern dienen der Gruppierung von Ausdrücken
++ --	Inkrement und Dekrement
**	Exponentialoperator
! ~	Logisches und bitweises Nicht
* / % x	Multiplikation, Division, Modulo, Zeichenkettenwiederholung
+ - .	Addition, Subtraktion, Verkettung
<< >>	Bitweise Verschiebung
< <= >= > lt gt le ge	Ungleichheitsoperatoren
== != <=> eq ne cmp	Gleichheitsoperatoren
&	Bitweises Und
^	Bitweises Oder bzw. bitweises exklusives Oder
&&	Logisches Und
	Logisches Oder
? ... : ...	Fragezeichenoperator: wenn ... dann ... sonst ...
= += -= *= usw.	Zuweisungsoperatoren
,	Komma
not	Logisches Nicht
and	Logisches Und
or	Logisches Oder



Wenn Sie sich bei sehr umfangreichen Ausdrücken über die tatsächliche Reihenfolge der Abarbeitung unschlüssig sind, können Sie beliebig viele runde Klammern setzen und damit eine Rangordnung festlegen. Die Verwendung von runden Klammern zur Gruppierung von Ausdrücken erhöht zusätzlich die Lesbarkeit Ihres Programms.

4.5 Übung

Variablen, Geltungsbereiche und Operatoren

Übungsdatei: --

Ergebnisdatei: *Übung.pl*

1. Welche Typen von Variablen gibt es, und wie unterscheiden Sie sich?
2. Erläutern Sie den Geltungsbereich von Variablen in Perl. Was müssen Sie beachten, und wie können Sie den Geltungsbereich verändern?
3. Erläutern Sie die in Perl zur Verfügung stehenden Datentypen.
4. Erklären Sie die Begriffe „Ausdruck“ und „Operator“.
5. Erläutern Sie die Unterschiede zwischen den Vergleichsoperatoren für Zahlen und Zeichenketten. Was müssen Sie beachten?
6. Bearbeiten Sie den folgenden Ausschnitt aus einem Perl-Programm. Geben Sie jeweils das zu erwartende Ergebnis der Ausdrücke an.

```
$a=220; $b=11; $c=8; $string1="Hallo "; $string2="Welt";
$ergebnis = $b * $c;
$ergebnis = $b - $string2;
$d = $c . $string1;
$ergebnis = $b * $d;
$ergebnis = $a % $c;
$ergebnis = $b & $c;
$ergebnis = $a / --$b;
$ergebnis = $string1 . $string2;
$ergebnis = $string2 x 2;
$ergebnis = ($b <= ($c + 2 + $b));
$ergebnis = ($string1 eq $b);
$b += 9;
```


5 Kontrollstrukturen

In diesem Kapitel erfahren Sie

- ✓ wie Sie Bedingungen erstellen
- ✓ welche Möglichkeiten zur Bedingungsauswahl es gibt
- ✓ wie Sie Anweisungen mit Schleifen wiederholt ausführen
- ✓ mit welchen Anweisungen Sie den Schleifen- oder Programmablauf steuern können

Voraussetzungen

- ✓ Arbeiten mit Variablen, Erstellen von Ausdrücken

5.1 Kontrollstrukturen verwenden

Anweisungen und Kontrollstrukturen

Ein Programm besteht aus einer Folge von Anweisungen. Die Anweisungen beschreiben dabei die Arbeitsschritte, die zur Lösung einer Problemstellung erforderlich sind. In seiner einfachsten Form enthält ein Programm nur Anweisungen, die der Reihe nach ausgeführt werden. Solche Programme haben Sie in den bisherigen Kapiteln kennen gelernt.

Meist muss ein Programm auf eine bestimmte Situation oder Aktion mit einer Entscheidung reagieren und damit von mehreren Anweisungen eine oder gar keine Anweisung durchführen oder bestimmte Anweisungen nicht nur einfach, sondern mehrfach hintereinander mit verschiedenen Werten durchlaufen.

Bedingungen formulieren

Bei der Programmierung von Kontrollstrukturen werden meist Bedingungen benötigt, die bestimmen, mit welcher Anweisung das Programm fortfahren soll.

Eine Bedingung besteht aus zwei oder mehr Werten (z. B. Variablen oder konstante Werte), die mit Vergleichsoperatoren zu einem Ausdruck verknüpft werden. Als Resultat liefert sie das Ergebnis `true` oder `false`. Falls mehrere Teilbedingungen vorliegen, können diese mit logischen Operatoren zur endgültigen Bedingung verbunden werden.

Im Folgenden finden Sie typische Beispiele für Bedingungen, wie sie auch in Kontrollstrukturen vorkommen können:

①	<code>\$anzahl_gesamt != \$anzahl</code>
②	<code>(\$projekt == 500) && (\$name eq "Meier")</code>
③	<code>\$zaehler >= 20</code>

- ① Die Bedingung ist wahr, wenn der Inhalt der Variablen `$anzahl_gesamt` ungleich dem Inhalt der Variablen `$anzahl` ist.
- ② Die Bedingung ist erfüllt, wenn die Variable `$projekt` gleich 500 ist und der Inhalt der Variablen `$name` der Zeichenkette "Meier" entspricht.
- ③ Wenn der Wert der Variablen `$zaehler` größer als 20 oder gleich 20 ist, gilt die Bedingung als erfüllt.

Wenn Sie mehrere Operatoren in einer Bedingung verwenden, bestimmt die Rangfolge der Operatoren, welcher Ausdruck zuerst ausgewertet wird. Um die Reihenfolge, in der Operatoren ausgewertet werden, gezielt zu verändern, können Sie Klammern verwenden. Ausdrücke in Klammern werden immer zuerst ausgewertet.

Als Beispiel soll eine Bedingung formuliert werden, die prüft, ob ein bestimmter Name vorliegt oder die Postleitzahl über einer Untergrenze liegt. Außer dieser Teilbedingung soll die Postleitzahl stets auch unter einer bestimmten Obergrenze liegen. Die Variable \$name soll "Meier" enthalten und die Variable \$plz den Wert 14524.

```
( ($name == "Meier") || (Plz >= 40000) ) && (Plz <= 60000)
```

Zuerst werden die Vergleiche in den inneren Klammern einzeln ausgewertet.

```
( true || false ) && false
```

Als Nächstes wird der geklammerte logische Ausdruck geprüft.

```
true && false
```

Als Letztes werden die beiden übrigen Teilbedingungen mit dem logischen Operatoren && verknüpft.

```
false
```

Ohne Klammersetzung wäre das Ergebnis true und damit falsch!

Damit Sie sicher sind, dass Bedingungsausdrücke in der richtigen Reihenfolge geprüft werden, sollten Sie auch dann Klammern setzen, wenn sie streng genommen auch ausgelassen werden dürfen. Dies erhöht gleichzeitig die Lesbarkeit Ihres Programms.



Anweisungsblöcke bilden

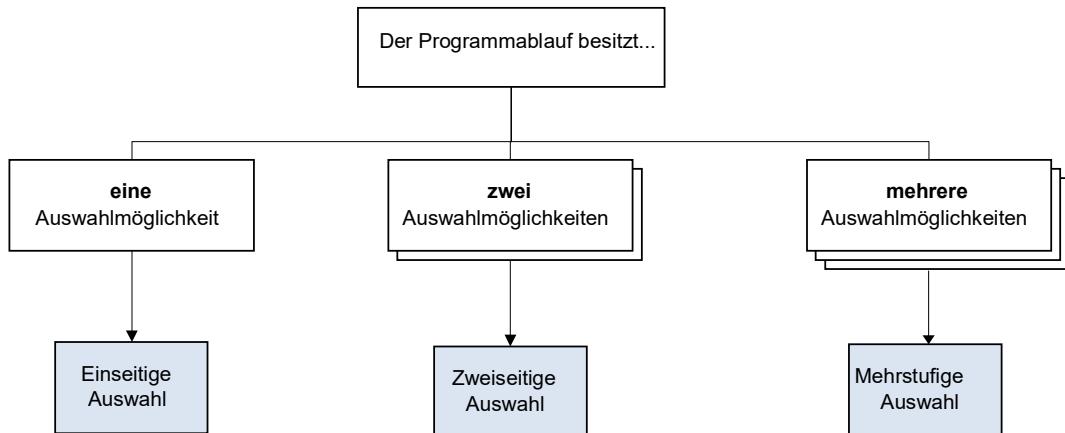
Eine Kontrollstruktur erlaubt die Auswahl oder Wiederholung einer oder mehrerer Anweisungen. Sollen mehrere Anweisungen ausgeführt werden, müssen Sie diese als Anweisungsblock zusammenfassen. Ein Block wird durch eine öffnende geschweifte Klammer eingeleitet `{` und mit einer schließenden geschweiften Klammer `}` beendet.

```
Bedingung {
    Anweisung1;
    Anweisung2;
    ...
}
```

5.2 Anweisungen zur Bedingungsauswahl

Bedingungsauswahl verwenden

Bei vielen Problemstellungen ist die Verarbeitung von Anweisungen von einer oder mehreren Bedingungen abhängig. Existieren verschiedene Alternativen, wird der Programmablauf in mehrere **Zweige** aufgeteilt, von denen nur ein einziger Zweig aufgrund der Bedingung ausgewählt und durchlaufen wird. Die Entscheidung dazu wird mit einer Bedingung gefällt, die nur einen Wert annehmen kann: true oder false.



Überblick über verschiedene Auswahlkonstrukte

Einseitige Auswahl mit der `if`-Anweisung

Die einseitige Auswahl ist dadurch gekennzeichnet, dass aufgrund einer Bedingung eine oder mehrere Anweisungen (Auswahlblock) ausgeführt werden oder nicht. Ist die Bedingung wahr, d. h., liefert sie `true`, dann werden die Anweisungen durchgeführt. Ist das Ergebnis der Bedingung dagegen `false`, werden sie nicht durchgeführt. Diese Auswahl wird in Perl mit der `if`-Anweisung realisiert.

Bedingung = wahr?	
ja	nein
Anweisung1	bleibt leer
Anweisung2	

Einseitige Auswahl

Syntax der `if`-Anweisung

```
if (Bedingung) {
    Anweisung1;
    Anweisung2;
}
```

```
Anweisung if (Bedingung);
```

- ✓ Die `if`-Anweisung wird mit dem Schlüsselwort `if` eingeleitet.
- ✓ Danach folgt die Bedingung, die erfüllt sein muss, damit der Anweisungsblock ausgeführt wird. Sie wird in runde Klammern eingeschlossen.
- ✓ Die Bedingung ist ein logischer Ausdruck, der einen der beiden Zustände `true` oder `false` zurückliefert.
- ✓ Soll nur eine Anweisung ausgeführt werden, so wird diese dem Schlüsselwort `if` vorangestellt. Das abschließende Semikolon folgt in diesem Fall nach der Bedingung.
- ✓ Mehrere Anweisungen werden durch geschweifte Klammern als Block zusammengefasst und nach der Bedingung angegeben.
- ✓ Ist der Bedingungsausdruck erfüllt, wird der Anweisungsblock ausgeführt. Andernfalls ignoriert Perl den Block und setzt die Programmausführung mit der nächsten Anweisung fort.

Beispiel: `if.pl`

Im Beispiel gewährt ein CD-Versender Rabatt, wenn CDs im Gesamtwert von mindestens 50 € bestellt werden. Nach der Eingabe der Anzahl bestellter CDs berechnet das Perl-Programm den Gesamtpreis und entscheidet mit einer `if`-Anweisung, ob ein Rabatt möglich ist.

```

#!/usr/bin/perl
use strict;
my $einzelpreis = 13;

print "Bitte geben Sie die Anzahl der bestellten CDs ein: ";
① my $anzahl=<STDIN>;
chomp($anzahl);

② my $gesamtpreis = $einzelpreis * $anzahl;

③ if ($gesamtpreis >= 50) {
④     print "Der Preis liegt über 50 EUR: Ein Rabatt ist möglich!\n";
}
⑤ print "Rechnungsbetrag bei $anzahl bestellter CDs: $gesamtpreis EUR\n";

```

- ① Die Anzahl der bestellten CDs wird über die Tastatur eingegeben. Das Perl-Programm liest dazu alle Eingaben bis zum Zeilenende. Mögliche Leerzeichen am Anfang oder Ende werden mit der `chomp`-Funktion entfernt.
- ② Der Gesamtpreis wird berechnet und in einer Variablen gespeichert.
- ③ Mit der Bedingung der `if`-Anweisung wird dabei überprüft, ob der Inhalt der Variablen `$gesamtpreis` größer oder gleich dem Wert 50 ist.
- ④ Ist die Bedingung wahr, wird diese Anweisung ausgeführt und eine Information über den zu gewährenden Rabatt angezeigt.
- ⑤ Der Gesamtpreis der bestellten CDs wird ausgegeben. Diese Anweisung wird immer ausgeführt, da sie nicht mehr von der Bedingung in der `if`-Anweisung abhängig ist.

```
F:\PerlProgramme\Kapitel_05>perl if.pl
Bitte geben Sie die Anzahl der bestellten CDs ein: 6
Der Preis liegt über 50 EUR: Ein Rabatt ist möglich!
Rechnungsbetrag bei 6 bestellter CDs: 78 EUR
F:\PerlProgramme\Kapitel_05>
```

Das Programm informiert über einen möglichen Rabatt

Auch wenn durch die `if`-Anweisung nur eine Anweisung ausgeführt werden soll, ist es oft sinnvoll, einen Anweisungsblock zu bilden. Die Übersichtlichkeit und Lesbarkeit des Programms wird dadurch verbessert.



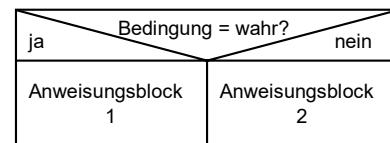
Beispiel

Wenn nur eine einzelne Anweisung in Abhängigkeit von einer Bedingung ausgeführt werden soll, können Sie die Kurzform der `if`-Anweisung verwenden. Im Beispiel soll ein Rabattwert von 10 % (0.1) gelten, wenn ein Gesamtpreis über 50 € vorliegt.

```
$rabatt=0.1 if ($gesamtpreis>50);
```

Zweiseitige Auswahl mit der `if-else`-Anweisung

Die zweiseitige Auswahl besitzt zwei Anweisungsblöcke, die in Abhängigkeit von der Auswahlbedingung ausgeführt werden. Dieser Typ der Auswahl kann damit auf beide Zustände der Auswahlbedingung reagieren. Wenn die Bedingung erfüllt ist (den Wert `true` liefert), dann wird der Anweisungsblock 1 ausgeführt, andernfalls der Anweisungsblock 2.



Zweiseitige Auswahl

Sie können die zweiseitige Auswahl verwenden, wenn es an einer Stelle im Programm genau zwei Auswahlmöglichkeiten gibt.

Syntax der `if-else`-Anweisung

```
if (Bedingung) {
    Anweisungsblock1;
} else {
    Anweisungsblock2;
}
```

- ✓ Die `if-else`-Anweisung beginnt mit dem Schlüsselwort `if`. Danach folgt in runden Klammern der Bedingungsausdruck.
- ✓ Ist die Bedingung erfüllt, wird der erste Anweisungsblock ausgeführt.
- ✓ Der alternative Anweisungsblock folgt nach dem Schlüsselwort `else`. Er wird ausgeführt, wenn die Bedingung nicht zutrifft. Der erste Anweisungsblock bleibt in diesem Fall unberücksichtigt.



Sie können auch innerhalb von Anweisungsblöcken einer Auswahl eine neue Auswahl verwenden, denn Auswahlanweisungen lassen sich beliebig verschachteln.

Beispiel: `ifelse.pl`

Das vorherige Beispiel wird hier weiter geführt. Falls der Gesamtpreis der bestellten CDs unter 50 € liegt, soll ebenfalls eine entsprechende Meldung angezeigt werden.

```
#!/usr/bin/perl
use strict;
my $einzelpreis = 13;

print "Bitte geben Sie die Anzahl der bestellten CDs ein: ";
my $anzahl=<STDIN>;
chomp($anzahl);

my $gesamtpreis = $einzelpreis * $anzahl;

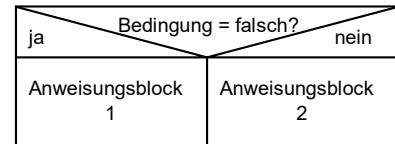
① if ($gesamtpreis >= 50) {
②     print "Der Preis liegt über 50 EUR: Ein Rabatt ist möglich!\n";
③ } else {
④     print "Der Preis liegt unter 50 EUR: Kein Rabatt möglich!\n";
}
print "Rechnungsbetrag bei $anzahl bestellter CDs: $gesamtpreis EUR\n";
```

- ① Am Beginn der zweiseitigen Auswahl steht die Auswahlbedingung. Der Vergleich des Gesamtpreises entscheidet, welche Meldung angezeigt wird.
- ② Die Bedingung ist erfüllt, wenn der Gesamtpreis größer oder gleich 50 € ist. Dann wird ein Hinweis auf den möglichen Rabatt ausgegeben.
- ③ Das Schlüsselwort `else` leitet den alternativen Anweisungsblock ein.
- ④ Falls der Gesamtpreis unter 50 € liegt, erfolgt die Meldung, dass kein Rabatt möglich ist.

Auswahl mit der `unless`-Anweisung

Wollen Sie einen Anweisungsblock nur dann ausführen, wenn die Auswahlbedingung **nicht** erfüllt ist, dann können Sie die `unless`- bzw. `unless-else`-Anweisung verwenden.

Dieser besondere Auswahltyp in Perl besitzt die gleiche Syntax wie die `if-else`-Anweisung. Die Auswahlbedingung wird hier jedoch nicht auf Wahrheit, sondern auf Unwahrheit überprüft.



Übersicht zur unless-Auswahl

Syntax der `unless-else`-Anweisung

Anweisung `unless` (Bedingung);

```

unless (Bedingung) {
    Anweisungsblock1;
} else {
    Anweisungsblock2;
}
  
```

- ✓ Die `unless`-Anweisung beginnt mit dem Schlüsselwort `unless`. Danach folgt in runden Klammern die Auswahlbedingung.
- ✓ Soll nur eine Anweisung ausgeführt werden, wenn die Bedingung nicht erfüllt ist, dann wird diese dem Schlüsselwort `unless` vorangestellt.
- ✓ Mehrere Anweisungen müssen als Block zusammengefasst werden.
- ✓ Ein alternativer Anweisungsblock wird mit dem Schlüsselwort `else` eingeleitet. Er wird ausgeführt, wenn die Auswahlbedingung erfüllt ist.

Bei der Verwendung der `unless`-Anweisung mit einem `else`-Block handelt es sich um eine umgekehrte `if-else`-Anweisung, die keine Vorteile bringt. Sinnvoll ist die `unless`-Anweisung nur dann, wenn Sie auf eine unwahre Bedingung reagieren wollen.



Zweiseitige Auswahl mit dem ternären Vergleichsoperator

Der ternäre (auch triadisch genannte) Vergleichsoperator `?` ermöglicht eine sehr kompakte zweiseitige Auswahl ähnlich der `if-else`-Anweisung. Die Verwendung ist immer dann sinnvoll, wenn Sie aufgrund einer Bedingung einer Variablen einen bestimmten Wert zuweisen möchten.

Der ternäre Vergleichsoperator erhält seinen Namen aufgrund der Tatsache, dass er als einziger Operator drei Operanden verlangt.



Syntax des Auswahl mit dem ternären Vergleichsoperator

Bedingung ? Operand1 : Operand2

- ✓ Die Anweisung beginnt mit der Auswahlbedingung. Zur besseren Übersicht kann diese in runde Klammern gesetzt werden.
- ✓ Der Ausdruck liefert in Abhängigkeit von der Bedingung einen der beiden Operanden als Ergebnis zurück.
- ✓ Nach dem ternären Operator `?` folgt der erste Operand. Er wird zurückgeliefert, wenn die Auswahlbedingung erfüllt ist.
- ✓ Durch einen Doppelpunkt getrennt wird der zweite Operand angegeben.

Eine zweiseitige Auswahl mit dem ternären Vergleichsoperator ist ein Ausdruck, der ein Ergebnis zurückliefert. Das Ergebnis können Sie einer Variablen zuweisen oder ausgeben. Die Operanden können dabei konstante Werte oder Ausdrücke sein, die einen Wert zurückliefern.



Beispiel: *ternärer_Operator.pl*

Ist der Gesamtpreis der bestellten CDs größer oder gleich 50 €, soll ein Rabatt von 5 % gewährt werden. Das Perl-Programm soll den Rabatt in Abhängigkeit von dieser Bedingung berechnen und anzeigen.

```
#!/usr/bin/perl
use strict;
my $einzelpreis = 13;

print "Bitte geben Sie die Anzahl der bestellten CDs ein: ";
my $anzahl=<STDIN>;
chomp($anzahl);

my $gesamtpreis = $einzelpreis * $anzahl;

① my $rabatt = ($gesamtpreis >= 50) ? $gesamtpreis * 0.05 : 0;

print "Rechnungsbetrag bei $anzahl bestellter CDs: $gesamtpreis EUR\n";
② print "Es sind $rabatt EUR Rabatt möglich. Endpreis: ",$gesamtpreis-$rabatt, " EUR\n";
```

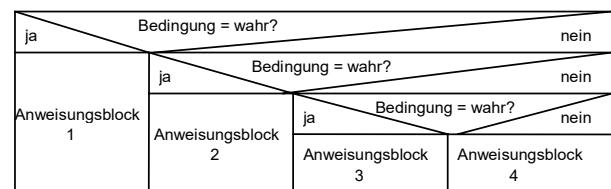
- ① Ein Vergleichsausdruck mit dem ternären Operator wird verwendet, um der Variablen \$rabatt den korrekten Rabattwert zuzuweisen. Ist der Gesamtpreis größer oder gleich 50 €, werden 5 % Rabatt vom Gesamtpreis berechnet und als Ergebnis zurückgeliefert. Andernfalls ist das Ergebnis des Ausdrucks Null.
- ② Hier erfolgt die Ausgabe des berechneten Rabattwerts und des Endpreises nach Abzug des Rabatts.

```
F:\PerlProgramme\Kapitel_05>perl ternärer_Operator.pl
Bitte geben Sie die Anzahl der bestellten CDs ein: 6
Rechnungsbetrag bei 6 bestellter CDs: 78 EUR
Es sind 3.9 EUR Rabatt möglich. Endpreis: 74.1 EUR
F:\PerlProgramme\Kapitel_05>
```

Berechnung des Rabatts aufgrund einer Bedingung

Mehrstufige Auswahl mit der *if-elsif-Anweisung*

Bei der mehrstufigen Auswahl existieren mehrere Anweisungsblöcke, von denen ein Block in Abhängigkeit von verschiedenen Bedingungen ausgeführt wird. Die Bedingungsabfragen sind dabei ineinander verschachtelt. Wenn die erste Bedingung erfüllt ist, wird Anweisungsblock 1 ausgeführt. Wenn nicht, fährt Perl mit der zweiten Bedingung fort usw., bis letztendlich der passende Anweisungsblock gefunden wurde.



Übersicht zur mehrstufigen Auswahl

Eine mehrstufige Auswahl realisieren Sie in Perl mit der *if-elsif-Anweisung*. Sie können die mehrstufige Auswahl verwenden, wenn es mehr als zwei Auswahlmöglichkeiten gibt.

Syntax der if-elsif-Anweisung

```
if (Bedingung1) {
    Anweisungsblock1;
} elsif (Bedingung2) {
    Anweisungsblock2;
}
...
} else {
    AnweisungsblockN;
}
```

- ✓ Die if-elsif-Anweisung wird wie eine if-else-Anweisung verwendet. Die mehrstufige Struktur wird durch den Einsatz des Schlüsselworts elsif erreicht.
- ✓ Die Anweisung kann beliebig viele elsif-Zweige enthalten. Jeder davon muss eine Auswahlbedingung besitzen.
- ✓ Die letzte Stufe stellt der else-Zweig dar. Die Anweisungen dieser Alternative werden ausgeführt, wenn keine der zuvor geprüften Bedingungen zutreffend war. Dieser Anweisungsblock kann auch entfallen.
- ✓ Der erste Anweisungsblock, dessen Bedingung zutrifft, wird abgearbeitet. Danach findet keine Bedingungsprüfung mehr statt. Nach Abarbeitung eines Anweisungsblocks wird die mehrstufige Auswahl verlassen.

Beachten Sie, dass das Schlüsselwort zum Einleiten einer zusätzlichen Auswahlbedingung elsif und nicht „elseif“ lautet.



Beispiel: ifelsif.pl

Als Beispiel kann wieder der CD-Versender dienen. Je nach Gesamtpreis der Rechnung kann er einen unterschiedlich hohen Rabatt anbieten. In einem Programm soll auf folgende Gesamtpreis-Beträge reagiert werden:

unter 50 €	kein Rabatt
ab 50 € bis unter 75 €	5 % Rabatt
ab 75 € bis unter 100 €	10 % Rabatt
ab 100 €	15 % Rabatt

```
#!/usr/bin/perl
use strict;
my $einzelpreis = 13;

print "\nBitte geben Sie die Anzahl der bestellten CDs ein: ";
my $anzahl=<STDIN>;
chomp($anzahl);

my $gesamtpreis = $einzelpreis * $anzahl;
my $rabatt;
① if ($gesamtpreis < 50) {
    $rabatt = 0;
② } elsif ($gesamtpreis < 75) {
    $rabatt = $gesamtpreis * 0.05;
③ } elsif ($gesamtpreis < 100) {
    $rabatt = $gesamtpreis * 0.1;
```

```

④ } else {
    $rabatt = $gesamtpreis * 0.15;
}

print "Rechnungsbetrag bei $anzahl bestellter CDs: $gesamtpreis EUR\n";
print "Es sind $rabatt EUR Rabatt möglich. Endpreis: ",$gesamtpreis-
    $rabatt, " EUR\n";

```

- ① Hier beginnt die if-elsif-Anweisung. Liegt der Gesamtpreis unter 50 €, ist kein Rabatt möglich.
- ② Das Schlüsselwort elsif leitet eine neue untergeordnete Bedingung ein, die überprüft wird, wenn der Gesamtpreis nicht unter 50 € liegt. Falls der Preis nun unter 75 € liegt, erfolgt die Berechnung des Rabatts mit 5 %.
- ③ Hier wird geprüft, ob der Gesamtpreis unter 100 € liegt. Wenn ja, erfolgt die Berechnung des Rabatts mit 10 %.
- ④ Falls alle bisherigen Bedingungen falsch waren, wird der Rabatt mit 15 % berechnet. Dies ist für einen Gesamtpreis größer 100 € der Fall.

```

F:\PerlProgramme\Kapitel_05>perl ifelsif.pl
Bitte geben Sie die Anzahl der bestellten CDs ein: 6
Rechnungsbetrag bei 6 bestellter CDs: 78 EUR
Es sind 7.8 EUR Rabatt möglich. Endpreis: 70.2 EUR

F:\PerlProgramme\Kapitel_05>perl ifelsif.pl
Bitte geben Sie die Anzahl der bestellten CDs ein: 10
Rechnungsbetrag bei 10 bestellter CDs: 130 EUR
Es sind 19.5 EUR Rabatt möglich. Endpreis: 110.5 EUR

F:\PerlProgramme\Kapitel_05>_

```

Das Programm berechnet den Rabatt in Abhängigkeit vom Gesamtpreis

5.3 Bedingte Wiederholungsschleifen

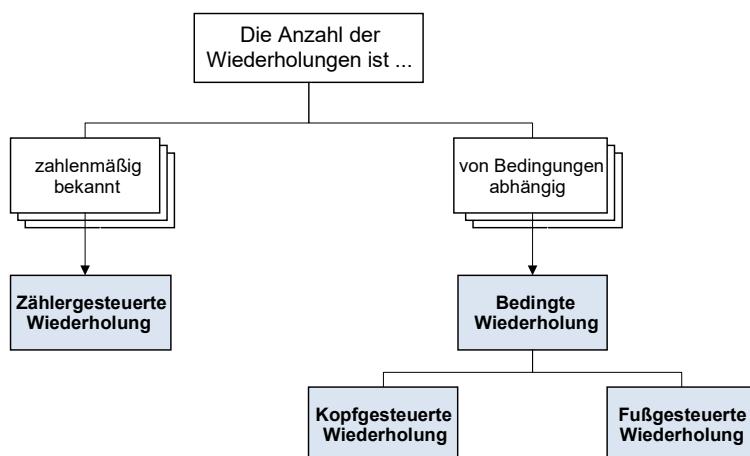
Anweisungen wiederholt ausführen

Manchmal müssen in einem Programm Anweisungen mehrmals wiederholt werden, beispielsweise um eine Liste von Werten abzuarbeiten. Oft ist es auch nicht vorhersehbar, wie oft Anweisungen ausgeführt werden sollen. Für diesen Zweck gibt es in Perl Strukturen, die eine Wiederholung ermöglichen. Damit können Sie alle Anweisungen, die sich wiederholen, als einen Anweisungsblock in knapper Form zusammenfassen.

Kontrollstrukturen für solche Wiederholungen werden auch **Schleifen** genannt, da die Anweisungen schleifenförmig abgearbeitet werden.

Eine Schleife besteht aus einer Schleifensteuerung und einem Schleifenkörper. Die Schleifensteuerung gibt an, wie oft oder unter welchen Bedingungen die Anweisungen abgearbeitet werden. Der Schleifenkörper enthält den zu wiederholenden Anweisungsblock.

Je nach Art der Schleifensteuerung werden verschiedene Arten von Wiederholungskonstrukten unterschieden. So kann die Schleifensteuerung beispielsweise vor (kopfgesteuerte Schleife) oder nach (fußgesteuerte Schleife) dem Schleifenkörper angeordnet sein, außerdem kann die Steuerung über einen Zähler oder eine Bedingung erfolgen.



Überblick über die verschiedenen Wiederholungsschleifen

Schleifen können wie Auswahlkonstrukte ineinander verschachtelt werden. Genauso können Sie verschiedene Kontrollstrukturen beliebig miteinander kombinieren, wenn es die Aufgabenstellung erfordert.



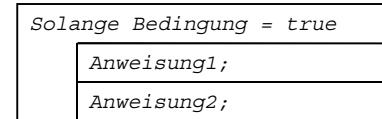
Kopfgesteuerte Schleifen

Nicht immer ist es möglich, im Voraus festzulegen, wie oft sich ein Anweisungsblock wiederholen soll. In solchen Fällen wird die Anzahl der Wiederholungen von einer Bedingung abhängig gemacht. Diese Bedingung wird bei jedem Schleifendurchlauf geprüft, und es wird entschieden, ob der Schleifenkörper noch einmal durchlaufen werden soll oder nicht.

Eine Form der bedingten Wiederholung ist die kopfgesteuerte Wiederholung. Dabei wird jeweils vor der Abarbeitung des Schleifenkörpers geprüft, ob die Anweisungen bearbeitet werden sollen.

Kopfgesteuerte Schleife mit der `while`-Anweisung

Solange die Bedingung am Anfang der Schleife erfüllt ist, werden die Anweisungen im Schleifenkörper ausgeführt, ansonsten wird die Schleife abgebrochen. Aufgrund des Schlüsselworts `while` (engl. while = solange) wird diese Schleife auch `while`-Schleife genannt.



Schema der `while`-Schleife

Es kann vorkommen, dass die Schleife nie durchlaufen wird, nämlich dann, wenn die Bedingung nie erfüllt ist (abweisende Schleife genannt). Genauso ist es auch möglich, dass die Bedingung immer erfüllt wird und damit die Schleife nie verlassen wird. Achten Sie daher darauf, vor Testläufen die Bedingung genau zu überprüfen, denn es besteht die Gefahr von Programmfehlern bzw. Endlosschleifen.



Syntax der kopfgesteuerten `while`-Schleife

```
while (Bedingung) {
    Anweisungsblock;
}
```

- ✓ Die Schleifensteuerung am Beginn der Schleife wird mit dem Schlüsselwort `while` eingeleitet. In runden Klammern folgt danach die Schleifenbedingung.
- ✓ Ist der Rückgabewert der Bedingung `true`, werden die Anweisungen im Schleifenkörper ausgeführt.
- ✓ Am Ende der Schleife erfolgt eine erneute Prüfung der Bedingung. Ist sie wieder erfüllt, wird die Schleife nochmals ausgeführt.

Beispiel: `ggTeiler.pl`

Als Beispiel soll ein Programm dienen, das für zwei eingegebene Zahlen den größten gemeinsamen Teiler bestimmt.

```

#!/usr/bin/perl
use strict;
print "\nErste Zahl: ";
my $zahl1=<STDIN>;
print "Zweite Zahl: ";
my $zahl2=<STDIN>

① while ($zahl2 != 0) {
②     my $temp = $zahl2;
③     $zahl2 = $zahl1 % $zahl2;
④     $zahl1 = $temp;
}

print "\nDer groesste gemeinsame Teiler ist $zahl1.\n";

```

- ① Die Schleife wird durchlaufen, solange die Variable \$zahl2 ungleich dem Wert 0 ist. Dies ist der Fall, wenn es bei der Division noch einen Rest gibt, der größte gemeinsame Teiler also noch nicht gefunden wurde.
- ② In der Variablen \$temp wird der Wert von \$zahl2 zwischengespeichert.
- ③ Der Variablen \$zahl2 wird mit dem Modulo-Operator der Rest aus der Division von \$zahl1 durch \$zahl2 zugewiesen.
- ④ Der in der Variablen \$temp gespeicherte ehemalige Wert von \$zahl2 (die nun den Rest der Division enthält) wird \$zahl1 zugewiesen.

```

F:\PerlProgramme\Kapitel_05>perl ggTeiler.pl
Erste Zahl: 345
Zweite Zahl: 125
Der groesste gemeinsame Teiler ist 5.
F:\PerlProgramme\Kapitel_05>

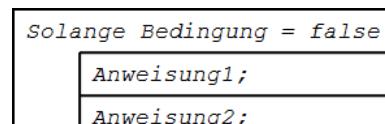
```

Berechnung des größten gemeinsamen Teilers

Kopfgesteuerte Schleife mit der `until`-Anweisung

Die until-Schleife (engl. until = bis) ist das Gegenstück der while-Schleife. Bei ihr werden die Anweisungen im Schleifenkörper nur ausgeführt, wenn die Schleifenbedingung nicht erfüllt ist.

Ähnlich der while-Schleife kann auch bei der until-Schleife der Schleifenkörper unendlich oft oder niemals ausgeführt werden.



Schema der while-Schleife

Syntax der kopfgesteuerten `until`-Schleife

```

until (Bedingung) {
    Anweisungsblock;
}

```

- ✓ Die until-Schleife wird mit dem Schlüsselwort `until` eingeleitet. Danach folgt in runden Klammern die Schleifenbedingung.
- ✓ Die Anweisungen im Schleifenkörper werden ausgeführt, solange die Schleifenbedingung nicht erfüllt ist.

Fußgesteuerte Schleifen

Bei der fußgesteuerten Wiederholung findet die Bedingungsprüfung am Ende der Schleife statt. Nach Abarbeitung des Schleifenkörpers wird geprüft, ob die Schleife noch einmal durchlaufen wird oder nicht. Auch bei diesen bedingten Schleifen gibt es eine Version mit dem Schlüsselwort `while` und eine mit dem Schlüsselwort `until`. Zum Einleiten der Schleife wird in Perl das Schlüsselwort `do` verwendet. Bis die Bedingung erfüllt (`do-while-Schleife`) bzw. nicht erfüllt (`do-until-Schleife`) ist, werden die Anweisungen des Schleifenkörpers wiederholt ausgeführt.

Im Gegensatz zur kopfgesteuerten Wiederholung wird hier der Schleifenkörper mindestens einmal durchlaufen. Sie können diese Schleife verwenden, wenn der Anweisungsblock der Schleife einmal auf jeden Fall ausgeführt werden soll. Erst dann wird geprüft, ob die Bedingung eine Wiederholung erfordert.



Syntax der fußgesteuerten `while-/until`-Schleife

```
do {
    Anweisungsblock;
} while (Bedingung)
```

```
do {
    Anweisungsblock;
} until (Bedingung)
```

- ✓ Beide Schleifen werden mit dem Schlüsselwort `do` eingeleitet. Danach folgt der Anweisungsblock des Schleifenkörpers.
- ✓ Der Anweisungsblock wird mindestens einmal ausgeführt, bevor eine Bedingungsprüfung erfolgt.
- ✓ Am Ende der Schleife folgt eines der Schlüsselwörter `while` oder `until` und der gewünschte Bedingungsausdruck.
- ✓ Mit dem Schlüsselwort `while` wird die Schleife so lange ausgeführt, bis die Bedingung falsch ist. Mit `until` wird der Anweisungsblock ausgeführt, bis die Anweisung wahr ist.
- ✓ Nach der Bedingung wird kein Semikolon angegeben.

Beispiel: `dountil.pl`

Das folgende Perl-Programm realisiert ein Kassensystem. In einer Schleife werden so lange Preise über die Tastatur eingelesen, bis ein leerer Wert eingegeben wird. Am Ende des Programms werden Gesamtpreis und Anzahl der summierten Produktpreise angezeigt.

```
#!/usr/bin/perl
use strict;
print "\nBitte geben Sie nacheinander die einzelnen Preise ein.\n";
print "Keine Eingabe und Return zum Beenden.\n\n";
my $preis=0;
my $anzahl=0;
my $gesamtpreis=0;
① do {
②     $preis=<STDIN>;
③     chomp($preis);
④     $gesamtpreis+=$preis;
⑤     $anzahl++;
⑥ } until ($preis == "");
⑦ $anzahl--;
print "Der Gesamtpreis bei $anzahl Produkten beträgt $gesamtpreis EUR.\n";
```

- ① Die Schleife wird mit dem Schlüsselwort `do` eingeleitet. Die nachfolgenden Anweisungen werden mindestens einmal ausgeführt.
- ② Der Einzelpreis wird von der Tastatur eingelesen und zum Gesamtpreis addiert.
- ③ Der Zähler für die Anzahl der eingegebenen Preise wird bei jedem Schleifendurchlauf um den Wert 1 erhöht.
- ④ Am Ende der Schleife erfolgt nach dem Schlüsselwort `until` die Bedingungsprüfung. Hier soll die Schleife so oft durchlaufen werden, bis ein leerer Einzelpreis eingegeben wurde.
- ⑤ Da auch der letzte Schleifendurchlauf als Preis gewertet wurde, muss der Zähler verringert werden. Danach erfolgt die Ausgabe des Gesamtpreises.



Sie können Schleifen mit den Schlüsselwörtern `while` und `until` gleichwertig verwenden und zu jeder Zeit gegeneinander austauschen. Meist bietet sich jedoch aufgrund der Logik der zu prüfenden Bedingung eine der Möglichkeiten an.



Beachten Sie, dass Variablen, die in Schleifen oder anderen Blöcken benötigt werden, in der Regel im globalen Kontext deklariert werden müssen, wenn Sie mit `use strict;` arbeiten und auf diese dort zugreifen wollen. Ebenso sollten Variablen in der Regel einen Vorgabewert haben.

```
F:\PerlProgramme\Kapitel_05>perl dountil.pl
Bitte geben Sie nacheinander die einzelnen Preise ein.
Keine Eingabe und Return zum Beenden.

4
5
6
7
8

Der Gesamtpreis bei 5 Produkten beträgt 30 EUR.

F:\PerlProgramme\Kapitel_05>_
```

Berechnung des Gesamtpreises aus eingegebenen Einzelpreisen

5.4 Zählergesteuerte Wiederholung

Bei der zählergesteuerten Wiederholung ist die Anzahl der Wiederholungen des Schleifenkörpers genau festgelegt. Dazu wird eine Zählervariable verwendet, deren Startwert am Beginn der Schleife festgelegt wird. Bei jedem Durchlauf wird der Wert der Zählervariablen im Schleifenkopf verändert und über eine Bedingung geprüft, ob die Schleife beendet werden soll.

Die zählergesteuerte Wiederholung wird in Perl, wie auch in anderen Sprachen, mit der `for`-Anweisung realisiert.

Syntax der `for`-Schleife

```
for (Ausdruck; Bedingung; Iteration) {
    Anweisungsblock;
}
```

- ✓ Die Schleife beginnt mit dem Schlüsselwort `for`. In runden Klammern folgen danach die Parameter zur Schleifensteuerung.
- ✓ Mit dem Parameter `Ausdruck` wird die Zählervariable benannt und initialisiert.
- ✓ Die Bedingung legt fest, ob die Schleife erneut durchlaufen wird. Vor jedem neuen Schleifendurchlauf erfolgt eine Prüfung der Bedingung. Ist sie wahr, wird der Anweisungsblock abgearbeitet. Liefert die Bedingung dagegen den Wert `false`, wird die Schleife beendet.
- ✓ Mit dem Ausdruck `Iteration` wird der Wert der Zählervariablen bei jedem Durchlauf verändert.

Beispiel: *for.pl*

Das folgende Perl-Programm ermittelt alle durch die Zahl 3 teilbaren Zahlen im Bereich von 1 bis 40.

```
#!/usr/bin/perl
use strict;
① for (my $i=1; $i<=40; $i++) {
②     print "$i\n" if ($i % 3 == 0);
}
```

- ① Hier beginnt die `for`-Schleife. Als Zählervariable wird die Variable `$i` verwendet und deren Startwert auf 1 festgelegt. Die angegebene Bedingung führt dazu, dass die Schleife so lange durchlaufen wird, wie der Wert `$i` kleiner oder gleich 40 ist. Mit dem Ausdruck `$i++` wird der Wert von `$i` bei jedem Durchlauf um 1 erhöht.
- ② Mit einer `if`-Anweisung erfolgt eine Prüfung, ob der aktuelle Wert der Zahl `$i` ohne Rest durch 3 teilbar ist. In diesem Fall erfolgt eine Ausgabe der Zahl.

Beachten Sie, dass in der `for`-Schleife mit `my $i` eine schleifenlokale Variable direkt im Schleifenkopf eingeführt wird. Die Variable `$i` steht damit im gesamten Schleifenkörper (und nur da) zur Verfügung.

**for-Schleife ohne Parameter**

```
for (;;) {
    Anweisungsblock;
}
```

- ✓ Im Schleifenkopf werden keine Parameter zur Steuerung übergeben.
- ✓ Die Schleife wird endlos ausgeführt und kann nur mit der Anweisung `last` abgebrochen werden.

Falls Ihr Programm eine endlose Schleife nicht mehr verlässt, können Sie es bei der Ausführung in der Konsole mit der Tastenkombination `Strg C` abbrechen. Beachten Sie, dass es bei CGI-Programmen derartige Möglichkeiten nicht gibt.



5.5 Anweisung zur Schleifen- und Programmsteuerung

Perl besitzt eine Anzahl von Anweisungen, mit denen Sie gezielt in den Schleifen- oder Programmablauf eingreifen können. So ist es z. B. möglich, einen Schleifendurchlauf auszulassen oder die Schleife bzw. das Programm zu beenden.

Die folgende Tabelle zeigt eine Übersicht der möglichen Anweisungen:

redo	Der aktuelle Schleifendurchlauf wird abgebrochen und ein neuer begonnen. Dabei wird die Schleifenbedingung nicht nochmals bewertet. In einer <code>for</code> -Schleife wird die Zählervariable nicht verändert.
next	Der aktuelle Schleifendurchlauf wird abgebrochen und ein neuer begonnen. Die Schleifenbedingung wird dabei wie üblich ausgewertet.
last	Sofortiger Abbruch der Schleife
exit	Beendet das Perl-Programm

Alle Anweisungen werden ohne Parameter aufgerufen.

Beispiel: *last.pl*

Das Beispiel aus dem vorherigen Abschnitt wird mit einer endlosen `for`-Schleife realisiert. Mit der `last`-Anweisung kann die Schleife dennoch abgebrochen werden.

```
#!/usr/bin/perl
use strict;
my $i = 0;
① for (;;) {
    $i++;
    print "$i\n" if ($i % 3 == 0);
② last if ($i>40);
}
```

- ① Die `for`-Schleife wird ohne Parameter angegeben. Sie wird daher endlos oft durchlaufen.
- ② Mit einer `if`-Anweisung erfolgt eine Prüfung, ob der Wert von `$i` größer ist als 40. Wenn dies der Fall ist, wird die Schleife mit `last` abgebrochen. Die Variable `$i` wurde außerhalb der Schleife deklariert und mit dem Wert 0 initialisiert.

5.6 Schnellübersicht

Was bedeutet ...	
Bedingung	Zwei oder mehr Werte, die über Vergleichsoperatoren und logische Operatoren verknüpft werden; eine Bedingung kann entweder wahr (<code>true</code>) oder falsch sein (<code>false</code>)
Einseitige Auswahl	Anweisungsblock, der in Abhängigkeit von einer Bedingung einmal oder gar nicht ausgeführt wird <code>if (Bedingung) { ... }</code>
Zweiseitige Auswahl	Zwei alternative Anweisungsblöcke stehen zur Auswahl. In Abhängigkeit von einer Bedingung wird einer der Anweisungsblöcke ausgeführt. <code>if (Bedingung) { ... } else { ... }</code>
Mehrstufige Auswahl	In Abhängigkeit von mehreren nacheinander angeordneten Bedingungen soll von mehreren Anweisungsblöcken einer ausgeführt werden. <code>if (Bedingung) { ... } elsif (Bedingung) { ... } else { ... }</code>
Zählergesteuerte Wiederholung	Es ist im Voraus bekannt, wie oft ein Anweisungsblock wiederholt werden soll. <code>for (Ausdruck; Bedingung; Iteration) { ... }</code>
Kopfgesteuerte Wiederholung	Anweisungen sollen in Abhängigkeit von einer Bedingung einmal, mehrmals oder gar nicht ausgeführt werden. Die Bedingung wird am Anfang der Schleife geprüft. <code>while/until (Bedingung) { ... }</code>
Fußgesteuerte Wiederholung	Anweisungen sollen in Abhängigkeit von einer Bedingung mindestens einmal oder mehrmals ausgeführt werden. Die Bedingung wird am Ende der Schleife geprüft. <code>do { ... } while/until (Bedingung)</code>

5.7 Übungen

Schleifen kennen und anwenden

Übungsdatei: --

Ergebnisdatei: --

1. Erklären Sie den Ablauf einer kopf- und einer fußgesteuerten Schleife.
2. Erläutern Sie die Unterschiede zwischen Schleifen mit der `while`- und der `until`-Anweisung.
3. Was ist eine Bedingung?
4. Welche Parameter werden bei einer zählergesteuerten Schleife verwendet?
5. Mit welcher Anweisung können Sie eine Schleife verlassen, ohne die Bedingung auszuwerten?

Zählergesteuerte Schleifen

Übungsdatei: --

Ergebnisdatei: *AddZahlen-E.pl*

1. Erstellen Sie ein Perl-Programm, das alle Zahlen von 1 bis zu einem über die Tastatur einzugebenden Wert addiert.
2. Verwenden Sie eine zählergesteuerte Schleife, um die Zahlen zu addieren. Speichern Sie die Summe in einer eigenen Variablen.
3. Geben Sie das Ergebnis auf dem Bildschirm aus. Zeigen Sie dabei auch den eingegebenen Endwert an.

Mit verschiedenen Kontrollstrukturen arbeiten

Übungsdatei: --

Ergebnisdatei: *Punkte-E.pl*

1. Erstellen Sie ein Programm, das aus einer eingegebenen Punktzahl eine Schulnote ermittelt. Verwenden Sie dazu folgende Aufstellung:

Punkte	Note
100-91	1
90-76	2
75-61	3
60-46	4
45-31	5
30-0	6

2. Prüfen Sie die Eingabe mit einer mehrseitigen Auswahl, und geben Sie jeweils die zugehörige Note aus.
3. Fügen Sie eine zusätzliche Prüfung ein, damit keine Punktewerte größer als 100 eingegeben werden können. Geben Sie in diesem Fall eine Meldung aus.
4. Die Auswertung der Punkte soll mehrfach ausgeführt werden können. Zeigen Sie dazu eine Aufforderung an, einen erneuten Durchlauf mit der Eingabe von *ja* oder *nein* zu bestätigen.
5. Verwenden Sie eine geeignete Schleife, um die Auswertung mindestens einmal und so lange auszuführen, bis die Aufforderung nicht mit *ja* bestätigt wird.

```
F:\PerlProgramme\Uebungsdateien\Kapitel_05>perl Punkte-E.pl
Bitte geben Sie die Punktzahl an: 78
Note 2
Noch mehr Punkte auswerten? ja/nein: ja
Bitte geben Sie die Punktzahl an: 65
Note 3
Noch mehr Punkte auswerten? ja/nein: nein
F:\PerlProgramme\Uebungsdateien\Kapitel_05>
```

Wiederholte Auswertung der Punkte

6 Listen und Datenfelder

In diesem Kapitel erfahren Sie

- ✓ was Listen, Arrays und Hashes sind
- ✓ wie Sie Daten in Arrays und Hashes speichern
- ✓ wie Sie auf einzelne Elemente zugreifen
- ✓ wie Sie in einer Schleife auf alle Elemente zugreifen

Voraussetzungen

- ✓ Variablen und Schleifen verwenden

6.1 Daten in Feldern speichern

Vorteile von Datenfeldern

Eine skalare Variable kann stets nur einen Wert speichern. Um viele gleichartige Daten in Form einer Liste oder Tabelle zu speichern, können Sie Datenfelder verwenden. Sie sind in der Lage, mehrere Werte unter einem Bezeichner (Variablennamen) zu speichern. Damit können Sie z. B. in einer Schleife über einen Zähler auf die einzelnen Werte zuzugreifen. Um beispielsweise bei einer Berechnung zehn Ergebnisse zu speichern, benötigen Sie statt zehn einzelner skalarer Variablen nur ein Datenfeld, das alle Ergebnisse aufnehmen kann. Für den Zugriff auf die einzelnen Werte werden ein **Datenfeldname** und ein **Index oder Schlüsselwert** benötigt.



Im Gegensatz zu anderen Sprachen müssen die einzelnen Elemente eines Datenfelds in Perl nicht den gleichen Datentyp besitzen. Sie können gleichzeitig Zahlen und Zeichenketten in einem Datenfeld speichern.

Arten von Datenfeldern

Beim Einsatz von Datenfeldern wird zwischen indizierten Datenfeldern (so genannten Arrays) und assoziativen Datenfeldern (so genannten Hashes) unterschieden. Zusätzlich ist der Begriff Liste mit Datenfeldern eng verbunden.

Liste	Eine Liste ist eine Anzahl bzw. Gruppe geordneter skalarer Daten, die in runden Klammern zusammengefasst werden. Eine Liste kann in einem Array gespeichert werden.
Array	Datenfeld, auf dessen Elemente über einen numerischen Index zugegriffen wird. Sie werden durch ein @ vor dem Variablenamen gekennzeichnet.
Hash	Datenfeld, auf dessen Elemente über einen alphanumerischen Schlüsselwert zugegriffen werden kann. Sie werden durch ein % vor dem Variablenamen gekennzeichnet.

Datenfeldern initialisieren

In Perl werden alle Datenfelder dynamisch verwaltet. Sie müssen sich daher nicht auf eine Anzahl von Elementen festlegen, die in einem Datenfeld gespeichert werden sollen. Die Reservierung des erforderlichen Speichers erledigt Perl automatisch. Sobald Sie einem Datenfeld einen Wert zuweisen, wird das Feld initialisiert. Wenn Sie zusätzliche Elemente einfügen, wird das Datenfeld entsprechend erweitert.



Skalare Variablen, Arrays und Hashes haben unterschiedliche Namensräume. Daher ist es möglich, in einem Programm gleichzeitig eine skalare Variable und ein Array mit gleichen Namen zu verwenden.

6.2 Arrays verwenden

In Arrays können Sie Daten als eindimensionale Listen oder mehrdimensionale Tabellen speichern. Arrays können als Ganzes oder elementweise angesprochen werden. Auf die einzelnen Elemente des Arrays greifen Sie über einen numerischen Indexwert zu. In jeder Dimension des Arrays werden die Werte daher mit einem fortlaufenden Index gespeichert.

Die Indizes von Arrays beginnen in Perl mit 0. Das erste Element besitzt daher den Indexwert 0, das zweite den Indexwert 1 und so weiter. Der Index eines Arrays mit n Elementen reicht von 0 bis n-1.



Listen

Eine Liste ist eine geordnete Folge von skalaren Werten, die in runden Klammern zusammengefasst werden. Listen sind eine einfache Möglichkeit, um eine Anzahl von Werten zusammenzufassen und in einem Array zu speichern.

Die folgende Tabelle zeigt einige Beispiele für Listen in Perl:

Liste	Erklärung
(2, 4, 6, 8, 10)	Die Liste enthält fünf Zahlen.
(2, "Leipzig", 10, "Frankfurt")	Die Liste enthält vier Elemente, gemischt aus Zahlen und Zeichenketten.
(18, \$name, 24, "Berlin")	Die Liste enthält neben Zahlen und einer Zeichenkette auch eine Variable. In der Liste wird dabei der Wert der Variablen gespeichert.
(2+2, 100, 2.5, 34)	Wenn die Liste Ausdrücke enthält, werden diese ausgewertet und das Ergebnis gespeichert.
("Note 1", "Note \$note")	Variablen in Zeichenketten werden ausgewertet, bevor die Zeichenkette in der Liste gespeichert wird.
()	Leere Liste
(1, 2, 6, @zahlen)	Die Elemente des Arrays @zahlen werden in die Liste eingefügt.

Intervalle in Listen angeben

Wenn Sie in einer Liste ein Intervall fortlaufender Werte zusammenfassen wollen, können Sie den Intervalloperator verwenden und damit Schreibarbeit sparen. Der Intervaloperator wird mit zwei Punkten dargestellt.

Beispiel

```
(1,2,3,4,5,6,7,8,9,10)
(2,4,8,10,11,12,13,14,15,16,20,22,24)
(a,b,c,d,e,f,g,h,i,j)
(1.5,2.5,3.5,4.5,5.5,6.5)
(-4,-3,-2,-1,0,1,2,3,4)
(1,2,3,4,5,20,21,22,23,24,25)
```

```
(1 .. 10)
(2,4,8,10 .. 16,20,22,24)
(a .. j)
(1.5 .. 6.5)
(-4 .. 4)
(1 .. 5,20 .. 25)
```

- ✓ Der Intervaloperator kann mit Zahlen und ASCII-Zeichen verwendet werden.
- ✓ Die Werte links und rechts des Operators bezeichnen die untere und obere Grenze des Bereichs. Die dazwischen liegenden Werte werden automatisch aufgefüllt.
- ✓ Der Operator kann auch mit Dezimalzahlen und negativen Zahlen verwendet werden.
- ✓ Intervallangaben sind auch innerhalb einer Liste mit weiteren Elementen möglich.

Werte in Arrays eingeben

Sie können einzelne Elemente eines Arrays mit Werten belegen oder einer Array-Variablen eine Liste zuweisen.

```
@arrayvariable = (Liste);
$arrayvariable[Index] = Wert;
$arrayvariable = @arrayvariable2;
```

- ✓ Beim Zuweisen einer Liste an ein Array wird das Array als Ganzes angesprochen und mit dem Zeichen `@` gekennzeichnet.
- ✓ Soll nur ein einzelner Wert gespeichert werden, muss der Indexwert des gewünschten Elements in eckigen Klammern angegeben werden. Da jedes Element einen skalaren Wert repräsentiert, wird die Variable mit dem Zeichen `$` gekennzeichnet.
- ✓ Einer Arrayvariablen kann auch ein anderes Array mit allen enthaltenen Werten zugewiesen werden.

Beispiel

Das Array `@namen` wird mit vier Elementen gefüllt. Die Indizes der einzelnen Werte sind in der Reihenfolge, in der die Werte in der Liste angegeben wurden.

```
@namen = ("Frank", "Franziska", "Walter", "Silke");
```

Das Array enthält danach die Werte in folgender Anordnung:

Index	0	1	2	3
Wert	Frank	Franziska	Walter	Silke



Bei der Eingabe von Zeichenketten in eine Liste kann die Angabe der Anführungszeichen auch entfallen. Perl wählt beim Erkennen von Buchstaben automatisch den richtigen Datentyp.

Zeichenketten zuweisen

Perl besitzt eine verkürzte Form, um ein Array zu erzeugen, das mehrere Zeichenketten ohne Leerzeichen enthält. Dafür wird die Funktion `qw` (quote word) verwendet.

Die folgenden drei Zuweisungsarten sind gleichwertig:

```
@namen = ("Frank", "Franziska", "Walter", "Silke");
@namen = (Frank, Franziska, Walter, Silke);
@namen = qw(Frank Franziska Walter Silke);
```

- ✓ Mit der `qw`-Funktion erstellen Sie eine Liste verschiedener Zeichenkettenelemente.
- ✓ Die einzelnen Elemente werden durch ein Leerzeichen voneinander getrennt.
- ✓ Das Ergebnis der `qw`-Funktion ist eine Liste, die einem Array zugewiesen werden kann.

Array über Tastatur einlesen

Sie kennen bereits die Eingabe eines skalaren Werts über die Tastatur durch das Auslesen der Standardeingabe `<STDIN>`. Auf die gleiche Art und Weise können Sie auch ein Array einlesen.



Anders als bei der Eingabe skalarer Werte beendet hier der Zeilenvorschub nur die Eingabe eines einzelnen Elements. Die Eingabe des Arrays wird erst durch eine besondere Steuersequenz beendet. Unter Windows verwenden Sie dazu **Strg Z** und unter Unix **Strg D**.

```
@namen = <STDIN>;
```

- ✓ Mit der Eingabe von **←** wird ein einzelnes Element abgeschlossen.
- ✓ Um die Eingabe zu beenden, geben Sie **Strg Z** bzw. **Strg D** ein und schließen Sie die Zeile mit **←** ab.
- ✓ Die einzelnen Elemente enthalten danach den Zeilenvorschub **\n**. Um ihn zu entfernen, können Sie die **chomp**-Funktion verwenden.

Gesamtes Array ausgeben

Mit der **print**-Funktion können Sie gleichzeitig alle Elemente eines Arrays ausgeben. Diese Funktion ist jedoch nur zu Testzwecken sinnvoll.

```
@namen = qw(Frank Franziska Walter);
print @namen;
```

```
@namen = qw(Frank Franziska Walter);
print "@namen";
```

Die Beispiele führen zu folgenden Ausgaben:

```
FrankFranziskaWalter
```

```
Frank Franziska Walter
```

- ✓ Wenn Sie eine Array-Variable an die **print**-Anweisung übergeben, werden alle Elemente ausgegeben.
- ✓ Damit Perl die einzelnen Elemente durch Leerzeichen trennt, kann das Array innerhalb einer Zeichenkette ausgegeben werden.

Beispiel: *ArrayEingabe.pl*

Das Array **@namen** wird mithilfe der **qw**-Funktion mit Elementen gefüllt und ausgegeben.

```
#!/usr/bin/perl
use strict;
① my @namen = qw(Peter Silke Franz Walter);
② print "@namen\n";
```

- ① Die **qw**-Funktion trennt die einzelnen Namen anhand der Leerzeichen und weist die Liste dem Array **@namen** zu.
- ② Hier erfolgt die Ausgabe des Arrays. Da es in Anführungszeichen gesetzt ist, werden die einzelnen Elemente durch Leerzeichen getrennt.

Array auf skalare Variablen aufteilen

Sie können die Elemente eines Arrays auf mehrere skalare Variablen aufteilen, in dem Sie Variablen in der Listen-Schreibweise mit runden Klammern angeben.

Beispiel

```
@namen = qw(Peter Silke Franz);
($name1,$name2,$name3) = @namen;
```

- ✓ Die skalaren Variablen müssen in runde Klammern eingeschlossen werden.
- ✓ Sie sollten so viele skalare Variablen angeben, wie Sie Elemente aus dem Array auslesen wollen.
- ✓ Die Elemente des Arrays werden, beginnend mit dem Indexwert 0, von links nach rechts auf die angegebenen Variablen aufgeteilt.

Auf Elemente eines Arrays zugreifen

Um auf einen einzelnen Wert eines Arrays zuzugreifen, geben Sie nach dem Arraynamen den Indexwert in eckigen Klammern an. Auf diese Art und Weise können Sie Werte im Array speichern oder aus diesem auslesen.

Syntax des Indexzugriffs

```
print $arrayname[Index];
$arrayname[Index] = Wert;
$variable = $arrayname[Index];
```

- ✓ Der Indexwert des gewünschten Elements wird in eckigen Klammern nach dem Namen des Arrays angegeben. Die Indizes eines Arrays beginnen beim Wert 0.
- ✓ Da ein einzelnes Element einen skalaren Wert darstellt, wird die Variable durch den Präfix `$` gekennzeichnet.
- ✓ Sie können den Wert eines Elements auslesen oder durch eine Wertzuweisung ersetzen.

Die einzelnen Elemente eines Arrays können Sie wie skalare Variablen in beliebigen Ausdrücken verwenden. Die Verwendung der Indexwerte eignet sich besonders für den Zugriff innerhalb einer Zählschleife.



Wenn Sie negative Indexwerte verwenden, beginnt Perl die Zählweise am Ende des Arrays. Der Indexwert `-1` bezeichnet daher das letzte Element der Liste. Das ist sehr ungewöhnlich und wird in den meisten anderen Programmiersprachen nicht so gemacht.

Indexwert des letzten Elements bestimmen

Durch den Präfix `$#` vor dem Namen des Arrays können Sie den höchsten Indexwert des Arrays ermitteln. Dabei handelt es sich um den Indexwert des letzten Elements, egal wie viele Elemente das Array enthält.

```
print $#arrayname; # Liefert den Indexwert des letzten Elements
print $arrayname[$#arrayname]; # Liefert den Wert des letzten Elements
$anzahl = @arrayname; # Liefert die Anzahl der Elemente im Array
```



Da die Indexierung mit dem Wert 0 beginnt, beträgt die Gesamtzahl der Elemente in einem Array `$#arrayname+1`. Wenn Sie das Array einer skalaren Variable zuweisen, erhalten Sie nicht den Indexwert des letzten Elements, sondern die Anzahl der Elemente im Array.

Beispiel: ArrayIndexzugriff.pl

Das folgende Beispiel zeigt einige Möglichkeiten für den Zugriff auf einzelne Elemente eines Arrays.

```
#!/usr/bin/perl
use strict;

① my @namen = qw(Peter Silke Franz Walter);
② $namen[2] = "Franziska";
③ my @werte=(1 .. 10);

④ for (my $i=0; $i<=$#namen; $i++) {
⑤   print "Der $i. Name lautet $namen[$i]\n";
}

⑥ my $zahl=++$werte[0];
print "Zahl = $zahl\n";
```

- ① Das Array @namen wird mithilfe der `qw`-Funktion mit Werten gefüllt.
- ② Der Wert des Elements mit dem Indexwert 2 wird von *Franz* in *Franziska* geändert.
- ③ Das Array @werte wird mit dem Intervalloperator gefüllt.
- ④ In einer `for`-Schleife werden alle Elemente des Arrays @namen ausgegeben. Der Ausdruck `$#namen` liefert den höchsten Indexwert.
- ⑤ Elemente eines Arrays können auch innerhalb von Zeichenketten verwendet werden.
- ⑥ Hier wird der Wert des Elements mit dem Index 0 zuerst um 1 erhöht (inkrementiert) und danach der Variablen \$zahl zugewiesen.

```
F:\PerlProgramme\Kapitel_06>perl ArrayIndexZugriff.pl
Der 0. Name lautet Peter
Der 1. Name lautet Silke
Der 2. Name lautet Franziska
Der 3. Name lautet Walter
Zahl = 2
```

Ausgabe des Beispielprogramms

Gleichzeitig auf mehrere Elemente zugreifen

Mit so genannten **Array-Scheiben** (engl. array slice) können Sie auf zwei oder mehr Elemente eines Arrays gleichzeitig zugreifen. Die Indexwerte der gewünschten Elemente werden dabei in eckigen Klammern an den Namen des Arrays angeschlossen. Da dabei stets eine Liste zurückgeliefert wird, muss der Arrayname durch ein vorangestelltes `@` gekennzeichnet werden.

Beispiel

Im Array @namen werden sechs Elemente gespeichert. Das Array @auswahl soll davon nur die Elemente mit den Indexwerten 0, 3 und 5 enthalten.

```
@namen = qw(Peter Silke Franz Walter Franziska Frank);
@auswahl = @namen[0,3,5];    # Array-Scheibe mit drei Elementen von @namen
@auswahl = @namen[0 .. 3];   # Array-Scheibe mit Intervalloperator definiert
@auswahl = @namen[0 .. $index];
# Array-Scheibe mit skalarer Variable definiert
```

- ✓ Um eine Array-Scheibe zu definieren, werden die Indexwerte der gewünschten Elemente in eckigen Klammern angegeben.
- ✓ Mehrere Indexwerte müssen durch Kommata getrennt werden.

- ✓ Eine Array-Scheibe ist selbst wieder ein Array.
- ✓ Die Indexwerte können auch mit dem Intervalloperator oder skalaren Variablen angegeben werden.

Funktionen für Arrays

Perl stellt eine Anzahl vordefinierter Funktionen für die Arbeit mit Arrays zu Verfügung. So können Sie Arrays sortieren, neue Elemente hinzufügen oder eine Zeichenkette in eine Liste aufspalten.

Funktion	Erläuterung	Beispiel
push	Fügt ein oder mehr skalare Werte am Ende des Arrays ein	<code>push (@werte, \$wert1, \$wert2);</code> Die Werte \$wert1 und \$wert2 werden am Ende des Arrays @werte eingefügt.
unshift	Fügt ein oder mehrere Werte am Anfang des Arrays ein. Die nachfolgenden Elemente werden verschoben.	<code>unshift (@werte, \$wert1);</code> Der Werte \$wert1 wird am Anfang des Arrays @werte eingefügt.
split	Zerlegt eine Zeichenkette anhand eines Trennzeichens in eine Liste	<code>@werte = split (//, \$zeichenkette);</code> Die Zeichenkette wird am Trennzeichen ";" aufgeteilt und die einzelnen Elemente im Array @werte gespeichert.
join	Fügt eine Liste zu einer Zeichenkette zusammen. Ein Trennzeichen für die Elemente kann angegeben werden.	<code>\$zeichenkette = join (";", @werte);</code> Die Elemente des Arrays @werte werden zu einer Zeichenkette zusammengeführt und durch das Zeichen ";" getrennt.
pop	Entfernt das letzte Element eines Arrays	<code>pop (@werte);</code> Das letzte Element des Arrays @werte wird gelöscht.
shift	Löscht das erste Element eines Arrays. Die folgenden Elemente rücken nach.	<code>shift (@werte);</code> Das erste Element des Arrays @werte wird gelöscht.
splice	Löscht beliebige Elemente eines Arrays. Angegeben wird der Indexwert des ersten und die Anzahl der zu löschen Elemente.	<code>splice (@werte, 1, 4);</code> Die vier Elemente mit den Indexwerten 1 bis 5 des Arrays @werte werden gelöscht.
splice	Ersetzen von Elementen eines Arrays. Angegeben wird der Indexwert des ersten Elements, die Anzahl und die neuen Werte.	<code>splice (@werte, 2, 2, "Berlin", "Bonn");</code> Die zwei Elemente mit den Indexwerten 2 bis 3 des Arrays @werte werden durch die Werte Berlin und Bonn ersetzt.
sort	Liefert die aufsteigend sortierte Form eines Arrays. Die Sortierung erfolgt dabei nach dem ASCII-Zeichensatz.	<code>@werte_sort = sort (@werte);</code> Die Elemente des Arrays @werte werden als Zeichenketten sortiert und die sortierte Liste dem Array @werte_sort übergeben.
reverse	Liefert die absteigend sortierte Form eines Arrays. Die Sortierung erfolgt dabei nach dem ASCII-Zeichensatz.	<code>@werte_sort = reverse (@werte);</code> Die Elemente des Arrays @werte werden als Zeichenketten sortiert und die sortierte Liste dem Array @werte_sort übergeben.
chomp	Löscht das Zeilenvorschubzeichen \n am Ende jedes Arrayelements	<code>chomp (@werte);</code> Das Zeilenvorschubzeichen \n wird am Ende aller Elemente des Arrays @werte entfernt.



Elemente numerisch sortieren

Mit der Perl-Funktion `sort` können Sie die Elemente eines Arrays nur nach dem ASCII-Zeichensatz sortieren lassen. Perl verwendet dazu intern die Vergleichsoperatoren `lt`, `gt` und `eq`. Zeichenketten werden auf diese Weise sinnvoll sortiert. Bei numerischen Werten, vor allem bei Zahlen mit mehreren Dezimalstellen, erhalten Sie jedoch ein falsches Ergebnis.

Sie können beim Aufruf der `sort`- oder der `reverse`-Funktion eine eigene Sortierfunktion angeben. Dies ist eine selbst definierte Funktion, die von Perl die zu vergleichenden Werte in den Variablen `$a` und `$b` erhält.

Beispiel: NumerischeSortierung.pl

Im Array `@werte` werden Zahlen mit mehreren Dezimalstellen gespeichert. Für eine korrekte numerische Sortierung wird eine eigene Sortierfunktion definiert, die auf dem numerischen Vergleich mit den Operatoren `<`, `>` und `==` beruht.

```
#!/usr/bin/perl
use strict;

my @werte=(7840,9856,3223,4323,2405,8801);

① my @werte_sort=sort(NumSort @werte);

② sub NumSort {
    if($a < $b)
        { return -1; }
    elsif($a == $b)
        { return 0; }
    else
        { return 1; }
}
```

- ① Beim Aufruf von `sort` wird der Name der selbst definierten Sortierfunktion `NumSort` übergeben. Perl verwendet nun statt der internen Sortierung diese Funktion.
- ② Hier beginnt die selbst definierte Funktion. Die zu sortierenden Werte werden von Perl in den Variablen `$a` und `$b` zur Verfügung gestellt. Je nach Ergebnis des Vergleichs muss die Funktion einen der Werte `-1`, `0` oder `1` zurückliefern.

6.3 Mehrdimensionale Arrays

In einem eindimensionalen Array wird für jedes Element nur ein Wert gespeichert. Der Zugriff erfolgt dabei über einen Indexwert. Dieses Verhalten spiegelt eine Liste oder einen Vektor wieder.

Wollen Sie jedoch Werte in Form einer Tabelle speichern, ist ein mehrdimensionales Array erforderlich. In einem mehrdimensionalen Array können Sie beispielsweise für eine Person außer dem Namen auch noch Straße, Postleitzahl und Ort speichern. Der Zugriff erfolgt dabei über zwei Indizes, die beim Vergleich mit einer Tabelle die Zeile und die Spalte des Werts bestimmen.

Die folgende Tabelle zeigt den Aufbau eines mehrdimensionalen Arrays:

Indexwert	0	1	2	3
0	Frank Meier	Waldstr. 5	64511	Braundorf
1	Sonja Siedler	Hainstr. 24	04109	Leipzig
2	Walter Schulze	Leipziger Str. 24	12001	Berlin

Würde die oben stehende Tabelle in einem Array @personen gespeichert, würden Sie den Wohnort (Indexwert 3) von *Sonja Siedler* (Indexwert 1) über die Angabe \$personen[1][3] erhalten.

Besonderheit von mehrdimensionalen Arrays in Perl

Perl kennt nur eindimensionale Listen und keine mehrdimensionalen Arrays in der Form, wie sie in anderen Sprachen verwendet werden. Sie können das Verhalten jedoch nachbilden, indem Sie in einem Element einer Liste eine zweite Liste speichern. Dabei sind beliebig viele Verschachtelungen erlaubt.

Jede in einem Array gespeicherte Unterliste ist eine zusätzliche Dimension.

Beispiel: MehrdimensionalesArray.pl

Für drei Personen werden Name, Straße, Postleitzahl und Wohnort in einem mehrdimensionalen Array gespeichert.

```
#!/usr/bin/perl
use strict;

① my @person1= ("Frank Meier", "Waldstr. 5", "64511", "Braunsdorf");
my @person2= ("Sonja Siedler", "Hainstr. 24", "04109", "Leipzig");
my @person3= ("Walter Schulze", "Leipziger Str. 24", "12001", "Berlin");

② my @personen= ( [@person1], [@person2], [@person3] );

③ print "Wohnort von $personen[1][0]: $personen[1][3]\n";
```

- ① Für jede Person wird eine Liste mit den verfügbaren Daten erstellt.
- ② Im Array @personen werden die einzelnen Arrays für jede Person als eigenes Element gespeichert. Damit eine Unterliste als neue Dimension begonnen wird, muss der Arrayname in eckige Klammern gesetzt werden.
- ③ Hier erfolgt die Ausgabe des Wohnorts (Indexwert 3) für die Person mit dem Indexwert 1.



Beachten Sie, dass Sie die Namen der Unterlisten in eckigen Klammern angeben. Andernfalls werden nur die Elemente der Liste in das neue Array kopiert, und es wird keine neue Dimension eingefügt.

Sie können auch folgende Schreibweise verwenden, um mehrdimensionale Arrays zu erstellen:

```
@arrayname = (\@array1, \@array2);
```

oder

```
$arrayname[0]=@array1;
$arrayname[1]=@array2;
```

- ✓ Statt den Arraynamen in eckige Klammern zu setzen, können Sie auch einen Backslash \ voranstellen. In diesem Fall wird eine Referenz auf die untergeordneten Arrays gespeichert. Eine Referenz ist ein Verweis auf den Speicherbereich eines Arrays.
- ✓ Sie können ein Array auch einem einzelnen Element zuweisen. Dies kann die Lesbarkeit des Programmcodes erhöhen.

6.4 Hashes verwenden

Hashes sind wie Arrays Datenfelder, in denen mehrere skalare Daten gespeichert werden können. Im Gegensatz zu Arrays erfolgt der Zugriff hier jedoch nicht über einen numerischen Index, sondern über eine Zeichenkette als alphanumerischen Schlüsselwert. Sie werden daher auch **assoziative Datenfelder** genannt. Für jedes Element des Datenfelds wird neben dem eigentlichen Wert ein Schlüssel gespeichert, der bei der Eingabe frei gewählt werden kann.

Werte in Hashes eingeben

Die Eingabe von Elementen in ein Hash erfolgt über das Zuweisen einer Liste, in der auf jeden Schlüssel der zu speichernde Wert folgt. Wie bei Arrays ist auch der Zugriff auf einzelne Elemente möglich.

```
%hashvariable = ("Schlüssel1", Wert1, "Schlüssel2", Wert2, ...);
%hashvariable = (Schlüssel1 => Wert1, Schlüssel2 => Wert2, ...);
$hashvariable{Schlüssel} = Wert;
%hashvariable = %hashvariable2;
```

- ✓ Beim Zuweisen einer Liste an ein Hash wird das Hash als Ganzes angesprochen und mit dem Zeichen % gekennzeichnet.
- ✓ In einer Liste geben Sie die Schlüssel und Werte durch Kommata getrennt an. Auf jeden Schlüssel muss der entsprechende Wert folgen. Die Schlüssel müssen mit Anführungszeichen als Zeichenketten gekennzeichnet werden. Die Werte können Zahlen oder ebenfalls Zeichenketten sein.
- ✓ Sie können die Schlüssel-Werte-Paare auch mithilfe des so genannten **Zeige-Operators =>** zusammenfassen. Die einzelnen Wertepaare werden durch Kommata getrennt. Bei dieser Eingabeweise kann die Angabe der Anführungszeichen bei den Schlüsseln entfallen, wenn diese keine Sonderzeichen (z. B. Umlaute) enthalten.
- ✓ Soll nur ein einzelner Wert gespeichert werden, muss der Schlüsselwert des gewünschten Elements in geschweiften Klammern angegeben werden. Da jedes Element einen skalaren Wert repräsentiert, wird die Variable mit dem Zeichen \$ gekennzeichnet.
- ✓ Einer Hash-Variablen kann auch ein anderes Hash mit allen enthaltenen Werten zugewiesen werden.

Beispiel

Im Hash %person werden Name, Adresse und Alter einer Person als Schlüssel-Wert-Paare gespeichert.

```
%person = (Vorname => "Frank", Name => "Meier", Strasse => Waldstr. 5,
"Postleitzahl" => "64511", Ort => "Braundorf", Alter => 27);
```

Das Hash enthält danach die Werte in folgender Anordnung:

Schlüssel	Vorname	Name	Strasse	Postleitzahl	Ort	Alter
Wert	Frank	Meier	Waldstr. 5	64511	Braundorf	27

Wollen Sie im Beispiel den Schlüsselwert Straße nennen, müssen Sie ihn wegen des Zeichens β in Anführungszeichen setzen.



Bei der Eingabe der Schlüssel-Wert-Paare mit dem Zeige-Operator => kann die Angabe der Anführungszeichen bei den Schlüsselwerten entfallen. Diese Eingabeform ist besonders empfehlenswert, da sie den Zusammenhang zwischen Schlüsseln und Werten am besten widerspiegelt.

Perl speichert die Elemente eines Hashes nicht in der Reihenfolge, wie Sie die Elemente eingeben, sondern in einer zufälligen Verteilung.



Sie können Hashes als Elemente eines Arrays verwenden oder Arrays als Wert eines Hash-Elements speichern. In jedem Fall sollten Sie dabei mit dem Zeichen `\$` die Referenz auf das Array oder Hash speichern. Der Zugriff erfolgt dann wie bei mehrdimensionalen Arrays.

Auf Elemente eines Hashes zugreifen

Um auf einen einzelnen Wert eines Hashes zuzugreifen, geben Sie nach dem Hashnamen den Schlüsselwert in geschweiften Klammern an. Auf diese Art und Weise können Sie Werte im Array speichern oder aus diesem auslesen.

Syntax des Zugriffs über Schlüsselwerte

```
print $hashname{Schlüssel};  
$hashname{Schlüssel} = Wert;  
$variable = $hashname{Schlüssel};
```

- ✓ Der Schlüsselwert des gewünschten Elements wird in geschweiften Klammern nach dem Namen des Hashes angegeben. Die genaue Schreibweise des Schlüsselwerts ist dabei von Bedeutung.
- ✓ Die Anführungszeichen des Schlüsselwerts können innerhalb der geschweiften Klammern entfallen, wenn keine Umlaute oder Sonderzeichen enthalten sind.
- ✓ Da ein einzelnes Element einen skalaren Wert darstellt, wird die Variable durch den Präfix `\$` gekennzeichnet.
- ✓ Sie können den Wert eines Elements auslesen oder durch eine Wertzuweisung ersetzen.
- ✓ Wenn Sie bei einer Wertzuweisung einen neuen Schlüssel verwenden, erstellt Perl automatisch ein neues Hash-Element.

Die einzelnen Elemente eines Hashes können Sie wie skalare Variablen in beliebigen Ausdrücken verwenden. Schlüsselwerte können auch durch skalare Variablen ersetzt werden.

Beispiel: HashZugriff.pl

Das folgende Beispiel zeigt einige Möglichkeiten für den Zugriff auf einzelne Elemente eines Arrays.

```
#!/usr/bin/perl
use strict;

① my %person = ( Vorname      => "Frank",
                  Name        => "Meier",
                  Strasse     => "Waldstr. 5",
                  Postleitzahl => "64511",
                  Ort         => "Braundorf",
                  Alter       => 27);

② print "\n$person{Vorname} $person{Name} wohnt in $person{Ort}.\n";
③ my $schluessel="Strasse";
④ print "$person{$schluessel}\n";
```

- ① Ein Hash %person wird mithilfe des Zeiger-Operators mit Werten gefüllt.
- ② Hier erfolgt eine Ausgabe verschiedener Elemente des Hashes. Die Schlüsselwerte werden auch innerhalb einer Zeichenkette ausgewertet.
- ③ Die Variable soll einen Schlüsselwert bestimmen.
- ④ Die Ausgabe des Hash-Elements erfolgt hier über die in ③ definierte Schlüsselvariable.

```
F:\PerlProgramme\Kapitel_06>perl HashZugriff.pl
Frank Meier wohnt in Braundorf.
Waldstr. 5
F:\PerlProgramme\Kapitel_06>
```

Ausgabe des Beispielprogramms

Funktionen für Hashes**Hashes mit unbekannten Schlüsselwerten**

Wenn Sie einen Hash innerhalb eines Programms dynamisch mit Schlüssel-Wert-Paaren füllen, beispielsweise durch Tastatureingaben, ist es möglich, dass Sie die Schlüsselwerte für den Zugriff auf einzelne Elemente nicht kennen.

Mit den Perl-Funktionen `keys` und `values` erhalten Sie eine Liste der Schlüssel (`keys`) bzw. Werte (`values`) eines Hashes. Gleichzeitig können Sie mit der `keys`-Funktion auch die Anzahl der Hash-Elemente ermitteln.

```
@schluessel = keys(%hashvariable);    # Array mit allen Schlüsseln
@werte = values(%hashvariable);        # Array mit allen Werten
$anzahl = keys(%hashvariable);         # Anzahl der Schlüssel-Wert-Paare
```

- Wird das Ergebnis der `keys`- oder `values`-Funktion einem Array zugewiesen, ist das Resultat eine Liste mit den Schlüsseln oder Werten des Hashes.
- Erfolgt dagegen eine Wertzuweisung an eine skalare Variable, dann ist das Ergebnis die Anzahl der Schlüssel-Wert-Paare.

Sie können das Array mit den Schlüsselwerten beispielsweise in einer `for`-Schleife durchlaufen, um zu allen Schlüsseln die entsprechenden Werte zu ermitteln.

Wie in Perl üblich, können Sie beim Aufruf der `values`- und `keys`-Funktion die runden Klammern auch weglassen.

Hash nach Schlüsseln oder Werten sortiert ausgeben

Um ein Hash sortiert auszugeben, müssen Sie die Liste der Schlüssel bzw. Werte mit der `keys`- bzw. `values`-Funktion ermitteln und sortieren. Danach können Sie über die sortierte Liste die einzelnen Elemente in einer `for`-Schleife ausgeben.

Beispiel: *HashSortiert.pl*

Im Hash `%farben` wird einigen englischen Farbnamen die deutsche Bezeichnung zugeordnet. Die Farben sollen nach der englischen Bezeichnung sortiert ausgegeben werden.

```
#!/usr/bin/perl
use strict;

① my %farben = (green  => "grün",
                 blue   => "blau",
                 yellow => "gelb",
                 red    => "rot",
                 brown  => "braun");

② my @farben_sort = sort(keys(%farben));
my $schluessel;
③ for ($i=0; $i<=$#farben_sort; $i++) {
④     $schluessel = $farben_sort[$i];
⑤     print "$schluessel = $farben{$schluessel}\n";
}
```

- ① Hier wird das Hash `%farben` initialisiert. Die englischen Farbbezeichnungen sind die Schlüsselwerte.
- ② Die `keys`-Funktion ermittelt die Liste der Schlüsselwerte. Diese wird mit der `sort`-Funktion gleichzeitig sortiert und einer Array-Variablen zugewiesen.
- ③ Die sortierte Schlüsselliste wird in einer `for`-Schleife durchlaufen. Dazu wird der Indexwert des Arrays `@farben_sort` ermittelt.
- ④ Der aktuelle Schlüsselwert wird mit dem Indexwert ermittelt und in einer skalaren Variablen gespeichert.
- ⑤ Mit dem englischen Farbnamen als Schlüssel kann nun auch die deutsche Bezeichnung aus dem Hash ermittelt werden.

```
F:\PerlProgramme\Kapitel_06>perl HashSortiert.pl
blue = blau
brown = braun
green = grün
red = rot
yellow = gelb
```

F:\PerlProgramme\Kapitel_06>_
Ausgabe des sortierten Hashes

Weitere Funktionen

Funktion	Erläuterung	Beispiel
<code>delete</code>	Löscht ein Schlüssel-Wert-Paar eines Hashes	<code>delete \$person{Alter};</code> Löscht das Element mit dem Schlüssel Alter aus dem Hash <code>%person</code>

Funktion	Erläuterung	Beispiel
exists	Ermittelt, ob ein Element mit einem bestimmten Schlüssel existiert	exists \$person{Alter} Prüft z. B. in einer if-Abfrage, ob im Hash %person ein Element mit dem Schlüssel Alter vorhanden ist; liefert true oder false als Ergebnis
each	Liefert das nächste Schlüssel-Wert-Paar als Liste	(\$schluessel, \$wert) = each %person Liefert z. B. in einer while-Schleife das nächste Schlüssel-Wert-Paar von %person

6.5 Arrays und Hashes in Schleifen durchlaufen

Um alle Elemente eines Arrays oder Hashes in einer Schleife zu durchlaufen, können Sie eine `for`-Anweisung verwenden. Da dabei jedoch eine Zählervariable mit einer Bedingung verwendet wird, ist der Einsatz recht aufwändig.

Eine elegantere Lösung ist die `foreach`-Schleife, die nacheinander alle Elemente eines Arrays oder, in Kombination mit der `keys`-Funktion, alle Elemente eines Hashes liefert.

Syntax der `foreach`-Schleife bei Arrays

```
foreach $variable (@arrayvariable) {
    Anweisungsblock;
}
```

- ✓ Die Schleife beginnt mit dem Schlüsselwort `foreach`. Danach folgt die Angabe einer skalaren Variablen, die nacheinander die einzelnen Elemente des Arrays aufnimmt.
- ✓ In runden Klammern wird der Name des Arrays angegeben.
- ✓ Danach folgt ein Anweisungsblock in runden Klammern, der für jedes Element ausgeführt wird.

Syntax der `foreach`-Schleife bei Hashes

```
foreach $schluessel (keys %hashvariable) {
    Anweisungsblock;
}
```

- ✓ Die Syntax ist die gleiche wie bei der Verwendung mit Arrays.
- ✓ Vor dem Namen des Hashes wird zusätzlich die `keys`-Funktion aufgerufen. Sie liefert ein Array der Schlüsselwörter.
- ✓ Die einzelnen Elemente werden im Anweisungsblock mit der Syntax `$hashvariable{$schluessel}` angesprochen.

Indem Sie zusätzlich die `sort`-Funktion verwenden, können Sie ein Hash auch sortiert durchlaufen:



```
foreach $schluessel (sort keys %hashvariable) {
    Anweisungsblock;
}
```

Beispiel: *foreach.pl*

Die Angaben einer Person werden im Hash `%person` gespeichert. Mit einer `foreach`-Schleife werden alle Informationen ausgegeben.

```
#!/usr/bin/perl
use strict;

my %person = ( "Vorname"      , "Frank",
               "Name"        , "Meier",
               "Strasse"     , "Waldstr. 5",
               "Postleitzahl" , "64511",
               "Ort"         , "Braundorf",
               "Alter"       , 27);

① foreach my $schluessel (keys %person) {
②   print "$schluessel: $person{$schluessel}\n";
}
```

- ① Die Schleife durchläuft alle Elemente des Hashes und liefert jeweils den Schlüssel. Dazu wird mit der `keys`-Funktion ein Array der Schlüsselwerte erzeugt.
- ② Der Schlüssel und der zugehörige Wert werden angezeigt.



Wenn Sie nicht mit `use strict;` arbeiten, können Sie bei dem Hash die Schlüssel auch ohne String-Notation notieren (beispielsweise einfach `Vorname` statt `"Vorname"`). Der strikte Modus erzwingt jedoch, dass die Schlüssel als Strings notiert werden.

Kurzform der `foreach`-Schleife

Perl bietet viele Möglichkeiten, um den Programmcode so knapp wie möglich zu halten. So kann in der `foreach`-Schleife die Angabe der skalaren Variable entfallen. Perl speichert den Wert dann in der implizierten Variable `$_`.

Um beispielsweise den Wert aller Elemente eines Arrays auszugeben, können Sie folgende Syntax verwenden:

```
foreach (@arrayvariable) {
  print $_;
}
```



Die vordefinierte Variable `$_` ist eine Besonderheit von Perl. Sie können sie auch in anderen Schleifen oder Anweisungsblöcken verwenden, um eine skalare Variable zu ersetzen. Grundsätzlich müssen Sie dabei folgende Regel beachten: Wenn Sie beim Aufruf einer Funktion oder Angabe einer Anweisung keine Variable zum Speichern des Ergebnisses angeben, verwendet Perl stattdessen die Variable `$_`. Im Gleichzug können Sie auch einen erforderlichen Parameter beim Aufruf einer Funktion, z. B. `print`, weglassen. Perl greift auch in diesem Fall auf die Variable `$_` zurück.

Die kürzeste Schreibweise der `foreach`-Schleife lautet daher folgendermaßen:

```
foreach (@arrayvariable) { print; }
```

Perl durchläuft in diesem Fall das Array und gibt jedes Element am Bildschirm aus.

6.6 Schnellübersichten

Was bedeutet ...	
skalare Variable	Variable, die einen einzelnen Wert speichern kann
Liste	Anzahl geordneter skalarer Variablen
Array	Datenfeld, auf dessen Elemente über einen numerischen Index zugegriffen wird
Hash	Datenfeld, auf dessen Elemente über einen alphanumerischen Schlüsselwert zugegriffen wird

Sie möchten ...	
ein Array mit einer Liste initialisieren	<code>@arrayvariable = (Wert1, Wert2, ...);</code>
ein Intervall in einer Liste angeben	<code>@arrayvariable = (Wert1 .. Wert2);</code>
auf ein Array-Element zugreifen	<code>\$arrayvariable[Index]</code>
Zeichenketten in einem Array speichern	<code>@arrayvariable = qw (Wert1 Wert2 ...);</code>
ein Array über die Tastatur einlesen	<code>@arrayvariable = <STDIN></code>
den höchsten Indexwert eines Arrays ermitteln	<code>\$#arrayvariable</code>
die Anzahl der Elemente in einem Array ermitteln	<code>\$anzahl = @arrayvariable;</code>
ein Hash mit Schlüssel-Wert-Paaren füllen	<code>%hashvariable = (Schlüssel1 => Wert1, Schlüssel2 => Wert2, ...)</code>
auf ein Hash-Element zugreifen	<code>\$hashvariable{Schlüssel}</code>
die Anzahl der Elemente in einem Hash ermitteln	<code>\$anzahl = keys(%hashvariable);</code>
alle Elemente eines Arrays durchlaufen	<code>foreach \$element (@arrayvariable) { ... }</code>
alle Elemente eines Hashes durchlaufen	<code>foreach \$element (keys %hashvariable) { ... }</code>

6.7 Übungen

Grundlagen zu Datenfeldern

Übungsdatei: --**Ergebnisdatei:** --

1. Erläutern Sie die Unterschiede zwischen einem Array und einem Hash. Wie werden jeweils die einzelnen Elemente angesprochen?
2. Was ist eine Liste?
3. Wie können Sie in einer Liste ein Intervall angeben?
4. Mit welchem Zeichen werden Arrays, Hashes und skalare Variablen gekennzeichnet?
5. Welche Daten können in einer Liste verwendet werden?
6. Mit welchen Schleifen können Sie Arrays und Hashes durchlaufen? Geben Sie die grundsätzliche Syntax der Schleifen an.

Arrays einsetzen

Übungsdatei: --**Ergebnisdatei:** *Übung2-E.pl*

1. Erstellen Sie ein Programm, das alle Vornamen, die Sie per Tastatur eingeben haben, sortiert ausgibt.
2. Verwenden Sie eine geeignete Schleife, um das Einlesen der Daten von der Tastatur und das Speichern in einem Datenfeld zu realisieren.
3. Löschen Sie bei jedem eingegebenen Namen das Zeilenvorschubzeichen.
4. Die Schleife soll beendet werden, wenn kein Name eingegeben wurde.
5. Achten Sie darauf, dass das Datenfeld kein leeres Element enthält. Entfernen Sie dazu das letzte Element des Arrays mit einer geeigneten Funktion.
6. Geben Sie alle eingegebenen Namen aufsteigend sortiert aus.

Mit Datenfeldern arbeiten

Übungsdatei: --

Ergebnisdatei: Übung3-E.pl

1. Erstellen Sie ein Programm, das für verschiedene deutsche Städte die Telefonvorwahl in einem Datenfeld speichert und nach Eingabe eines Ortes anzeigt.
2. Speichern Sie die folgenden Orte mit der entsprechenden Vorwahl in einem assoziativen Feld.

Ort	Vorwahl		Ort	Vorwahl
Berlin	030		Frankfurt	069
Hamburg	040		Leipzig	0341
München	089		Düsseldorf	0211
Stuttgart	0711		Köln	0221
Rostock	0381		Dresden	0351
Wiesbaden	0611		Hannover	0511
Erfurt	0361			

3. Geben Sie zuerst informativ alle gespeicherten Orte in alphabetischer Reihenfolge am Bildschirm aus. Trennen Sie die einzelnen Elemente mit einem Leerzeichen.
4. Realisieren Sie eine Abfrage, die eine Ortsangabe von der Tastatur einliest. Entfernen Sie den Zeilenvorschub am Ende der Eingabe.
5. Überprüfen Sie, ob eine Vorwahl für den eingegebenen Ort verfügbar ist. Ist dies der Fall, zeigen Sie den Ortsnamen und die entsprechende Vorwahl an. Andernfalls zeigen Sie eine Fehlermeldung an.
6. Ermöglichen Sie eine wiederholte Vorwahlsuche, indem Sie eine Abfrage in eine weitere Vorwahlsuche einbinden.

```
F:\PerlProgramme\Uebungsdateien\Kapitel_06>perl Übung3-E.pl
Die Vorwahlen folgender Städte sind gespeichert:
Berlin Dresden Düsseldorf Erfurt Frankfurt Hamburg Hannover Köln Leipzig München
Rostock Stuttgart Wiesbaden

Bitte gewünschten Ort eingeben: Berlin

Die Vorwahl für Berlin lautet 030.
Weitere Vorwahl suchen? (ja/nein) ja

Bitte gewünschten Ort eingeben: Wiesbaden

Die Vorwahl für Wiesbaden lautet 0611.
Weitere Vorwahl suchen? (ja/nein) _
```

Mögliche Ausgabe der Vorwahlsuche

7 Unterprogramme und Funktionen

In diesem Kapitel erfahren Sie

- ✓ wie Sie Unterprogramme erstellen und verwenden
- ✓ wie Sie Argumente an Unterprogramme übergeben und Ergebnisse zurückliefern
- ✓ wie Sie Referenzen auf Variablen erstellen
- ✓ welche internen Perl-Funktionen es gibt und wie Sie diese verwenden

Voraussetzungen

- ✓ Arbeiten mit Variablen, Ausdrücken und Kontrollstrukturen

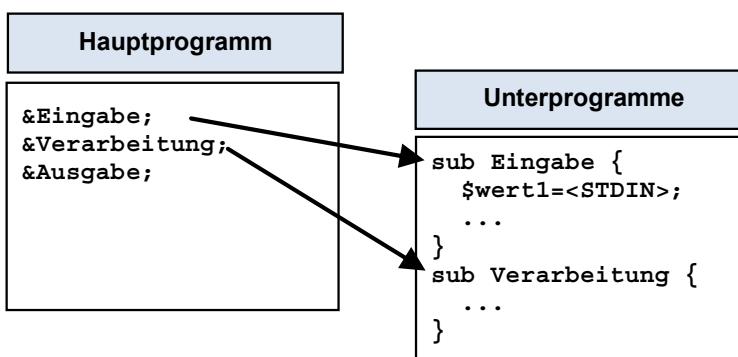
7.1 Unterprogramme in Perl

Unterprogramme zum Strukturieren eines Programms

Unterprogramme bzw. **Subroutinen** sind Anweisungsblöcke, die Sie vom Hauptprogramm aus aufrufen können. Besonders bei sehr umfangreichen und komplexen Programmen ist es sinnvoll, Anweisungsblöcke in eigene Unterprogramme zu strukturieren und den Programmverlauf übersichtlicher zu gestalten.

Beispiel

Ein Perl-Programm, das verschiedene Benutzereingaben erwartet, die eingegebenen Werte verarbeitet und danach ausgibt, kann in drei sinnvolle Blöcke gegliedert werden. Für jeden der Blöcke Eingabe, Verarbeitung und Ausgabe können Sie ein eigenes Unterprogramm erstellen, um alle zusammengehörenden Anweisungen zu gruppieren.



Gliedern eines Programms in Unterprogramme



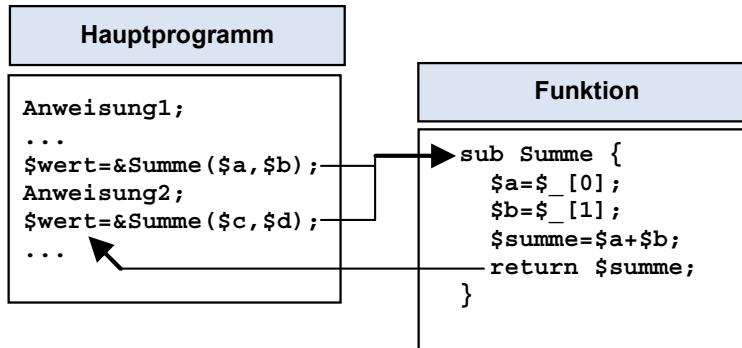
Alle Unterprogramme müssen im Programmcode explizit aufgerufen werden. Erfolgt kein Aufruf, werden die Anweisungen des Unterprogramms auch nicht ausgeführt.

Unterprogramme mit Parameterübergabe

Einem Unterprogramm können Sie beim Aufruf Variablenwerte oder Konstanten als **Argumente** (auch **Parameter** genannt) übergeben. Die Werte können im Unterprogramm verarbeitet werden. Als Resultat kann das Unterprogramm ein Ergebnis zurückliefern. In diesem Fall wird das Unterprogramm auch Funktion genannt, wobei in Perl der Begriff „Funktion“ in der Regel nur für eingebaute Unterprogramme der Umgebung benutzt wird.

Im Programmverlauf ist der Einsatz einer Funktion immer dann notwendig bzw. sinnvoll, wenn ein Anweisungsblock an mehreren Programmstellen mit verschiedenen Werten bearbeitet werden soll. Statt den Anweisungsblock immer wieder in den Programmcode zu schreiben, können Sie ihn in ein Unterprogramm kapseln und beliebig oft aufrufen.

Beim Einsatz von Unterprogrammen mit Argumenten können Sie Werte übergeben, ohne globale Variablen und Konstanten zu verwenden. Gleichzeitig können die in einem Unterprogramm zusammengefassten Anweisungen beliebig oft aufgerufen werden, ohne mehrfach im Programm vorhanden zu sein. Der Programmcode wird dadurch übersichtlicher und leichter nachvollziehbar.



Aufruf eines Unterprogramms mit Parametern und einem Rückgabewert

Früher wurde in einigen Programmiersprachen zwischen Funktionen (liefern ein Ergebnis zurück) und Prozeduren (liefern kein Ergebnis) unterschieden. Beide Formen werden in Perl auf die gleiche Weise als Unterprogramme definiert und verwendet. Es steht Ihnen in Perl frei, mit dem Schlüsselwort `return` einen Wert zurückzuliefern oder nicht.

Perl verfügt bereits über eine Anzahl vordefinierter Funktionen, beispielsweise `chomp()`. Genauso wie bei eigenen Funktionen erwarten diese eine Anzahl Argumente und liefern meist ein Ergebnis zurück. Im Gegensatz zu selbst erstellten Unterprogrammen wird ihnen beim Aufruf **nicht** das Zeichen `&` vorgestellt.

7.2 Unterprogramme erstellen und verwenden

Bei der Deklaration geben Sie Perl den Namen des Unterprogramms bekannt und erstellen die Anweisungen, die innerhalb des Unterprogramms bearbeitet werden sollen.

Syntax der Deklaration von Unterprogrammen

```
sub UnterprogrammName {
    Anweisung;
    return Wert;
}
```

- ✓ Unterprogramme werden mit dem Schlüsselwort `sub` gekennzeichnet.
- ✓ Danach folgt der Name des Unterprogramms. Mit diesem Namen wird das Unterprogramm aufgerufen.
- ✓ In geschweiften Klammern wird der Anweisungsblock des Unterprogramms definiert.
- ✓ Soll das Unterprogramm einen Wert zurückliefern, muss er mit dem Schlüsselwort `return` festgelegt werden. Der Rückgabewert kann eine Variable, das Ergebnis eines Ausdrucks oder ein konstanter Wert sein.

Für die Vergabe von Namen für Unterprogramme gelten die gleichen Regeln wie für Variablen. Sie können nur Buchstaben, Zahlen und den Unterstrich verwenden. Unterprogrammnamen müssen mit einem Buchstaben beginnen.

Variablen in Unterprogrammen

Wenn Sie in einem Unterprogramm Variablen verwenden, werden diese, wie in Perl üblich, als globale Variablen deklariert. Oft sollen die Variablen jedoch ausschließlich innerhalb des Unterprogramms verfügbar sein. Um dieses Verhalten zu erreichen, können Sie die Variablen mit den Schlüsselwörtern `local` oder `my` als lokale Variablen kennzeichnen. So vermeiden Sie auch Überschneidungen mit gleichnamigen Variablen im Hauptprogramm.



Erinnern Sie sich an die Unterschiede zwischen `local` und `my`. Mit `local` wird eine globale Variable nur lokal verdeckt, mit `my` wird eine neue Variable angelegt.

Syntax des Aufrufs von Unterprogrammen

<code>&UnterprogrammName;</code>	<code>\$Variable = &UnterprogrammName;</code>
--------------------------------------	---

- ✓ Beim Aufruf des Unterprogramms wird dem Namen das Zeichen `&` vorangestellt.
- ✓ Liefert das Unterprogramm einen Wert zurück, dann kann er einer Variablen zugewiesen oder in einem Ausdruck verwendet werden.
- ✓ Der Aufruf eines Unterprogramms ist eine eigene Anweisung und muss mit einem Semikolon `;` abgeschlossen werden.



Der Aufruf von Unterprogrammen kann auch vor der Deklaration des Unterprogramms erfolgen. Dies ist möglich, da Perl zuerst den Quelltext des Programms vollständig durchläuft und in einen Zwischencode übersetzt. Sie müssen daher ein Unterprogramm nicht zwingend vor der ersten Verwendung definieren.

Beispiel: Volumen.pl

Das folgende Perl-Programm soll das Volumen eines Quaders, z. B. eines Raumes, berechnen. Die Eingabe der Maße, die Berechnung und die Ausgabe werden jeweils in einem Unterprogramm bearbeitet.

```
#!/usr/bin/perl
use strict;
my $werte;
my @werte;
my $volumen;
① &Eingabe;
&Berechnung;
&Ausgabe;
② sub Eingabe {
    print "Volumen-Berechnung eines Quaders\n\n";
    print "Bitte geben Sie Höhe, Breite und Tiefe in m ein. ";
    print "Trennen Sie die Werte durch ein Leerzeichen.\n";
    $werte=<STDIN>;
    chomp($werte);
    @werte=split(/ /,$werte);
}
③ sub Berechnung {
    $volumen=$werte[0]*$werte[1]*$werte[2];
}
④ sub Ausgabe {
    print "Das Volumen beträgt $volumen m³\n\n";
}
```

- ① Im Hauptteil des Programms erfolgt der Aufruf der drei Unterprogramme.

```
F:\PerlProgramme\Kapitel_07>perl Volumen.pl
Volumen-Berechnung eines Quaders
Bitte geben Sie Höhe, Breite und Tiefe in m ein. Trennen Sie die Werte durch ein Leerzeichen.
5 6 7
Das Volumen beträgt 210 m³

F:\PerlProgramme\Kapitel_07>_
```

Berechnung eines Volumens mit Unterprogrammen

- ② Im ersten Unterprogramm erfolgt die Eingabe der Werte über die Tastatur. Dazu wird eine Zeile von der Standardeingabe gelesen.
- ③ Die Werte sollen mit einem Leerzeichen getrennt eingegeben werden. Die `split`-Funktion teilt die Zeichenkette `$werte` an den Leerzeichen, um ein Array mit den einzelnen Werten zu erhalten.
- ④ In diesem Unterprogramm erfolgt die Berechnung, indem alle drei Werte des Arrays multipliziert werden.
- ⑤ Auch die Ausgabe wird in einem eigenen Unterprogramm vorgenommen.

7.3 Übergabe von Argumenten und Rückgabe von Werten

Argumente übergeben

Argumente sind Parameter, die Sie beim Aufruf eines Unterprogramms übergeben und innerhalb des Unterprogramms verarbeiten können. Die Argumente werden in runde Klammern eingeschlossen und an den Namen des Unterprogramms angeschlossen. In vielen Programmiersprachen erfolgt die Definition der übergebenen Argumente beim Deklarieren des Unterprogramms. In Perl werden die Argumente jedoch stets mithilfe des vordefinierten Arrays `@_` zur Verfügung gestellt.

Syntax des Aufrufs von Unterprogrammen mit Argumenten

<code>&UnterprogrammName (Argument1, ...);</code>	<code>&UnterprogrammName Argument1, ...;</code>
---	---

- ✓ Beim Aufruf des Unterprogramms wird dem Namen das Zeichen `&` vorangestellt.
- ✓ Die Argumente werden durch Kommata getrennt nach dem Namen des Unterprogramms angegeben.
- ✓ Die Argumentenliste sollte in runde Klammern eingeschlossen werden. Die Klammern können jedoch auch entfallen.

Durch die besondere Art der Parameterübergabe kennt Perl keine benannten Argumente. Den ersten übergebenen Parameter erhalten Sie über die Variable `$_[0]`, den zweiten über `$_[1]` usw. Dies hat jedoch den Vorteil, dass Sie die Parameterübergabe sehr flexibel halten können. Ein Unterprogramm kann so einmal zwei und ein anderes Mal vier Argumente erhalten.

Wenn Sie einem Unterprogramm weniger Argumente übergeben, als erforderlich ist, erhalten Sie in Perl keine Fehlermeldung. Die fehlenden Argumente sind in diesem Fall nicht definiert. Bei der Programmierung der Funktion müssen Sie dies beachten.



Rückgabewert

Wenn nötig, können Sie in einer Funktion einen Rückgabewert definieren. Dies kann beispielsweise das Resultat einer Berechnung oder ein Wahrheitswert sein als Variable oder konstanter Wert. Der Rückgabewert wird mit dem Schlüsselwort `return` festgelegt. Sie können skalare Variablen, aber auch Arrays oder Hashes zurückgeben.

Das Schlüsselwort `return` hat eine doppelte Funktion: Neben dem Festlegen eines Rückgabewerts beendet es auch gleichzeitig das Unterprogramm.



Sie können die `return`-Anweisung auch weglassen. In diesem Fall nimmt Perl den Wert der zuletzt ausgeführten Anweisung als Rückgabewert.

Beispiel

```
sub Summe {
    $a=10;
    $b=$a+5;    # Letzte Anweisung, Rückgabewert des Unterprogramms ist 15
}
```

oder

```
sub Summe {
    $a=10;
    $b=$a+5;
    return $b; # Rückgabewert ist ebenfalls 15
}
```



Vor allem, wenn das Unterprogramm Verzweigungen enthält, muss die zuletzt ausgeführte Anweisung nicht die letzte Anweisung im Anweisungsblock sein. Verwenden Sie im Zweifelsfall besser die `return`-Anweisung.

Vorteile von Unterprogrammen mit Argumenten und Rückgabewert

Der Vorteil von Unterprogrammen liegt vor allem im Kapseln der Anweisungen und der Definition fester Schnittstellen. Dies erleichtert insbesondere die Programmentwicklung mit mehreren Programmierern oder das Verwenden von Unterprogrammen in mehreren Projekten.

Werden für ein Unterprogramm die zu übergebenden Argumente und der Rückgabewert festgelegt, können die Anweisungen im Unterprogramm geändert werden, ohne dass andere Programmteile betroffen sind.

Beispiel: *KleinsteZahl.pl*

Das Perl-Programm soll mithilfe einer Funktion die kleinste von einer beliebigen Anzahl Zahlen bestimmen. Die Zahlen werden, mit Leerzeichen getrennt, über die Tastatur eingegeben. Die eingegebenen Werte werden dem Unterprogramm als Argument übergeben. Als Ergebnis liefert es die kleinste Zahl.

```
#!/usr/bin/perl
use strict;
my $werte;

&Eingabe;
&Ausgabe;

① sub Eingabe {
    print "Geben Sie eine beliebige Anzahl Zahlen ein.\n\n";
    print "Trennen Sie die Zahlen jeweils durch ein Leerzeichen.\n";
    $werte=<STDIN>;
    chomp($werte);
}
```

```

sub Ausgabe {
    print "Die kleinste Zahl ist ".&Kleinste($werte);
}

sub Kleinste {
    my $kleinste;
    my @werte;
    @werte=split(/ /,$_[0]);
    foreach (@werte) {
        if (!$kleinste) {
            $kleinste=$_;
        } elsif ($_<$kleinste) {
            $kleinste=$_;
        }
    }
    return $kleinste;
}

```

- ① Die Eingabe und Ausgabe der Werte wird in eigenen Unterprogrammen organisiert.
- ② Hier erfolgt der Aufruf des Unterprogramms &Kleinste mit der eingegebenen Zeichenkette als Argument. Das Unterprogramm liefert ein Ergebnis, das innerhalb der print-Anweisung ausgegeben wird.
- ③ An dieser Stelle beginnt die Deklaration des Unterprogramms &Kleinste.
- ④ Die Variablen \$kleinste und @werte werden als lokale Variablen deklariert.
- ⑤ Der Zugriff auf das übergebene Argument erfolgt mit dem Arrayelement \$_[0]. Die Zahlen in der Zeichenkette werden anhand des Leerzeichens aufgeteilt.
- ⑥ Nacheinander werden die eingegebenen Zahlen im Array @werte durchlaufen. Da keine skalare Variable in der foreach-Schleife angegeben wird, kann auf das aktuelle Element mit der Variable \$_ zugegriffen werden.
- ⑦ Ist die Variable \$kleinste noch nicht definiert (dies ist beim ersten Durchlauf der Fall), erhält sie den Wert des aktuellen Arrayelements.
- ⑧ Ist der Wert des aktuellen Arrayelements kleiner als die Variable \$kleinste, dann wird dieser Wert in der Variablen gespeichert. Als Resultat enthält die Variable immer den bisher kleinsten Zahlenwert.
- ⑨ Mit dem Schlüsselwort return wird der Wert der Variablen \$kleinste als Ergebnis zurückgegeben.

```

F:\PerlProgramme\Kapitel_07>perl KleinsteZahl.pl
Geben Sie eine beliebige Anzahl Zahlen ein.
Trennen Sie die Zahlen jeweils durch ein Leerzeichen.
3 4 8 99 12 3 1
Die kleinste Zahl ist 1
F:\PerlProgramme\Kapitel_07>_

```

Ermitteln der kleinsten Zahl einer Reihe

Rückgabewert und Ausführungskontext

In Perl ist es möglich, dass ein Unterprogramm mehrere unterschiedliche Werte zurückgeben kann, je nachdem, ob es im skalaren oder im Listenkontext aufgerufen wird. Ein skalarer Kontext liegt vor, wenn das Ergebnis eines Unterprogramms einer skalaren Variablen zugewiesen wird. Soll das Ergebnis dagegen in einem Array gespeichert werden, liegt ein Listenkontext vor.

Mit der Perl-Funktion wantarray, die den Wahrheitswert true oder false liefert, können Sie prüfen, ob ein Listenkontext vorliegt.

Beispiel: *Zerlegung.pl*

In einem Unterprogramm soll eine Zeichenkette mit einer Dezimalzahl anhand des Punktes als Dezimaltrennzeichen zerlegt werden. Je nach Ausführungskontext liefert das Unterprogramm entweder nur den ganzzahligen Teil als skalaren Wert oder beide Teile als Array.

```
#!/usr/bin/perl
use strict;

① my $ganze_zahl=&Zerlegung("24.75");
print "Der ganzzahlige Teil ist: $ganze_zahl\n";

② my @zahl_teile=&Zerlegung("24.75");
print "Der ganzzahlige Teil ist: $zahl_teile[0]\n";
print "Der gebrochene Teil ist: $zahl_teile[1]\n";

sub Zerlegung {
    my @teile=split(/\./,$_[0]);
    ③ wantarray ? ($teile[0],$teile[1]) : $teile[0];
}
```

- ① Zuerst wird das Unterprogramm im skalaren Kontext ausgeführt. Dabei wird das Ergebnis einer skalaren Variablen zugewiesen.
- ② Hier erfolgt der Aufruf im Listenkontext, denn das Ergebnis des Unterprogramms wird einem Array zugewiesen.
- ③ Die übergebene Zeichenkette wird am Punkt zerlegt. Da der Punkt in Perl als Platzhalterzeichen gilt, muss er durch einen vorangestellten Backslash als Escape-Zeichen angegeben werden.
- ④ Die Anweisung `wantarray` liefert einen Wahrheitswert, auf den mit dem Fragezeichenoperator reagiert wird. Liefert die Anweisung `true`, wird eine Liste zurückgegeben, anderenfalls ein skalarer Wert.

```
F:\PerlProgramme\Kapitel_07>perl Zerlegung.pl
Der ganzzahlige Teil ist: 24
Der gebrochene Teil ist: 75
F:\PerlProgramme\Kapitel_07>
```

Zerlegung einer Dezimalzahl

7.4 Referenzen verwenden

Arrays als Argumente von Unterprogrammen

Wenn Sie ein Array als Argument übergeben, verliert es seine Eigenständigkeit. Es wird in seine Elemente aufgegliedert, auf die Sie über das vordefinierte Array `@_` innerhalb des Unterprogramms zugreifen können. Vor allem, wenn Sie skalare Variablen und Arrays gleichzeitig als Argumente übergeben wollen, ist dies von Bedeutung.

Beispiel

```
&Summe ($zahl1, @zahlen, $zahl2);
```

Innerhalb des Unterprogramms enthält das Array `@_` die Werte in folgender Reihenfolge:

```
@_=($zahl1,$zahlen[0],$zahlen[1],...,$zahlen[n],$zahl2);
```

Soll das Array erhalten bleiben, müssen Sie es als Referenz, d. h. als Zeiger auf einen Speicherplatz, übergeben.

Beispiel

```
&Summe ($zahl1, \@zahlen, $zahl2);
```

In diesem Fall wird eine Referenz auf das Array @zahlen übergeben:

```
$zahl1=$_[0];
$zahlen=@{$_[1]};
$zahl2=$_[2];
```

Referenzen auf Variablen erstellen

Referenzen auf eine Variable erstellen Sie durch das Voranstellen eines Backslashes \ vor den Variablennamen. Sie können Referenzen auf skalare Variablen, Arrays, Hashes oder andere Referenzen erstellen. Die Referenz selbst ist ein skalarer Wert und kann in einer entsprechenden Variablen oder in einem Arrayelement gespeichert werden.

Beispiel	Erklärung
\$referenz = \\$wert;	Referenz auf eine skalare Variable
\$referenz = \@werte;	Referenz auf ein Array
\$referenz = \%werte;	Referenz auf ein Hash

Dereferenzierung

Eine Referenz enthält nur die Speicheradresse einer Variablen. Um wieder auf den eigentlichen Wert der Variablen zugreifen zu können, müssen Sie eine Dereferenzierung durchführen. Dabei wird die Speicherreferenz aufgelöst.

Bei der Dereferenzierung wird der Referenzvariablen das ursprüngliche Kennzeichen des Datentyps (\$, @ oder %) vorangestellt.

Beispiel	Erklärung
\$referenz = \\$wert; print \$\$referenz;	Referenz auf eine skalare Variable Ausgabe des ursprünglichen Wertes
\$referenz = \@werte; @array = @{\$referenz}; print \$\$referenz[0];	Referenz auf ein Array Zugriff auf das ursprüngliche Array Ausgabe eines einzelnen Elements
\$referenz = \%werte; %hash = %\$referenz;	Referenz auf ein Hash Zugriff auf den ursprünglichen Hash

Eine alternative Schreibweise für die Dereferenzierung ist die Blockschreibweise. Dazu wird die Referenzvariable in geschweifte Klammern gesetzt. Davor wird wiederum das Symbol des ursprünglichen Datentyps gesetzt.

```
$referenz = \@werte;      # Speichern einer Referenz auf das Array @werte
$array = @{$referenz};    # Ursprüngliches Array an eine neue Variable übergeben

$variable = @{$referenz}[2];  # Zugriff auf ein einzelnes Arrayelement
```

Für den Zugriff auf einzelne Arrayelemente existiert in Perl eine weitere elegante Schreibweise:

```
$referenz = \@werte;          # Speichern einer Referenz auf das Array @werte
$variable = $referenz->[2];   # Zugriff auf das Element mit dem Index 2
$referenz->[1] = 15;         # Wertzuweisung an das Element mit dem Index 1
```

Referenzen auf Hashes können Sie auf die gleiche Weise verwenden wie Referenzen auf Arrays. Auch der Zugriff auf einzelne Elemente erfolgt in dieser Form, dann jedoch mit Schlüsselwörtern statt Indexwerten und geschweiften statt eckigen Klammern:

```
$referenz = \%einwohner;
$variable = @{$referenz}{ "Berlin" };
$variable = $referenz->{ "Berlin" };
```



Mit Referenzen können Sie sehr komplizierte Datenstrukturen schaffen. So können beispielsweise Elemente eines Arrays Referenzen auf Hashes enthalten. Die Referenzen können dabei beliebig verschachtelt werden.

7.5 Vordefinierte Perl-Funktionen

Neben den Unterprogrammen, die Sie selbst erstellen, enthält Perl bereits eine große Zahl vordefinierter Funktionen. Die Perl-internen Funktionen sind zu jeder Zeit verfügbar und können an allen Stellen des Programm-codes aufgerufen werden. Eine der wichtigsten dieser Funktionen ist `print`.

Syntax des Aufrufs von Perl-internen Funktionen

funktionsname (Argumente) ;	<code>\$variable = funktionsname (Argumente) ;</code>
-----------------------------	---

- ✓ Der Aufruf von internen Funktionen erfolgt wie bei selbst erstellten Unterprogrammen.
- ✓ Vor dem Funktionsnamen wird **kein** `@`-Zeichen angegeben.
- ✓ Falls ein Ergebnis zurückgeliefert wird, kann es einer Variablen zugewiesen oder in einem Ausdruck verwendet werden.



Da Sie beim Aufruf der Funktion die runden Klammern nicht angeben müssen, ist oft nicht mehr ersichtlich, dass es sich tatsächlich um eine Funktion handelt. Es ist dringend zu empfehlen, bei Funktionsaufrufen die runden Klammern zu notieren.

Die folgenden Tabellen zeigen eine Auswahl der wichtigsten internen Funktionen. Eine ausführliche Auflistung finden Sie in der Perl-Dokumentation `perfunc`.

Funktionen für Zeichenketten

Funktion	Erklärung	Beispiel	Ergebnis
<code>chomp (\$str) ;</code>	Löscht den Zeilenumbruch <code>\n</code> am Ende der Zeichenkette	<code>chomp ("Welt\n") ;</code>	"Welt"
<code>chop (\$str) ;</code>	Löscht das letzte Zeichen einer Zeichenkette	<code>chomp ("Welt") ;</code>	"wel"
<code>chr (\$wert) ;</code>	Liefert das Zeichen für einen ASCII-Wert	<code>chr(65) ;</code>	"A"

Funktion	Erklärung	Beispiel	Ergebnis
<code>index(\$str,\$ss);</code>	Liefert die Position der Zeichenkette \$ss in der Zeichenkette \$str	<code>index("Hallo Welt","Welt");</code>	6
<code>lc(\$str);</code> <code>lcfirst(\$str);</code>	Umwandlung in Kleinbuchstaben Nur erstes Zeichen umwandeln	<code>lc("Welt");</code>	"welt"
<code>uc(\$str);</code>	Umwandlung in Großbuchstaben	<code>uc("Welt");</code>	"WELT"
<code>ucfirst(\$str);</code>	Nur erstes Zeichen umwandeln		
<code>length(\$str);</code>	Ermittelt die Länge der Zeichenkette	<code>length("Welt");</code>	4
<code>sprintf("Format", \$variable);</code>	Formatierte Ausgabe von Werten	<code>sprintf("%.2f",23.5);</code>	23.50
<code>substr(\$str, \$beginn, \$laenge);</code>	Liefert einen Teil der Zeichenkette \$str beginnend ab \$beginn und mit einer optionalen Länge von \$laenge	<code>substr("Hallo Welt",6);</code>	"Welt"
<code>split(//,\$str);</code>	Wandelt eine Zeichenkette anhand eines Trennzeichens in ein Array, erstes Argument ist ein regulärer Ausdruck für das Trennzeichen	<code>split(/ /, "Hallo Welt");</code>	"Hallo" "Welt"
<code>join(\$z,@array);</code>	Verbindet die Elemente eines Arrays mit dem Zeichen \$z zu einer Zeichenkette	<code>join(" ",@array);</code>	"Hallo Welt"

Funktionen für Zahlen

Funktion	Erklärung	Beispiel	Ergebnis
<code>abs(\$zahl)</code>	Liefert den Absolutwert einer Zahl	<code>abs(-2)</code>	2
<code>cos(\$zahl)</code> <code>sin(\$zahl)</code>	Winkelfunktionen	<code>sin(0.5)</code>	0.47942
<code>exp(\$zahl)</code>	Exponentialfunktion	<code>exp(0)</code>	1
<code>int(\$zahl)</code>	Liefert den ganzzahligen Teil einer Dezimalzahl	<code>int(24.75)</code>	24
<code>rand(\$zahl)</code>	Liefert eine Zufallszahl, die kleiner ist als \$zahl	<code>rand(6)</code>	5.4071
<code>srand(\$zahl)</code>	Initialisiert den Zufallszahlengenerator	<code>srand(time ^ \$\$)</code>	

Funktionen für Datum und Uhrzeit

Funktion	Erklärung	Beispiel	Ergebnis
<code>time()</code>	Ermittelt die aktuelle Zeit im Unix-Format (Anzahl der Millisekunden seit dem 01.01.1970)	<code>time()</code>	1016627068
<code>localtime(\$zeit)</code>	Konvertiert eine Zeitangabe im Unix-Format in eine Zeichenkette oder ein Array	<code>localtime(time())</code>	Wed Mar 20 10:10:45 2002



Die Funktion für das Dateisystem und für die Arbeit mit Arrays oder Hashes werden in den entsprechenden Kapiteln aufgeführt.

7.6 Schnellübersicht

Sie möchten ...	
ein Unterprogramm erstellen	sub UnterprogrammName { ... }
ein Unterprogramm aufrufen	&UnterprogrammName (Argumente);
auf übergebene Argumente zugreifen	@_ bzw. \$_[0], \$_[1] usw.
ein Ergebnis zurückliefern	return \$wert;
den Ausführungskontext prüfen	wantarray
eine Referenz auf eine Variable erstellen	\\$variable, \@array, \%hash
auf den Wert einer referenzierten Variablen zugreifen	\$\$referenz, @\$referenz, %\$referenz
über eine Referenz auf ein einzelnes Element eines Arrays zugreifen	\$\$referenz[Index], \${\$referez}[Index], \$referenz->[Index]

7.7 Übungen

Grundlagen zu Unterprogrammen

Übungsdatei: --

Ergebnisdatei: --

1. Nennen Sie Vorteile für das Verwenden von Unterprogrammen.
2. Über welche vordefinierte Variable erhalten Sie in einem Unterprogramm Zugriff auf die übergebenen Argumente?
3. Was bewirkt die Anweisung `return`?
4. Was sind Referenzen, und wofür können Sie diese einsetzen?
5. Erläutern Sie, wie Sie über eine Referenz auf den ursprünglichen Wert einer Variablen zugreifen können. Nennen Sie verschiedene Möglichkeiten.

Unterprogramme einsetzen

Übungsdatei: --

Ergebnisdatei: *Übung2-E.pl*

1. Erstellen Sie ein Programm, das Zufallszahlen für das Lottospiel 6 aus 49 erzeugt.
2. Initialisieren Sie den Zufallszahlengenerator mit der entsprechenden Funktion.
3. Kapseln Sie die Erzeugung einer einzelnen Lottozahl in einem Unterprogramm `LottoZahl`.
4. Beachten Sie, dass die Funktion zur Erzeugung einer Zufallszahl beim Wert 0 beginnt und Dezimalzahlen liefert.
5. Rufen Sie das Unterprogramm in einer geeigneten Schleife sechsmal auf, und geben Sie das Ergebnis aus.
6. Verändern Sie das Programm so, dass es keine doppelten Zufallszahlen erzeugt. Zu diesem Zweck können Sie die erzeugte Zahl als Schlüsselwert in einem Hash verwenden.
7. Prüfen Sie beim Erzeugen der Zufallszahl, ob die aktuelle Zahl bereits als Schlüsselwert im Hash vorliegt. Ist dies der Fall, soll eine neue Zahl generiert werden. Verwenden Sie dafür eine geeignete Schleife.

Parameter verwenden

Übungsdatei: --

Ergebnisdatei: *Übung3-E.pl*

1. Das Programm soll mithilfe eines Unterprogramms prüfen, ob ein über die Tastatur eingegebener Städtename in einem Array vorhanden ist.
2. Erzeugen Sie ein Array mit verschiedenen Städtenamen.
3. Erstellen Sie eine Tastaturabfrage, um den Namen einer Stadt einzulesen. Denken Sie an das Entfernen des Zeilenvorschubs am Ende der Eingabe.
4. Die Unterfunktion soll zwei Argumente erhalten. Das erste Argument ist die gesuchte Stadt. Als zweites Argument soll das Array mit den Städtenamen übergeben werden. Damit das Array erhalten bleibt, müssen Sie es als Referenz übergeben.
5. Erzeugen Sie im Unterprogramm zwei lokale Variablen für den Namen der Stadt und die Städteliste.
6. Weisen Sie den lokalen Variablen die übergebenen Werte zu. Dereferenzieren Sie dabei den Verweis auf das Array.
7. Durchlaufen Sie in einer Schleife die Elemente des Arrays, und prüfen Sie, ob der Stadtnname vorhanden ist. In diesem Fall soll der Wert 1 (`true`) zurückgegeben werden.
8. Ist die Stadt nicht gefunden worden, soll das Unterprogramm den Wert 0 (`false`) liefern.
9. Bearbeiten Sie den Vergleich so, dass die Groß- und Kleinschreibung keine Rolle spielt.
10. Rufen Sie das Unterprogramm auf, und geben Sie je nach zurückgeliefertem Ergebnis eine entsprechende Meldung aus.

8 Datei- und Verzeichnisfunktionen

In diesem Kapitel erfahren Sie

- ✓ wie Sie Dateien lesen und in Dateien schreiben können
- ✓ wie Sie Statusinformationen über Dateien ermitteln
- ✓ welche Funktionen des Dateisystems Perl unterstützt
- ✓ wie Sie Verzeichnisse auslesen

Voraussetzungen

- ✓ Perl-Funktionen, Variablen verwenden

8.1 Mit Dateien arbeiten

Grundlagen zu Dateisystemen

Um Daten auf einem Computer zu speichern, werden Dateien verwendet. Diese werden in einer Verzeichnisstruktur verwaltet. Die Gesamtheit aus Verzeichnissen und den darin gespeicherten Dateien wird auch **Dateisystem** genannt.

Perl verfügt über verschiedene Funktionen für den Zugriff auf das Dateisystem eines Computers. So können Sie Dateien lesen, in Dateien schreiben, Verzeichnisse auslesen oder statistische Daten über Dateien ermitteln. Der prinzipielle Zugriff ist dabei unter allen Betriebssystemen gleich.



Um Verzeichnisnamen in einer Dateiangabe zu trennen, wird unter Windows normalerweise ein Backslash und unter Unix-artigen Betriebssystemen ein Slash verwendet. Ein Backslash leitet in Zeichenketten jedoch eine Escape-Zeichenfolge ein, z. B. \n. Um dennoch ein Backslash zu verwenden, müssen Sie ihn doppelt schreiben: \\ . Um dies zu umgehen, können Sie in Perl auch unter Windows den Slash verwenden.

Normale Schreibweise	Schreibweise in Perl	Betriebssystem
C:\Perl\html	c:\\perl\\html	Windows
C:\Perl\html	c:/perl/html	Windows
/perl/html	/perl/html	Unix



Beachten Sie, dass unter Unix-artigen Betriebssystemen im Gegensatz zu Windows bei der Schreibweise von Datei- und Verzeichnisnamen zwischen Groß- und Kleinschreibung unterschieden wird.

Dateien öffnen

Um eine Datei auszulesen oder in sie zu schreiben, muss sie zuerst geöffnet werden. Dies geschieht mit der Perl-Funktion `open`. Sie definiert ein so genanntes **Datei-Handle** (Deskriptor), über die im weiteren Programmverlauf der Dateizugriff erfolgt. Konnte die Datei erfolgreich geöffnet werden, liefert die Funktion den Wert `true`.

Syntax der `open`-Funktion

```
open(DATEIHANDLE, $dateiname);
```

- ✓ Die Funktion `open` benötigt zwei Argumente. Das erste ist das Datei-Handle, über das nach dem Ausführen der Funktion auf die Datei zugegriffen werden kann. Das zweite Argument ist der Dateiname.
- ✓ Das Datei-Handle wird ohne Datentypkennzeichen (\$) verwendet und sollte zur besseren Unterscheidung von Variablenamen in Großbuchstaben angegeben werden.
- ✓ Als Dateiname kann eine Zeichenkette oder eine skalare Variable angegeben werden. Sie kann zusätzlich eine Verzeichnisstruktur enthalten.
- ✓ Die Funktion liefert im Erfolgsfall den Wert `true`, sonst den Wert `false`.

Falls sich die zu öffnende Datei im gleichen Verzeichnis wie das Perl-Programm befindet, müssen Sie nur den Dateinamen angeben. Sonst ist der vollständige Dateipfad notwendig.

Da die Funktion einen Wahrheitswert zurückgibt, können Sie z. B. mit einer `if`-Abfrage prüfen, ob das Öffnen erfolgreich war und entsprechend reagieren.



Beispiel

Es soll die Datei `namen.txt` im aktuellen Arbeitsverzeichnis geöffnet werden. War das Öffnen nicht erfolgreich, wird mit einer entsprechenden Meldung darauf reagiert.

```
if (open(DATEI, "namen.txt")) {
    ... # Öffnen erfolgreich, Datei kann bearbeitet werden
} else {
    print "Die Datei namen.txt konnte nicht geöffnet werden.\n";
}
```

Zugriffsmodi verwenden

Normalerweise wird die bei der `open`-Anweisung angegebene Datei zum Lesen geöffnet. In diesem Fall erhalten Sie eine Fehlermeldung, wenn Sie versuchen, in die Datei zu schreiben. Der Zugriffsmodus für die Datei wird durch zusätzliche Zeichen vor dem Dateinamen bestimmt.

<code>open(DATEI, "dateiname.erw");</code>	Datei wird zum Lesen geöffnet. Die Datei muss vorhanden sein.
<code>open(DATEI, "<dateiname.erw");</code>	Datei wird zum Lesen geöffnet. Die Datei muss vorhanden sein.
<code>open(DATEI, ">dateiname.erw");</code>	Datei wird zum Schreiben geöffnet, der Inhalt wird vorher gelöscht. Ist die Datei nicht vorhanden, wird sie erstellt.
<code>open(DATEI, ">>dateiname.erw");</code>	Datei wird zum Anhängen geöffnet, das Schreiben erfolgt am Ende der vorhandenen Datei. Ist die Datei nicht vorhanden, wird sie erstellt.
<code>open(DATEI, "+<dateiname.erw");</code>	Datei wird im Lese-/Schreibmodus geöffnet und muss vorhanden sein.
<code>open(DATEI, " dateiname.erw");</code>	Die Datei ist eine ausführbare Programmdatei. Das Programm wird geöffnet und ausgeführt.

Auf Fehler beim Öffnen einer Datei reagieren

Da vom erfolgreichen Öffnen einer Datei meist der gesamte weitere Programmverlauf abhängt, sollten Sie stets den Rückgabewert der `open`-Anweisung auswerten. Oft kann das Programm im Fall eines Fehlers nicht ausgeführt werden und soll mit einer Meldung beendet werden. Neben der `if`-Struktur bietet Perl für diesen Zweck die Funktionen `die` und `warn`, die zusammen mit einem logischen Operator verwendet werden.

<code>warn</code>	Gibt eine Fehlermeldung aus und setzt den Programmablauf nach der Anweisung fort
<code>die</code>	Gibt eine individuelle Meldung aus und beendet das Programm

Beide Funktionen werden meist über den logischen Or-Operator `||` mit einer Funktion verknüpft, die einen Wahrheitswert zurückliefert.

Beispiel

```
open(DATEI,"datei.txt") || die("Die Datei konnte nicht geöffnet werden.\n");
```

statt

```
if(!open(DATEI,"datei.txt")) {
    print "Die Datei konnte nicht geöffnet werden.\n";
    exit;
}
```



Über die von Perl vordefinierte Variable `$!` können Sie zusätzlich die Meldung der fehlerhaft ausgeführten Funktion anzeigen.

```
open(DATEI,"datei.txt") || die("Die Datei konnte nicht geöffnet
werden.\nFehler: $!\n");
```

Unbenanntes Datei-Handle

Unter bestimmten Voraussetzungen kann es notwendig sein, ein unbenanntes (anonymes) Datei-Handle zu verwenden. Dies ist z. B. dann sinnvoll, wenn Sie ein lokales Datei-Handle benötigen, das ausschließlich innerhalb eines Unterprogramms gültig ist. Standardmäßig sind Datei-Handles global gültig.

Ein unbenanntes Datei-Handle erstellen Sie mit der Funktion `gensym` und weisen es einer skalaren Variablen zu. Damit die `gensym`-Funktion verfügbar ist, müssen Sie am Anfang des Programms mit der Anweisung `use Symbol;` das Modul `Symbol` einbinden.

Beispiel

Mit der Funktion `gensym` wird in einem Unterprogramm ein unbenanntes Datei-Handle erzeugt und in der Variablen `$dateihandle` gespeichert.

```
#!/usr/bin/perl
use strict;
use Symbol;
...
sub DateiBearbeiten {
    my $dateihandle = gensym();
    open ($dateihandle,"beispiel.txt");
    ...
}
```

Dateien schließen

Mit der `close`-Funktion können Sie eine Datei schließen und das Datei-Handle freigeben. Am Ende eines Perl-Programms werden alle offenen Dateien automatisch geschlossen.

Syntax der `close`-Funktion

```
close (DATEIHANDLE) ;
```

- ✓ Die `close`-Funktion benötigt als Argument das Datei-Handle der zu öffnenden Datei.
- ✓ Die Funktion schließt die Datei und gibt das Datei-Handle frei.

8.2 Textdateien lesen

Datei-Handle `STDIN`

Sie kennen bereits die Eingabe von Daten über die Tastatur. Dabei haben Sie Daten aus der Standardeingabe-Datei `STDIN` gelesen. Perl verwaltet die Standardeingabe wie eine Datei im Lesemodus und stellt das entsprechende Datei-Handle automatisch zur Verfügung.

Perl stellt außerdem die Datei-Handles `STDOUT` für die Standardausgabe und `STDERR` für die standardmäßige Fehlerausgabe zur Verfügung.

Wenn Sie Textdateien bearbeiten wollen, können Sie über den Datei-Handler die Datei wie eine Tastatureingabe einlesen.

Dateien zeilenweise lesen

Wenn Sie das Datei-Handle einer skalaren Variablen zuweisen, wird immer eine Zeile der Datei bis zum Zeilenumbruch `\n` eingelesen.

```
$zeile = <DATEI>;
```

Der Dateizeiger wird dabei auf die nächste Zeile gesetzt, sodass Sie fortlaufend die gesamte Datei durchlaufen können. Versuchen Sie, eine Zeile nach dem Ende der Datei zu lesen, erhalten Sie den Rückgabewert `false`. Dies können Sie verwenden, um eine Datei innerhalb einer Schleife vollständig zu durchlaufen.

```
while ($zeile = <DATEI>) {  
    ... # Bearbeite aktuelle Zeile aus der Datei in der Variablen $zeile  
}
```

Beispiel: `Zeilenweise.pl`

Das folgende Perl-Programm liest eine Datei zeilenweise ein und zeigt die Zeilen zusammen mit einer Zeilennummer am Bildschirm an.

```

#!/usr/bin/perl
use strict;
my $zeile;
my $zlnr=0;
print "Das Programm liest eine Beispieldatei zeilenweise aus.\n\n";
① open(DATEI,"beispieldatei.txt") || die("Datei kann nicht geöffnet werden!");
② while($zeile=<DATEI>) {
③     chomp($zeile);
④     print "$zeile (".++$zlnr.")\n";
}
⑤ close(DATEI);

```

- ① Mit der `open`-Anweisung wird die Datei `beispieldatei.txt` im aktuellen Arbeitsverzeichnis zum Lesen geöffnet. Falls die Operation nicht erfolgreich war, wird das Programm mit einer Fehlermeldung beendet.
- ② In einer `while`-Schleife wird die Datei zeilenweise durchlaufen.
- ③ Die Zeilen enthalten noch den Zeilenumbruch, der hiermit entfernt wird.
- ④ Mit der Variablen `$zlnr` wird eine fortlaufende Zeilennummer generiert und zusammen mit der eingelesenen Zeile ausgegeben.
- ⑤ Am Ende des Programms wird die Datei wieder geschlossen.

```

F:\PerlProgramme\Kapitel_08>perl Zeilenweise.pl
Das Programm liest eine Beispieldatei zeilenweise aus.
Das ist die erste Zeile der Datei. <1>
Das ist die zweite Zeile der Datei. <2>
Das ist die dritte Zeile der Datei. <3>
Das ist die vierte Zeile der Datei. <4>
F:\PerlProgramme\Kapitel_08>_

```

Zeilenweises Auslesen einer Datei

Dateien vollständig einlesen

Neben dem zeilenweisen Abarbeiten können Sie den Inhalt einer Datei auch mit einem Mal vollständig in ein Array einlesen. Jedes Array-Element enthält danach eine Zeile der Datei.

```
@zeilen = <DATEI>;
```

8.3 In Dateien schreiben und Dateien sperren

Schreiben in Dateien

Um in eine Datei zu schreiben, können Sie, wie bei der Bildschirmausgabe, die `print`-Anweisung mit dem entsprechenden Datei-Handle verwenden. Wird `print` ohne die Angabe eines Datei-Handles verwendet, nimmt Perl automatisch das Handle `STDOUT` (normalerweise der Bildschirm) an.

Syntax des Schreibens in eine Datei

```

print DATEI $variable;
print DATEI "Text";

```

- ✓ Nach dem Schlüsselwort `print` wird das Datei-Handle einer zum Schreiben geöffneten Datei angegeben. Fehlt diese Angabe, nimmt Perl die Standardausgabe `STDOUT`.
- ✓ Danach folgt der zu schreibende Ausdruck als Variable oder konstanter Wert.

Beachten Sie, dass Dateien mit der `open`-Funktion standardmäßig nur zum Lesen geöffnet werden. Ein Schreibvorgang löst in diesem Fall einen Fehler aus. Geben Sie beim Öffnen vor dem Dateinamen daher einen geeigneten Modus an, z. B. `>>` zum anhängenden Schreiben.



Beispiel: *TextSpeichern.pl*

Das Perl-Programm liest Eingaben über die Tastatur und speichert diese, zusammen mit einer Zeilennummer, in einer Datei.

```
#!/usr/bin/perl
use strict;

my $datei="eingaben.txt";
my $zlnr=0;
my $text="";

print "Das Programm schreibt Tastatureingaben in eine Datei.\n\n";
① open(DATEI,>$datei) || die("Datei $datei kann nicht geöffnet werden!");
do {
    print "Bitte Text eingeben (Ende - kein Text):\n";
    $text=<STDIN>;
    if ($text ne "\n") {
②     print DATEI ++$zlnr.": $text";
    }
③ } while($text ne "\n");
④ close(DATEI);
```

- ① Mit der `open`-Funktion wird die Datei, deren Name über eine Variable festgelegt ist, zum Schreiben geöffnet. Ist das Öffnen nicht erfolgreich, erfolgt eine Fehlermeldung.
- ② An dieser Stelle wird mit der `print`-Funktion in die Datei geschrieben. Dazu wird das Datei-Handle `DATEI` angegeben. Zusätzlich zum eingegebenen Text erfolgt die Ausgabe einer Zeilennummer. Da die Eingabe noch den Zeilenumbruch enthält, muss er beim Schreiben nicht mit angegeben werden.
- ③ Die Schleife wird so lange durchlaufen, bis kein Text eingegeben wurde. In diesem Fall enthält die Variable `$text` nur den Zeilenumbruch `\n`.
- ④ Mit der `close`-Funktion wird die Datei wieder geschlossen.

Dateien sperren

Es kann vorkommen, dass mehrere Prozesse gleichzeitig in eine Datei schreiben wollen. Gerade beim Einsatz eines Perl-Programms als CGI-Anwendung ist dies sehr wahrscheinlich.

Während Sie in Ihrem Programm den Inhalt einer Datei einlesen, bearbeiten und wieder in die Datei schreiben, kann ein anderes Programm bereits neue Daten in die Datei geschrieben haben. Um den Datenverlusten in solchen Situationen vorzubeugen, können Sie die Datei mit der `flock`-Funktion für einen weiteren Zugriff sperren.

Syntax der `flock`-Funktion

```
flock(DATEI,Sperroption);
```

- ✓ Die `flock`-Funktion erwartet als erstes Argument ein Handle auf eine vorher geöffnete Datei.
- ✓ Das zweite Argument ist ein numerischer Wert, der die Sperroption bestimmt.

Die Sperroption kann als Zahl oder Konstante angegeben werden. Konstanten können Sie nur verwenden, wenn Sie am Anfang des Programms das Modul Fcntl mit dem Befehl `use Fcntl ':flock';` einbinden.

Sperroption	Konstante	Erklärung
1	LOCK_SH	Zugriff erfolgt <i>shared</i> , d. h., nur der aktuelle Prozess darf in die Datei schreiben, alle anderen Prozesse dürfen nur lesen.
2	LOCK_EX	Zugriff erfolgt <i>exclusive</i> , d. h., nur der aktuelle Prozess darf die Datei bearbeiten, für alle anderen ist jeglicher Zugriff verboten.
4	LOCK_NB	<i>non-blocking</i> , Zugriffsversuch wird beendet, ohne auf die Freigabe einer Datei durch einen blockierenden Prozess zu warten.
8	LOCK_UN	<i>unlock</i> , Zugriffsschutz wird aufgehoben

Beispiel: *counter.pl*

Das folgende Programm besitzt die grundlegende Funktionsweise eines CGI-Programms, das die Zugriffe auf eine bestimmte Datei zählt. In einer Textdatei wird für verschiedene Dateien einer Webseite die Anzahl der Zugriffe gespeichert. Das Perl-Programm liest diese Textdatei ein, erhöht den Zähler für die betreffende Datei der Webseite und speichert den neuen Zählerstand. Damit kein Zugriff verloren geht, wird die Datei beim Zugriff exklusiv gesperrt. In der Textdatei werden Dateiname der Webseite und Zählerstand durch Kommata getrennt.

```
#!/usr/bin/perl
use strict;
① use Fcntl ':flock';

② my $zeile="";
my @zeile;
my $datei="counter.txt";
my $seite="index.html";

③ open(DATEI,"+<$datei") || die("Datei $datei kann nicht geoeffnet werden!");
④ flock(DATEI, LOCK_EX);
⑤ @daten=<DATEI>;
⑥ seek(DATEI,0,0);
foreach(@daten) {
⑦   @zeile=split(/;/,$_);
⑧   if ($zeile[0] eq $seite) {
⑨     print DATEI "$zeile[0];".++$zeile[1]."\n";
⑩   } else {
⑪     print DATEI $_;
⑫   }
⑬ }
⑭ flock(DATEI, LOCK_UN);
close(DATEI);
```

- ① Damit für die `flock`-Funktion die entsprechenden Konstanten verwendet werden können, muss das Fcntl-Modul geladen werden. Dies geschieht mit der `use`-Anweisung.
- ② An dieser Stelle werden die Namen der Zählerdatei und der zuzählenden Webseiten-Datei festgelegt.
- ③ Da der Zählerstand zuerst eingelesen und dann neu geschrieben werden soll, wird die Zählerdatei mit der `open`-Funktion für den gleichzeitigen Lese- und Schreibzugriff geöffnet.

- ④ Damit kein anderer Prozess einen neuen Zählerstand speichern kann, solange die Bearbeitung in diesem Programm läuft, wird die Datei exklusiv gesperrt. Auch das Lesen durch andere Prozesse in dieser Zeit ist nicht sinnvoll, da der aktuelle Zugriff noch nicht verzeichnet ist.
- ⑤ Die Zählerdatei wird vollständig in das Array @daten eingelesen.
- ⑥ Nach dem Auslesen der Datei befindet sich der Dateizeiger am Ende. Um ein Schreiben am Dateianfang zu ermöglichen, wird der Zeiger mit der seek-Funktion zurückgesetzt.
- ⑦ Die gesamte Datei wird in einer Zeile durchlaufen und jede Zeile anhand des Kommas als Trennzeichen aufgeteilt.
- ⑧ Als erstes Element jeder Zeile wird der Dateiname der Webseite angegeben. Stimmt dieser mit dem gewünschten Namen überein, wird die Zählung in der folgenden Anweisung ausgeführt.
- ⑨ Der neue Zählerstand für die gewünschte Seite wird hier in die Zählerdatei geschrieben. Jede Zeile in der Datei muss mit einem Zeilenumbruch abgeschlossen werden.
- ⑩ Alle übrigen Zeilen der Zählerdatei werden unverändert in die Datei geschrieben. Hier ist der Zeilenumbruch noch vom Einlesen vorhanden.
- ⑪ Am Ende des Programms wird die exklusive Dateisperre wieder aufgehoben.

Das Programm liest eine vorhandene Zählerdatei ein und speichert den neuen Zählerstand. Für die zuzählende Datei muss jedoch bereits ein Eintrag vorhanden sein. Das mögliche Ergebnis könnte so aussehen:

Zählerdatei vor dem Programmaufruf	... und nach dem Programmaufruf
<pre>hilfe.html;1; index.html;1; faq.html;1;</pre>	<pre>hilfe.html;1; index.html;2; faq.html;1;</pre>

8.4 Statusinformationen über Dateien ermitteln

Perl bietet mit den Dateiprüfoperatoren, auch **-x-Dateiprüfungen** genannt, eine einfache und nützliche Möglichkeit, um verschiedene Statusinformationen über Dateien zu ermitteln. So können Sie beispielsweise bereits vor dem Öffnen einer Datei prüfen, ob sie überhaupt existiert oder leer ist.

Syntax der Dateiprüfoperatoren

```
-x "dateiname"
-x $variable
```

- ✓ Die Dateiprüfung beginnt immer mit einem Bindestrich und dem Kennbuchstaben eines Dateiprüfoperators.
- ✓ Danach folgt der Name einer Datei als Zeichenkette oder Variable. Der Dateiname kann auch Verzeichnisangaben enthalten.
- ✓ Der Ausdruck liefert einen Wahrheitswert oder einen numerischen Wert zurück.

Viele Dateiprüfoperatoren können auch für Verzeichnisangaben verwendet werden.

Die folgende Tabelle zeigt die wichtigsten Dateiprüfoperatoren in Perl:

Dateiprüfoperator	Beschreibung
-e	Prüft, ob die Datei existiert
-f	Prüft, ob die Angabe eine reguläre Datei ist
-d	Prüft, ob die Angabe ein Verzeichnis ist

Dateiprüfoperator	Beschreibung
-l	Prüft, ob die Angabe ein Symlink (Verweis, Verknüpfung) auf eine Datei ist
-S	Prüft, ob die Angabe ein Socket ist (nur unter Unix)
-s	Ermittelt, ob die Datei nicht leer ist und liefert die Dateigröße in Bytes
-z	Ermittelt, ob die Datei leer ist
-T	Ermittelt, ob es sich um eine Textdatei handelt
-B	Ermittelt, ob es sich um eine Binärdatei handelt
-x	Prüft, ob die Datei ausführbar ist
-A	Ermittelt, wie lange der letzte Zugriff auf die Datei zurückliegt (in Tagen)
-M	Ermittelt, wie lange die letzte Änderung der Datei zurückliegt (in Tagen)
-W	Prüft, ob der aktuelle Benutzer in die Datei schreiben kann
-R	Prüft, ob der aktuelle Benutzer die Datei lesen kann
-w	Prüft, ob die Datei beschreibbar ist
-r	Prüft, ob die Datei lesbar ist

Beispiel

Mit einer `if`-Abfrage wird vor dem Öffnen der Datei geprüft, ob die Datei vorhanden ist.

```
$datei="beispiel.txt";
if (-e $datei) {
    open(DATEI,$datei);
} else {
    print "Die Datei $datei ist nicht vorhanden!\n";
}
```

8.5 Dateien zeichenweise bearbeiten

Funktionen für zeichenweisen Zugriff

Das zeilenweise Einlesen eignet sich nur für Textdateien, da hier alle Zeilen durch einen Zeilenumbruch abgeschlossen sind. Dies sind alle Dateien, die Sie auch mit einem Texteditor öffnen und bearbeiten können.

Binäre Dateien, die zusätzliche Steuerzeichen enthalten können und nicht zeilenweise strukturiert sind, sollten Sie besser zeichen- oder blockweise einlesen. In diesem Fall beachtet Perl das Zeilenumbruchzeichen nicht und liest alle Zeichen so ein, wie sie in der Datei vorhanden sind.

Folgende Funktionen stehen Ihnen für diese Zugriffsweise zur Verfügung:

<code>read();</code>	Liest eine bestimmte Anzahl Zeichen ab der aktuellen Position des Dateizeigers in eine Variable ein
<code>seek();</code>	Setzt den Dateizeiger an eine neue Position
<code>tell();</code>	Ermittelt die aktuelle Position des Dateizeigers

<code>eof();</code>	Liest ein einzelnes Zeichen aus einer Datei
<code>getc();</code>	Umschalten zur binären Bearbeitung von Dateien unter Betriebssystemen, die zwischen Textdateien und binären Dateien unterscheiden
<code>binmode();</code>	Prüft, ob der Dateizeiger das Dateiende erreicht hat

Dateizeiger positionieren und prüfen

Beim zeichenweisen Zugriff auf eine Datei ist der Dateizeiger von besonderer Bedeutung. Er bestimmt, ab welcher Stelle die nächsten Daten aus einer Datei gelesen werden. Gleichzeitig können Sie mit ihm prüfen, ob bereits das Ende der Datei erreicht ist. Für die Angabe der Dateizeigerposition werden Bytes verwendet.

```
seek(DATEI, Position, Bezug);
```

- ✓ Mit der `seek`-Funktion wird der Dateizeiger positioniert. Sie beginnt mit dem Schlüsselwort `seek`.
- ✓ Als erstes Argument wird das Datei-Handle einer geöffneten Datei angegeben.
- ✓ Das zweite Argument ist die Position des Dateizeigers in Bytes abhängig von der Bezugsposition im dritten Argument.

Folgende Bezugsposition können Sie verwenden:

<code>0</code>	Position des Dateizeigers vom Anfang der Datei aus
<code>1</code>	Relative Verschiebung des Dateizeigers von der aktuellen Position aus
<code>2</code>	Position des Dateizeigers vom Ende der Datei aus

Mit der `tell`-Funktion können Sie die aktuelle Position des Dateizeigers bestimmen.

```
$position = tell(DATEI);
```

- ✓ Die `tell`-Anweisung erwartet als Argument das Datei-Handle einer geöffneten Datei.
- ✓ Der Rückgabewert ist die Position des Dateizeigers in Bytes, vom Beginn der Datei aus gerechnet.

Mit der `eof`-Funktion ist es möglich zu überprüfen, ob der Dateizeiger bereits das Dateiende überschritten hat. Diese Prüfung ist vor allem im Zusammenhang mit dem Auslesen einer Datei notwendig, da die betreffenden Funktionen den Dateizeiger automatisch verschieben.

```
eof(DATEI);
```

- ✓ Die `eof`-Funktion erwartet als Argument das Datei-Handle einer geöffneten Datei.
- ✓ Der Rückgabewert ist `true`, wenn der Dateizeiger das Ende der Datei überschritten hat.

Datei zeichenweise lesen

Mit der `getc`-Funktion lesen Sie ein einzelnes Zeichen aus einer Datei. Der Dateizeiger wird dabei um ein Byte verschoben.

```
$zeichen = getc(DATEI);
```

- ✓ Die `getc`-Funktion erwartet als Argument das Datei-Handle einer geöffneten Datei.
- ✓ Die Funktion liefert ein einzelnes, aus der Datei ausgelesenes Zeichen zurück und verschiebt den Dateizeiger um ein Byte.

Beispiel: Zeichenweise.pl

Mit der `getc`-Funktion wird eine Datei zeichenweise ausgelesen und auf dem Bildschirm angezeigt.

```
#!/usr/bin/perl
use strict;
my $zeichen="";
open(DATEI, "<beispieldatei.txt") || die("Datei kann nicht geoeffnet werden!");
① while(!eof(DATEI)) {
②     $zeichen=getc(DATEI);
    print $zeichen;
}
close(DATEI);
```

- ① In einer bedingten Schleife erfolgt das Auslesen der Daten, bis das Dateiende erreicht ist. Mit der `eof`-Funktion wird dabei die Position des Dateizeigers überwacht.
- ② Die `getc`-Funktion liefert ein einzelnes Zeichen aus der geöffneten Datei.

Eine sehr mächtige Funktion, um Daten aus einer Datei zu lesen, ist die `read`-Funktion. Sie ermöglicht eine genaue Kontrolle der Position und Anzahl der zu lesenden Zeichen.

```
read(DATEI, $variable, Bytes [, Offset]);
```

- ✓ Die `read`-Anweisung erwartet als erstes Argument das Datei-Handle einer geöffneten Datei.
- ✓ Das zweite Argument ist eine skalare Variable, in der die eingelesenen Daten gespeichert werden.
- ✓ Als drittes Argument wird die Anzahl der zu lesenden Daten in Bytes angegeben.
- ✓ Optional kann noch ein vierter Argument verwendet werden. Es bestimmt ein Offset in Bytes, ab dem die Daten in der Variablen gespeichert werden.
- ✓ Die Funktion liefert die Anzahl der gelesenen Zeichen zurück.

Binärmodus verwenden

Einige Betriebssysteme, wie beispielsweise Windows, unterscheiden beim Zugriff auf Dateien zwischen Text- und Binärmodus. Im Textmodus wird dabei aus den zwei aufeinander folgenden Steuerzeichen `\r` (Wagenrücklauf) und `\n` (Zeilenvorschub) am Ende einer Zeile automatisch ein einzelnes `\n`. Bei binären Dateien können dadurch Fehler auftreten. Mit der Perl-Funktion `binmode` schalten Sie den binären Modus ohne die automatische Umwandlung ein. Unter Unix-artigen Betriebssystemen ist die Funktion bedeutungslos.

```
binmode(DATEI);
```

- ✓ Die `binmode`-Funktion erwartet als erstes Argument das Datei-Handle einer geöffneten Datei.
- ✓ Die Funktion schaltet in den binären Modus zum Lesen und Schreiben in die Datei, bis die Datei mit der `close`-Anweisung geschlossen wird.

Unter Windows muss der Binärmodus auch beim Schreiben einer Datei beachtet werden. Dies ist besonders wichtig, wenn Sie Perl für Ihre CGI-Anwendungen benutzen und den Webserver unter Windows betreiben. Wenn Sie z. B. eine Grafikdatei ausgeben wollen, müssen Sie den Binärmodus auch für die Standardausgabe `STDOUT` setzen.

Beispiel: *BildAusgabe.pl*

Das folgende Perl-Programm ist ein Beispiel für eine CGI-Anwendung. Das Programm liest eine Bilddatei im GIF-Format ein und gibt Sie auf der Standardausgabe wieder aus. Damit das Programm unter Windows funktionsfähig ist, wird der Binärmodus verwendet. Um eine sinnvolle Darstellung des Bildes zu erhalten, sollte das Programm mit einem Browser von einem Webserver aufgerufen werden. Sie können das Programm auch in einer Konsole starten, dann werden die Binärzeichen der Grafikdatei ausgegeben.

```
#!/usr/bin/perl
use strict;

my $datei="button.gif";
my $bild;
open(BILD, "<$datei");
① binmode(BILD);
② my $groesse= -s $datei;
③ read(BILD,$bild,$groesse);
close(BILD);

④ binmode(STDOUT);
⑤ print "Content-Type: image/gif\n\n";
⑥ print $bild;
```

- ① Nach dem Öffnen der Datei wird der binäre Modus für das Datei-Handle eingeschaltet.
- ② Die Grafikdatei muss blockweise eingelesen werden. Daher wird an dieser Stelle mit dem Dateiprüfoperator `-s` die Dateigröße in Bytes ermittelt.
- ③ Mit der `read`-Funktion wird die Grafikdatei vollständig in die Variable `$bild` eingelesen. Dazu wird die vorher ermittelte Dateigröße verwendet.
- ④ Wird der Webserver, der das Perl-Programm aufruft, unter Windows betrieben, muss zuerst für die Standardausgabe der Binärmodus eingeschaltet werden.
- ⑤ Damit der Webbrowser die Daten als Grafik erkennt, muss der korrekte Typ der Daten angegeben werden.
- ⑥ Das Bild wird ausgegeben. Der Webserver leitet die Daten dann an den aufrufenden Webbrowser weiter.

Eine sinnvolle Ausgabe des Beispielprogramms erhalten Sie bei der Ausführung als CGI-Programm. Dazu muss ein Webserver installiert sein. Nähere Informationen zur Installation des Webservers Apache und zur Verwendung von CGI-Programmen finden Sie in Kapitel 11.



Programmstart über die Konsole

Das Programm kann auch in einer Konsole ausgeführt werden, aber die binären Bilddaten werden dann einfach ausgegeben und das Bild nicht dargestellt.

8.6 Funktionen des Dateisystems

Perl stellt Ihnen einige Funktionen zur Verfügung, um auf der Ebene des Dateisystems zu arbeiten. So können Sie beispielsweise Dateien umbenennen, verschieben oder löschen. Einige Funktionen sind speziell auf Verzeichnisse ausgerichtet.

Die folgende Tabelle zeigt die wichtigsten Funktionen für das Dateisystem:

<code>rename();</code>	Datei- oder Verzeichnis umbenennen
<code>unlink();</code>	Datei löschen
<code>chmod();</code>	Rechte für Dateien oder Verzeichnisse ändern
<code>mkdir();</code>	Verzeichnis erstellen
<code>chdir();</code>	Verzeichnis wechseln
<code>rmdir();</code>	Verzeichnis löschen
<code>stat();</code>	Eigenschaften einer Datei ermitteln

Dateien umbenennen oder löschen

```
rename(AlterName, NeuerName);  
unlink(Dateiname);
```

- ✓ Mit der `rename`-Funktion wird eine Datei umbenannt. Als erstes Argument wird dabei der alte Name und als zweites Argument der neue Name angegeben. War das Umbenennen erfolgreich, wird der Wert `true` zurückgeliefert.
- ✓ Die `unlink`-Funktion kann zum Löschen von Dateien verwendet werden.

Mit der `rename`-Funktion können Sie neben Dateien auch Verzeichnisse umbenennen. Die `unlink`-Funktion kann dagegen nur Dateien löschen. Für Verzeichnisse müssen Sie die Funktion `rmdir()` verwenden.



Datei- oder Verzeichnisrechte verändern

Perl stammt aus der Welt des Betriebssystems Unix. Unter Unix-artigen Betriebssystemen werden für alle Dateien und Verzeichnisse Zugriffsrechte für drei Gruppen von Benutzer erteilt:

- ✓ Besitzer der Datei bzw. des Verzeichnisses
- ✓ Benutzergruppe, zu der der Besitzer gehört
- ✓ alle anderen Benutzer

Für jede der drei Gruppen wird die Lese-, Schreib- und Ausführungsberechtigung erteilt oder verwehrt. Diese Berechtigungen werden oft mit den Kürzeln `r` (engl. `read` = lesen), `w` (engl. `write` = schreiben) und `x` (engl. `execute` = ausführen) bezeichnet. Eine vollständige Angabe der Benutzerrechte besitzt daher drei `r w x`-Gruppen. Soll ein Recht gewährt werden, wird eine binäre 1 gesetzt, sonst eine binäre 0. Aus der 9-stelligen Binärzahl wird danach ein Oktalwert gebildet.

Besitzer <code>r w x</code>	Gruppe <code>r w x</code>	Andere Benutzer <code>r w x</code>	Erklärung	Oktalwert
1 1 1	1 1 1	1 1 1	Alle Benutzer können die Datei lesen, beschreiben und ausführen.	777
1 1 1	1 0 0	1 0 0	Der Besitzer hat alle Datei-Rechte, alle anderen Benutzer dürfen die Datei nur lesen.	744
1 1 1	1 0 1	1 0 1	Der Besitzer hat alle Datei-Rechte, alle anderen Benutzer dürfen die Datei lesen und ausführen (besonders für CGI-Programme).	755

Mit der `chmod`-Funktion verändern Sie die Zugriffsberechtigung für eine Datei oder ein Verzeichnis. Dabei müssen Sie den Oktalwert der neuen Berechtigung angeben.

```
chmod(Berechtigung, Dateiname);
```

- ✓ Als erstes Argument der Funktion wird die neue Zugriffsberechtigung als Oktalwert angegeben. Damit Perl die Oktalzahl erkennt, muss ihr eine Null vorangestellt werden.
- ✓ Das zweite Argument ist der Name der Datei, für die die Berechtigung gelten soll.

Beispiel

```
chmod(0777,"beispiel.txt");
chmod(0755,"programm.pl");
chmod(0744,$datei);
```



Verzeichnis erstellen, wechseln oder löschen

```
mkdir(Verzeichnisname, Berechtigung) ;
chdir(Pfadname) ;
rmdir(Verzeichnisname) ;
```

- ✓ Mit der `mkdir`-Funktion erstellen Sie ein neues Verzeichnis. Als erstes Argument wird der Verzeichnisname angegeben. Danach folgt die Zugriffsberechtigung des neuen Verzeichnisses als Oktalzahl.
- ✓ Die Funktion `chdir()` erlaubt den Wechsel des aktuellen Arbeitsverzeichnisses. Als Argument wird der vollständige Pfad zum Verzeichnis angegeben.
- ✓ Die `rmdir`-Funktion löscht das als Argument angegebene Verzeichnis.

Eigenschaften einer Datei ermitteln

Die `stat`-Funktion liefert ein Array mit verschiedenen Statusinformationen einer Datei, u. a. Dateigröße und letzte Zugriffszeit.

```
@array = stat("Dateiname") ;
```

- ✓ Die `stat`-Funktion liefert als Ergebnis ein Array mit verschiedenen statistischen Funktionen über eine Datei.
- ✓ Der Dateiname wird als Argument übergeben.

Das Array enthält 13 Elemente mit folgender Bedeutung:

Indexwert	Erklärung
0	Eindeutige Gerätenummer des Dateisystems, auf dem sich die Datei befindet
1	Eindeutige Nummer (Inode) der Datei innerhalb des Dateisystems
2	Dateityp und Zugriffsrechte als Oktalzahl
3	Anzahl der Links auf die Datei
4	Benutzer-Nummer (User-ID) des Besitzers der Datei
5	Gruppen-Nummer (Group-ID) des Besitzers der Datei
6	RAW-Gerätenummer bei Gerätedateien
7	Dateigröße in Bytes
8	Letzte Zugriffszeit als Unix-Zeitangabe
9	Letzte Änderungszeit als Unix-Zeitangabe
10	Letzte Zugriffszeit für einen Zugriff, der eine Inode-Änderung zur Folge hatte
11	Ideale Blockgröße für blockweises Lesen oder Schreiben der Datei
12	Anzahl belegter Speicherblöcke

8.7 Mit Verzeichnissen arbeiten

Sie kennen bereits die Dateisystemfunktionen `mkdir()`, `chdir()` und `rmdir()`, um Verzeichnisse zu erstellen, das aktuelle Arbeitsverzeichnis zu wechseln oder ein Verzeichnis zu löschen. Perl verfügt jedoch über weitere Funktionen, um gezielt den Inhalt eines Verzeichnisses auszulesen.

Der Zugriff auf Verzeichnisse erfolgt in Perl ähnlich wie bei Dateien. So müssen Sie ein Verzeichnis zuerst öffnen, um ein Verzeichnis-Handle zu erhalten. Danach können Sie das Verzeichnis mithilfe eines Verzeichniszeigers Datei für Datei durchlaufen.

<code>opendir();</code>	Verzeichnis für eine Bearbeitung öffnen
<code>readdir();</code>	Einträge aus einem Verzeichnis auslesen
<code>seekdir();</code>	Verzeichniszeiger auf einen Eintrag positionieren
<code>telldir();</code>	Aktuelle Position des Verzeichniszeigers ermitteln
<code>rewinddir();</code>	Verzeichniszeiger auf ersten Eintrag positionieren
<code>closedir();</code>	Verzeichnis schließen

Verzeichnis öffnen und schließen

Wie bei der Arbeit mit Dateien müssen Sie auch Verzeichnisse vor dem Auslesen mit der Funktion `opendir` öffnen. Sie erhalten ein Verzeichnis-Handle, über das alle folgenden Zugriffe erfolgen müssen. Am Ende der Bearbeitung sollten Sie das Verzeichnis wieder schließen.

```
opendir(VERZEICHNIS, Verzeichnisname);
closedir(VERZEICHNIS);
```

- ✓ Mit der `opendir`-Funktion wird ein Verzeichnis für eine folgende Bearbeitung geöffnet. Als erstes Argument wird das Verzeichnis-Handle angegeben. Danach folgt der Name des zu öffnenden Verzeichnisses.
- ✓ Die `opendir`-Funktion gibt den Wert `true` zurück, wenn das Öffnen erfolgreich war.
- ✓ Die `closedir`-Funktion schließt ein geöffnetes Verzeichnis.

Verzeichnis auslesen

Mit der `readdir`-Funktion lesen Sie die in einem Verzeichnis vorhandenen Einträge aus. Je nach Ausführungs-kontext liefert die Funktion dabei den nächsten Verzeichniseintrag als skalaren Wert oder ein Array mit allen Einträgen.

```
$eintrag = readdir(VERZEICHNIS);
@eintraege = readdir(VERZEICHNIS);
```

- ✓ Die `readdir`-Funktion liefert die Einträge eines Verzeichnisses. Als Argument wird dabei das Verzeichnis-Handle eines geöffneten Verzeichnisses übergeben.
- ✓ Wird die Funktion im skalaren Kontext verwendet, gibt sie den nächsten Verzeichniseintrag zurück und setzt den Verzeichniszeiger auf den folgenden Wert.
- ✓ Im Listenkontext liefert die Funktion ein Array mit allen Verzeichniseinträgen.

Einträge in einem Verzeichnis sind Dateinamen, die Namen weiterer Verzeichnisse und die Platzhalter für das aktuelle Verzeichnis (`.`) und das übergeordnete Verzeichnis (`..`).



Beispiel: *VerzeichnisAuslesen.pl*

Das Perl-Programm soll ausgehend von einem Startverzeichnis alle Dateien in diesem Verzeichnis und in allen untergeordneten Verzeichnissen auflisten.

```
#!/usr/bin/perl
use strict;
① use Symbol;

② my $startverzeichnis= "f:/PerlProgramme";

③ &VerzeichnisAuslesen($startverzeichnis);
exit;

sub VerzeichnisAuslesen {
④   my $verzeichnish=gensym();
⑤   my $verzeichnis = $_[0];
print "\n$verzeichnis\n";
⑥   opendir($verzeichnish,$ verzeichnis) ||
      die("Das Verzeichnis $verzeichnis kann nicht geöffnet werden!");
⑦   while(my $eintrag=readdir($verzeichnish)) {
⑧     next if ($eintrag eq ".." || $eintrag eq "...");
print "$eintrag\n";
⑨     if (-d "$verzeichnis/$eintrag") {
&VerzeichnisAuslesen("$verzeichnis/$eintrag");
      }
    }
⑩   closedir($verzeichnish);
}
```

- ① Damit die gensym-Funktion zum Erstellen eines unbenannten Verzeichnis-Handles zur Verfügung steht, muss mit der use-Anweisung das Symbol-Modul eingebunden werden.
- ② Hier wird das Startverzeichnis festgelegt. Die Angabe muss an die jeweiligen Gegebenheiten angepasst werden.
- ③ Das Auslesen des Verzeichnisses ist als Unterprogramm ausgeführt. Hier wird es mit dem Startverzeichnis als Argument aufgerufen.
- ④ Das Verzeichnis-Handle soll lokal nur im Unterprogramm gültig sein. Zu diesem Zweck wird ein unbenanntes Handle mit der gensym-Funktion erzeugt.
- ⑤ Die ebenfalls lokale Variable \$verzeichnis nimmt den als Argument übergebenen Verzeichnisnamen auf.
- ⑥ Hier wird das Verzeichnis geöffnet. Dabei wird das in der Variable \$verzeichnish gespeicherte Handle verwendet.
- ⑦ Nacheinander werden alle Verzeichniseinträge in einer Schleife durchlaufen. Nach dem letzten Eintrag liefert die readdir-Funktion den Wert false und beendet die Schleife.
- ⑧ Die Platzhalter für das aktuelle Verzeichnis und das übergeordnete Verzeichnis werden ausfiltriert.

```
f:/PerlProgramme/Uebungsdateien/Kapitel_12
kontakt.cgi
kontakt.html
Kapitel_13

f:/PerlProgramme/Uebungsdateien/Kapitel_13
gaestebuch.cgi
Kapitel_14

f:/PerlProgramme/Uebungsdateien/Kapitel_14
newsletter.cgi
Kapitel_15

f:/PerlProgramme/Uebungsdateien/Kapitel_15
newsletter_senden.cgi

F:/PerlProgramme/Kapitel_08>_
```

Auslesen eines Verzeichnisses mit allen Unterverzeichnissen

- ⑨ Mit Dateiprüfoperator `-d` wird ermittelt, ob es sich bei dem aktuellen Verzeichniseintrag um ein Unterverzeichnis handelt. Ist dies der Fall, wird das Unterprogramm mit diesem Verzeichnisnamen als Argument rekursiv aufgerufen.
- ⑩ Am Ende des Unterprogramms wird das Verzeichnis wieder geschlossen.

Das Auslesen der Verzeichnisse erfolgt durch einen rekursiven Aufruf des Unterprogramms, das sich dadurch praktisch selbst aufruft. Da das Unterprogramm somit für verschiedene Verzeichnisse quasi gleichzeitig aktiv ist, müssen alle Variablen und das Verzeichnis-Handle lokal deklariert werden.



Verzeichniszeiger verwenden

Wie beim Zugriff auf Dateien gibt es auch beim Auslesen von Verzeichnissen einen Zeiger. Für das Positionieren des Zeigers werden vom Dateisystem gelieferte Adressen und keine fortlaufenden Nummern verwendet. Der Dateizeiger wird eintragsweise verschoben.

```
$position = telldir(VERZEICHNIS);
```

- ✓ Mit der `telldir`-Funktion wird die aktuelle Position des Verzeichniszeigers ermittelt. Dabei wird das Handle eines geöffneten Verzeichnisses als Argument verlangt.
- ✓ Die Funktion liefert die Adresse des aktuellen Verzeichniseintrags als Ergebnis.

```
seekdir(VERZEICHNIS, Position);
```

- ✓ Mit der `seekdir`-Funktion wird der Verzeichniszeiger verschoben. Als erstes Argument benötigt die Funktion ein Handle auf ein geöffnetes Verzeichnis.
- ✓ Als zweites Argument muss die Adresse eines Verzeichniseintrags angegeben werden. Dabei sind nur Adressen gültig, die mit der `telldir`-Funktion ermittelt wurden.

```
rewinddir(VERZEICHNIS);
```

- ✓ Mit der `rewinddir`-Funktion wird der Verzeichniszeiger auf den ersten Eintrag verschoben. Als Argument benötigt die Funktion ein Handle auf ein geöffnetes Verzeichnis.

8.8 Der Diamond-Operator (<>)

Zwar sind Eingaben vom Terminal heutzutage eher selten, aber Perl stellt neben dem Standard-Handle `<STDIN>` die Zeichenkombination `<>` als eine Kurzform dafür bereit, wobei das eigentlich eine Spezialform des sogenannten Rhombus-Operators oder Diamond-Operators ist. Sie werden diese Schreibweise in alten Perl-Skripten oft finden, können sie aber auch heutzutage als Abkürzung einsetzen, wenn Sie eine Terminaleingabe benötigen.

Beispiel: *eingabeDiamond.pl*

Das Perl-Programm nimmt mit dem Diamond-Operator eine Eingabe entgegen und gibt den Wert wieder aus.

```
#!/usr/bin/perl
use strict;
print "Geben Sie Ihren Namen an: ";
① my $name = <>;
② print "Hallo $name\n";
```

- ① Der Diamond-Operator wird als Alias für die Standardeingabe verwendet
- ② Der eingelesene Wert wird ausgegeben.

```
F:\PerlProgramme\Kapitel_08>perl eingabeDiamond.pl
Geben Sie Ihren Namen an: Ralph
Hallo Ralph
```

```
F:\PerlProgramme\Kapitel_08>
```

Verwenden des Diamond-Operators

Wenn kein Filehandle in Verbindung mit dem Operator verwendet wird, wird Perl indirekt die spezielle Variable @ARGV untersuchen. Wenn @ARGV keine Elemente hat, liest der Operator entweder von der Standardeingabe bzw. Tastatur oder einer umgeleiteten Datei. Sie können den Operator so ganz einfach in einer Schleife zum Einlesen von mehreren Informationen verwenden.

Beispiel: eingabeDiamond2.pl

```
#!/usr/bin/perl
use strict;
print "Geben Sie Ihren Namen an: ";
① while (<>) {
    print "$.: $_";
}
② print "\n$. Zeilen insgesamt\n\n";
```

- ① Der Diamond-Operator wird in der Bedingung der Schleife verwendet und liest eine Eingabe nach der anderen. In \$ steht die Anzahl der Lesevorgänge und in \$_ der jeweils eingelesene Inhalt.
- ② Die Anzahl der Lesevorgänge wird ausgegeben. Beachten Sie, dass Sie aus Gründen der Einfachheit die Schleife mit **Ctrl C** beenden müssen.

```
F:\PerlProgramme\Kapitel_08>perl eingabeDiamond2.pl
Eingabe 1
1: Eingabe 1
Eingabe 2
2: Eingabe 2
Und noch was
3: Und noch was
3 Zeilen insgesamt
Terminating on signal SIGINT<2>
Terminating on signal SIGINT<2>
F:\PerlProgramme\Kapitel_08>
```

Verwenden des Diamond-Operators in einer Schleife

Der Diamond Operator ist empfehlenswert, wenn Sie den Inhalt von mehreren Dateien einlesen möchten. Rufen Sie dazu das Programm mit den Namen von Dateien als Parameter auf.

Beispiel: perl eingabeDiamond2.pl counter.txt eingaben.txt

```
F:\PerlProgramme\Kapitel_08>perl eingabeDiamond2.pl counter.txt eingaben.txt
1: hilfe.html;1;
2: index.html;4;
3: faq.html;1;
4: 1: Das ist neuer Text
4 Zeilen insgesamt

F:\PerlProgramme\Kapitel_08>
```

Der Inhalt von mehreren Dateien wird eingelesen und verarbeitet

Perl erstellt das @ARGV-Array aus der Befehlszeile. Jeder Dateiname wird dem @ARGV-Array als Element hinzugefügt. Wenn das Programm den Diamond-Operator ausführt, werden die angegebenen Dateien zeilenweise ausgelesen. Ist das Ende einer Datei erreicht, wird die nächste Datei genommen usw. Am Ende der letzten Datei wird die Schleife beendet.

Sie können auch innerhalb von einem Programm einfach dem Array @ARGV Werte mit Dateinamen zuweisen und dann den Operator einsetzen.

Beispiel:

```
@ARGV = ("counter.txt", "eingaben.txt");
while (<>) {
    print();
}
```

8.9 Schnellübersicht

Sie möchten ...	
eine Datei öffnen	open(DATEIHANDLE, "Dateiname");
auf Fehler beim Öffnen einer Datei reagieren	open(DATEIHANDLE, "Dateiname") die("Meldung");
eine Datei schließen	close(DATEIHANDLE);
ein unbenanntes Datei-Handle verwenden	use Symbol; \$handle=getsym();
eine Textdatei zeilenweise lesen	\$zeile=<DATEIHANDLE>;
eine Datei zeichenweise lesen	\$zeichen=getc(DATEIHANDLE);
eine Datei in Blöcken lesen	read(DATEIHANDLE,\$block,Bytes);
den Dateizeiger verschieben	seek(DATEIHANDLE, Position, Bezug);
in eine Datei schreiben	print DATEIHANDLE Daten;
eine Datei sperren	flock(DATEIHANDLE, Sperroption);
Statusinformationen über eine Datei ermitteln	- [x] Dateiname
unter Windows mit binären Dateien arbeiten	binmode(DATEIHANDLE);
eine Datei löschen	unlink(Dateiname);
Zugriffsberechtigungen setzen	chmod(Berechtigung, Dateiname);
ein Verzeichnis öffnen	opendir(VERZEICHNISHANDLE, Verzeichnisname);
einen Verzeichniseintrag lesen	\$eintrag=readdir(VERZEICHNISHANDLE);
ein Verzeichnis schließen	closedir(VERZEICHNISHANDLE);
das aktuelle Arbeitsverzeichnis wechseln	chdir(Pfadname);

8.10 Übungen

Dateien lesen und schreiben

Übungsdatei: --

Ergebnisdatei: --

1. Wie prüfen Sie, ob das Öffnen einer Datei erfolgreich war?
2. Beim Öffnen einer Datei können Sie einen Zugriffsmodus angeben. Erläutern Sie die verschiedenen Modi!
3. Mit welcher Perl-Funktion können Sie ein Programm anhalten und gleichzeitig eine Warnmeldung ausgeben?
4. Erläutern Sie das zeilenweise und das zeichenweise Lesen einer Datei. Wann wird welches Verfahren eingesetzt?
5. Was müssen Sie beachten, wenn Sie unter Windows Dateien lesen oder schreiben wollen?
6. Erläutern Sie die Vergabe von Zugriffsrechten für Dateien und Verzeichnissen unter Unix. Welche Perl-Funktion verwenden Sie dafür?
7. Warum kann es sinnvoll sein, den Zugriff auf eine Datei zu sperren? Erläutern Sie dies an einem Beispiel, und geben Sie die Perl-Funktion an, mit der Sie eine Datei sperren können.
8. Erklären Sie die Vorgehensweise beim Auslesen eines Verzeichnisses. Welche Perl-Funktionen verwenden Sie?

Mit Dateien und Verzeichnissen arbeiten

Übungsdatei: *Verzeichnisinhalt.txt*

Ergebnisdatei: *Übung2-E.pl*

1. Erstellen Sie ein Perl-Programm, das den Inhalt eines über die Tastatur eingegebenen Verzeichnisses in eine Datei schreibt.
2. Programmieren Sie eine Eingabemöglichkeit für den Verzeichnisnamen.
3. Öffnen Sie die Datei *verzeichnisinhalt.txt* im gleichen Verzeichnis wie das Perl-Programm im Schreibmodus. Fangen Sie mögliche Fehler ab.
4. Öffnen Sie den eingegebenen Dateipfad als Verzeichnis. Fangen Sie mögliche Fehler ab.
5. Durchlaufen Sie das Verzeichnis in einer geeigneten Schleife.
6. Es sollen nur Dateinamen beachtet werden. Filtern Sie alle anderen Einträge und Unterverzeichnisse aus.
7. Zusätzlich soll die Dateigröße ausgegeben werden. Ermitteln Sie diese mit einem Dateiprüfoperator. Rechnen Sie die in Bytes zurückgegebene Dateigröße in KByte um (Division durch 1024).
8. Schreiben Sie die Informationen in die Datei.
9. Schließen Sie Datei und Verzeichnis wieder.

In Dateien suchen

Übungsdatei: --

Ergebnisdatei: *Übung3-E.pl*

1. Erstellen Sie ein Programm, das einen Teil eines Dateinamens und einen Verzeichnisnamen entgegennimmt und beginnend ab diesem Verzeichnis nach Dateien sucht, die den Suchbegriff enthalten.
2. Programmieren Sie die Eingabe eines Dateinamens und eines Verzeichnisnamens.
3. Da die Suche rekursiv auch in allen Unterverzeichnissen durchgeführt werden soll, müssen Sie die Suche in einem Unterprogramm organisieren.
4. Das Unterprogramm soll als Argumente das zu durchsuchende Verzeichnis und den Suchbegriff erhalten.
5. Verwenden Sie im Unterprogramm nur lokale Variablen und ein unbenanntes Verzeichnis-Handle.
6. Prüfen Sie für jeden Verzeichniseintrag, ob der Suchbegriff im Dateinamen enthalten ist. Dabei soll keine Unterscheidung nach Groß- oder Kleinbuchstaben erfolgen. Stellen Sie zusätzlich sicher, dass Sie nur in Dateinamen und nicht in Verzeichnisnamen suchen. Bei einem Sucherfolg soll der komplette Dateipfad ausgegeben werden.
7. Falls Sie ein Unterverzeichnis finden, soll dieses ebenfalls durchsucht werden.
8. Fügen Sie einen Zähler als globale Variable ein, der am Ende des Programms ausgibt, wie viele Dateien gefunden wurden.

```
F:\PerlProgramme\Uebungsdateien\Kapitel_08>perl Übung3-E.pl
Bitte geben Sie den Teil eines Dateinamens ein: pl
Bitte geben Sie einen Verzeichnisnamen ein: f:\perlprogramme
f:\perlprogramme\Kapitel_02\HalloWelt.pl
f:\perlprogramme\Kapitel_03\DatenAusgabe.pl
f:\perlprogramme\Kapitel_03\Heredoc.pl
f:\perlprogramme\Kapitel_04\Typkonvertierung.pl
f:\perlprogramme\Kapitel_04\Variablen.pl
f:\perlprogramme\Kapitel_04\Variablen2.pl
f:\perlprogramme\Kapitel_04\Variablen2_Fehler.pl
f:\perlprogramme\Kapitel_04\Variablen3.pl
f:\perlprogramme\Kapitel_04\Variablen3_Fehler.pl
f:\perlprogramme\Kapitel_04\Variablen3_Fehler2.pl
f:\perlprogramme\Kapitel_04\Variablen4.pl
f:\perlprogramme\Kapitel_04\Variablen5.pl
f:\perlprogramme\Kapitel_05\dountil.pl
f:\perlprogramme\Kapitel_05\for.pl
```

Mögliche Ergebnisse der Dateisuche

9 Funktionen für Zeichenketten

In diesem Kapitel erfahren Sie

- ✓ wie Sie Zeichenketten formatiert ausgeben
- ✓ wie Sie Zeichenketten durchsuchen
- ✓ wie Sie Teile aus Zeichenketten auslesen
- ✓ wie Zeichenketten modifiziert werden

Voraussetzungen

- ✓ Perl-Funktionen, Variablen verwenden

9.1 Zeichenketten formatiert ausgeben

Sie kennen bereits die Ausgabe von Zeichenketten mit der `print`-Anweisung. Perl verfügt über zwei weitere Funktionen für diesen Zweck, die gleichzeitig das Formatieren der Ausgabe ermöglichen: `sprintf()` und `printf()`. Mit diesen Funktionen ist es beispielsweise möglich, Zahlen mit einer bestimmten Anzahl Kommastellen auszugeben.

<code>sprintf()</code>	Liefert als Rückgabewert eine nach einer Formatangabe formatierte Zeichenkette
<code>printf()</code>	Gibt eine formatierte Zeichenkette auf dem Bildschirm aus oder schreibt sie in einer Datei



Beide Funktionen formatieren eine Zeichenkette, die `sprintf`-Funktion liefert jedoch ein Ergebnis, das z. B. in einer Variablen gespeichert werden kann. Die `printf`-Funktion gibt die formatierte Zeichenkette sofort aus und ist daher ein Äquivalent für den Ausdruck `print sprintf(...);`.

Syntax der `printf`- und `sprintf`-Funktion

```
$variable = sprintf("Formatierung", Wert1, Wert2,...);
printf [DATEIHANDLE] "Formatierung", Variable1, Variable2,...;
```

- ✓ Die Funktionen `sprintf()` und `printf()` wandeln die Werte einer oder mehrerer Variablen nach den Vorgaben einer Formatierung in eine Zeichenkette.
- ✓ Das erste Argument beider Funktionen ist eine Zeichenkette mit den Formatierungsanweisungen.
- ✓ Die Formatierung kann keine, eine oder mehrere Anweisungen enthalten.
- ✓ Nach der Formatierung werden alle auszugebenden Werte angegeben und durch Kommata getrennt.
- ✓ Die Funktion `sprintf()` liefert als Rückgabewert die formatierte Zeichenkette. Die `printf`-Funktion gibt die Zeichenkette am Bildschirm aus oder schreibt sie in eine Datei.

Formatierung angeben

Eine Formatierungsanweisung besteht aus einem Prozentzeichen (%), gefolgt von einem oder mehreren Optionen, die beispielsweise den auszugebenden Datentyp und die Anzahl der Ziffern angeben. In der Formatierungsangabe der `sprintf`- bzw. `printf`-Funktion muss für jeden auszugebenden Wert eine Formatierungsanweisung angegeben werden. Die Formatierungsanweisungen werden dabei von links beginnend durch die entsprechenden Werte ersetzt.

Sie können in der Formatierungsanweisung auch weitere Texte unterbringen. Perl interpretiert nur Formatierungsanweisungen, die mit einem Prozentzeichen beginnen.



Beispiel

```
$variable = sprintf("%03.4f", $zahl);
printf "Der Preis für %d Stück beträgt %.02f.", $anzahl, $preis;
```

Typ-Angaben

Das wichtigste Element einer Formatierungsanweisung ist die Angabe des Datentyps des auszugebenden Werts. Falls der Wert einen anderen Datentyp besitzt, wird er umgewandelt. Folgende Typ-Angaben sind in Perl möglich:

Typ	Erklärung	Beispiel	Ausgabe
%%	Prozentzeichen wird ausgegeben	printf "%%";	%
%c	ASCII-Wert wird als Zeichen ausgegeben	printf "%c", 65;	A
%s	Zeichenkette wird als solche ausgegeben	printf "%s", "Hallo";	Hallo
%d	Ganzzahl mit Vorzeichen wird als Dezimalzahl ausgegeben	printf "%d", -65;	-65
%u	Ganzzahl ohne Vorzeichen wird als Dezimalzahl ausgegeben	printf "%u", 65;	65
%o	Ganzzahl ohne Vorzeichen wird als Oktalzahl ausgegeben	printf "%o", 65;	101
%x	Ganzzahl ohne Vorzeichen wird als Hexadezimalzahl ausgegeben; hexadezimale Ziffern 10 bis 15 werden als Kleinbuchstaben a bis f ausgegeben	printf "%x", 58;	3a
%X	Ganzzahl ohne Vorzeichen wird als Hexadezimalzahl ausgegeben; hexadezimale Ziffern 10 bis 15 werden als Großbuchstaben A bis F ausgegeben	printf "%X", 58;	3A
%e	Fließkommazahl wird in Exponentialschreibweise ausgegeben	printf "%e", 100.5;	1.005000e+12
%f	Fließkommazahl ausgeben	printf "%f", 100.5;	100.500000
%b	Ganzzahl ohne Vorzeichen wird als Binärzahl ausgegeben	printf "%b", 65;	1000001

Formatierung mit Flags beeinflussen

Jede Formatierungsanweisung können Sie mit so genannten Flags weiter beeinflussen. Flags sind einzelne Zeichen, mit denen Sie beispielsweise die Anzahl der Dezimalstellen oder ein Füllzeichen festlegen können. Flags werden zwischen dem Prozentzeichen % und der Typ-Angabe eingesetzt.

Flag	Erklärung	Beispiel	Ausgabe
Leerzeichen	Positiven Zahlen wird ein Leerzeichen statt dem Zeichen + vorangestellt	printf "% d", 65;	65
+	Positiven Zahlen wird das Zeichen + vorangestellt	printf "%+d", 65;	+65

Flag	Erklärung	Beispiel	Ausgabe
Zahl	Mindestanzahl an Zeichen bzw. Ziffern, fehlende Zeichen/Ziffern werden durch Leerzeichen aufgefüllt	printf "%6d", 65;	65
.Zahl	Fließkommazahlen: Anzahl der Kommastellen Zeichenketten: maximale Anzahl an Zeichen Ganzzahlen: minimale Länge	printf "%.2", 100.5;	100.50
0	Statt Leerzeichen wird die 0 zum Auffüllen verwendet	printf "%06d", 65;	000065
-	Format wird von links statt wie üblich von rechts aufgefüllt	printf "%-6d", 65;	65



Sie können mehrere Flags miteinander kombinieren, um das gewünschte Ausabeformat festzulegen. Beispielsweise bewirkt die Angabe %.02f, dass eine Fließkommazahl mit zwei Stellen nach dem Komma dargestellt wird. Fehlende Kommastellen werden dabei durch die Zahl 0 ersetzt.

Beispiel: PreisFormat.pl

Das Perl-Programm berechnet nach der Eingabe einer Anzahl den Gesamtpreis eines bestellten Artikels und gibt die Informationen formatiert aus.

```
#!/usr/bin/perl

use strict;
my $produkt="Glasvase";
my $preis=12.9;

① print "\nBestellte Anzahl: ";
my $anzahl=<STDIN>;
chomp($anzahl);

② printf "Gesamtpreis für Artikel %s: %.02f EUR\n",
$produkt, ($preis*$anzahl);
③ printf " (%d Stück zu je %.02f EUR)\n", $anzahl, $preis;

④ $gesamtpreis=sprintf("%.2f", $preis*$anzahl);
```

- ① Die Anzahl der bestellten Einheiten des Artikels wird von der Tastatur eingelesen und in der Variablen \$anzahl gespeichert.
- ② Hier wird der Gesamtpreis ausgegeben. Die Formatierungsanweisung %s gibt eine Zeichenkette aus. Als zweite Anweisung wird %.02f verwendet. Der Buchstabe f steht für die Ausgabe einer Fließkommazahl. Die Option .02 definiert zwei Kommastellen, wobei fehlende Ziffern durch den Wert 0 ersetzt werden.

```
F:\PerlProgramme\Kapitel_09>perl PreisFormat.pl
Bestellte Anzahl: 7
Gesamtpreis für Artikel Glasvase: 90.30 EUR
(7 Stück zu je 12.90 EUR)

F:\PerlProgramme\Kapitel_09>
```

Formatierte Ausgabe verschiedener Werte

- ③ Die erste Formatierungsanweisung gibt den Wert als Ganzzahl aus. Die zweite Anweisung dient wieder der Ausgabe eines Preises.
- ④ Mithilfe der Funktion `sprintf()` wird der Gesamtpreis formatiert in einer Variablen gespeichert.

9.2 Teile einer Zeichenkette suchen und ausschneiden

Um herauszufinden, ob ein Zeichen oder ein Text in einer anderen Zeichenkette enthalten sind, können Sie die Funktionen `index()` bzw. `rindex()` verwenden. Die Funktion liefert Ihnen die Zeichenposition der ersten Übereinstimmung. Die Anzahl der Zeichen einer Zeichenkette ermitteln Sie mit der Funktion `length()`.

Zusätzlich haben Sie die Möglichkeit, einen Text aus einer Zeichenkette auszuschneiden. Dafür verwenden Sie die `substr`-Funktion.

Das erste Zeichen einer Zeichenkette besitzt die Zeichenposition 0. Die letzte Zeichenposition ist daher die um den Wert 1 verringerte Länge der Zeichenkette.



Syntax der `index`- und `rindex`-Funktion

```
$position = index(Zeichenkette, Suchtext[, Startposition]);  
$position = rindex(Zeichenkette, Suchtext[, Startposition]);
```

- ✓ Die `index`-Funktion liefert die Position des ersten Auftretens des Suchtexts in der Zeichenkette. Die Funktion `rindex()` ermittelt dagegen die Position des letzten Auftretens. Beide Funktionen besitzen die gleiche Syntax.
- ✓ Das erste Argument bezeichnet die zu durchsuchende Zeichenkette.
- ✓ Das zweite Argument gibt den Suchtext bzw. das Suchzeichen an, dessen Auftreten in der Zeichenkette geprüft wird.
- ✓ Optional kann eine Startposition für den Beginn der Suche angegeben werden.

Beide Funktionen liefern als Rückgabewert die Zeichenposition der Übereinstimmung. Wird der Suchtext am Anfang der Zeichenkette gefunden lautet das Ergebnis 0. Dieser Wert wird von Perl als `false` gewertet! Ist die Suche nicht erfolgreich gewesen, dann lautet das Ergebnis -1.



Syntax der `length`-Funktion

```
$laenge = length(Zeichenkette);
```

- ✓ Die Funktion `length()` ermittelt die Anzahl der Zeichen in einer gegebenen Zeichenkette.

Syntax der `substr`-Funktion

```
$teil = substr(Zeichenkette, Startposition);  
$teil = substr(Zeichenkette, Startposition, Länge);
```

- ✓ Die `substr`-Funktion gibt einen bestimmten Teil einer Zeichenkette zurück.
- ✓ Das erste Argument ist die auszulesende Zeichenkette.
- ✓ Als zweites Argument wird die Startposition angegeben.
- ✓ Optional kann die Anzahl der auszulesenden Zeichen angegeben werden. Fehlt die Angabe, wird bis zum Ende der Zeichenkette ausgelesen.

Die substr-Funktion kann auch zum Ersetzen von Teilen einer Zeichenkette verwendet werden. In diesem Fall liefert die Funktion als Rückgabewert die ersetzenen Zeichen und verändert die Zeichenkette selbst.

```
$teil = substr($zeichenkette, Startposition, Länge, Ersetzung);
```

- ✓ Die auszulesende Zeichenkette muss als Variable übergeben werden.
- ✓ Ab der gegebenen Startposition wird eine Anzahl an Zeichen ausgelesen und durch den Text des vierten Arguments ersetzt.
- ✓ Die Funktion liefert den ausgelesenen Teil der Zeichenkette.



Mehr Möglichkeiten zum Suchen und Ersetzen von Texten in Zeichenketten bieten reguläre Ausdrücke, mit denen sich auch eine Suche mit Platzhaltern realisieren lässt.

Beispiel: *EmailZerlegung.pl*

Das Perl-Programm soll eine über die Tastatur eingegebene E-Mail-Adresse in die Bestandteile Name, Domain und Toplevel-Domain zerlegen.

```
#!/usr/bin/perl
use strict;
print "\nBitte geben Sie eine E-Mail-Adresse ein: ";
my $email=<STDIN>;
chomp($email);

① my $pos1=index($email, "@");

② if ($pos1<0) {
    print "\nDie E-Mail-Adresse $email ist ungültig!\n";
    exit;
}

③ my $name=substr($email,0,$pos1);
④ my $pos2=rindex($email, ".");
⑤ my $domain=substr($email,$pos1+1,$pos2-$pos1-1);
⑥ my $toplevel=substr($email,$pos2+1);

print "Die E-Mail-Adresse lautet: $email\n";
⑦ print "Sie besteht aus ".length($email)." Zeichen.\n\n";
⑧ print "Namensteil:\t$name\n";
print "Domain:\t$domain\n";
print "Toplevel-Domain:$toplevel\n";
```

- ① Mit der Funktion `index()` wird an dieser Stelle die Position des Zeichens @ ermittelt. Dieses Zeichen trennt den Namensteil vom Rest der E-Mail-Adresse.
- ② Wurde das Zeichen nicht gefunden, dann liefert die `index`-Funktion den Wert -1. Da in diesem Fall die E-Mail-Adresse nicht gültig ist, wird das Programm mit einer Fehlermeldung beendet.
- ③ Um den Namensteil zu ermitteln, wird mithilfe der Funktion `substr` die Zeichenfolge vom Beginn der E-Mail-Adresse bis zur Position des Zeichens @ ausgelesen.
- ④ Die Zeichenfolge zwischen dem Zeichen @ und dem letzten Punkt in der E-Mail-Adresse wird als Domänenname bezeichnet. Die Funktion `rindex()` findet die Position des letzten Punktes in der E-Mail-Adresse.

- ⑤ Mit den zwei ermittelten Positionsangaben kann der Domainname ausgelesen werden. Die Anzahl der auszulesenden Zeichen ist dabei die Differenz zwischen beiden Positionen.
- ⑥ Alle Zeichen nach dem letzten Punkt bezeichnen die Toplevel-Domain.
- ⑦ Die Gesamtzahl an Zeichen in der E-Mail-Adresse wird mit der length-Funktion ausgegeben.
- ⑧ Hier erfolgt die Ausgabe der bei der Zerlegung ermittelten Bestandteile der E-Mail-Adresse. Die Ausgabe wird mithilfe von Tabulatoren (\t) formatiert.

```
F:\PerlProgramme\Kapitel_09>perl EmailZerlegung.pl
Bitte geben Sie eine E-Mail-Adresse ein: info@rjs.de
Die E-Mail-Adresse lautet: info@rjs.de
Sie besteht aus 11 Zeichen.

Namensteil:           info
Domain:               rjs
Toplevel-Domain:     de

F:\PerlProgramme\Kapitel_09>
```

Zerlegung einer E-Mail-Adresse

9.3 Zeichenketten modifizieren und umwandeln

Letztes Zeichen entfernen

Mit den Funktionen `chop()` und `chomp()` können Sie das letzte Zeichen einer Zeichenkette entfernen. Während die Funktion `chop()` in jedem Fall das letzte Zeichen entfernt, egal um welches es sich dabei handelt, entfernt die Funktion `chomp()` nur Separatorzeichen. Das Separatorzeichen wird dabei von der vordefinierten Variablen `$/` bestimmt. Standardmäßig wird nur der Zeilenumbruch `\n` entfernt.

Syntax der `chomp-` und `chop-`Funktion

```
chop($zeichenkette);
chomp($zeichenkette);
```

- ✓ Die Funktion `chop()` entfernt das letzte Zeichen einer Zeichenkette.
- ✓ Die Funktion `chomp()` entfernt das letzte Zeichen nur dann, wenn es sich um ein definiertes Separatorzeichen handelt.
- ✓ Beide Funktionen modifizieren die als Argument übergebene Zeichenkettenvariable.

Schreibweise ändern

Mit der Perl-Funktion `uc()` können Sie alle Kleinbuchstaben einer Zeichenkette in Großbuchstaben umwandeln. Im Gegensatz dazu wandelt die Funktion `lc()` alle Großbuchstaben in Kleinbuchstaben um.

Mit den Funktionen `ucfirst()` und `lcfirst()` wird statt der gesamten Zeichenkette nur das erste Zeichen verändert.

Alle vier Funktionen können keine Umlaute und den Buchstaben ß umwandeln. Diese Zeichen werden unverändert übernommen.



Syntax der `uc-`, `lc-`, `ucfirst-` und `lcfirst-`Funktion

```
$variable = uc(Zeichenkette);
$variable = lc(Zeichenkette);
$variable = ucfirst(Zeichenkette);
$variable = lcfirst(Zeichenkette);
```

- ✓ Die Funktion `uc()` bzw. `lc()` wandelt alle Kleinbuchstaben in Großbuchstaben bzw. umgekehrt.
- ✓ Die Funktionen `ucfirst()` und `lcfirst()` arbeiten wie die `uc-` bzw. `lc-`Funktion, wandeln jedoch nur das erste Zeichen der Zeichenkette um.
- ✓ Alle Funktionen liefern die modifizierte Zeichenkette als Rückgabewert.

9.4 Weitere Funktionen für Zeichenketten

Nachfolgend finden Sie eine Auflistung weiterer Zeichenketten-Funktionen, die jedoch nicht weiter betrachtet werden, da ihr Einsatz recht selten ist:

Funktion	Erklärung
<code>reverse(Zeichenkette)</code>	Dreht eine Zeichenkette Zeichen für Zeichen um
<code>crypt(Zeichenkette, Schlüssel)</code>	einseitige Verschlüsselung der Zeichenkette; die Verschlüsselung ist nicht umkehrbar und wird beispielsweise unter Unix zum Speichern von Passwörtern verwendet
<code>chr(Zeichenwert)</code>	Liefert das entsprechende Zeichen eines ASCII-Werts
<code>ord(Zeichen)</code>	Liefert den ASCII-Wert eines Zeichens

9.5 Schnellübersicht

Sie möchten ...	
einen Wert formatiert ausgeben	<code>printf "Formatierung", Wert1, Wert2, ...;</code>
einen Wert formatiert speichern	<code>\$variable=sprintf("Formatierung", Wert1, Wert2, ...);</code>
eine Zahl mit zwei Kommastellen ausgeben	<code>printf "%,.02f", Wert;</code>
die erste Position eines Zeichens/ Texts in einer Zeichenkette ermitteln	<code>index(Zeichenkette, Suchtext);</code>
die letzte Position eines Zeichens/ Texts in einer Zeichenkette ermitteln	<code>rindex(Zeichenkette, Suchtext);</code>
die Länge einer Zeichenkette ermitteln	<code>length(Zeichenkette);</code>
einen Teil aus einer Zeichenkette auslesen	<code>substr(Zeichenkette, Startposition, Länge);</code>
eine Zeichenkette in Großbuchstaben umwandeln	<code>uc(Zeichenkette);</code>
eine Zeichenkette in Kleinbuchstaben umwandeln	<code>lc(Zeichenkette);</code>

9.6 Übungen

Zahlen einlesen und ausgeben

Übungsdatei: --**Ergebnisdatei:** *Übung1-E.pl*

1. Erstellen Sie ein Perl-Programm, das zwei Zahlen von der Tastatur einliest.
2. Dividieren Sie die erste Zahl durch die zweite Zahl.
3. Erzeugen Sie eine Ausgabe mithilfe der `printf`-Funktion, die die beiden eingegebenen Zahlen und das Ergebnis ausgibt.
4. Erstellen Sie nun eine Ausgabe, in der nur das Ergebnis mit drei Nachkommastellen angegeben wird.
5. Geben Sie das Ergebnis zuletzt in der Exponentialschreibweise an.

Zahlen einlesen und umwandeln

Übungsdatei: --**Ergebnisdatei:** *Übung2-E.pl*

1. Erstellen Sie ein Perl-Programm, das eine Dezimalzahl von der Tastatur einliest.
2. Geben Sie diese Zahl in hexadezimaler, oktaler und binärer Schreibweise aus.
3. Verändern Sie das Programm so, dass Sie mehrere Werte eingeben und umwandeln können. Das Programm soll mit der Eingabe einer 0 beendet werden.

Zeichenkette einlesen und umwandeln

Übungsdatei: --**Ergebnisdatei:** *Übung3-E.pl*

1. Erstellen Sie ein Perl-Programm, das die Eingabe einer beliebigen Zeichenkette erlaubt.
2. Bestimmen Sie die Länge der Zeichenkette, und geben Sie diese aus.
3. Ermitteln Sie das erste und letzte Wort der Zeichenkette, und geben Sie das Wort in Großbuchstaben aus.

Web-Adressen einlesen und zerlegen

Übungsdatei: --**Ergebnisdatei:** *Übung4-E.pl*

1. Erstellen Sie ein Perl-Programm, das die Eingabe einer Web-Adresse erlaubt.
2. Es wird davon ausgegangen, dass die Adresse in der Form `http://subdomain.domainname.toplevel` eingegeben wird.
3. Trennen Sie zuerst den Vorsatz `http://` von der Web-Adresse.
4. Zerlegen Sie dann die Web-Adresse in die entsprechenden Teile, und geben Sie Subdomain, Domainname und Toplevel-Domain aus.

10 Reguläre Ausdrücke

In diesem Kapitel erfahren Sie

- ✓ was reguläre Ausdrücke sind
- ✓ wie Suchmuster für reguläre Ausdrücke definiert werden
- ✓ wie Sie Texte nach komplexen Suchmustern durchsuchen
- ✓ wie Sie mithilfe von regulären Ausdrücken Textteile suchen und ersetzen

Voraussetzungen

- ✓ Arbeiten mit Variablen und Operatoren

10.1 Suchen und Ersetzen mit regulären Ausdrücken

Perl wurde ursprünglich zum Verarbeiten umfangreicher Textdaten entwickelt, wie sie beispielsweise in Logdateien oder Quelltexten von Programmen entstehen. In diesen Textdaten soll nach bestimmten Mustern gesucht und ersetzt werden können. Um Textteile genau beschreiben zu können, werden in Perl so genannte reguläre Ausdrücke (engl. regular expressions) verwendet.

Auch mit Zeichenkettenfunktionen wie der `index`-Funktion ist es möglich, nach dem Vorkommen eines Wortes oder Zeichens in einem Text zu suchen. Reguläre Ausdrücke erlauben jedoch das Definieren von Mustern, die auch auf verschiedene Textteile zutreffen können. So können Sie mit regulären Ausdrücken beispielsweise mit einer Anweisung alle durch spitze Klammern `<>` begrenzte HTML-Befehle aus einer Datei filtern. In vielen anderen Sprachen würde dies umfangreichen Programmcode bedeuten.



Ein regulärer Ausdruck ist ein genau definiertes Suchmuster, mit dem Sie Inhalte aus Variablen oder Textdateien herausziehen oder ersetzen können. Im Gegensatz zu normalen Zeichenkettenfunktionen bieten sie deutlich mehr Freiheiten beim Vergleichen von Textstellen.

Reguläre Ausdrücke wurden nicht für Perl erfunden. Auch viele andere Programme unter Unix verwenden diese flexible Möglichkeit zum Suchen und Ersetzen von Textteilen, so z. B. der Editor `ed` oder das Suchprogramm `grep`.

Suchmuster planen

Ein Suchmuster für einen regulären Ausdruck kann eine Anzahl fester Zeichen umfassen. Zusätzlich können Sie jedoch auch Escape-Zeichen und besondere Steuerzeichen angeben, die beispielsweise für beliebige andere Zeichen stehen oder die Wiederholung eines Zeichens festlegen.

Wenn Sie in einem Text nach der Zeichenfolge `AND` suchen wollen, dann umfasst das Suchmuster genau diese Zeichen in der geforderten Schreibweise. In regulären Ausdrücken können Sie jedoch auch nur Bruchteile eines vollständig gesuchten Musters angeben, z. B. wenn Sie nur wissen, dass der gesuchte Text mit einem `A` beginnt, mit `ND` endet und mindestens drei Zeichen enthält.

Wenn Sie ein Suchmuster erstellen wollen, sollten Sie überlegen, wie die gesuchte Zeichenfolge aufgebaut ist. Tritt sie nur einmal auf, oder ist es eine wiederkehrende Zeichenfolge. Gibt es in der Zeichenfolge redundante Teile, die in allen Fundstellen auftauchen, können Sie das Muster so aufbauen, dass nur nach diesen Teilen gesucht wird und der Rest ignoriert wird.

10.2 Mustererkennung mit regulären Ausdrücken

Einfache Suchmuster definieren

Suchmuster für reguläre Ausdrücke werden in Perl in Slashes `/` eingeschlossen. Innerhalb des Suchmusters können Sie Buchstaben und Ziffern verwenden. Alle anderen Zeichen, z. B. `*` oder `?`, haben eine besondere Bedeutung. Wollen Sie dennoch nach diesen Zeichen suchen, müssen Sie ihnen einen Backslash `\` voranstellen.

Ein besonderes Zeichen ist der Punkt `.`. Er ist ein Platzhalter für ein beliebiges Zeichen: Buchstabe, Ziffer oder Sonderzeichen.

Die folgende Tabelle zeigt einige einfache Beispiele für Suchmuster in regulären Ausdrücken:

Suchmuster	Erklärung
<code>/Welt/</code>	Sucht nach dem Auftreten des Wortes <i>Welt</i>
<code>/Hallo Welt/</code>	Sucht nach dem Auftreten der Zeichenfolge <i>Hallo Welt</i>
<code>\(c\)/</code>	Sucht nach dem Auftreten der Zeichenfolge <i>(c)</i>
<code>/14\..5/</code>	Sucht nach der Zahl <i>14.5</i>
<code>/\$text/</code>	Sucht mit dem Inhalt der Variablen <code>\$text</code> als Suchmuster
<code>/b.n/</code>	Sucht nach einem b gefolgt von einem beliebigen Zeichen und einem n, z. B. bin, Bund, Band, binde usw.
<code>/Welt/i</code>	Sucht nach dem Wort <i>Welt</i> ohne Groß- oder Kleinschreibung zu beachten

Damit bei der Suche nicht zwischen Groß- und Kleinschreibung unterschieden wird, können Sie an den regulären Ausdruck die Option `i` anfügen: `/Muster/i`. Mit der Option `/Muster/o` erreichen Sie, dass der reguläre Ausdruck nur einmal im Programmablauf kompiliert wird. Vor allem wenn Sie die Mustererkennung in einer Schleife verwenden, kann dies Rechenzeit sparen. Beide Optionen können Sie auch kombinieren.

Folgende Zeichen besitzen in regulären Ausdrücken eine besondere Bedeutung und müssen bei der Verwendung mit einem Backslash gekennzeichnet werden:

<code>/ . + * ? ^ \$ () [] { }</code>

Suchmuster verwenden

Um Suchmuster auf eine Variable anzuwenden, werden in Perl so genannte **Match-Operatoren** (engl. match = übereinstimmen) verwendet:

<code>=~</code>	Prüft, ob der angegebene reguläre Ausdruck in einer Variablen gefunden wird. Ist die Suche erfolgreich, besitzt der Ausdruck den Wert <code>true</code> , andernfalls den Wert <code>false</code> .
<code>!~</code>	Prüft, ob der angegebene reguläre Ausdruck in einer Variablen nicht vorhanden ist. War die Suche nicht erfolgreich, dann besitzt der Ausdruck den Wert <code>true</code> , andernfalls den Wert <code>false</code> .

Mit den Match-Operatoren erstellen Sie einen Ausdruck, der einen Wahrheitswert liefert. Diesen Ausdruck können Sie beispielsweise in einer `if`-Abfrage verwenden.

Syntax einer Suche mit regulären Ausdrücken

```
$variable =~ /Suchmuster/; # Prüft auf das Vorhandensein des Suchmusters
$variable !~ /Suchmuster/; # Prüft auf das Nicht-Vorhandensein des Suchmusters
```

- ✓ Für das Definieren einer Suche mit regulären Ausdrücken werden die Match-Operatoren `=~` oder `=!` verwendet.
- ✓ Auf der linken Seite des Operators wird die zu durchsuchende Variable angegeben. Rechts des Operators erfolgt die Angabe des Suchmusters.
- ✓ Der gesamte Ausdruck liefert einen Wahrheitswert, der beispielsweise in einer Abfrage verwendet werden kann.

Beispiel

In einer Variablen wird der Link auf eine Webseite gespeichert. Mit einem regulären Ausdruck soll geprüft werden, ob der Link den Ausdruck `http://` enthält.

```
$link = "http://www.herd़t.com/";
if ($link =~ /http:\:\/\//) {
    print "Der Link $link verweist auf eine Webseite.";
} else {
    print "Der Link $link scheint nicht auf eine Webseite zu verweisen";
}
```



Reguläre Ausdrücke eignen sich vor allem für Suchausdrücke mit variablen Mustern. Wenn Sie nach einzelnen Zeichen oder festen Teilen einer Zeichenkette suchen wollen, ist die Perl-Funktion `index` die bessere Wahl.

10.3 Optionen für reguläre Ausdrücke

Zeichenklassen verwenden

Reguläre Ausdrücke entwickeln ihr Stärken besonders bei der Suche nach Mustern, die auf verschiedene Möglichkeiten passen. Um solche Suchmuster zu definieren, können Sie so genannte Zeichenklassen verwenden.

Zeichenklassen werden in eckige Klammern eingeschlossen und legen eine Liste möglicher Entsprechungen für genau ein Zeichen fest. Wenn Sie beispielsweise nach den Zahlen 12, 13 oder 14 suchen wollen, können Sie die zweite Ziffer durch die Zeichenklasse `[234]` darstellen. Die einzelnen Zeichen innerhalb der eckigen Klammern werden dabei quasi durch ein logisches ODER verbunden.



Wenn Sie größere Zeichenbereiche in einer Zeichenklasse angeben wollen, genügt es, wenn Sie das erste und letzte Zeichen des Bereichs angeben und mit einem Bindestrich verbinden.

Die folgende Tabelle zeigt einige Beispiele für die Verwendung von Zeichenklassen innerhalb von Suchmustern:

Suchmuster	Erklärung
/200 [012] /	Sucht nach den Zahlen 2000, 2001 und 2002
/201 [0-9] /	Sucht nach den Zahlen 2010 bis 2019
/m[ui]t/	Sucht nach den Wörtern mit und mut

Suchmuster	Erklärung
/b [uai]n[dt]/	Sucht nach den Wörtern bunt, bind, band, bund usw.
/\. [\n\r] . /	Sucht nach einem Punkt, dem ein Zeilenvorschub oder Wagenrücklauf als Zeichen für ein Zeilenende und danach ein beliebiges Zeichen folgt
/ [Pp]erl/	Sucht nach den Wörtern Perl und perl
/ [aeiou] /	Prüft, ob ein kleingeschriebener Vokal enthalten ist

Zeichen ausschließen

Wenn Sie wollen, dass an einer Stelle des Suchmusters alle Zeichen bis auf eine bestimmte Auswahl vorkommen dürfen, dann können Sie eine Zeichenklasse negieren. Dazu geben Sie als erstes Zeichen innerhalb der eckigen Klammern das Ausschlusszeichen `\^` an.

Beispiel

Ein Suchmuster soll alle Jahreszahlen zwischen 2000 und 2009 außer den Jahren 2001 und 2004 finden.

```
$jahr =~ /200[^14]/;
```

Abkürzungen für Zeichenklassen

Perl stellt einige Abkürzungen für häufig verwendete Zeichenklassen zur Verfügung, die durch einen Buchstaben und einen vorangestellten Backslash `\` gekennzeichnet werden. Diese Abkürzungen stehen wiederum für ein einzelnes Zeichen, werden jedoch ohne eckige Klammern verwendet.

Abkürzung	Erklärung	Überschrift
\d	Steht für eine beliebige Ziffer	[0-9]
\D	Steht für ein Zeichen, das keine Ziffer ist	[^0-9]
\w	Steht für ein Zeichen, das ein Buchstabe, eine Ziffer oder ein Unterstrich ist (je nach System werden auch Umlaute eingeschlossen)	[a-zA-Z0-9_]
\W	Steht für ein Zeichen, das weder Buchstabe noch Ziffer noch Unterstrich ist (je nach System werden auch Umlaute ausgeschlossen)	[^a-zA-Z0-9_]
\s	Steht für ein Leerzeichen, einen Zeilen- oder Seitenumbruch bzw. einen Tabulator (so genannte Whitespace-Zeichen)	[\n\r\t\f]
\S	Steht für ein beliebiges Zeichen, das kein Steuerzeichen und kein Leerzeichen ist	[^\n\r\t\f]

Beispiel

```
$zahl =~ /\d\d\d\d/;      # vierstellige Zahl
$zahl =~ /\d\d\.\d\d/;    # Zahl mit zwei Stellen vor und nach dem Dezimalpunkt
$text =~ /\w\s\w/;        # Steuerzeichen von zwei normalen Zeichen umschlossen
```

Wiederholungen angeben

Wenn ein bestimmtes Zeichen im Suchmuster mehrfach vorkommen soll, können Sie dies mit verschiedenen Sonderzeichen, den so genannten Quantoren, angeben. Dabei ist es möglich, festzulegen, wie oft das Zeichen mindestens oder maximal hintereinander vorkommen soll.

Der Quantor wird nach dem Zeichen angegeben, dessen Wiederholung er festlegt. Der Quantor selbst steht nicht für ein Zeichen im Suchmuster.

Folgende Quantoren für reguläre Ausdrücke stehen unter Perl zur Verfügung:

<code>X*</code>	Das Zeichen X ist gar nicht, einmal oder mehrfach vorhanden.
<code>X+</code>	Das Zeichen X ist einmal oder mehrfach vorhanden.
<code>X?</code>	Das Zeichen X ist gar nicht oder maximal einmal vorhanden.
<code>X{n,m}</code>	Das Zeichen X ist mindestens n-mal und maximal m-mal vorhanden.

Quantoren können auch für Zeichenklassen, Abkürzungen von Zeichenklassen oder den universellen Platzhalter `.` verwendet werden. Erst im Zusammenspiel dieser verschiedenen Möglichkeiten entwickeln reguläre Ausdrücke ihre vollständigen Möglichkeiten.

Beispiele für reguläre Ausdrücke mit Wiederholungen

Die folgende Tabelle zeigt einige Suchmuster für reguläre Ausdrücke, die Wiederholungen verwenden.

Suchmuster	Erklärung
<code>/ [A-Z] .+ /</code>	Großbuchstabe gefolgt von mindestens einem beliebigen Zeichen
<code>/<[^>]+>/</code>	Öffnende spitze Klammer, danach mindestens ein Zeichen, das keine schließende spitze Klammer ist, gefolgt von einer spitzen Klammer: sucht nach HTML-Tags in der Form <...>
<code>/<html>.*</html>/i</code>	Bereich mit beliebigen Zeichen, der von den HTML-Tags <code><html></code> und <code></html></code> eingeschlossen ist. Es wird nicht zwischen Groß- und Kleinschreibung unterschieden.
<code>/\s\d{2,4}\s/</code>	Eine Zahl mit mindestens zwei und maximal vier Ziffern, die von Leerzeichen oder anderen Steuerzeichen eingeschlossen ist.
<code>/[a-z]+\d+/</code>	Kleinbuchstabe (oder auch keiner) gefolgt von mindestens einer Ziffer

Alle Quantoren versuchen so viele Zeichen wie möglich in das Suchmuster einzupassen. Wenn Sie beispielsweise das Suchmuster `/<html>.+</html>/` verwenden, kann der durchsuchte Text das HTML-Tag `</html>` auch mehrfach enthalten. In diesem Fall passt der reguläre Ausdruck auf alle Zeichen bis zum letzten Vorkommen von `</html>`. Dieses Verhalten der Perl-Quantoren wird auch greedy (engl. greedy = gierig) genannt.

Beispiel: *HTMLCheck.pl*

Das Perl-Programm übernimmt einen Dateinamen von der Kommandozeile und prüft mit einem regulären Ausdruck, ob die Datei HTML-Tags enthält. Ist dies der Fall, wird eine entsprechende Meldung ausgegeben.

```

#!/usr/bin/perl
use strict;

① my $datei=$ARGV[0];
② if (!$datei) {
    print "Prüft Datei auf enthaltene HTML-Tags\n\n";
③     print "$0 Dateiname\n";
} else {
④     open(DATEI,<$datei") ||
        die("Datei $datei kann nicht geoeffnet werden!");
⑤     my @datei=<DATEI>;
⑥     my $inhalt=join("",@datei);
⑦     if ($inhalt=~/<[^>]+>/) {
        print "Die Datei $datei enthält HTML-Tags.\n
            Es scheint sich um eine HTML-Datei zu handeln.\n\n";
    } else {
        print "Die Datei $datei enthält keine HTML-Tags.\n\n";
    }
    close(DATEI);
}

```

- ① Über das Array `@ARGV` werden Parameter der Kommandozeile an das Programm übergeben. Da nur der erste Parameter von Bedeutung ist, wird er über den Indexwert 0 ausgelesen und in der Variablen `$datei` gespeichert.
- ② Falls kein Dateiname übergeben wurde, soll eine kurze Hilfe zur Verwendung des Programms angezeigt werden.
- ③ Der aktuelle Name des Perl-Programms wird durch die von Perl vordefinierte Variable `$0` zurückgegeben.
- ④ Die zu überprüfende Datei wird an dieser Stelle zum Lesen geöffnet.
- ⑤ Der gesamte Inhalt der Datei wird in einem Array eingelesen.
- ⑥ Die Datei soll vollständig und nicht zeilenweise nach HTML-Tags überprüft werden. Zu diesem Zweck wird das Array mit der `join`-Funktion in eine skalare Variable umgewandelt. Die Variable enthält danach alle Zeilen hintereinander.
- ⑦ Mit einem regulären Ausdruck wird überprüft, ob die Datei mindestens einen regulären Ausdruck enthält. Je nach Ergebnis der Überprüfung wird eine entsprechende Meldung ausgegeben.

```

F:\PerlProgramme\Kapitel_10>perl HtmlCheck.pl beispiel.
txt
Die Datei beispiel.txt enthält keine HTML-Tags.

F:\PerlProgramme\Kapitel_10>perl HtmlCheck.pl webseitedemo.txt
Die Datei webseitedemo.txt enthält HTML-Tags.
Es scheint sich um eine HTML-Datei zu handeln.

F:\PerlProgramme\Kapitel_10>

```

Verschiedene Ergebnisse der HTML-Prüfung

Finden der erstmöglichen Übereinstimmung

Reguläre Ausdrücke suchen stets die erste mögliche Übereinstimmung des Suchmusters mit dem zu durchsuchenden Text. Dabei spielt es keine Rolle, ob das Suchmuster mehrfach gefunden werden kann.

Dieses Verhalten kann besonders beim Verwenden des Quantors * zu Problemen führen. Bei diesem Quantor kann das vorangestellte Zeichen beliebig oft oder gar nicht vorhanden sein. Unter bestimmten Voraussetzungen ist das Suchmuster daher stets wahr.

Beispiel

```
$text="Hallo Welt!";
$text =~ /w*/; # Suchmuster wird gefunden, da w auch nicht vorhanden sein darf
```

Verankerung von Mustern

Beim Anwenden eines regulären Ausdrucks wird das Suchmuster von Anfang bis Ende des zu durchsuchenden Texts bewegt. Die Suche ist erfolgreich, wenn ein Teil des Textes an einer beliebigen Stelle zu dem verwendeten Suchmuster passt.

Wenn Sie festlegen wollen, dass ein bestimmtes Muster nur am Anfang oder am Ende des Textes oder an einer Wortgrenze beginnt, dann können Sie Perls Musteranker verwenden.

Folgende Anker stehen Ihnen bei der Arbeit mit regulären Ausdrücken zur Verfügung:

^	Der Anker steht für den Beginn des Textes.
\$	Der Anker steht für das Ende des Textes.
\b	Der Anker steht für eine Wortgrenze.
\B	Der Anker steht für das Innere eines Wortes (Gegenstück zu \b).



Der Musteranker `^` wird stets am Beginn des Suchmusters angegeben, der Musteranker `$` dagegen am Ende. Eine Wortgrenze mit dem Anker `\b` bezeichnet den Übergang eines Buchstabens, einer Ziffer oder Unterstrichs (Zeichenklasse `\w`) zu einem Leerzeichen oder Sonderzeichen (Zeichenklasse `\s`).

Beispiele für Musterverankerungen

Die folgende Tabelle zeigt einige Suchmuster für reguläre Ausdrücke, die Musteranker verwenden.

Suchmuster	Erklärung
/^ [A-Z] . . . /	Wort am Anfang des Texts mit vier Buchstaben, das erste Zeichen muss ein Großbuchstabe sein.
/Hallo.+Welt\$/	Das Wort <i>Hallo</i> gefolgt von mindestens einem beliebigen Zeichen, wiederum gefolgt vom Wort <i>Welt</i> am Ende des Texts.
/\bbin\b/	Findet das Wort <i>bin</i> als allein stehendes Wort, das z. B. in Leerzeichen eingeschlossen ist.
/\Bbin\B/	Findet den Wortteil <i>bin</i> nur innerhalb eines Wortes, z. B. in <i>verbinden</i> , <i>anbinden</i> usw.
/\bWelt\$/	Das Wort <i>Welt</i> allein stehend am Ende des Texts, davor muss sich eine Wortgrenze befinden.
/^Welt\$/	Der durchsuchte Text darf nur aus dem Wort <i>Welt</i> bestehen.

Sie können die Musteranker nicht nur einzeln, sondern auch gemeinsam mit anderen für ein bestimmtes Suchmuster verwenden. Damit können Sie ein Suchmuster besonders sicher beschreiben.

Beachten Sie, dass der Musteranker `\A` nicht mit dem Ausschlusszeichen `\A` innerhalb einer Zeichenklasse übereinstimmt.



Alternativen angeben

Ähnlich wie Sie mittels Zeichenklassen mehrere Alternativen für ein einzelnes Zeichen angeben, können Sie auch mehrere Wörter als Alternativen innerhalb eines Suchmusters verwenden. Die einzelnen Wörter werden mit dem Pipe-Zeichen `|` in der Form einer logischen ODER-Verknüpfung verbunden.

Beispiel

```
$browser =~ /Netscape|Mozilla/i;      # Findet entweder Netscape oder Mozilla
$jahr =~ /2015|2016|2017/;            # Findet entweder 2015, 2016 oder 2017
```

oder

```
$browser =~ /Netscape/ || $browser =~ /Mozilla/;
$jahr =~ /2015/ || $jahr =~ /2016/ || $jahr =~ /2017/;
```

Mit der Angabe von Alternativen können Sie die Suche mit verschiedenen Suchmustern in einem regulären Ausdruck vereinen. Die einzelnen Alternativen können auch Zeichenklassen oder Wiederholungen enthalten.



```
$browser =~ / [Nn]etscape | [Mm]ozilla/;
```

10.4 Übereinstimmungen speichern

Perl besitzt die Fähigkeit, Teile eines regulären Ausdrucks beim Übereinstimmen mit dem Suchmuster zu speichern. Dies ist besonders dann sinnvoll, wenn Sie ein Suchmuster verwenden, das verschiedene Übereinstimmungen zulässt. Die gefundene Stelle im durchsuchten Text lässt sich dann speichern und im weiteren Programmverlauf oder innerhalb des regulären Ausdrucks verwenden.

Mit runden Klammern geben Sie die Teile des Suchmusters an, deren Übereinstimmung gespeichert werden soll. Auf die gespeicherten Übereinstimmungen greifen Sie mit den speziellen Variablen `$1` für den Inhalt der ersten Klammern, `$2` für den Inhalt der zweiten Klammer usw. zu.

Beispiel

Eine Datumsangabe wird mithilfe eines regulären Ausdrucks in seine Bestandteile zerlegt.

```
$datum="Leipzig, den 01.04.2016";
$datum =~ /(.*), den (\d\d)\.(\d\d)\.(\d\d\d\d)/;
print "$1\n"; # Leipzig
print "$2\n"; # 01
print "$3\n"; # 04
print "$4\n"; # 2016
```

Sie können die gefundenen Übereinstimmungen auch innerhalb des regulären Ausdrucks verwenden. In diesem Fall werden sie nicht mit `$1`, `$2` usw. gekennzeichnet sondern durch `\1`, `\2` usw. Dabei bestimmt die Ziffer wieder das Klammpenpaar, auf das Bezug genommen wird.

Beispiel: DatumFinden.pl

Das Perl-Programm soll eine Textdatei nach Datumswerten im Jahr 2015, 2016 oder 2017 durchsuchen, bei denen Tag und Monat gleich sind, beispielsweise 05.05.2017. Durch einen Tabulator getrennt, folgt auf jedes Datum eine Ortsangabe, die ebenfalls ausgelesen werden soll.

```
#!/usr/bin/perl
use strict;

my $datei="beispiel.txt";
open(DATEI,"<$datei") || die("Datei $datei kann nicht geoeffnet werden!");
① while(<DATEI>) {
②   if (/^(\d\d)\.\1\.(201[567])\t(.+)$/) {
③     print "Datum: $1.$1.$2 Ort: $3\n";
   }
close(DATEI);
```

- ① Die Datei wird in einer `while`-Schleife zeilenweise ausgelesen. Da keine Variable zum Speichern der Daten angegeben wurde, verwendet Perl standardmäßig die Variable `$_`.
- ② An dieser Stelle erfolgt die Anwendung des Suchmusters. Wird dabei keine zu prüfende Variable angenommen, verwendet Perl auch hier standardmäßig `$_`. Das Suchmuster enthält verschiedene Klammerungen, um die zu suchenden Bestandteile zu gliedern.
- ③ Wurde eine Übereinstimmung gefunden, erfolgt an dieser Stelle die Ausgabe des Datums und des Orts. Die vordefinierten Variablen `$1`, `$2` und `$3` liefern dabei die gefundenen Textstellen.

Das Suchmuster des regulären Ausdrucks im Beispiel verwendet verschiedene Techniken für die Beschreibung der zu suchenden Daten:

/^	(\d\d)	\.	\1	\.	(201[567])	\t	(.+)	\$/
Beginn der Zeile	Zahl mit zwei Ziffern	Punkt	Suchergebnis der ersten Klammer	Punkt	Ziffern 201 gefolgt von 5, 6 oder 7	Tabulator	Mindestens ein und unbegrenzt viele Zeichen	Ende der Zeile
	<code>\$1 bzw. \1</code>				<code>\$2</code>		<code>\$3</code>	

Weitere Informationen zu Übereinstimmungen bei der Suche

Außer dem Zugriff auf gespeicherte Übereinstimmungen liefert Perl mit verschiedenen vordefinierten Variablen noch weitere Informationen über den Text, auf den ein Suchmuster passte.

<code>\$&</code>	Liefert den gesamten übereinstimmenden Teil eines Textes
<code>\$`</code>	Liefert alle Zeichen vor dem übereinstimmenden Suchmuster
<code>\$`</code>	Liefert alle Zeichen, die auf das übereinstimmende Suchmuster folgen

Gruppieren mit Klammern

Mit runden Klammern können Sie nicht nur Übereinstimmungen für eine spätere Verwendung speichern, sondern auch das Suchmuster gruppieren. Auf diese Weise können Sie Quantoren nicht nur für einzelne Zeichen, sondern für Gruppen von Zeichen verwenden. Der Quantor wird dabei an die schließende runde Klammer angegeschlossen und gibt an, wie oft die Zeichen innerhalb der Gruppierung wiederholt werden sollen.

Beispiel

```
/ (Welt){4} /          # Welt vier Mal hintereinander
/Jahr.*(2016|2017)+/
# 2016 od. 2017 mind. einmal und beliebig oft hintereinander
```

Mit Gruppierungen können Sie besonders umfangreiche Suchmuster in zusammengehörige Bereiche gliedern. Dies erhöht die Übersichtlichkeit und hilft beim nachträglichen Ändern von Suchmustern.



Anzahl übereinstimmender Zeichen einschränken

Perls Quantoren für wiederholte Zeichen und Zeichengruppen verhalten sich grundsätzlich *greedy*, d. h., sie nehmen so viele Zeichen auf wie möglich, damit der reguläre Ausdruck gerade noch gilt. Besonders beim Speichern von Übereinstimmungen kann dieses Verhalten jedoch zu Fehlern führen.

Mit einem Fragezeichen **?**, das Sie dem Quantor nachstellen, können Sie das so genannte Non-Greedy-Verhalten erzwingen. Dann nimmt der Quantor nur noch so viele Zeichen auf, bis das Suchmuster zum ersten Mal eine Übereinstimmung findet.

Beispiel: *NonGreedy.pl*

Ein Teil einer HTML-Datei soll durchsucht werden, um den Text zwischen den HTML-Tags **** und **** zu ermitteln.

```
#!/usr/bin/perl
use strict;

my $html = "Ein <b>Beispiel</b> für das <b>Non-Greedy-Verhalten</b> von
Perl.";
① $html =~ /<b>(.+)</b>/i;
print "$1\n";
② $html =~ /<b>(.+?)</b>/i;
print "$1\n";
```

- ① Hier erfolgt die Anwendung des Quantors **+**. Es werden alle Zeichen bis zum letzten Auftreten von **** eingelesen.
- ② Durch das Anfügen eines Fragezeichens wird an dieser Stelle das Non-Greedy-Verhalten erzwungen. Der Quantor nimmt nur so viele Zeichen auf, bis das HTML-Tag **** zum ersten Mal gefunden wird.

```
F:\PerlProgramme\Kapitel_10>perl NonGreedy.pl
Beispiel</b> für das <b>Non-Greedy-Verhalten
Beispiel

F:\PerlProgramme\Kapitel_10>
```

Greedy- und Non-Greedy-Verhalten im Vergleich

Falls Sie bei einem regulären Ausdruck fehlerhafte Übereinstimmungen erhalten, sollten Sie das Non-Greedy-Verhalten erzwingen oder das Suchmuster genauer definieren.



10.5 Regeln für das Verwenden regulärer Ausdrücke

Besonders in komplizierten regulären Ausdrücken ist es schwierig, Fehler zu finden. Daher sollten Sie bereits beim Planen der Suchmuster beachten, wie Perl reguläre Ausdrücke verarbeitet. Die folgenden Regeln sollen Ihnen dabei helfen:

- ✓ Die Suche wird von links nach rechts durchgeführt. Sie beginnt mit dem ersten Zeichen eines Texts und wird bei der Übereinstimmung abgebrochen. In diesem Fall ist das Ergebnis des regulären Ausdrucks `true`, sonst `false`.
- ✓ Variablen innerhalb des Suchmusters werden vor der Suche durch ihren jeweiligen Inhalt ersetzt. Falls die Variable Sonderzeichen enthält, müssen diese durch einen vorangestellten Slash kenntlich gemacht werden. Die Variable kann auch Quantoren oder Zeichenklassen enthalten.
- ✓ Das Suchmuster wird von links nach rechts, beginnend mit dem ersten Zeichen abgearbeitet. Wird bei dem ersten Zeichen eine Übereinstimmung festgestellt, dann wird mit dem nächsten Zeichen des Suchmusters fortgefahren.
- ✓ Enthält das Suchmuster Alternativen, die durch das Zeichen `|` getrennt sind, werden diese ebenfalls von links nach rechts auf ihr Vorkommen im zu durchsuchenden Text überprüft. Beim ersten Auftreten einer Übereinstimmung wird die Suche abgebrochen.
- ✓ Alle Quantoren arbeiten greedy und nehmen so viele Zeichen auf, dass das Suchmuster gerade noch erfüllt ist. Mit dem Operator `?!` nach dem Quantor wird dieses Verhalten aufgehoben.
- ✓ Mit runden Klammern werden Bereiche eines Suchmusters gruppiert. Perl speichert die gefundenen Übereinstimmungen dieser Gruppierungen und erlaubt den Zugriff über die Variablen `$1`, `$2` usw.; die Ziffer gibt dabei die Nummer der Klammerung an.
- ✓ Bei der Suche wird nach Groß- und Kleinschreibung unterschieden. Alle Zeichen im Suchmuster müssen exakt so angegeben werden, wie sie der durchsuchte Text enthalten soll. Mit der Schreibweise `/ . . . /i` erfolgt eine Suche ohne Beachten der Schreibweise.

10.6 Ersetzen von Textteilen mit regulären Ausdrücken

Substitution verwenden

Mit regulären Ausdrücken können Sie Texte nicht nur nach Übereinstimmungen mit einem Suchmuster überprüfen, sondern auch fundene Textteile durch andere Texte ersetzen. Dieser Vorgang wird auch Substitution genannt.

Bei der Substitution wird ein Text nach einem Muster durchsucht. Wird eine Übereinstimmung gefunden, wird diese durch einen angegebenen Text ersetzt. Das Suchmuster kann, wie bei der Mustererkennung, alle Möglichkeiten von regulären Ausdrücken enthalten, beispielsweise Quantoren, Zeichenklassen und Alternativen.

Syntax der Substitution

```
$variable =~ s/Suchmuster/Ersetzung/ [Option];
```

- ✓ Zuerst erfolgt die Angabe der zu bearbeitenden Variablen gefolgt vom Operator `=~`.
- ✓ Danach folgt die Angabe des Substitutions-Operators `s///`. In die ersten beiden Slashes wird das Suchmuster eingeschlossen, danach folgt der einzufügende Text.
- ✓ Anschließend können verschiedene Optionen angegeben werden.
- ✓ Der Ausdruck liefert als Ergebnis die Anzahl der Ersetzungen.

Folgende Optionen sind möglich:

<code>s///g</code>	Standardmäßig ersetzt Perl nur das erste Vorkommen des Suchmusters. Mit der Option <code>g</code> (global) werden alle Vorkommen ersetzt.
<code>s///i</code>	Bei der Prüfung des Suchmusters wird mit dieser Option nicht zwischen Groß- und Kleinschreibung unterscheiden.
<code>s///o</code>	Das Suchmuster wird nur einmal im Programmablauf kompiliert. Besonders in Schleifen kann so Rechenzeit gespart werden.
<code>s///m</code>	Beim Durchsuchen des Textes werden Zeilenumbrüche beachtet.
<code>s///s</code>	Der zu durchsuchende Text enthält mehrere Zeilen, die Zeilenumbrüche werden jedoch nicht beachtet.
<code>s///e</code>	Es werden auch Anweisungen und Aufrufe von Unterprogrammen und Perl-Funktionen innerhalb des Suchmuster oder der Ersetzung ausgewertet.

Sie können auch mehrere Optionen kombinieren. Um beispielsweise eine globale Suche ohne Beachtung der Schreibweise durchzuführen, können Sie den regulären Ausdruck `s/Suchmuster/Ersetzung/gi` verwenden. Die Reihenfolge der Optionen ist dabei nicht von Bedeutung.



Beispiel für Substitutionen

Die folgende Tabelle enthält einige Beispiele für die Angabe von Suchmuster und Ersetzung bei einer Substitution:

Regulärer Ausdruck	Erklärung
<code>s/Welt/Erde/g</code>	Ersetzt alle Vorkommen des Wortes <code>Welt</code> durch <code>Erde</code>
<code>s/201[012]/2020/</code>	Ersetzt das erste Vorkommen der Zahl <code>2010</code> , <code>2011</code> oder <code>2012</code> durch die Zahl <code>2020</code>
<code>s/\d+/x/g</code>	Ersetzt für den gesamten Text das Auftreten mindestens einer und beliebig vieler Ziffern hintereinander durch den Buchstaben <code>x</code>
<code>s/2015 2016/dieses Jahr/g</code>	Ersetzt alle Vorkommen von <code>2015</code> oder <code>2016</code> durch die Zeichenkette <code>dieses Jahr</code>
<code>s/\$variable1/\$variable2/g</code>	Ersetzt alle Vorkommen des Inhalts von <code>\$variable1</code> durch den Inhalt von <code>\$variable2</code>
<code>s/XX/\$zahl * 2/eg</code>	Ersetzt alle Vorkommen von <code>XX</code> durch das Ergebnis der Multiplikation von <code>\$zahl</code> mit <code>2</code>

Beispiel: `HeuteErsetzen.pl`

Mit dem folgenden Perl-Programm wird der Platzhalter `HEUTE` in einer Datei durch den aktuellen Wochentag ersetzt und die Datei neu erstellt.

```

#!/usr/bin/perl
use strict;
my $datei="beispiel2.txt";
① my @wochentage = qw(Sonntag Montag Dienstag Mittwoch Donnerstag Freitag
    Samstag);
② my @datum=localtime(time);
③ open(DATEI,"+<$datei") || die("Datei $datei kann nicht geoeffnet werden!");
④ my @datei=<DATEI>;
my $inhalt=join("",@datei);

⑤ $inhalt =~ s/HEUTE/$wochentage[$datum[6]]/g;

⑥ seek(DATEI,0,0);
⑦ print DATEI $inhalt;
close(DATEI);

```

- ① Hier wird ein Array mit den verschiedenen Wochentagen definiert.
- ② Die Perl-Funktion localtime liefert ein Array mit verschiedenen Informationen zu einem Datum. Als Argument wird die aktuelle Zeit mit der Funktion time uebergeben.
- ③ Die Datei wird zum Lesen und Schreiben geoeffnet.
- ④ Der gesamte Inhalt der Datei wird in einem Array eingelesen und anschließend mit der Perl-Funktion join() zu einer skalaren Variablen zusammengeföhrt. Dies ist notwendig, weil die Substitution nur für skalare Variablen angewendet werden kann.
- ⑤ Hier erfolgt das Suchen nach dem Platzhalter HEUTE und das Ersetzen durch den aktuellen Wochentag. Dieser ist als numerischer Wert im sechsten Element des Arrays @datum gespeichert. Über das Array @wochentage wird der Name des Wochentags zurückgegeben. Durch die Option erfolgt ein Ersetzen aller Vorkommen des Platzhalters.
- ⑥ Der Dateizeiger wird vor dem Schreiben der Datei wieder auf den Anfang der Datei gesetzt.
- ⑦ Hier erfolgt das Schreiben des Dateiinhalts mit allen ersetzen Werten.

Besser lesbare Substitutionsausdrücke

Wenn Sie innerhalb des Suchmusters oder der Ersetzung einen Slash angeben wollen, beispielsweise als Datei-angabe /usr/bin/perl, muss diesem ein Backslash vorangestellt werden. Die Aneinanderreihung mehrerer Slashes und Backslashes kann schnell unübersichtlich werden.

Für besser lesbare Substitutionsausdrücke erlaubt Ihnen Perl, statt des Slashes bei der Angabe von Suchmuster und Ersetzung ein beliebiges anderes Zeichen zu verwenden, das im regulären Ausdruck möglichst nicht verwendet wird.

Normale Substitution	Besser lesbare Schreibweise
s/\usr\bin\perl/\usr\local\bin\perl/g	s! /usr/bin/perl ! /usr/local/bin/perl ! g
s;(.*)?<i>\1</i>;ig	s ;(.*)? ;<i>\1</i> ;ig



Wenn Sie den Slash durch ein anderes Zeichen ersetzen, müssen Sie an drei Stellen des Substitutionsausdrucks das gleiche Zeichen verwenden. Wollen Sie das Zeichen gleichzeitig innerhalb des regulären Ausdrucks verwenden, müssen Sie ihm einen Backslash \ voranstellen.

Gespeicherte Übereinstimmungen verwenden

Auch bei der Substitution können Sie Teile des Suchmusters mit runden Klammern gruppieren und bei einer Übereinstimmung auf die gespeicherten Werte zurückgreifen. So können Sie beispielsweise eine gefundene Textstelle über die Variable \1 in der Ersetzung verwenden.

Innerhalb des Substitutionsausdrucks erfolgt der Zugriff auf die gespeicherten Übereinstimmungen wie bei der Mustererkennung durch einen Slash  und darauf folgend der Nummer der Klammerung, z. B. \1, \2 usw. Außerhalb des regulären Ausdrucks ist der Zugriff über die Variablen \$1, \$2 usw. möglich. Beachten Sie, dass die Variablen bei globaler Suche nur die letzte Übereinstimmung enthalten.

Beispiel

In einem Text sollen alle Datumswerte in die HTML-Tags **** und **** eingeschlossen werden. Die Datumswerte können ein- oder zweistellige Tages- und Monatsangaben sowie zwei- oder vierstellige Jahresangaben enthalten. Zusätzlich soll die Anzahl der Ersetzungen ermittelt werden.

```
$anzahl = ($text =~ s/(\d+\.\d+\.\d\d+)/<b>\1</b>/g);
```

Enthält die Variable \$text beispielsweise den Datumswert **4.10.2002**, wird dieser durch **4.10.2002** ersetzt. Die Variable \$anzahl liefert die Anzahl der Ersetzungen, die durchgeführt wurden.

Zeichenweises Ersetzen

Bei der Substitution werden Teile eines Textes durch andere Texte ersetzt. Wollen Sie dagegen nur einzelne Zeichen ersetzen, können Sie die Übersetzungsfunktion der regulären Ausdrücke verwenden. Dabei wird das Suchmuster Zeichen für Zeichen abgearbeitet und stets durch ebenfalls ein einzelnes Zeichen ersetzt. Dies können Sie beispielsweise verwenden, um alle Kleinbuchstaben in Großbuchstaben umzuwandeln.

Die Übersetzungsoption wird mit dem Operator **tr///** durchgeführt. Dabei steht das Kürzel **tr** für **translate** (engl. to translate = übersetzen).

Syntax der Übersetzung

```
$variable =~ tr/Musterzeichen/Ersetzungszeichen/ [Option];
```

- ✓ Zuerst erfolgt die Angabe der zu bearbeitenden Variablen und dem Operator **=~**.
- ✓ Danach erfolgt die Angabe des Übersetzungsoperators **tr///**. In die ersten beiden Slashes werden die Zeichen eingeschlossen, die als Muster dienen. Danach folgen die Zeichen, die als Ersetzung eingefügt werden.

Bei der Übersetzung wird das Suchmuster zeichenweise durchlaufen. Jedes Zeichen, das eine Übereinstimmung im Text besitzt, wird durch das Ersetzungszeichen an der gleichen Stelle ersetzt. Gibt es weniger Ersetzungszeichen als Suchzeichen, wird das letzte Ersetzungszeichen verwendet.

Folgende Optionen sind bei der Übersetzung möglich:

tr///c	Alle Zeichen außer den angegebenen werden ersetzt.
tr///d	Alle angegebenen Zeichen werden gelöscht.
tr///s	Es werden nur zusammenhängende Gruppen der zu suchenden Zeichen durch ein einzelnes Ersatzzeichen ersetzt.

Beispiele für Übersetzungsausdrücke

Übersetzungsausdruck	Erklärung
<code>tr/ae/io/</code>	Jedes <i>a</i> wird durch ein <i>i</i> und jedes <i>e</i> durch ein <i>o</i> ersetzt.
<code>tr/a-z/A-Z/</code>	Alle Kleinbuchstaben werden durch Großbuchstaben ersetzt.
<code>tr/0-9/x/</code>	Alle Ziffern werden durch den Buchstaben <i>x</i> ersetzt.
<code>tr/a-z/ /s</code>	Alle zusammenhängenden Gruppen von Kleinbuchstaben werden durch ein Leerzeichen ersetzt.
<code>tr/0-9/ /c</code>	Ersetzt alle Zeichen außer den Ziffern 0 bis 9 durch ein Leerzeichen.

10.7 Schnellübersicht

Sie möchten ...	
ein Suchmuster definieren	<code>/Suchmuster/</code>
eine Zeichenklasse definieren	<code>[Zeichen]</code>
einen Platzhalter für ein beliebiges Zeichen verwenden	<code>.</code>
Wiederholungen für ein Zeichen angeben	<code>* + {n,m}</code>
prüfen, ob ein Suchmuster in einer Variablen vorhanden ist/ nicht vorhanden ist	<code>\$variable =~ /Suchmuster/;</code> <code>\$variable !~ /Suchmuster/;</code>
auf gespeicherte Übereinstimmungen zugreifen	<code>\$1, \$2 usw.</code> <code>\1, \2 usw.</code>
einen Textteil durch einen anderen ersetzen	<code>s/Suchmuster/Ersetzung/</code>
eine zeichenweise Übersetzung durchführen	<code>tr/Suchzeichen/Ersetzungszeichen/</code>

10.8 Übungen

Grundlagen zu regulären Ausdrücken

Übungsdatei: --

Ergebnisdatei: --

1. Erläutern Sie den Begriff „regulärer Ausdruck“.
2. Wie geben Sie in einem Suchmuster eine Zeichenklasse an?
3. Was sind Quantoren, und wozu werden sie verwendet? Zählen Sie alle Quantoren auf.
4. Wie können Sie auf gefundene Übereinstimmungen des Suchmusters zugreifen?
5. Erläutern Sie das Suchen und Ersetzen mit regulären Ausdrücken.
6. Welche Option verwenden Sie, um bei einer Substitution alle Vorkommen des Suchmusters zu ersetzen?

Mit regulären Ausdrücken arbeiten

Übungsdatei: --

Ergebnisdatei: *Übung2-E.pl*

1. Erstellen Sie ein Perl-Programm, das alle HTML-Tags in einer HTML-Datei löscht.
2. Der Dateiname soll über die Kommandozeile übergeben werden. Falls keine Dateiname übergeben wurde, soll ein kurzer Hilfetext angezeigt werden.
3. Prüfen Sie mit einem regulären Ausdruck, ob der übergebene Dateiname die Endung *.html* oder *.htm* besitzt. Ist dies nicht der Fall, soll das Programm mit einer Fehlermeldung abgebrochen werden. Erstellen Sie den regulären Ausdruck so, dass Sie gleichzeitig den Dateinamen (inkl. möglicher Pfadangaben) und die Dateiendung trennen.
4. Speichern Sie den Dateinamen ohne Endung in einer Variablen.
5. Öffnen Sie die Datei, und lesen Sie den gesamten Inhalt in ein Array ein.
6. Fügen Sie die Zeilen zu einer skalaren Variablen zusammen.
7. Löschen Sie alle HTML-Tags, indem Sie diese in einem regulären Ausdruck durch eine leere Zeichenkette ersetzen.
8. Öffnen Sie eine neue Datei mit dem übergebenen Dateinamen und der Endung *.txt*.
9. Schreiben Sie den Inhalt der alten Datei ohne HTML-Tags in diese Datei.

11 Datum und Zeit ausgeben

In diesem Kapitel erfahren Sie

- ✓ wie Sie das aktuelle Datum und die aktuelle Uhrzeit ermitteln
- ✓ wie Sie Unix-Zeitangaben verwenden
- ✓ wie Sie Datums- und Zeitangaben ausgeben
- ✓ wie Sie Datumsdifferenzen berechnen

Voraussetzungen

- ✓ Arbeiten mit Variablen und regulären Ausdrücken

11.1 Datums- und Zeitfunktionen in Perl

Alle Datums- und Zeitfunktionen in Perl basieren auf der so genannten Unix-Zeit. Unter dem Betriebssystem Unix wird die Zeit in Sekunden seit dem 01.01.1970 um 0:00 Uhr angegeben. Perl greift dies auf und verwendet die gleiche Vorgehensweise auch unter allen anderen Betriebssystemen. Eine typische Unix-Zeitangabe lautet beispielsweise 1018341620.

Folgende Datums- und Zeitfunktionen stehen Ihnen zur Verfügung:

<code>time()</code>	Ermittelt die aktuelle Unix-Zeit in Sekunden
<code>localtime()</code>	Ermittelt aus einer Unix-Zeitangabe die einzelnen Elemente von Datum und Uhrzeit bezogen auf die Zeitzone des Computers
<code>gmtime()</code>	Ermittelt wie <code>localtime()</code> die einzelnen Elemente von Datum und Uhrzeit, jedoch bezogen auf die Greenwich-Zeitzone (GMT)



In der Praxis werden Sie meist die aktuelle Unix-Zeit mit `time()` ermitteln und diese mit der `localtime`-Funktion für eine weitere Verwendung zerlegen. Die `localtime`-Funktion kann jedoch außer der aktuellen Unix-Zeit auch jede andere Unix-Zeitangabe verarbeiten.

Bei einem Computer in Deutschland und bei entsprechend eingestellter Zeitzone geht Perl bei einem Aufruf der `localtime`-Funktion von der mitteleuropäischen Zeitzone (MEZ) aus. Die Greenwich-Zeitzone wird für Berechnungen von Datums- und Zeitwerten aus unterschiedlichen Zeitzonen verwendet, da sie als Standard gilt.

Syntax der `time`- und `localtime`-Funktion

```
$unixzeit=time();
@datum = localtime($unixzeit); # Listenkontext
$datum = localtime($unixzeit); # skalarer Kontext
```

- ✓ Die `time`-Funktion erwartet keinen Parameter und liefert die Unix-Zeit als skalaren Wert.
- ✓ Die Funktion `localtime()` benötigt als Parameter eine Unix-Zeitangabe. Um den aktuellen Zeitpunkt zu verwenden, kann die Angabe auch entfallen.

- ✓ Rückgabewert der `localtime`-Funktion ist bei einer Verwendung im Listenkontext ein Array mit verschiedenen Datums- und Zeitelementen.
- ✓ Im skalaren Kontext liefert die Funktion eine Zeichenkette mit dem aktuellen Datum und der aktuellen Uhrzeit.

Der Rückgabewert der `localtime`-Funktion im skalaren Kontext ist in der Praxis wenig sinnvoll, da die Datumsangabe im amerikanischen Format mit englischen Monats- und Wochentagsbezeichnungen erfolgt, beispielsweise `Tue Apr 9 10:51:43 2002`.



Datums- und Zeitberechnungen

Die Verwendung der Unix-Zeitangaben vereinfacht die Berechnung mit Datums- und Zeitwerten durch die Verwendung von Sekundenwerten. Um einen Zeitpunkt in der Zukunft zu ermitteln, addieren Sie eine bestimmte Anzahl Sekunden. Für einen Zeitpunkt in der Vergangenheit können Sie einen Sekundenwert subtrahieren. Mithilfe der `localtime`-Funktion wandeln Sie danach die neue Unix-Zeitangabe in reales Datum um.

Beispiel

Für das Berechnen der Fälligkeit einer Rechnung soll das Perl-Programm den Datumswert in 14 Tagen ermitteln.

```
$heute = time();
$in_14_tagen = $heute + (14 * 24 * 60 * 60);
@datum = localtime($in_14_tagen);
```

Um das Datum in 14 Tagen zu ermitteln, muss die Anzahl der Sekunden berechnet werden, die in 14 Tagen verstreichen. Da jeder Tag 24 Stunden, jede Stunde 60 Minuten und jede Minute 60 Sekunden besitzt, berechnet sich die Sekundenzahl mit der Formel `14 * 24 * 60 * 60`.

11.2 Datum und Zeit ausgeben

Im Listenkontext liefert die `localtime`-Funktion ein Array mit allen wichtigen Elementen einer Datums- und Zeitangabe. Mithilfe dieser Elemente können Sie ein Datum und/oder einen Zeitpunkt in der gewünschten Form ausgeben.

Folgende Elemente beinhaltet das zurückgegebene Array:

Indexwert	Erklärung	Wertbereich
0	Sekunden	0 bis 59
1	Minuten	0 bis 59
2	Stunden	0 bis 23
3	Tag des Monats	1 bis 31
4	Monat	0 bis 11
5	Anzahl der Jahre seit 1900	0 bis ...
6	Wochentag	0 bis 6
7	Tag des Jahres	0 bis 364 (bis 365 in Schaltjahren)
8	Sommerzeit	true in der Sommerzeit, sonst false



Wenn Sie die einzelnen Elemente gleich in mehrere skalare Werte aufteilen wollen, können Sie folgende Syntax verwenden:

```
($sekunden, $minuten, $stunden, $monatstag, $monat, $jahr, $wochentag,
$jahrestag, $sommerzeit) = localtime($unixzeit);
```

Jahr ausgeben

Perl liefert die Anzahl der Jahre seit 1900; für das Jahr 2022 beispielsweise den Wert 122. Um das aktuelle Jahr zu erhalten, addieren Sie zu diesem Wert 1900:

```
$jahr = $jahr + 1900;
```

Um das Jahr zweistellig auszugeben, können Sie die formatierte Ausgabe mit der `sprintf`-Funktion verwenden:

```
$jahr = sprintf("%02d", $jahr % 100);
```

Monat ausgeben und in Monatsnamen umwandeln

Beachten Sie, dass die Angabe des Monats mit dem Wert 0 beginnt. Dies hilft bei der Umwandlung des numerischen Werts in einen Monatsnamen als Wort über ein Array. Für eine korrekte numerische Monatsangabe erhöhen Sie diesen Wert um 1.

Um den Monatsnamen als Wort auszugeben, definieren Sie ein entsprechendes Array mit den Namen aller Monate. Danach können Sie mit dem Monat als Indexwert darauf zugreifen.

```
@monatsnamen = qw(Januar Februar März April Mai Juni Juli August September
Oktober November Dezember);
@datum=localtime();
print $monatsname[$datum[4]];
```

Wochentag ausgeben

Der Wochentag wird als numerischer Wert angegeben und beginnt mit dem Wert 0 für Sonntag. Mithilfe eines Arrays können Sie diesen Wert in den Namen eines Wochentags umwandeln.

```
@wochentage = qw(Sonntag Montag Dienstag Mittwoch Donnerstag Freitag Samstag);
@datum=localtime();
print $wochentage[$datum[6]];
```

Uhrzeit und Datum mit führender Null ausgeben

Da alle Zeit- und Datumswerte numerisch angegeben werden, enthalten Sie bei Werten kleiner als 10 keine führende Null. Um diese zweistellige Ausgabe dennoch zu erhalten, können Sie die `sprintf`-Funktion verwenden oder die führende Null in einer `if`-Abfrage als Zeichenkette anfügen.

```
@datum=localtime();
print "$datum[2]:$datum[1]:$datum[0]\n"; # z. B. 8:5:15
print sprintf("%02d:%02d:%02d", $datum[2], $datum[1], $datum[0]); # z. B. 08:05:15
```

oder

```
@datum=localtime();
$sekunden = $sekunden < 10 ? $sekunden = "0".$sekunden : $sekunden;
$minuten = $minuten < 10 ? $minuten = "0".$minuten : $minuten;
$stunden = $stunden < 10 ? $stunden = "0".$stunden : $stunden;
print "$stunden:$minuten:$sekunden";
```

Die Angabe %02d in der Formatangabe der sprintf-Funktion bedeutet eine Ausgabe als ganze Dezimalzahl mit zwei Stellen und führender Null bei weniger als zwei Stellen. Mehr Informationen zur sprintf- bzw. printf-Funktion finden Sie in der Perl-Hilfeseite *perfunc*.



Beispiel: *Datum.pl*

Das Perl-Programm zeigt das aktuelle Datum an und liest danach die Anzahl der Tage für eine Datumsberechnung in der Zukunft ein. Es wird ermittelt, welches Datum in der eingegebenen Anzahl an Tagen vorliegt.

```
#!/usr/bin/perl
use strict;

① my @wochentage = qw(Sonntag Montag Dienstag Mittwoch Donnerstag Freitag
    Samstag);
my @monatsnamen = qw(Januar Februar März April Mai Juni Juli August
    September Oktober November Dezember);

② my $heute=time();
my @datum=localtime($heute);
③ $datum[5]+=1900;
④ print "Heute ist $wochentage[$datum[6]], der $datum[3] .
    $monatsnamen[$datum[4]] $datum[5].\n";

⑤ print "Anzahl der Tage: ";
my $stage=<STDIN>;
chomp($stage);

⑥ @datum=localtime($heute+($stage*24*60*60));
$datum[5]+=1900;
print "In $stage Tagen ist $wochentage[$datum[6]], der $datum[3] .
    $monatsnamen[$datum[4]] $datum[5].\n";
```

- ① Für eine Ausgabe des Wochentags und Monatsnamens werden zwei Arrays definiert.
- ② An dieser Stelle wird das aktuelle Datum als Unix-Zeitangabe ermittelt und danach mit der localtime-Funktion umgewandelt.
- ③ Für die korrekte Angabe des Jahres wird der entsprechende Wert um 1900 erhöht.
- ④ Hier erfolgt die Ausgabe des aktuellen Datums. Dabei wird auf die vorher definierten Arrays zugegriffen.

```
F:\PerlProgramme\Kapitel_11>perl Datum.pl
Heute ist Mittwoch, der 9. September 2015.
Anzahl der Tage: 177
In 177 Tagen ist Freitag, der 4. März 2016.
F:\PerlProgramme\Kapitel_11>
```

Berechnung eines Datums in der Zukunft

- ⑤ Von der Tastatur wird die Anzahl der Tage eingelesen, die als Berechnungsgrundlage dienen soll.
- ⑥ Hier erfolgt die Berechnung des zukünftigen Datums. Dabei wird die vorher gespeicherte, aktuelle Unix-Zeitangabe verwendet. Danach erfolgt die Ausgabe des neuen Datums.

11.3 Datum in Unix-Zeitangabe umwandeln

Perl verfügt über keine eigenen Funktionen zum Umwandeln einer Datums- oder Zeitangabe in das Unix-Format. Im Modul `Time::Local`, das zur Standard-Installation von Perl gehört, sind jedoch die Funktionen `timelocal()` und `timegm()` als Umkehrfunktionen von `localtime()` und `gmtime()` enthalten.

 Um diese Funktionen nutzen zu können, müssen Sie das Modul am Anfang des Perl-Programms mit dem Befehl `use Time::Local;` einbinden.

Syntax der `timelocal`- und `timegm`-Funktion

```
$unixzeit = timelocal($sekunden, $minuten, $stunden, $tag, $monat, $jahr);
$unixzeit = timegm($sekunden, $minuten, $stunden, $tag, $monat, $jahr);
```

- ✓ Die Funktionen `timelocal()` und `timegm()` erwarten als Parameter die Angabe eines Datums bzw. einer Uhrzeit mit skalaren Werten für Sekunden, Minuten, Stunden, Tag des Monats, Monat und Jahr.
- ✓ Die Funktionen liefern als Resultat eine Unix-Zeitangabe, die dem übergebenen Datum entspricht.

Beide Funktionen überprüfen die übergebenen Werte auf Gültigkeit und beenden das Programm gegebenenfalls mit einer Fehlermeldung. Folgende Wertebereiche für die einzelnen Argumente sind zu beachten:

Übergebenes Element	Wertebereich
Sekunden	0 bis 59
Minuten	0 bis 59
Stunden	0 bis 23
Tag des Monats	1 bis 31
Monat	0 bis 11
Jahr	0 bis ... (Anzahl der Jahre seit 1900)

Beispiel: `AnzahlTage.pl`

Das Perl-Programm berechnet die Anzahl der Tage, die seit einem Datum in der Vergangenheit bis heute vergangen sind. Das Datum wird in einem festen Format von der Tastatur eingelesen und mit einem regulären Ausdruck zerlegt.

```

#!/usr/bin/perl
use strict;

① use Time::Local;

print "Berechnung der Anzahl an Tagen zwischen einem Datum und
heute.\n\n";
print "Bitte geben Sie ein Datum in der Vergangenheit ein
(TT.MM.JJJJ):\n";
my $datum=<STDIN>;
chomp($datum);

② $datum =~ /(\d\d)\.(\d\d)\.(\d\d\d\d)/;
my $tag=$1;
③ my $monat=$2-1;
④ my $jahr=$3-1900;

⑤ my $datum_unix=timelocal(0,0,0,$tag,$monat,$jahr);
⑥ my $stage=int((time()-$datum_unix)/60/60/24);

print "Zwischen dem $datum und heute liegen $stage Tage.\n\n";

```

- ① Um die Funktion `timelocal()` zu nutzen, muss das Modul `Time::Local` eingebunden werden.
- ② Das über die Tastatur eingegebene Datum wird hier mit einem regulären Ausdruck zerlegt. Durch die Gruppierung kann auf die einzelnen Elemente des Datums zugegriffen werden.
- ③ Die `timelocal`-Funktion erwartet die Monatsangabe im Bereich 0 bis 11. Daher muss der eingegebene Wert um 1 verringert werden.
- ④ Auch das Jahr muss angepasst werden. Um die Anzahl der Jahre von 1900 bis zum eingegebenen Jahr zu erhalten, wird die Differenz mit 1900 gebildet.
- ⑤ Mit der `timelocal`-Funktion wird hier die Unix-Zeitangabe für das eingegebene Datum ermittelt. Da die Uhrzeit keine Rolle spielt, wird der Wert 0 für Sekunden, Minuten und Stunden angenommen.
- ⑥ Um die Differenz zum aktuellen Datum zu berechnen, wird vom Ergebnis der `time`-Funktion die in ⑤ ermittelte Unix-Zeitangabe abgezogen. Diese Angabe in Sekunden wird in Tage umgerechnet. Da die Kommastellen nicht relevant sind, erfolgt anschließend das Umwandeln in eine Ganzzahl mit der `int`-Funktion.

```

F:\PerlProgramme\Kapitel_11>perl AnzahlTage.pl
Berechnung der Anzahl Tagen zwischen einem Datum und heute.

Bitte geben Sie ein Datum in der Vergangenheit ein (TT.MM.JJJJ):
12.12.2002
Zwischen dem 12.12.2002 und Heute liegen 4654 Tage.

F:\PerlProgramme\Kapitel_11>_

```

Berechnung der Differenz zwischen zwei Daten

11.4 Übungen

Grundlagen zu Datums- und Zeitangaben

Übungsdatei: --

Ergebnisdatei: --

1. Erläutern Sie die Unix-Zeitangaben, die unter Perl verwendet werden.
2. Welche Werte liefert die `localtime`-Funktion in einem Array zurück?
3. Was müssen Sie bei der Jahresangabe der `localtime`-Funktion beachten?
4. Wie wandeln Sie die numerischen Angaben für Wochentag und Monat in die entsprechenden Namen um?

Datums- und Zeitangaben verwenden

Übungsdatei: --

Ergebnisdatei: *Übung2-E.pl*

1. Das Perl-Programm soll für alle Dateien in einem Verzeichnis ermitteln, vor wie viel Tagen der letzte Zugriff stattfand.
2. Programmieren Sie die Eingabe eines Verzeichnisnamens über die Tastatur.
3. Erstellen Sie ein Unterprogramm, das alle Dateien im übergebenen Verzeichnis ermittelt.
4. Durchlaufen Sie im Unterprogramm alle Verzeichniseinträge, und filtern Sie alle Unterverzeichnisse und die Platzhalter `.` und `..` aus.
5. Ermitteln Sie für jede Datei mithilfe der `stat`-Funktion die Unix-Zeitangabe für die letzte Änderung der Datei. Speichern Sie diese zusammen mit dem Dateinamen in einem Hash. Der Dateiname soll dabei als Schlüsselwort dienen.
6. Das Unterprogramm soll ein Hash mit allen Dateinamen zurückgeben.
7. Ermitteln Sie die aktuelle Zeit, und speichern Sie diese in einer Variablen.
8. Damit die Anzeige der Tage seit der letzten Änderung korrekt ist, müssen Sie die Anzahl Sekunden seit Mitternacht ermitteln. Verwenden Sie dazu die `localtime`-Funktion und addieren Sie das Ergebnis zur aktuellen Unix-Zeitangabe.
9. Durchlaufen Sie in einer Schleife alle Elemente des Hashs. Die Ausgabe soll dabei sortiert erfolgen.
10. Berechnen Sie für jede Datei die Anzahl der Tage seit der letzten Änderung, indem Sie von der korrigierten aktuellen Unix-Zeitangabe den entsprechenden Wert subtrahieren. Rechnen Sie die Sekunden in Tage um, und bilden Sie daraus einen ganzzahligen Wert.
11. Je nach Anzahl vergangener Tage soll eine Anzeige erfolgen, dass der letzte Zugriff heute, gestern oder vor x Tagen erfolgt.

```
F:\PerlProgramme\Uebungsdateien\Kapitel_11>perl Übung2-E.pl
Bitte geben Sie ein Verzeichnis ein: c:\Windows

CtHdaLoc.reg Letzte Änderung war vor 82 Tagen.
HelpPane.exe Letzte Änderung war vor 61 Tagen.
Professional.xml Letzte Änderung war vor 61 Tagen.
WLXPGSS.SCR Letzte Änderung war vor 527 Tagen.
WMSysPr9.prx Letzte Änderung war vor 61 Tagen.
WindowsShell.Manifest Letzte Änderung war vor 61 Tagen.
WindowsUpdate.log Letzte Änderung war Heute.
atiogl.xml Letzte Änderung war vor 415 Tagen.
ativpsrm.bin Letzte Änderung war vor 24 Tagen.
bfsvc.exe Letzte Änderung war vor 61 Tagen.
bootstat.dat Letzte Änderung war Gestern.
diagerr.xml Letzte Änderung war vor 24 Tagen.
diagwrn.xml Letzte Änderung war vor 24 Tagen.
explorer.exe Letzte Änderung war vor 29 Tagen.
hh.exe Letzte Änderung war vor 61 Tagen.
mib.bin Letzte Änderung war vor 61 Tagen.
notepad.exe Letzte Änderung war vor 24 Tagen.
regedit.exe Letzte Änderung war vor 61 Tagen.
splwow64.exe Letzte Änderung war vor 61 Tagen.
system.ini Letzte Änderung war vor 748 Tagen.
twain_32.dll Letzte Änderung war vor 61 Tagen.
win.ini Letzte Änderung war vor 748 Tagen.
winhlp32.exe Letzte Änderung war vor 61 Tagen.
write.exe Letzte Änderung war vor 61 Tagen.

F:\PerlProgramme\Uebungsdateien\Kapitel_11>
```

Mögliche Ausgabe des Perl-Programms

12 CGI-Programme mit Perl

In diesem Kapitel erfahren Sie

- ✓ wie Sie mit dem Webserver Apache umgehen
- ✓ wie das CGI als Schnittstelle zwischen Webserver und Perl-Programm funktioniert
- ✓ wie Sie Daten an CGI-Programme übergeben
- ✓ wie Sie Formulardaten decodieren und in einem CGI-Programm verwenden

Voraussetzungen

- ✓ Perl-Funktionen, Unterprogramme und reguläre Ausdrücke verwenden

12.1 Interaktive Webseiten mit CGI-Programmen

Interaktivität im WWW

Das World Wide Web (WWW) hat mit statischen HTML-Seiten begonnen. Sollen die Besucher einer Webseite jedoch Daten eingeben oder sollen Teile einer Webseite erst beim Abruf erstellt werden, werden CGI-Programme oder allgemein Skripte auf dem Server, aber auch im Client selbst, verwendet.

Mit CGI-Programmen können Sie auf dem Webserver Daten aus Formularen in HTML-Seiten verarbeiten und auf die eingegebenen Daten reagieren. Gleichzeitig werden CGI-Programme auch zum Erstellen von dynamischen HTML-Seiten verwendet. So ist es beispielsweise möglich, dass der Besucher einer Seite in einer Datenbank nach Produkten sucht und diese per Mausklick bestellt. Mit CGI-Programmen können Sie dabei die geforderte Interaktivität realisieren.

CGI als Schnittstelle zum Webserver

Beim Aufruf einer Webseite in Ihrem Browser richten Sie eine Anfrage an einen Webserver im Internet. Dieser reagiert darauf und sendet Ihnen die angeforderte HTML-Seite über das Internet-Protokoll HTTP (Hypertext Transfer Protocol). Der Webserver selbst besitzt in der Regel keine Möglichkeiten, um interaktiv auf Formulareingaben zu reagieren oder dynamische HTML-Seiten zu erstellen. Er ist ausschließlich für den Transport der Daten zum Besucher einer Webseite bzw. für den Empfang von Formulardaten zuständig.

Ein normales Perl-Programm erwartet Eingaben von der Standardeingabe (bisher die Tastatur) und sendet alle Ausgaben an die Standardausgabe (bisher der Monitor). Damit das Perl-Programm mit einem Webserver kommunizieren kann, muss eine standardisierte Schnittstelle vorliegen: das CGI.

Das CGI (Common Gateway Interface) ist die Standardschnittstelle zwischen einem Webserver und einem externen Programm, z. B. einem Perl-Programm. Ruft der Besucher einer Webseite eine mit einem CGI-Programm übereinstimmende Adresse auf, übergibt der Webserver die gesendeten Daten zusammen mit einigen Statusinformationen über die Standardeingabe an das externe Programm. Das CGI-Programm kann diese Daten verarbeiten und Daten auf die Standardausgabe schreiben, die der Webserver wiederum an den Browser übermittelt.

CGI-Programme

CGI-Programme können in beliebigen Programmiersprachen erstellt werden. Wichtig ist dabei jedoch, dass sie auf dem Computer, auf dem der Webserver läuft, ausgeführt werden können. Da Perl sehr mächtige Funktionen zur Verarbeitung von Texten besitzt, leicht erlernbar und auf der häufigsten Webserver-Plattform Unix einfach verfügbar ist, ist Perl eine wichtige Standardtechnologie für CGI-Programme.

Ein in Perl erstelltes CGI-Programm ist im Wesentlichen ein normales Perl-Programm. Es sollte die Dateiendung `.pl` oder `.cgi` tragen. Für eine besonders einfache Programmierung wird dabei meist das CGI-Modul eingebunden.

CGI-Programme befinden sich im Dateisystem des Webservers in der Regel in einem speziellen Verzeichnis mit dem Namen `cgi-bin`. Ein Webserver wie Apache startet bei einer Standardkonfiguration externe Programme aus Sicherheitsgründen nur dann, wenn sie sich in diesem Verzeichnis befinden.

12.2 Installation des Webservers Apache

Um Perl als Programmiersprache für CGI-Programme zu verwenden, muss ein Webserver installiert oder frei zugänglich sein. Eine leistungsfähige und kostenlos verfügbare Webserver-Software für viele Betriebssysteme ist der Apache-Webserver.

Obwohl manche Linux-Distributionen einen Apache-Webserver bereits automatisch mitbringen, können Sie diesen auch von Hand installieren und einrichten. Unter Windows müssen Sie das ggf. sowieso nachträglich machen. Sie können den Apache-Webserver im Internet unter <http://www.apache.org/> herunterladen.

Wenn Sie Apache als isoliertes Programm installieren, benötigen Sie für den Datenbankzugriff mit Perl noch MySQL und Unterstützung für Perl. Die Administration und Konfiguration von Apache ist nicht trivial.

Für den Betrieb des Webservers zum Testen von Perl-CGI-Programmen ist es ausreichend, wenn Sie das Softwarepaket XAMPP (<https://www.apachefriends.org/de/>) bestehend aus Apache, MySQL sowie Unterstützung von Perl, PHP und einigem mehr installieren. Dessen Installation ist viel einfacher und die Konfiguration und Administration sind komfortabler, als wenn Sie die Einzelprogramme installieren.

In dem Buch wird XAMPP verwendet. Im Anhang finden Sie Ausführungen zur Installation von XAMPP für verschiedene Betriebssysteme und Informationen, wie Sie Apache und MySQL, über das XAMPP-Control Panel starten und stoppen können.



Apache manuell betreiben

Sofern Sie ein Betriebssystem auf Unix-Basis betreiben und Apache automatisch bereits installiert wurde, können Sie Apache auch manuell starten. Ist Apache unter `/usr/local/apache` installiert, gehen Sie wie folgt vor:

- ▶ Starten Sie den Webserver mit dem Befehl:
`/usr/local/apache/bin/apachectl start`

Der Ort der Installation kann allerdings abweichen und auch unter Unix-basierenden Betriebssystemen ist XAMPP rein für die Programmierung mit Perl meist die bequemere Lösung.

Zusammenspiel von Webserver, CGI und Perl testen

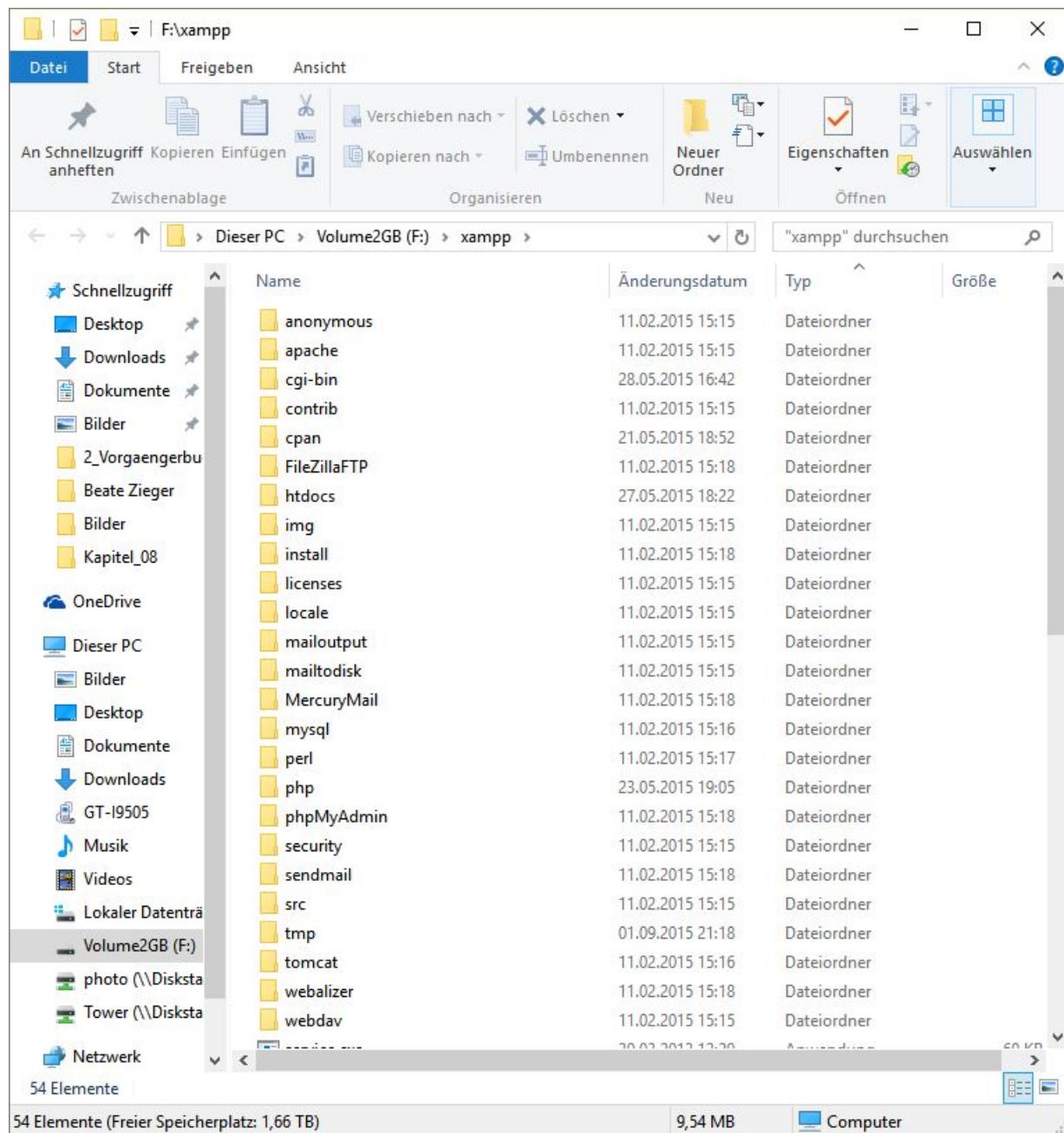
Im Folgenden wird davon ausgegangen, dass ein funktionsfähig installierter Webserver vorhanden ist, beispielsweise Apache. Bei der direkten Konfiguration des Webservers muss ein Verzeichnis für die CGI-Programme eingerichtet und der korrekte Pfad zum Perl-Interpreter angegeben werden. Bei XAMPP ist dieses bereits automatisch der Fall.

Webserver konfigurieren und testen

Bei der Installation des Apache-Webservers werden verschiedene Unterverzeichnisse angelegt. Im Unterverzeichnis `conf` finden Sie die zentrale Konfigurationsdatei des Webservers `httpd.conf`. Diese Datei erlaubt sehr umfangreiche Einstellungsmöglichkeiten. Unter der Adresse <https://httpd.apache.org/docs/current/> finden Sie zahlreiche Hilfestellungen zur Konfiguration und dem Betrieb des Apache-Webservers. Mit XAMPP werden Sie in der Regel keine manuellen Konfigurationen vornehmen müssen.

Im Unterverzeichnis `htdocs` werden standardmäßig alle HTML-Dateien gespeichert, die über den Webserver abrufbar sind. Das Unterverzeichnis `cgi-bin` ist den CGI-Programmen vorbehalten.

Beim Einsatz von XAMPP werden diese Verzeichnisse gegenüber einer „normalen“ Installation von Apache im Verzeichnissystem etwas verschoben, aber die Bedeutung ist gleich. Sie finden die Verzeichnisse direkt im Installationsverzeichnis von XAMPP.



Das Installationsverzeichnis von XAMPP

Webserver testen

Ist der Apache-Webserver gestartet, können Sie die erfolgreiche Installation durch einen Aufruf der Startseite testen. Die Adresse zum Aufrufen des lokalen Webservers lautet <http://127.0.0.1/>. Alternativ können Sie auch die Kurzform <http://localhost/> angeben.

Wenn die Webserver-Software auf einem Rechner läuft, können Sie diesen Webserver von jedem Browser aus ansprechen, der den Server-Rechner erreichen kann. Dazu geben Sie im Browser die URL des Server-Rechners ein. Dabei ist es gleichgültig, ob Sie mit der URL einen Server-Rechner im Internet oder im lokalen Netzwerk adressieren. Den Server-Rechner können Sie über seinen (DNS-)Namen oder dessen IP-Adresse adressieren. Verwenden Sie einen Namen, muss sichergestellt sein, dass dieser in die IP-Adresse aufgelöst werden kann.



In einem lokalen Netzwerk geben Sie in der Regel für den Zugriff auf einen Webserver die IP-Adresse direkt ein (z. B. <http://192.168.178.49>), während Sie im Internet meist die DNS-Namen von einem Webserver verwenden (z. B. <https://www.perl.org/>).

Werden der Webserver und der Browser auf dem gleichen Rechner ausgeführt, liegt eine besondere Situation vor. Die beiden Programme sollen über http miteinander Daten austauschen! Dabei muss zwingend die Adresse der Gegenstelle angegeben werden, auch wenn die sich auf dem gleichen Rechner befindet.

Die Netzwerkadresse 127.0.0.1 steht für eine symbolische beziehungsweise virtuelle Netzwerkverbindung und benötigt keinen physikalischen Netzwerkadapter. Man spricht in diesem Zusammenhang von localhost als Adresse oder **Loop Back** als Zugriffsverfahren (zu einem virtuellen Netzwerkadapter). Damit können Sie aus dem Browser heraus den eigenen Rechner ansprechen und den Webserver aufrufen, der auf dem gleichen Rechner ausgeführt wird.

- ▶ Starten Sie einen Browser.
- ▶ Geben Sie in die Adresszeile die Adresse <http://127.0.0.1> ein.

Bei erfolgreicher Installation und aktivem Webserver erscheint die Startseite von Apache oder in unserem Fall XAMPP. Sollte dies nicht der Fall sein, prüfen Sie zuerst, ob der Webserver tatsächlich gestartet ist. Von dort gelangen Sie entweder durch automatische Weiterleitung oder einen Klick auf einen entsprechend Hyperlink zu einer Webseite, über die Sie XAMPP komfortabel verwalten können.

The screenshot shows the XAMPP Control Panel for Windows. The main window title is "XAMPP für Windows". The sidebar on the left lists various services: XAMPP 5.6.3 (status: green), Willkommen, Status, Sicherheitscheck, Dokumentation, Komponenten, Applications, PHP (phpinfo(), CD-Verwaltung, BioRhythmus, Instant Art, Telefonbuch), Perl (perlinfo(), Gästebuch), J2ee (Info, Tomcat examples), and Tools (phpMyAdmin, FileZilla FTP). The main content area displays the Apache welcome page. It features a heading "Willkommen zu XAMPP für Windows!", a "Herzlichen Glückwunsch: XAMPP ist erfolgreich auf diesem Rechner installiert!", and instructions for OpenSSL support. It also mentions "Install applications on XAMPP using Bitnami" and shows icons for various Bitnami applications like WordPress, Drupal, Joomla!, and others.

Die Verwaltungsseite von XAMPP

12.3 Perl als CGI-Anwendung einrichten

Damit Sie Perl-Programme als CGI-Programme ausführen können, müssen Sie diese im Verzeichnis *cgi-bin* unterhalb des Apache-Verzeichnisses speichern. Wenn Sie mit einem selbst installierten und konfigurierten Apache arbeiten, müssen Sie unter Umständen einige Änderungen an der Konfiguration des Webservers vornehmen. Bei der Verwendung von XAMPP sind keine Änderungen nötig.

 Sie können CGI-Programme auch in anderen Verzeichnissen ausführen, dann sind jedoch weitere Änderungen in der Datei *httpd.conf* notwendig. Mehr Informationen finden Sie in der Dokumentation des Apache-Webservers.

Shebang-Zeile prüfen

Bei CGI-Programmen besitzt die Shebang-Zeile eine besondere Bedeutung. Im Gegensatz zur Ausführung eines Perl-Programms in der Konsole verwendet der Apache-Webserver diese Zeile, um den Pfad zum Perl-Interpreter zu ermitteln.

 Wenn Sie Perl-Programme als CGI-Programme verwenden, müssen Sie auch unter Windows darauf achten, dass die Shebang-Zeile den vollständigen Pfad zum Perl-Interpreter wiedergibt. Dabei können Sie als Pfadtrennzeichen den Slash **/** statt des unter Windows üblichen Backslashes **** verwenden.

<code>#!/usr/bin/perl</code>	Beispiel für eine Shebang-Zeile unter Unix
<code>#!c:/perl/bin/perl.exe</code>	Beispiel für eine Shebang-Zeile unter Windows

Ausführungsrechte unter Unix-Systemen vergeben

Das Verzeichnis *cgi-bin* muss für den Benutzer, unter dessen Namen der Apache-Webserver ausgeführt wird, freigegeben sein. Wenn Sie nicht mit XAMPP arbeiten und die Rechte nicht richtig gesetzt sind, können Sie dies mit dem `chmod`-Befehl festlegen.

Alle CGI-Programme müssen als ausführbare Dateien definiert werden. Sie erreichen dies mit dem Befehl `chmod 755 Dateiname` **[←]**.

Header ausgeben

Bevor der Webserver eine HTML-Datei an den Browser schickt, sendet er verschiedene Kopfdaten, den so genannten Header (engl. head=Kopf). Der Header enthält beispielsweise Angaben zum Webserver und viele weitere Informationen:

```
HTTP/1.1 200 OK
Date: Fri, 08. Apr 2016 10:25:13 GMT
Server: Apache/2.0.35
Connection: Keep-Alive
Content-Type: text/html
...
```

Innerhalb eines CGI-Programms ist vor allem die Ausgabe der Content-Type-Zeile wichtig, da sie den Browser über die Art der gesendeten Daten informiert (mittels des sogenannten MIME-Typs). Die Angabe beginnt stets mit den Wörtern `Content-Type`. Danach folgt als übergeordneter Datentyp beispielsweise `text` (Texte) oder `image` (Bilder). Nach einem Slash / wird der genaue Dokumententyp angegeben:

Content-Type	Erklärung
text/html	Standard-Dokumententyp für die Ausgabe einer HTML-Datei
text/plain	Ausgabe wird als ASCII-Text interpretiert
image/gif	Ausgabe wird als GIF-Bilddatei interpretiert
image/jpeg	Ausgabe wird als JPEG-Bilddatei interpretiert

Die Ausgabe der Header-Daten im CGI-Programm muss durch eine Leerzeile, d. h. zwei aufeinander folgende Zeilenumbrüche, beendet werden:

```
print "Content-Type: text/html\n\n";
```

Sie müssen im CGI-Programm nicht alle Header-Daten manuell ausgeben. Der Webserver prüft die Ausgabe des Programms und fügt alle fehlenden Header-Daten automatisch hinzu. Die Ausgabe des Dokumententyps ist jedoch zwingend erforderlich.



Beispiel: *CGI-Test.cgi*

Das folgende Programm gibt beim Aufruf als CGI-Programm eine HTML-formatierte Meldung im Browser aus.

```
① #! "F:/xampp/perl/bin/perl.exe"
use strict;
② print "Content-Type: text/html\n\n";
③ print <<ENDE;
<html><body><h3>Hallo Welt!</h3>
<p>Dieses Perl-Programm wurde als CGI-Programm gestartet.</p>
</body></html>
ENDE
```

- ① Die Shebang-Zeile muss bei Ihrer lokalen Installation angepasst werden.
- ② CGI-Programme müssen als erste Zeile einen so genannten Inhaltstyp (Content-Type) ausgeben. Dieser bestimmt, wie der Browser die gesendeten Daten interpretiert. Für HTML-Seiten lautet diese Zeile Content-Type: text/html.
Die Zeile muss zwingend mit zwei Zeilenumbrüchen abgeschlossen werden.
- ③ Ab hier erfolgt die Ausgabe der eigentlichen Elemente der HTML-Seite.



Ausgabe des CGI-Programms im Browser

12.4 Datenübergabe an CGI-Programme

HTML-Formulare erstellen

Die häufigste Art, CGI-Programme zu starten und dabei Daten zu übergeben, erfolgt über Formulare in einer Webseite. Im `form`-Tag der HTML-Seite wird dabei der Name des auszuführenden CGI-Programms angegeben.

Die folgende Tabelle zeigt die wichtigsten Elemente eines HTML-Formulars:

HTML-Tag	Erklärung
<code><form action="CGI-Programm" method="Methode"> </form></code>	Eröffnet ein HTML-Formular, das ein bestimmtes CGI-Programm aufruft. Dabei kann die Methode der Datenübertragung festgelegt werden (GET oder POST).
<code><input type="text" name="Name" value="Wert"></code>	Texteingabefeld mit einem bestimmten Namen und einem Vorgabewert
<code><textarea name="Name"></textarea></code>	Eingabefeld für größere Textbereiche
<code><select name="Name"> <option value="Wert">Bezeichnung1</option> ... </select></code>	Auswahlfeld; die einzelnen Optionen werden mit dem HTML-Tag <code><option ...></code> angegeben.
<code><input type="checkbox" name="Name" value="Wert"></code>	Kontrollfeld, das einzeln aktiviert oder deaktiviert werden kann
<code><input type="radio" name="Name" value="Wert"></code>	Kontrollfeld, das für einen Wert innerhalb einer Gruppe aktiviert werden kann
<code><input type="hidden" name="Name" value="Wert"></code>	Feld zur versteckten Übergabe von Parametern
<code><input type="submit" value="Beschreibung"></code>	Schaltfläche zum Absenden des Formulars mit einer bestimmten Beschriftung

Beispiel: `name.html`, `name.cgi`

Die Webseite zeigt ein Formular, in das ein Name eingegeben werden soll. Dieser Name wird an das CGI-Programm `name.cgi` übergeben und mit einer entsprechenden Seite beantwortet.



Beachten Sie, dass sich die Webseite im Verzeichnis `htdocs` befinden muss, das CGI-Programm aber im Verzeichnis `cgi-bin`. Die Pfadangabe des CGI-Programms beginnt mit einem Slash /. Das bedeutet, dass vom Wurzelverzeichnis des Webservers aus navigiert wird. In dem Beispiel wird die Konfiguration von XAMPP vorausgesetzt. Auf anderen Apache-Installationen kann sich das unterscheiden.

```

<html>
<head>
<title>Name eingeben</title>
</head>
<body>
<h3>Name an ein CGI-Programm übergeben</h3>
① <form action="/cgi-bin/Kapitel_12/name.cgi" method="GET">
② Bitte geben Sie Ihren Namen ein: <input type="text" name="name">
<br>
③ <input type="submit" value="Absenden">
</form>
</body>
</html>

```

- ① Hier startet das Formular. In der Option `action` des HTML-Tags wird der Name des CGI-Programms mit Pfadangaben übergeben. Mit `method` wird die Methode zum Übertragen der Daten festgelegt, in diesem Fall die GET-Methode, bei der die Daten als Zeichenkette an die Webadresse angehängt werden.
- ② An dieser Stelle wird ein Texteingabefeld mit dem Namen `name` angezeigt. Unter diesem Namen kann später im CGI-Programm auf die eingegebenen Daten zugegriffen werden.
- ③ Mit diesem HTML-Tag wird eine Schaltfläche zum Absenden des Formulars eingefügt.

Durch das Formular wird das CGI-Programm `name.cgi` aufgerufen, das eine Antwortseite erstellt:

```

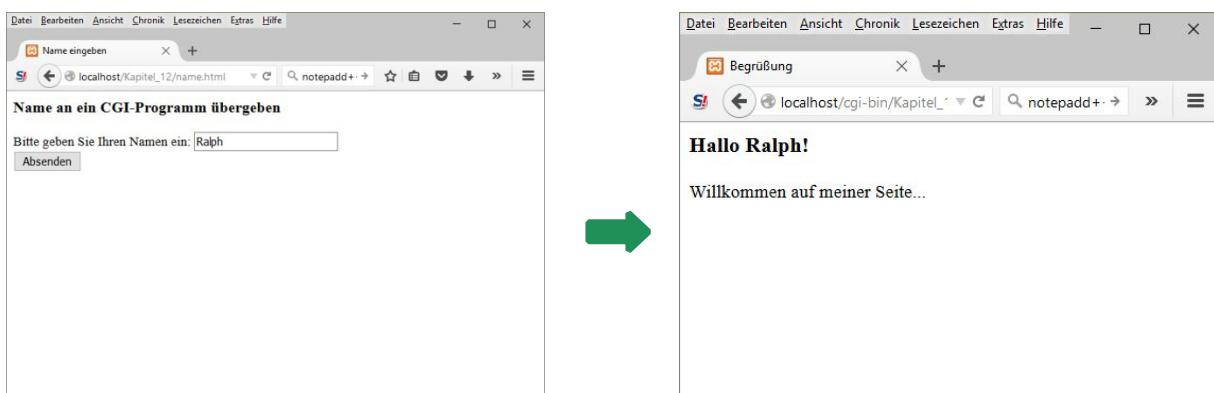
#!/F:/xampp/perl/bin/perl.exe"
use strict;
① print "Content-Type: text/html\n\n";

② my $query_string=$ENV{'QUERY_STRING'};
③ ($my $key, my $value)=split(/=/,$query_string);

print "<html>";
print "<head><title>Begrüßung</title>";
print "</head><body>";
④ print "<h3>Hallo $value!</h3>Willkommen auf meiner Seite...</h3>";
print "</body></html>";

```

- ① Hier wird der Header ausgegeben, um zu bestimmen, dass es sich um ein HTML-Dokument handelt.
- ② Die Datenübergabe des Webservers an das CGI-Programm erfolgt über Umgebungsvariablen. In der Umgebungsvariablen `QUERY_STRING` sind die Daten enthalten, die bei der Übertragungsmethode GET an die Webadresse des Skripts angehängt wurden.
- ③ Die Daten werden als Wertepaare in der Form `key=value` übergeben. Die `split`-Funktion trennt die Wertepaare und speichert die Daten in entsprechenden Variablen.
- ④ Hier erfolgt die Ausgabe des im HTML-Formular eingegebenen Namens.



Übergabe eines Werts an ein CGI-Programm

GET- und POST-Übertragungsmethode

Innerhalb des `form`-Tags einer HTML-Seite wird mit der `method`-Option festgelegt, wie die Formulardaten an das CGI-Programm übertragen werden. Bei beiden Methoden werden zuerst Wertepaare aus dem Namen eines Formularelements und dessen Wert gebildet und diese mit dem Zeichen `&` verbunden.

Folgende Übertragungsmethoden sind möglich:

GET	Die Daten werden mit einem Fragezeichen <code>?</code> direkt an die aufgerufene Adresse des CGI-Programms angehängt. <code>http://webserver.de/cgi-bin/name.cgi?name=Ralph</code>
POST	Die Daten werden innerhalb einer HTTP-Anfrage an den Webserver gesendet. Die übertragenen Informationen sind dabei beim Aufruf eines CGI-Programms nicht direkt sichtbar.



Statt mit dem Zeichen `&` können die einzelnen Wertepaare auch durch ein Semikolon `:` verbunden werden. Diese Form ist zwar zulässig, wird jedoch nur selten verwendet.

GET-Übertragungsmethode

Bei dieser Methode werden die Daten zusammen mit der Adresse (URL) eines CGI-Programms als so genannter Query-String an den Webserver übermittelt.

Der Webserver wertet die URL aus und setzt vor den Aufruf des CGI-Programms folgende Umgebungsvariablen:

Variable	Wert	Zugriff in Perl
REQUEST_METHOD	GET	<code>\$ENV{ 'REQUEST_METHOD' }</code>
QUERY_STRING	Enthält die übertragenen Wertepaare als zusammenhängende Zeichenkette, den Query-String.	<code>\$ENV{ 'QUERY_STRING' }</code>

Die GET-Methode ist die Standardmethode beim Übertragen von Formulardaten, weist jedoch einige Nachteile auf:

- ✓ Die Maximallänge der URL ist von Browser zu Browser unterschiedlich und liegt bei ca. 2 KB.
- ✓ Auch die Länge der Daten, die in einer Umgebungsvariablen an das CGI-Programm übergeben werden können, sind je nach Betriebssystem beschränkt.

- ✓ Die übertragenen Daten sind beim Aufruf des CGI-Programms direkt sichtbar und können manipuliert werden. Auch Werte aus Passwort-Feldern werden bei der Übertragung im Klartext angezeigt.
- ✓ Der an die URL angehängte Query-String führt zu unansehnlichen, langen Einträgen in der Adresszeile des Browsers.

Die GET-Methode ermöglicht jedoch trotz aller Nachteile eine einfache Möglichkeit der Datenübergabe, die vor allem beim Aufruf eines CGI-Programms per Hyperlink und zum Testen verwendet werden kann.



POST-Übertragungsmethode

Bei der POST-Methode werden die Daten vom Browser innerhalb einer HTTP-Anfrage an den Webserver gesendet. Dadurch ist es möglich, deutlich mehr Daten zu übertragen, die zudem in der Adresszeile des Browsers nicht sichtbar sind – was bei GET der Fall wäre.

Der Webserver übergibt die Daten dem CGI-Programm über die Standardeingabe. In Perl können Sie die Daten daher wie Tastatureingaben über das Datei-Handle `STDIN` einlesen. Zusätzlich definiert der Webserver folgende Umgebungsvariablen:

Variable	Wert	Zugriff in Perl
<code>REQUEST_METHOD</code>	<code>POST</code>	<code>\$ENV{ 'REQUEST_METHOD' }</code>
<code>CONTENT_LENGTH</code>	Länge der über die Standardeingabe an das CGI-Programm übermittelten Formulardaten.	<code>\$ENV{ 'CONTENT_LENGTH' }</code>

Formulardaten im CGI-Programm auswerten

Ein CGI-Programm muss unabhängig von der verwendeten Übertragungsmethode funktionieren. Um die Formulardaten auszuwerten, sollten Sie folgende Schritte abarbeiten:

- ▶ Prüfen Sie mithilfe der Umgebungsvariablen `REQUEST_METHOD`, welche Übertragungsmethode verwendet wurde.
- ▶ Bei der GET-Methode lesen Sie die Umgebungsvariable `QUERY_STRING` aus.
- ▶ Im Falle der POST-Methode lesen Sie die in der Umgebungsvariablen `CONTENT_LENGTH` angegebene Anzahl an Zeichen von der Standardeingabe.
- ▶ Unabhängig von der Übertragungsmethode müssen die Formulardaten anhand des Zeichens `&` bzw. `=` in Wertepaare aufgeteilt werden.
- ▶ Danach können die Wertepaare am Gleichheitszeichen in Name und Wert des Formularelements getrennt werden.

Beispiel: `formular.html`, `formulardaten.cgi`

Das Perl-Programm definiert ein Unterprogramm, das Formulardaten in einem Hash speichert, wobei der Name eines Formularelements als Schlüsselwort verwendet wird. Das Unterprogramm kann beide möglichen Übertragungsmethoden verarbeiten.

```

#!/F:/xampp/perl/bin/perl.exe"
use strict;

print "Content-Type: text/html\n\n";
① my %formular=&Formular_Auswerten;
② foreach my $name (keys %formular) {
    print "Das Formularelement <b>$name</b> besitzt den Wert
    <b>$formular{$name}</b>.<br>";
}

sub Formular_Auswerten {
    my ($formulardaten, @wertepaare, $wertepaar, %formular);

③ if (lc($ENV{'REQUEST_METHOD'}) eq "get") {
④ $formulardaten=$ENV{'QUERY_STRING'};
    } else {
⑤ read(STDIN, $formulardaten, $ENV{'CONTENT_LENGTH'});
    }
    my $key;
    my $value;

⑥ my @wertepaare=split(/&/,$formulardaten);
    foreach my $wertepaar (@wertepaare) {
⑦ ($key, $value)=split(/=/,$wertepaar);
    $formular{$key}=$value;
    }
    return %formular;
}

```

- ① An dieser Stelle wird das Unterprogramm zum Auswerten der Formulardaten aufgerufen. Das Resultat ist ein Hash mit den übergebenen Wertpaaren.
- ② Für jedes Formularelement werden der Name und der eingegebene Wert ausgegeben.
- ③ Hier wird geprüft, ob die Übertragung der Formulardaten mit der GET-Methode erfolgt.
- ④ Ist dies der Fall, wird die Umgebungsvariable QUERY_STRING ausgelesen.
- ⑤ Andernfalls liegt die POST-Methode vor und es wird die in der Umgebungsvariablen CONTENT_LENGTH angegebene Anzahl an Zeichen von der Standardeingabe eingelesen.
- ⑥ Die Zeichenkette mit den Formulardaten wird anhand des Zeichens & in einzelne Wertpaare aufgeteilt.
- ⑦ Jedes der Wertpaare wird innerhalb einer foreach-Schleife am Gleichheitszeichen in Name bzw. Schlüssel und Wert des Formularelements getrennt und danach in einem Hash gespeichert. Der Name dient dabei als Schlüsselwort.

URL-Codierung bei der Datenübergabe

Codierung der übertragenen Daten

Sowohl bei der GET- als auch bei der POST-Übertragungsmethode codiert der Browser alle Daten, die er an den Webserver sendet, damit auch Sonderzeichen wie Umlaute oder Leerzeichen korrekt übertragen werden.

Beispiel

Ein HTML-Formular besitzt die Eingabefelder `vorname`, `name` und `text`. Die eingegebenen Daten werden über die GET-Methode an das CGI-Programm *FormularDaten.cgi* übergeben.

Eingegeben Werte	Aufgerufene Adresse
<code>vorname=üöäÜÖÄ</code> <code>name=?&%\$§</code> <code>text=ÄÜÖ</code>	<code>/cgi-bin/Kapitel_12/formulardaten2.cgi?</code> <code>vorname=%FC%F6%E4%DC%D6%C4&name=%3F%26%25%24%A7&text=%C4%DC%D6</code>

Die Werte sind bewusst gewählt, damit die Codierung der Zeichen deutlich wird.



Formulardaten vor...

... und nach der URL-Decodierung

Die Umwandlung der Sonderzeichen beim Senden der Formulardaten an den Webbrowser wird URL-Codierung genannt. Dabei werden alle Sonderzeichen und Leerzeichen innerhalb des Namens und Werts eines Formularfelds umgewandelt:

- ✓ Sonderzeichen werden in ihre zweistellige hexadezimale Entsprechung umgewandelt und mit einem Prozentzeichen (%) gekennzeichnet.
- ✓ Leerzeichen werden durch das Pluszeichen (+) dargestellt.
- ✓ Name und Wert eines Formularfelds werden durch ein Gleichheitszeichen (=) verbunden.

Übertragene Daten decodieren

Im Fall der Übermittlung von Sonderzeichen zeigt sich, dass Perl doch schon recht alt und an einigen Stellen unkomfortabel ist. In vielen anderen Sprachen wie PHP oder ASP.NET werden URL-codierte Daten vom Browser automatisch zurückgewandelt. In Perl müssen Sie die Daten vom Browser manuell zurückverwandeln: Dazu teilen Sie die Formulardaten anhand des Zeichens (&) (bzw. (=)) in die einzelnen Wertepaare auf. Anschließend können Sie die Wertepaare in Namen und Werte trennen und decodieren. Dazu müssen die hexadezimal codierten Zeichen in ihre eigentliche Entsprechung umgewandelt werden.

Die Decodierung können Sie mit zwei regulären Ausdrücken durchführen:

```
$variable =~ s/\+/ /g;
$variable =~ s/%(..)/pack("c",hex($1))/ge;
```

- ✓ Der erste reguläre Ausdruck wandelt alle Plus-Zeichen (+) in ein Leerzeichen um.
- ✓ Im zweiten regulären Ausdruck wird nach einem Prozentzeichen (%) gesucht. Der Hexadezimalcode für ein Zeichen ist immer zweistellig. Daher werden die folgenden zwei Zeichen gruppiert, um die Übereinstimmung zu speichern.
- ✓ Der Ersetzungsausdruck enthält den Aufruf der Perl-Funktion pack (). Diese Funktion wandelt einen Dezimalwert in ein einzelnes Zeichen (engl. character).
- ✓ Der gefundene Hexadezimalwert \$1 wird mithilfe der hex-Funktion in eine Dezimalzahl umgewandelt.



Die Option e im regulären Ausdruck ermöglicht das Verwenden einer Perl-Funktion innerhalb des Ersetzungsausdrucks. Gleichzeitig müssen Sie aber statt des Ausdrucks \1 die Variable \$1 verwenden, um auf die gespeicherte Übereinstimmung zuzugreifen.

Beispiel: *formular2.html*, *formulardaten2.cgi*

Das CGI-Programm aus dem vorherigen Beispiel wird so weiterentwickelt, dass es die übertragenen Formulardaten vor der Anzeige decodiert. Die Decodierung wird in einem eigenen Unterprogramm realisiert.

```
#! "F:/xampp/perl/bin/perl.exe"
use strict;
print "Content-Type: text/html\n\n";
my %formular=&Formular_Auswerten;
foreach my $name (keys %formular) {
    print "Das Formularelement <b>$name</b> besitzt den Wert
<b>$formular{$name}</b>.<br>";
}

sub Formular_Auswerten {
    my ($formulardaten, @wertepaare, $wertepaar, %formular);

    if (lc($ENV{'REQUEST_METHOD'}) eq "get") {
        $formulardaten=$ENV{'QUERY_STRING'};
    } else {
        read(STDIN, $formulardaten, $ENV{'CONTENT_LENGTH'});
    }

    my $key;
    my $value;
    my @wertepaare=split(/&/,$formulardaten);
    foreach my $wertepaar (@wertepaare) {
        ($key, $value)=split(/=/,$wertepaar);
        $key=&URL_Decode($key);
        $value=&URL_Decode($value);
        $formular{$key}=$value;
    }
    return %formular;
}

sub URL_Decode {
    my $wert=$_;
    $wert=~s/\+/ /g;
    $wert=~s/%(..)/pack("c",hex($1))/ge;
    return $wert;
}
```

- ① Nacheinander werden der Name und der Wert jedes Formularelements durch einen Aufruf des Unterprogramms `URL_Decode()` decodiert.
- ② Die Perl-Funktion `shift` liefert das erste Element eines Arrays und löscht es danach. Da kein Arrayname übergeben wurde, verwendet die Funktion standardmäßig das Array `@_` und erhält damit Zugriff auf die an das Unterprogramm übergebenen Parameter. Die Zeile ist daher ein Ersatz für den Ausdruck `my $wert=$_[0];`.

- ③ Zuerst werden alle Plus-Zeichen in der codierten Zeichenkette durch Leerzeichen ersetzt.
- ④ Mit diesem regulären Ausdruck erfolgt das Umwandeln der im Hexadezimalcode übermittelten Sonderzeichen.

Auch in Perl gibt es mittlerweile vorgefertigte Funktionen bzw. Module, die die URL-Codierung im Hintergrund erledigen. Insbesondere das Modul CGI (<http://search.cpan.org/~leejo/CGI-4.21/lib/CGI.pod>) kodiert bzw. dekodiert Formulardaten automatisch richtig (vgl. Kapitel 13).



Umgebungsvariablen der CGI-Schnittstelle

Neben den Umgebungsvariablen REQUEST_METHOD und QUERY_STRING definiert der Webserver beim Aufruf eines CGI-Programms noch eine Anzahl weiterer Umgebungsvariablen, die Ihnen beispielsweise die IP-Adresse des aufrufenden Browsers oder die verwendete Webserver-Software übermitteln.

Die folgende Tabelle zeigt Ihnen die wichtigsten Umgebungsvariablen der CGI-Schnittstelle, auf die Sie in Perl über den Hash %ENV Zugriff erhalten:

Umgebungsvariable	Erklärung
CONTENT_LENGTH	Anzahl der über die Standardeingabe übergebenen Zeichen bei der POST-Übertragungsmethode
CONTENT_TYPE	Gibt an, welchen MIME-Typ die bei der POST-Methode übertragenen Daten besitzen
DOCUMENT_ROOT	Wurzelverzeichnis der im Webserver aufrufbaren Dateien
GATEWAY_INTERFACE	Bestimmt die Version der CGI-Schnittstelle
HTTP_ACCEPT	Liste von MIME-Typen, die der Browser akzeptiert
HTTP_ACCEPT_LANGUAGE	Liefert die vom Browser unterstützten Sprachen
HTTP_COOKIE	Enthält Namen und Wert der vom Browser gesendeten Cookies als Wertepaare
HTTP_REFERER	Gibt die URL der Webseite an, die das CGI-Programm aufgerufen hat
HTTP_USER_AGENT	Bezeichnung (Name, Versionsnummer) des aufrufenden Browsers
PATH_INFO	Pfadinformationen, die zusätzlich zum Namen des CGI-Programms übermittelt werden
QUERY_STRING	Zeichenkette mit den bei der GET-Methode übertragenen Daten
REMOTE_ADDR	Liefert die IP-Adresse des aufrufenden Browsers
REMOTE_HOST	Hostname des aufrufenden Browsers
REQUEST_METHOD	Gibt an, welche Übertragungsmethode für Formulardaten verwendet wurde
SCRIPT_NAME	Liefert den Pfad und Namen des CGI-Programms
SERVER_ADDR	IP-Adresse des Webservers
SERVER_NAME	Hostname des Webservers
SERVER_SOFTWARE	Name und Versionsnummer des Webservers

Über die Umgebungsvariable HTTP_USER_AGENT können Sie ermitteln, welcher Webbrowser beim Aufruf verwendet wurde. Nachdem Sie die Angabe mit einem regulären Ausdruck ausgewertet haben, können Sie für jeden Browsertyp eine spezielle Antwortseite zurückgeben.



12.5 Schnellübersichten

Was bedeutet ...	
CGI	Engl. Common Gateway Interface, Schnittstelle zwischen Webserver und externem Programm
GET-Übertragungsmethode	Übertragung der Formulardaten durch das Hinzufügen eines Query-Strings an die Adresse des CGI-Programms
Query-String	Zeichenkette, in der alle Wertepaare der Formulardaten durch das Zeichen <code>&</code> zusammengefügt sind
POST-Übertragungsmethode	Übertragung der Formulardaten unsichtbar innerhalb einer HTTP-Anforderung an den Webserver
URL-Codierung	Codierung von Sonder- und Leerzeichen beim Übertragen der Formulardaten an den Webserver
Header	Daten, die vor dem eigentlichen Dokument gesendet werden und dabei verschiedene Statusinformationen übermitteln
Content-Type	Eine Header-Zeile, die der Webserver an den Browser sendet, um den Dokumententyp festzulegen

Sie möchten ...	
ein Perl-Programm als CGI-Programm verwenden	Shebang-Zeile korrekt angeben, Header ausgeben, unter Unix die erforderlichen Zugriffsrechte setzen
Header für eine HTML-Seite ausgeben	<pre>print "Content-Type: text/html\n\n";</pre>
Umgebungsvariablen auslesen	<pre>\$ENV{ 'Variablenname' };</pre>
Übertragungsmethode der Formulardaten prüfen	<pre>if (lc(\$ENV{ 'REQUEST_METHOD' }) eq "get") { ... }</pre>
Formulardaten bei der GET-Methode ermitteln	<pre>\$daten=\$ENV{ 'QUERY_STRING' }</pre>
Formulardaten bei der POST-Methode ermitteln	<pre>read(STDIN, \$formulardaten, \$ENV{ 'CONTENT_LENGTH' });</pre>
Formulardaten decodieren	<pre>\$variable =~ s/\+/ /g; \$variable =~ s/%(..)/pack("c",hex(\$1))/ge;</pre>

12.6 Übungen

Grundlagen zur CGI-Programmierung

Übungsdatei: --

Ergebnisdatei: --

1. Welche Aufgabe besitzt das CGI? Wie funktioniert der Datenaustausch zwischen Webserver und Perl-Programm?
2. Was müssen Sie beim Verwenden eines Perl-Programms als CGI-Programm beachten?
3. Was bewirkt die Ausgabe der Zeile Content-Typ: text/html?
4. Erläutern Sie die Unterschiede zwischen den Übertragungsmethoden POST und GET.
5. Was sind Umgebungsvariablen, und welchen Zweck erfüllen sie beim CGI?
6. Wie erhalten Sie im CGI-Programm die bei der POST-Methode gesendeten Formulardaten?
7. Was verstehen Sie unter URL-Codierung? Und wie wird die Decodierung durchgeführt?

CGI-Programme verwenden

Übungsdatei: --

Ergebnisdatei: *kontakt.html, kontakt.cgi*

1. Erstellen Sie eine HTML-Seite mit einem Kontaktformular (vgl. nachstehende Abbildung).
2. Die eingegebenen Daten sollen mit einem CGI-Programm verarbeitet und in einer Datei gespeichert werden.
3. Erstellen Sie ein Unterprogramm, das die Übertragungsmethode prüft und die Formulardaten auf die jeweils geeignete Art und Weise ermittelt.
4. Trennen Sie die Formulardaten in einzelne Wertepaare und diese wiederum in Name und Wert des Formularelements.
5. Speichern Sie die Formulardaten in einem Hash.
6. Erstellen Sie ein weiteres Unterprogramm, das die URL-Decodierung übernimmt. Arbeiten Sie dabei mit geeigneten regulären Ausdrücken.
7. Decodieren Sie die Namen und Werte der Formularelemente vor der Speicherung im Hash.
8. Die Eingabefelder für den Vornamen, Nachnamen und die E-Mail-Adresse müssen ausgefüllt werden. Definieren Sie im Programm ein Array mit den Namen der Pflichtfelder.
9. Durchlaufen Sie alle Pflichtfelder, und beenden Sie das Programm mit einer Information, wenn eines der Felder nicht ausgefüllt wurde.
10. Speichern Sie alle Formulardaten jeweils durch Kommata getrennt in der Datei *kontakt.csv*.
11. Geben Sie eine Information über das erfolgreiche Speichern der Daten aus.
12. Prüfen Sie die Ausführung des CGI-Programms mit der GET- und der POST-Übertragungsmethode, indem Sie das Formular bearbeiten.

The figure consists of three side-by-side screenshots of Microsoft Internet Explorer windows.
 1. The left window shows a 'Kontakt-Formular' (Contact Form) with fields for Vorname (Peter), Name (Teich), Strasse, PLZ Ort, and E-Mail. A 'Speichern' (Save) button is at the bottom. The address bar shows 'http://127.0.0.1/kontakt.html'.
 2. The middle window shows an error message: 'Sie haben vergessen, Ihren Vornamen, Nachnamen und Ihre E-Mail-Adresse anzugeben.' with a link 'Bitte gehen Sie noch einmal zurück.' The address bar shows 'http://127.0.0.1/cgi-bin/kontakt.cgi'.
 3. The right window shows a success message: 'Vielen Dank, Peter Teich!' followed by 'Wir haben Ihre Daten gespeichert und wünschen Ihnen noch einen angenehmen Tag.' The address bar shows 'http://127.0.0.1/cgi-bin/kontakt.cgi'.

Formular zur Kontaktaufnahme und die Ausgaben des CGI-Programms

13 Pakete, Module und OOP in Perl

In diesem Kapitel erfahren Sie

- ✓ was Pakete in Perl sind
- ✓ was Module sind
- ✓ was Namensräume sind
- ✓ wie Sie mit Perl objektorientiert programmieren

Voraussetzungen

- ✓ Perl-Funktionen, Unterprogramme und Syntax

13.1 Pakete und Module

Module in Perl

Perl bietet viele fertige Funktionen für Routine-Aufgaben, die fest im Interpreter „eingebaut“ sind. Je mehr solcher Funktionen im Perl-Interpreter eingebaut sind, desto größer, langsamer und ressourcenunfreundlicher wird er.

Daher gibt es einen zweiten Typ Funktionen, die nicht direkt in den Perl-Interpreter „eingebaut“ sind, sondern in Form von zusätzlichen Perl-Programmen zur Verfügung stehen. Diese speziellen Perl-Programme werden **Module** genannt und haben in der Regel die Dateierweiterung *.pm*.

Module stellen eine Folge von Perl-Anweisungen, Funktionen als auch Variablen für bestimmte Aufgaben über externe Dateien zur Verfügung. Module können Sie sich wie eine Bibliothek vorstellen. Um diese Module verwenden zu können, binden Sie die Module in Ihre Skripte bzw. Programme ein. Anschließend können Sie auf die Funktionen und Variablen der Module zugreifen oder die Anweisungen in dem Modul werden an der Stelle abgearbeitet, wo das Modul eingebunden wird.

Mittlerweile gibt es sehr viele Perl-Module, die frei verfügbar sind und nahezu jede denkbare Anwendung abdecken (oft auch redundant).

Es wird zwischen zwei wichtigen Typen von öffentlich verfügbaren Modulen unterschieden:

- ✓ Standardmodule
- ✓ CPAN-Module

Standardmodule werden bereits zusammen mit dem Perl-Interpreter ausgeliefert, während **CPAN-Module** im Internet zum Download zur Verfügung stehen.

Standardmodule werden daher bei jeder typischen Installation des Perl-Interpreters mit installiert und können wie die fest eingebauten Funktionen von Perl verwendet werden. Sie sind in der Regel auch dann verfügbar, wenn Sie ein Perl-Programm von Ihrem Rechner auf einen anderen Rechner verlagern.

Wenn Sie dagegen CPAN-Module downloaden und in Ihre eigenen Programme einbinden, ist die Portierung eines Programms auf einen anderen Rechner nicht ganz so unproblematisch. Denn Ihr Programm funktioniert dort erst, nachdem dort die gleichen CPAN-Module installiert sind.

CPAN-Module installieren

CPAN-Module installieren Sie wie folgt:

- ▶ Geben Sie die erste Anweisung ein:
cpan App::cpanminus
- ▶ Installieren Sie danach jedes beliebige Modul mit:
cpanm Module::Name

Beim Datenbankmodul DBI in Kapitel 15 finden Sie eine praktische Anwendung.



Module einbinden mit `require` oder `use`

Zum Einbinden von Modulen stehen in Perl die Funktionen `use` und `require` zur Verfügung. Die Funktion `use` ist empfehlenswert, weil sie konsequenter den Modul-Gedanken mit der Kapselung von Funktionalität verfolgt.

Bei der Verwendung von `use` wird das Modul in der Regel zu Beginn des Hauptprogramms eingebunden. Unabhängig davon steht ein mit `use` eingebundenes Modul bereits zur Verfügung, bevor die erste Code-Zeile Ihres Programms ausgeführt wird.

Bei der Verwendung von `require` wird das Modul dagegen erst an der Stelle eingebunden, an der die `require`-Anweisung steht. Vor dem entsprechenden Aufruf stehen Funktionen oder Variablen des Moduls noch **nicht** zur Verfügung. Steht der `require`-Aufruf beispielsweise in einem Zweig einer bedingten Anweisung, wird er nur dann ausgeführt, wenn das Script in den entsprechenden Zweig gelangt.

Ein weiterer wichtiger Unterschied zwischen `use` und `require` ist, dass Sie bei `use` genau angeben können, welche Funktions- und Variablennamen Sie aus einem Modul importieren wollen (mit nachgestelltem Bezeichner der Funktion oder Variablen), während `require` diese Möglichkeit nicht anbietet.

Eigene Modulen erstellen

Es ist in Perl möglich, eigene Module zu schreiben. Dies ist beispielsweise sinnvoll, um Code so zu verteilen, dass es nur noch ein schlankes Hauptprogramm mit eingebundenen Modulen für spezielle Aufgaben gibt. Diese Modularisierung ist eng mit der objektorientierten Denkweise verknüpft (vgl. Abschnitt 13.2).

Wenn Sie Perl-Module selbst erstellen, müssen Sie folgende Regeln und Besonderheiten beachten:

- ✓ Module sind Perl-Code, der in der Regel mit der Endung `.pm` versehen ist.
- ✓ Module liefern als Rückgabewert `true`. Sie können als letzte Anweisung `1;` notieren, was in Perl dem Wert `true` entspricht.
- ✓ Module können Sie entweder mit `use` oder `require` einbinden.

Beispiel für ein eigenes Modul (`zufallszahlen.pm`)

```
(1) #!/usr/bin/perl
use strict;
my $i = 0;
my $z = 0;
while ( $i < 7 ) {
    $z = int( 1 + rand(49) );
    print "$z\n";
    $i++;
}
1;
```

- ① Das Beispiel zeigt Code, der Zufallszahlen berechnet und ausgibt. Der einzige relevante Unterschied zu einer „normalen“ Perl-Datei `zufallszahlen.pl` besteht in der Dateierweiterung und dem booleschen Rückgabewert als letzte Zeile im Quellcode ②.

Beispiel: `use_modul.pl`

In diesem Beispiel wird das Modul `zufallszahlen.pm` mit `use` eingebunden:

```
#!/usr/bin/perl
use strict;

① print "Ausgabe aus dem Hauptprogramm\n\n";
② use zufallszahlen;
print "\nWeitere Ausgabe des Hauptprogramms\n";
```

- ① Im Programm wird im Quellcode zuerst eine Ausgabe aus dem Hauptprogramm vorgenommen.
 ② Dann wird das Modul mit `use` eingebunden.

Sie erkennen aber an der Ausgabe in der Konsole, dass **zuerst** die Ausgaben des Moduls erfolgen, bevor die im Quellcode zuerst notierte Ausgabe aus dem Hauptprogramm ① erfolgt. Sie sehen hier den beschriebenen Umgang mit den Modulen bei der Einbindung mit `use`. Das Modul steht bereits zur Verfügung, bevor überhaupt die erste Code-Zeile des Programms ausgeführt wird.

```
F:\PerlProgramme\Kapitel_13>perl use_modul.pl
3
12
28
23
38
29
3
Ausgabe aus dem Hauptprogramm

Weitere Ausgabe des Hauptprogramms
F:\PerlProgramme\Kapitel_13>
```

Ausgabe bei der Einbindung des Moduls mit `use`

Bei Verwendung von `require` wird das Modul dagegen erst an der Stelle eingebunden, an der die `require`-Anweisung steht. Das zeigt die folgende Abwandlung.

Beispiel: `require_modul.pl`

In diesem Beispiel wird das Modul mit `require` eingebunden:

```
#!/usr/bin/perl
use strict;

① print "Ausgabe aus dem Hauptprogramm\n\n";
② require zufallszahlen;
print "\nWeitere Ausgabe des Hauptprogramms\n";
```

- ① An der Ausgabe in der Konsole erkennen Sie deutlich, dass zuerst die Ausgabe aus dem Hauptprogramm erfolgt und dann erst die Ausgabe aus dem Modul ②.

```
F:\PerlProgramme\Kapitel_13>perl require_modul.pl
Ausgabe aus dem Hauptprogramm

16
20
10
21
13
7
22

Weitere Ausgabe des Hauptprogramms
F:\PerlProgramme\Kapitel_13>
```

Ausgabe bei der Einbindung des Moduls mit `require`

Auf den ersten Blick spricht dieser Unterschied vielleicht sogar eher für die Verwendung von `require` und weniger für den von `use`. In der Praxis enthalten Moduldateien meist keinen direkt auszuführenden Code, sondern bestehen aus Funktionen (Subroutinen), die im einbindenden Programm aufgerufen werden können. Die Notation von `use` sollte allerdings in der Praxis ganz oben im Programm erfolgen.

Speicherorte für Module und die Liste @INC

Moduldateien können in einem der Verzeichnisse abgelegt werden, die in der vordefinierten Variablen `@INC` definiert sind, oder in einem Unterverzeichnis davon. Bei einer typischen Perl-Installation enthält die Liste `@INC` eine Reihe von Unterverzeichnissen des Perl-Installationsverzeichnisses sowie das jeweils aktuelle Verzeichnis. Das aktuelle Verzeichnis enthält meistens das Hauptprogramm, das die Moduldatei einbindet.

Beispiel: `INC_auswerten.pl`

Das nachfolgende Beispiel wertet die vordefinierte Variable `@INC` aus und zeigt Ihnen, in welchen Verzeichnissen bei Ihrer Perl-Installation Sie Moduldateien ablegen können.

```
#!/usr/bin/perl
use strict;
① foreach (@INC) {
    print "$_\n";
}
```

- ① Das Programm gibt einfach mit einer Schleife den Inhalt der Liste `@INC` zeilenweise aus und damit werden die verfügbaren Verzeichnispfade angezeigt.

```
F:\PerlProgramme\Kapitel_13>perl INC_auswerten.pl
f:/Strawberry/perl/site/lib/MSWin32-x64-multi-thread
f:/Strawberry/perl/site/lib
f:/Strawberry/perl/vendor/lib
f:/Strawberry/perl/lib
.

F:\PerlProgramme\Kapitel_13>_
```

Eine Liste mit erlaubten Verzeichnissen für Module

In der üblichen Konfiguration ist der letzte Eintrag der Liste ein einzelner Punkt (.). Dies steht für „aktuelles Verzeichnis“ und bewirkt, dass der Perl-Interpreter, nachdem er eine eingebundene Datei in den allgemeinen Verzeichnispfaden nicht gefunden hat, im aktuellen Verzeichnis sucht.

Falls Sie Ihre Moduldatei in keinem der hier angegebenen Verzeichnisse ablegen wollen, können Sie Pfade auch direkt angeben. Dann müssen Sie allerdings vor dem Einbinden einer solchen Moduldatei die Liste `@INC` um den Verzeichnispfad erweitern, in dem die Moduldatei liegt:

```
#!/usr/bin/perl
use strict;
use lib "/httpd/docs/cgi-shared/speziell";
use lib "/httpd/docs/cgi-shared/module"
...
```

Durch eine Anweisung `use lib` (ein sogenanntes Pragma – siehe unten) mit anschließender Angabe eines Verzeichnispfades wird dieser Pfad an den Anfang der Liste `@INC` übernommen.

Adressierungs-Syntax beim Einbinden von Modulen

Module können standardmäßig in einem der Verzeichnisse von `@INC` oder einem Unterverzeichnis davon abgelegt werden. Daraus ergeben sich die möglichen Formen der Adressierung eines Moduls beim Einbinden mit `use` oder `require`. Beispiele:

<ul style="list-style-type: none"> ✓ require "CGI.pm"; ✓ require CGI; 	<ul style="list-style-type: none"> ✓ use "CGI.pm"; ✓ use CGI;
---	---

Alle vier Beispiele zeigen das Einbinden eines Moduls, das in einer Datei namens *CGI.pm* abgelegt ist. Die Datei befindet sich direkt in einem der verfügbaren Verzeichnisse. Für die Verwendung von `require` und `use` gelten im Wesentlichen die gleichen Regeln. Entweder notieren Sie den vollständigen Dateinamen der Moduldatei in Anführungszeichen oder Sie notieren den Dateinamen ohne die Dateiendung *.pm* und ohne Anführungszeichen. Wenn sich die Datei in einem Unterverzeichnis eines der Verzeichnisse von `@INC` befindet, ist die relative Pfadangabe erforderlich. Beispiele:

<ul style="list-style-type: none"> ✓ require "Mathematik/Statistik.pm"; ✓ require Mathematik::Statistik; 	<ul style="list-style-type: none"> ✓ use "Mathematik/Statistik.pm"; ✓ use Mathematik::Statistik;
--	--

Hier zeigen die Beispiele das Einbinden eines Moduls *Statistik.pm*. Die Datei befindet sich unterhalb eines der `@INC`-Verzeichnisse in einem Unterverzeichnis namens *Mathematik*. In diesem Fall ist allerdings die Syntax ohne Anführungszeichen vorzuziehen. Denn wenn Sie mit Anführungszeichen arbeiten, müssen Sie die einzubindende Datei mit vollqualifiziertem Namen ansprechen. Die Syntax ohne Anführungszeichen kommt dagegen ohne die Dateiendung *.pm* aus. Es wird erwartet, dass die eingebundene Moduldatei diese Endung hat. Die Zeichenfolge, welche die Verzeichnistrenner symbolisiert, also Hierarchien darstellt, besteht aus einem doppelten Doppelpunkt (`::`).

Packages, Namensräume und Module

Im Zusammenhang mit Modulen sind auch **Packages** oder **Pakete** zu sehen. Unmittelbar haben Pakete und Module erst einmal nichts miteinander zu tun, aber erst Pakete und **Namensräume** erlauben die gefahrlose Verwendung von Modulen oder allgemein fremdem Perl-Code.

Angenommen, in Ihrem Perl-Programm definieren Sie eine Subroutine mit einem bestimmten Namen und binden ein Modul ein, in dessen Code ebenfalls eine Subroutine mit gleichem Namen definiert ist. Damit der Perl-Interpreter beim Aufruf der Subroutine weiß, welches der beiden Unterprogramme gemeint ist, muss es eine Möglichkeit geben, ihm das mitzuteilen.

Zu diesem Zweck unterstützt Perl das Konzept der **Namensräume**. Ein Namensraum bezeichnet grundsätzlich erst einmal nur einen Bereich, in dem ein bestimmter Bezeichner eindeutig ist. Namensräume werden nicht nur in der Programmierung, sondern auch bei XML, Dateisystemen oder Verzeichnissen verwendet.

Jeder Namensraum in Perl stellt ein sogenanntes Package (engl. für *Paket*) dar. Namensraum und Package werden weitgehend synonym verwendet. Jedes Perl-Programm, in dem Sie keine explizite Angabe eines Namensraumes (welches Package) vornehmen, benutzt ein **Default-Package**. Es besitzt den impliziten Namen `main`.

Mit der Anweisung `package` können Sie im Code ein Paket explizit bezeichnen. Innerhalb eines Codes können verschiedene Namensräume/Packages für bestimmte Bereiche markiert werden. Das nachfolgende Beispiel *Namensraum.pl* zeigt das und es wird dabei zunächst bewusst auf die Verwendung von `use strict;` verzichtet:

```

① #!/usr/bin/perl
$text = "Namensraum main.\n";
print $text;

② package rjs;
$text = "Namensraum rjs.\n";
print $text;

③ package main;
print $text;

```

- ① Das Beispiel verdeutlicht den Default-Namensraum mit dem Namen `main`. Zunächst wird eine Variable `$text` definiert und mit einem Wert belegt. Dieser Wert wird gleich anschließend ausgegeben. Da bis zum Ausführungszeitpunkt noch keine Angabe zum Namensraum gemacht wurde, befindet sich das Programm zu dem Zeitpunkt im Default-Namensraum `main`.
- ② Im zweiten Schritt wird mit Hilfe der Funktion `package` ein neuer eigener Namensraum definiert. Im Quellcode unterhalb wird wiederum die Variable mit dem Namen `$text` definiert und ein neuer Wert zugewiesen. Dieser neue Wert wird wieder ausgegeben. Dabei stellt sich die Frage, ob damit der gleichnamigen Variablen im Default-Namensraum nur ein neuer Wert zugewiesen oder wirklich eine neue Variable angelegt wurde? Beachten Sie, dass das Programm explizit auf die Verwendung von `use strict;` verzichtet und ebenso bewusst keine Deklaration der Variablen mit `my` vorgenommen wird. Zu diesem Ausführungszeitpunkt existieren damit zwei Skalare mit gleichem Namen, der aus dem Package `rjs` und der aus dem Default-Package `main`.
- ③ Zur Kontrolle wird im Beispiel wieder mit der Funktion `package` explizit auf das Package `main` „umgeschaltet“. Von den beiden existierenden Skalaren gleichen Namens wird der im Programm zuerst definierte ausgegeben, derjenige, der zum Default-Namensraum `main` gehört.

```
F:\PerlProgramme\Kapitel_13>perl Namensraum.pl
Namensraum main.
Namensraum rjs.
Namensraum main.
```

F:\PerlProgramme\Kapitel_13>_

Mehrere Namensräume in einer Datei

Wie das Beispiel zeigt, kann der Namensraum innerhalb einer Datei beliebig gewechselt werden. Dadurch lässt sich auf der einen Seite ein hohes Maß an Flexibilität erreichen, wovon Module in Perl profitieren. Allerdings ist das Ansprechen von Variablen oder Funktionen in anderen Packages so nicht möglich, wenn die Variablen oder Funktionen mit `my` deklariert sind, was bei der Verwendung von `use strict;` notwendig ist.

Betrachten Sie die Abwandlung `Namensraum2.pl`, wo jetzt die Verwendung von `use strict;` eingesetzt und damit die explizite Deklaration erzwungen wird:

```
#!/usr/bin/perl
use strict;
① my $text = "Namensraum main.\n";
...
```

- ① Mit `my` legen Sie eine Variable global fest. Wenn die Variable in einem anderen Namensraum geändert wird, ist der Wert damit auch im Default-Namensraum geändert.

```
F:\PerlProgramme\Kapitel_13>perl Namensraum2.pl
Namensraum main.
Namensraum rjs.
Namensraum rjs.
```

F:\PerlProgramme\Kapitel_13>

Mit my legen Sie eine Variable global fest

Dieses unterschiedliche Verhalten macht die Verwendung verschiedener Namensräume in einer Datei etwas trickreich. Allgemein ist die Verwendung von mehreren Namensräumen in einer Datei selten sinnvoll und da der strikte Modus und die Anwendung von `my` in der Praxis empfohlen wird, um eine saubere Kapselung von Variablen und Funktionen zu erhalten, sollten Sie verschiedene Pakete nur in unterschiedlichen Modulen/Dateien vornehmen.

Beispiel für ein Modul mit Package: *zufallszahlen2.pm*

Das Beispiel zeigt Code innerhalb eines Moduls, der Zufallszahlen berechnet und ausgibt.

```
#!/usr/bin/perl
① package rjs;
use strict;

② sub Zufall () {
    my $i = 0;
    my $z = 0;
    while ( $i < 7 ) {
        $z = int( 1 + rand(49) );
        print "$z\n";
        $i++;
    }
}
1;
```

- ① Die package-Anweisung ordnet dem Code den Namensraum `rjs` zu.
- ② Die Zufallszahlen werden dieses Mal innerhalb einer Subroutine mit Namen `Zufall` berechnet und ausgegeben.

Beispiel für die Moduleinbindung mit `use`: *use_modul2.pl*

```
#!/usr/bin/perl
use strict;
① use zufallszahlen2;

print "Ausgabe aus dem Hauptprogramm\n\n";
② rjs::Zufall();
print "\nWeitere Ausgabe des Hauptprogramms\n";
```

- ① In dem Programm wird das Modul `zufallszahlen2` mit `use` eingebunden.
- ② Der entscheidende Punkt ist, dass der Aufruf der Subroutine `Zufall` aus dem Modul **qualifiziert** erfolgen muss. Denn auf Variablen und Funktionen, die im Modul unterhalb des Packages Testmodul definiert werden, kann nur durch Angabe des entsprechenden Namensraums zugegriffen werden. Die Anweisung `rjs::Zufall()`; zeigt, wie auf das andere Package zugegriffen wird. Dem Namen einer Variablen oder einer Funktion wird der Name des Packages und ein doppelter Doppelpunkt (:) vorangestellt. Skalare, Listen und Hashes beginnen weiterhin mit ihren typischen Kennzeichen \$, @ und %. Zwischen dem Kennzeichen und dem Variablenamen stehen jedoch auch hier der Package-Name und der doppelte Doppelpunkt.



Das Ansprechen von Variablen oder Funktionen in anderen Packages oder Modulen mit Package-Name `::Variablen/Funktionsname` ist nicht möglich, wenn die Variablen oder Funktionen mit `my` deklariert sind. Das ist kein Nachteil, denn damit wird eine saubere Kapselung von Variablen und Funktionen erzwungen.

Die Moduldokumentationen und *perldoc*

Für die meisten öffentlichen Module (Standardmodule als auch CPAN-Module) gibt es in der Regel integrierte Dokumentationen im Quellcode, die mit dem Programm `perldoc` – Bestandteil von Perl – in der Konsole ausgegeben werden können.

Der Aufruf geht wie folgt:

`perldoc [Pfad]Modulname`

```
F:\PerlProgramme\Kapitel_13>perldoc CGI
NAME
    CGI - Handle Common Gateway Interface requests and responses
SYNOPSIS
    use CGI;
    my $q = CGI->new;
    # Process an HTTP request
    @values = $q->multi_param('form_field');
    $value   = $q->param('param_name');
    $fh      = $q->upload('file_field');
-- Fortsetzung --
```

Die Ausgabe der Dokumentation des CGI-Moduls – Teil 1

Standardmodule

Es gibt in Perl eine große Anzahl an Standardmodulen, die Sie nach Kategorien sortieren können und die in den Dokumentationen von Perl ausführlich beschrieben sind. Die Mehrzahl der Modulnamen bei den Standardmodulen beginnt mit einem Großbuchstaben. Es hat sich eingebürgert, Module auf diese Weise zu benennen. Nachfolgend werden einige wichtige Standardmodule vorgestellt:

- ✓ Über das Modul CPAN laden und installieren Sie Module bei Bedarf aus dem CPAN-Verzeichnis.
- ✓ Das Modul Config erlaubt den Zugriff auf die Konfigurationsdaten von Perl.
- ✓ Mit dem Modul Data::Dumper stellen Sie die interne Struktur von Variablen dar.
- ✓ Das Modul Digest::MD5 erlaubt es, MD5-Prüfsummen zu berechnen.
- ✓ Das Modul Env importiert Umgebungsvariablen mit einzelnen Namen und macht dadurch die Verwendung des vordefinierten Hashes %ENV überflüssig.
- ✓ Das Modul Hash::Util enthält verschiedene Funktionen zum Arbeiten mit Hashes.
- ✓ Das Modul List::Util enthält verschiedene Funktionen zum Arbeiten mit Listen.
- ✓ Mit dem Modul Text::ParseWords können Sie einen Text in die einzelnen Wörter zerlegen und diese in einer Liste speichern. Wörter in Anführungszeichen gelten dabei als ein zusammenhängender Ausdruck.
- ✓ Das Modul Encode ist das allgemeine API für die Module der Encode-Familie. Diese Familie an Modulen enthält zahlreiche Untermodule wie Encode::Alias, Encode::Byte, Encode::Encoding, Encode::MIME::Header oder Encode::Unicode.
- ✓ Das Modul Locale::Constants ist für Lokalisierungen wichtig. Es definiert verschiedene Konstanten, die beim Arbeiten mit den ebenfalls bei Lokalisierung bedeutsamen Modulen Locale::Country, Locale::Currency, Locale::Language und Locale::Script genutzt werden können.
- ✓ Die Module Math::BigFloat, Math::BigInt und Math::BigRat unterstützen den Umgang mit sehr großen Zahlen. Math::Complex unterstützt das Rechnen mit komplexen Zahlen.
- ✓ Das Modul Math::Trig stellt die in Perl selber fehlenden trigonometrischen Funktionen und Konstanten bereit.
- ✓ Das Modul DirHandle bietet eine objektorientierte Alternative zu den Funktionen opendir, Seite readdir und closedir.
- ✓ Die Untermodule von File (etwa File::Basename, File::Copy oder File::Path) unterstützen die Arbeit mit Dateien und Verzeichnissen. Ebenso das IO und dessen Untermodule wie IO::Dir, IO::File, IO::Handle oder IO::Socket, die zudem noch für eine objektorientierte Prozesskommunikation über ein Netzwerk genutzt werden.
- ✓ Unter Time finden Sie Module für Datum und Zeit wie Time::Local oder Time::localtime.
- ✓ Die Module für CGI-Scripts basieren auf CGI und dessen Untermodulen wie CGI::Apache, CGI::Cookie oder CGI::Fast. Gerade CGI::Fast als Schnittstelle zum Fast-CGI-Standard beschleunigt CGI-Aktionen erheblich, weil sie als Prozesse dauerhaft im Arbeitsspeicher des Server-Rechners gehalten werden.
- ✓ Die Module im Bereich Net sind Module für allgemeine Internet-Funktionen. Mit Net::Domain ermittelt man etwa den Host- und Domainnamen des aktuellen Rechners, mit Net::FTP kann mit FTP-Servern kommunizieren oder Net::POP3 und Net::SMTP dienen der E-Mail-Kommunikation.

Pragmas

Pragmas sind spezielle Module, die Anweisungen enthalten, mit denen das Verhalten von Perl zur Kompilier- und zur Laufzeit gesteuert werden kann. Diese Pragmas sind ebenfalls Teil der Standardbibliothek von Perl. Die bekannteste und wichtigste Pragma-Anweisung ist `strict`. Pragmas werden wie gewöhnliche Module mit der Funktion `use` importiert. Mit der Anweisung `no` können Sie ein Pragma wieder ausschalten.

Beispiel: *Pragmas.pl*

```
#!/usr/bin/perl
① use strict;
$a = 7;
② no strict;
print "$a\n";
print "$b\n";
```

- ① Zuerst wird das Pragma `strict` mit `use strict;` angeschaltet ...
- ② ...und danach mit `no strict;` wieder ausgeschaltet.



Zu allen Pragmas gibt es ausführliche Dokumentationen in den Manpages von Perl oder dem `perldoc`-System.

Auswahl wichtiger Pragmas:

- ✓ Mit dem `constant`-Pragma können Sie zur Kompilierzeit konstante Variablen erzeugen.
- ✓ Das `diagnostics`-Pragma erzeugt bei Bedarf erweiterte Auskünfte in den Warnungen von Perl. Es entspricht dem Perl-Schalter `-w`, hat aber den Vorteil, dass Sie es bei Bedarf für verschiedene Teile des Skriptes ein- und ausschalten können.
- ✓ Mit dem `integer`-Pragma können Sie festlegen, dass nur mit Ganzzahlarithmetik zu rechnen ist.
- ✓ Mit dem `lib`-Pragma können Sie weitere Verzeichnisse in das `@INC`-Array aufnehmen.

Erweiterte Verwendung von `use` (@EXPORT, @EXPORT_OK und `qw`)

Module können selbst wieder andere Module einbinden. Das stört eine saubere Kapselung, denn einzelne Module sollten bei einem guten Design des Codes abgeschlossene „Dateninseln“ darstellen. Um „öffentliche“ Schnittstellen von Modulen festlegen zu können (das ist stark von der objektorientierten Programmierung beeinflusst), haben Sie die Möglichkeit, einzelne Bezeichner (Variablennamen oder Namen von Subroutinen) in mehreren Packages (dem einbindenden und dem eingebundenen) sichtbar zu machen. Sie können dazu die Symbole aus dem einen Modul exportieren bzw. in das einbindende Programm oder Modul importieren.

Ein Modul kann deshalb erzwingen, dass Sie beim Einbinden genau angeben müssen, welche Variablen oder Funktionen Sie in den Namensraum des einbindenden Programms importieren möchten.

Dazu gibt es das sogenannte **Exporter-Modul**. Die ist ein Standardmodul von Perl und erwartet im eingebundenen Package zwei Listen namens `@EXPORT` und `@EXPORT_OK`. Ein Modul kann das Exporter-Modul einbinden und diese beiden Listen mit denjenigen Variablen- und Funktionsnamen füllen, die in ein einbindendes Modul exportiert werden können. Im einbindenden Skript können Sie angeben, welche der exportierbaren Namen des Moduls Sie in den Namensraum Ihres Programms importieren möchten.

13.2 Objektorientierte Programmierung (OOP)

Seit vielen Jahren hat sich objektorientierte Programmierung durchgesetzt. Auch in Perl können Sie objektorientiert programmieren. Sie müssen es aber nicht, wie Sie an den bisherigen Beispielen gesehen haben. Sie können in Perl auch prozedurale und objektorientierte Programmierung mischen.

Die objektorientierte Programmierung geht stark vom natürlichen Denken von Menschen aus. Denn das Denken des modernen Menschen ist nicht so „prozedural“ ausgerichtet wie die klassische Programmierung. Es vollzieht sich nicht in Sprungadressen und Kontrollstrukturen, sondern wird an größere eigenständige Einheiten (Objekte) gebunden, die in der Lage sind, Informationen und Fähigkeiten beizusteuern, miteinander zu kommunizieren und dadurch etwas Neues zu schaffen. Ein Objekt hat Eigenschaften und kann etwas tun. Einige der Eigenschaften und Fähigkeiten stammen von anderen Objekten, die das Objekt selbst eingebunden und damit geerbt hat, und andere wiederum sind ganz eigene und besondere Eigenschaften und Fähigkeiten dieses Objekts.

In dem Buch werden wichtige Techniken der OOP besprochen, es ist allerdings kein umfassender Einstieg zur objektorientierten Programmierung.



Hintergründe und Konzepte der OOP

Ziel der Entwicklung komplexer Software-Systeme ist, eine möglichst hohe Software-Qualität zu erreichen. Diese kann als innere und äußere Software-Qualität betrachtet werden. Die innere Software-Qualität bezeichnet die Sicht des Entwicklers. OOP bietet beispielsweise durch die Möglichkeiten der Abstraktion, Hierarchiebildung, Kapselung, Wiederverwendbarkeit und Schnittstellenbildung hervorragende Ansätze zur Gewährleistung einer hohen inneren Software-Qualität. Die äußere Software-Qualität spiegelt die Sicht des Kunden wieder. Dieser erwartet z. B. Korrektheit, Stabilität, Anwendungsfreundlichkeit oder Erweiterbarkeit einer Software. Ein Paradigma der OOP ist, dass eine hohe innere Software-Qualität zu einer hohen äußeren Software-Qualität führt. Und OOP bestätigt dieses Paradigma.

Die wichtigsten theoretischen Konzepte in der objektorientierten Software-Entwicklung sind:

- ✓ **Objekte** dienen als Abstraktion eines realen Gegenstands oder Konzepts und verfügen über einen spezifischen Zustand und ein spezifisches Verhalten.
- ✓ **Klassen** dienen als Baupläne für Objekte. Sie repräsentieren die Objekttypen. Sie fassen alle relevanten Eigenschaften und Verhaltensweisen der repräsentierten Objekttypen zusammen (sie klassifizieren diese).
- ✓ Eine **Instanz** ist ein konkretes, reales Objekt einer bestimmten Klasse. Mit dem Objekt arbeiten Sie in Ihrem Code über einen individuellen Namen, unter dem Sie es ansprechen können.
- ✓ **Attribute** beschreiben die **Eigenschaften** eines Objekts (den spezifischen Zustand). Objekte unterscheiden sich durch Attribute voneinander.
- ✓ **Methoden** beschreiben die objekt- und klassenbezogenen **Funktionalität** (das spezifische Verhalten).
- ✓ Über eine Konstruktor-Funktion (kurz Konstruktor) wird eine Instanz eines Objektes erzeugt. Wenn ein Objekt nicht mehr gebraucht wird, kann es beseitigt werden. Dazu dient die Destruktor-Funktion (kurz Destruktor).
- ✓ Zwischen Objekten bestehen Beziehungen, die als Assoziationen, Kompositionen und Aggregationen bezeichnet werden. Eine **Assoziation** beschreibt in der Regel eine Beziehung zwischen zwei (meistens) oder mehr Klassen. Eine ganz besondere Assoziation ist die **Vererbung**. Diese beschreibt einen Mechanismus zum Generalisieren und Spezialisieren von Objekttypen. In der objektorientierten Programmierung ist die Aggregation ebenso eine besondere Art der Assoziation – aber zwischen den Objekten selbst. Sie beschreibt eine sogenannte schwache Beziehung zwischen Objekten. Ein Objekt wird hier als Teil eines anderen, ganzen Objekts gesehen. Es kann aber auch ohne das umgebende Objekt existieren. Im Fall einer Komposition ist ein Objekt auch Teil eines anderen, ganzen Objekts. Es kann jedoch nicht ohne das umgebende Objekt existieren.
- ✓ Die sogenannte Polymorphie dient zur flexiblen Auswahl geeigneter Elemente identischen Namens anhand anderer Unterscheidungsmerkmale.

Objektorientierte Programmierung in Perl

Der Perl-Interpreter unterstützt objektorientiertes Programmieren seit der Version 5.0. Um eine Klasse zu erstellen, brauchen Sie in Perl ein eigenes Package, also einen eigenen Namensraum.

Subroutinen, die innerhalb des Packages notiert sind, sind die Methoden von Instanzen dieser Klasse. Auch Objekteigenschaften können Sie innerhalb solcher Methoden definieren.

 Mehrere Packages lassen sich zwar auch innerhalb einer Perl-Datei definieren (vgl. Abschnitt 13.1), aber im Normalfall ist ein Package eine separate Perl-Datei und umgekehrt. Ein Package ist in Perl meist ein Modul mit der entsprechenden Dateiendung `.pm`. Von daher haben in Perl modul-orientiertes und objektorientiertes Programmieren viel miteinander zu tun.

Ein Package ist allerdings nicht automatisch eine Klasse. Die Klasse muss mit einer expliziten Syntax deklariert und verwendet werden.

Im folgenden Beispiel wird gezeigt, wie Sie ein Modul als Objektklasse erstellen, in einem Perl-Programm eine Instanz dieses Objekts erzeugen und mit einer Methode des Objekts arbeiten.

Ein einfaches Beispiel für ein objektorientiertes Modul: `OO_Modul.pm`

```
#!/usr/bin/perl
① package OO_Modul;
use strict;

② sub new {
    ③     my $Obj = shift;
    ④     my $Ref = {};
    ⑤     bless($Ref,$Obj);
    ⑥     return($Ref);
}

⑦ sub Zufall () {
    my $obj = shift;
    my $endzahl = shift;
    my $i = 0;
    my $z = 0;
    while ( $i < $endzahl ) {
        $z = int( 1 + rand(49) );
        print "\n$z";
        $i++;
    }
}
1;
```

- ① Die Moduldatei beginnt mit einer `package`-Anweisung, die den Namen des Moduls nennt. Der Name sollte identisch mit dem Vornamen der Moduldatei sein. Wenn die Moduldatei also wie im Beispiel `OO_Modul.pm` heißt, sollte die Anweisung `package OO_Modul;` lauten.
- ② Die Moduldatei im Beispiel enthält insgesamt zwei Subroutinen (②,⑦). Die erste davon ist die Konstruktorfunktion. Sie erhält üblicherweise den Namen `new`, könnte aber theoretisch auch einen anderen Namen erhalten.

Wie eine Konstruktor-Funktion aufgebaut ist, kann von Fall zu Fall variieren. In jedem Fall muss die Konstruktor-Funktion jedoch einen Aufruf der Funktion `bless` ⑤ enthalten. Diese bindet eine Referenz (einen Zeiger) an eine Klasse, sodass anschließend über diese Referenz auf das Objekt zugegriffen werden kann.

- ③ Mit `my $Obj = shift;` wird der erste Parameter eingelesen, den der Konstruktor `new` übergeben bekommt. Die Konstruktor-Funktion bekommt, wenn sie aufgerufen wird, automatisch als ersten Parameter den Klassennamen übergeben, im Beispiel also den Namen `OO_Modul`. Das ist im Beispiel der Wert, der anschließend in `$Obj` steht.
- ④ Mit `my $Ref = {};` wird eine Referenz deklariert, damit die Funktion `bless` den Zeiger an das Objekt binden kann.

Nun ist die Frage, wie aus der skalaren Variable `$Ref` eine Referenz werden soll? Sie muss sich ja auf irgend etwas beziehen, worauf sie zeigt. Ideal ist für diesen Zweck die Zuweisung `{}`. Dies ist einfach eine Referenz auf einen leeren unbenannten Hash. Ideal ist das deshalb, weil dieser Hash auch für die Definition von Objektdaten geeignet ist – was allerdings in diesem einfachen Beispiel nicht geschieht. So bleibt es bei dem leeren Hash, und `$Ref` ist ein Zeiger.

- ⑤ Mit `bless ($Ref, $Obj);` wird die nötige Verbindung zwischen Referenz und Objekt hergestellt
- ⑥ Anschließend wird `$Ref` zurückgegeben.
- ⑦ Die zweite Subroutine in der Moduldatei dient als Methode. Solche Methoden sind in Perl meist recht ähnlich aufgebaut. Alle Subroutinen, die als Methoden eines Objekts fungieren, bekommen automatisch mindestens einen Parameter übergeben, und zwar den Zeiger auf das Objekt, der in der Konstruktor-Funktion erzeugt und an das Objekt gebunden wurde.

Nun könnten sich Methoden den Inhalt weiterer Parameter auch direkt mit `$_[1]` usw. holen. Aber die Notation mit dem Aufruf der `shift`-Funktion hat den Vorteil, dass die Parameterliste dabei sauber abgearbeitet und der Reihe nach gelöscht wird. Mit dem jeweils folgenden Aufruf von `shift` kommt die Subroutine also an den zweiten Parameter und der stellt den Endwert der Schleife dar, in der nachfolgend Zufallszahlen ausgegeben werden.

Ein einfaches Beispiel zum Verwenden des objektorientierten Moduls: `use_oo_modul.pl`

Das eigentliche Perl-Programm zum Verwenden des objektorientierten Moduls ist einfach.

```
#!/usr/bin/perl
① use strict;
② use OO_Modul;

my $obj = OO_Modul -> new();
print "Ziehung von 3 Zahlen:\n"
③ $obj->Zufall(3);
print "\nNeue Ziehung von 4 Zahlen:\n";
$obj->Zufall(4);
```

- ① Mit der Funktion `use` und der Anweisung `use OO_Modul;` binden Sie die Moduldatei `OO_Modul.pm` ein.
- ② Mit `my $obj = OO_Modul -> new();` wird die Konstruktor-Funktion der Moduldatei aufgerufen. Dies ist die entscheidende Anweisung, mit der eine neue Instanz des `OO_Modul`-Objekts erzeugt wird! Die Anweisung besteht darin, dass ein Skalar (in dem im Beispiel mit dem Namen `$obj`) deklariert und mit einem Wert initialisiert wird. Der zugewiesene Wert besteht darin, dass der Name der gewünschten Klasse mit Hilfe des Zeigeoperators `->` auf die Konstruktor-Funktion `new` zeigt. Da die Konstruktor-Funktion eine Referenz zurückgibt, die durch den `bless`-Aufruf an das Objekt gebunden ist, wird `$obj` automatisch zu einer Referenz auf die neu erzeugte Objektinstanz.

- ③ Über \$obj und den Zeigeoperator -> kann das Programm fortan auf die Methoden des eingebundenen Moduls zugreifen. Dies wird zweimal gemacht, wobei der Konstruktor-Funktion jeweils ein Wert übergeben und dieser in der Methode verwendet wird.

```
F:\PerlProgramme\Kapitel_13>perl use_oo_modul.pl
Ziehung von 3 Zahlen:
2
41
16
Neue Ziehung von 4 Zahlen:
48
46
44
43
F:\PerlProgramme\Kapitel_13>
```

Das Objekt wurde erzeugt und die Methode wird zweimal aufgerufen



Für den Destruktor ist der Methodename DESTROY zwingend vorgegeben. Der Destruktor ist aber nur notwendig, wenn bei der Auflösung einer Instanz noch besondere Anweisungen ausgeführt werden sollen. Darauf wird hier nicht weiter eingegangen, denn in der Regel wird ein Objekt automatisch beseitigt, wenn es nicht mehr benötigt wird.

13.3 Schnellübersichten

Was bedeutet ...	
Instanz	Ein aus einer Klasse erzeugtes Objekt
Methode	Eine Subroutine, die über ein Objekt bereitgestellt wird und dessen Funktionalität realisiert
Modul	Eine externe Zusammenfassung von Code, der in einem Programm eingebunden werden kann
Namensraum	Ein Bereich, in dem ein Bezeichner eindeutig sein muss
Pragma	Das sind spezielle Module, die Anweisungen enthalten, mit denen das Verhalten von Perl zur Kompilier- und zur Laufzeit gesteuert werden kann.
Konstruktor	Eine Funktion, mit der ein Objekt erzeugt werden kann

Sie möchten ...	
ein Modul erstellen	Benennen Sie die Perl-Datei mit der Dateierweiterung .pm und notieren Sie als letzte Anweisung 1 ;
ein Modul einbinden	Verwenden Sie entweder use oder require
ein Paket festlegen	Verwenden Sie die Anweisung package und einen Bezeichner
ein Pragma anschalten oder ausschalten	Verwenden Sie use zum Anschalten und no zum Ausschalten
einen Namensraum verwenden	Stellen Sie den Namensraum voran und notieren Sie dann zweimal den Doppelpunkt
eine Klasse erstellen	Erzeugen Sie ein Modul und dort eine Konstruktor-Funktion sowie die gewünschten Methoden und Eigenschaften.

13.4 Übungen

Grundlagen zu OOP

Übungsdatei: --

Ergebnisdatei: --

1. Wie nennt man ein Objekt noch?
2. Mit welchen Anweisungen kann man Module einbinden?
3. Für welchen Wahrheitswert steht in Perl die Zahl 5?
4. Wie nennt man das Gegenstück zum Konstruktor?
5. Wie nennt man ein Paket oder Package noch?
6. Wie nennt man eine Referenz noch?

Objektorientiert programmieren

Übungsdatei: --

Ergebnisdatei: *HTMLGen.pl*, *HTMLGen.pm*

1. Erstellen Sie eine Moduldatei *HTMLGen.pm*.
2. Ordnen Sie den Code dem Paket zu, das sich aus dem Namen des Moduls ergibt.
3. Die Moduldatei soll vier Subroutinen enthalten. Die erste davon ist der Konstruktor mit Namen `new`.
4. Der Konstruktor bindet mit der Funktion `bless` eine Referenz an eine Klasse.
5. Mit `my $Obj = shift;` wird im Konstruktor der erste Parameter eingelesen, den der Konstruktor `new` übergeben bekommt.
6. Mit `my $Ref = {};` wird eine Referenz deklariert.
7. Mit `bless ($Ref, $Obj);` wird die nötige Verbindung zwischen Referenz und Objekt hergestellt.
8. Abschließend wird vom Konstruktor `$Ref` zurückgegeben.
9. Die vier übrigen Subroutinen in der Moduldatei sind sich alle recht ähnlich und fungieren als die Methoden der Klasse. Sie erzeugen mit Hilfe von `print`-Anweisungen HTML-Code.
10. In allen Methoden wird mit dem Aufruf von `shift` zuerst der Klassename ermittelt.
11. In der Methode `Beginn` wird mit dem zweiten Aufruf von `shift` der Wert für den Titel der Webseite ermittelt. Dieser soll der Variable `$Titel` zugewiesen werden.
12. Die Subroutine `Beginn` schreibt zudem den nötigen HTTP-Header für HTML-Ausgaben ("Content-type: text/html\n\n") sowie den Anfang der Webseite. Also der folgende String Anweisungen sollte generiert werden: "<html><head><title>\$Titel</title></head><body>\n"
13. Die Methode `Text` schreibt übergebenen Text in einen Div-Container.
14. Die Methode `Ende` schreibt das Ende der Webseite ("</body></html>\n").
15. Das Perl-Programm *HTMLGen.pl*, das Sie erstellen sollen, bindet mit `use` das Modul ein.
16. Mit dem Konstruktor wird ein Objekt erzeugt.
17. Die Methoden `Beginn`, `Text` und `Ende` werden nacheinander aufgerufen.

```
F:\PerlProgramme\Uebungsdateien\Kapitel_13>perl HTMLGen.pl
Content-type: text/html

<html><head><title>00-Perl</title></head><body>
<div>Generierter Text!</div>
</body></html>

F:\PerlProgramme\Uebungsdateien\Kapitel_13>
```

So sieht die Ausgabe in der Konsole aus – in einem Browser würde der HTML-Code interpretiert

14 Erweiterte CGI-Funktionalität

In diesem Kapitel erfahren Sie

- ✓ wie Sie das CGI-Programm mithilfe des CGI-Moduls erstellen
- ✓ wie Sie über das CGI-Modul auf Formulardaten zugreifen
- ✓ wie Sie HTML-Seiten mit Methoden des CGI-Moduls ausgeben
- ✓ wie Sie Cookies verwenden

Voraussetzungen

- ✓ HTML-Formulare und CGI-Programme erstellen
- ✓ Grundlagen zu Modulen
- ✓ Grundlagen zu OOP mit Perl

14.1 Das Modul *CGI.pm* verwenden

Für CGI-Programmierung mit Perl wird meist das standardmäßig vorhandene Modul `CGI.pm` verwendet. Es stellt eine Vielzahl von Funktionen zur Verfügung, die Sie im Zusammenhang mit CGI-Programmen verwenden können, und erleichtert damit die Programmierung. Es berücksichtigt viele Besonderheiten und spezielle Einsatzfälle, die bei der manuellen Auswertung von Formulardaten viel Programmcode benötigen.

Das Modul binden Sie durch die Anweisung `use CGI;` am Beginn des Programms ein. Da die Anwendung der Funktionen des CGI-Moduls über Objekte erfolgt, ist eine gewisse Erfahrungen in der objektorientierten Programmierung sinnvoll. Am Beginn des CGI-Programms müssen Sie eine neue Instanz des CGI-Objekts erzeugen und die Referenz darauf einer Variablen zuweisen:

```
use CGI;  
$cgivariable = new CGI;
```

- ✓ Mit der `use`-Anweisung wird das Modul *CGI.pm* eingebunden.
- ✓ Die Anweisung `new` (der Konstruktor des Moduls bzw. der Modulklasse) erstellt eine neue Instanz des CGI-Objekts. Die Referenz darauf kann in einer Variablen gespeichert werden.



Um mit dem CGI-Modul zu arbeiten, benötigen Sie nur wenige Kenntnisse über das Anwenden von Objekten in der Programmierung mit Perl. Die Grundlagen, die im Kapitel 13 vermittelt wurden, genügen auf jeden Fall. Weitere Informationen zur objektorientierten Programmierung finden Sie in der Perl-Hilfdatei `perlobj`.

Zugriff auf Konstanten und Methoden des CGI-Objekts

Das CGI-Objekt fasst alle bei der CGI-Programmierung notwendigen Variablen bzw. Konstanten (Eigenschaften) und Funktionen (Methoden) zusammen. Der Aufruf erfolgt dabei über die Objektvariable und den Operator `->`.

```
$cgivariable->eigenschaftenname;  
$cgivariable->methodenname (Argumente);
```

- ✓ Nach dem Namen der Objektvariablen wird der Operator `->` angegeben. Er ermöglicht den Zugriff auf Eigenschaften und Methoden des Objekts.
- ✓ Danach folgt der Name einer Konstante oder Methode.

Die Namen aller Eigenschaften und Methoden sowie Beispiele zu deren Verwendung finden Sie in der Dokumentation des Moduls *CGI.pm*.

Durch das Einbinden des Moduls mit der Anweisung `use CGI qw/:standard/;` ersparen Sie sich ggf. die Angabe der Objektinstanz `$cgivariable->` vor dem Namen von Eigenschaften und Methoden des Moduls.



14.2 Header ausgeben

Standard-Header ausgeben

Mit der `header`-Methode können Sie verschiedene Header-Zeilen erzeugen. Ohne Parameter liefert die Methode einen Header mit dem Content-Typ *text/html* zurück.

```
print $cgivariable->header (Argumente) ;
```

- ✓ Die `header`-Methode gibt als Ergebnis einen formatierten HTML-Header zurück.
- ✓ Wird kein Argument angegeben, erhalten Sie eine Header-Zeile für den Dokumententyp *text/html*.
- ✓ Die Methode liefert eine Zeichenkette, die mit der `print`-Anweisung ausgegeben werden kann.

Beispiel

Die folgenden Beispiele zeigen die Ausgabe eines Dokumententyps:

```
print $cgivariable->header;                                # Content-Type: text/html
print $cgivariable->header('text/html');                  # Content-Type: text/html
print $cgivariable->header('text/plain');                # Content-Type: text/plain
print $cgivariable->header(-type='text/plain');          # Content-Type: text/plain
```

Umleitung erzeugen

Mit einem speziellen Header, dem so genannten Location-Header, können Sie den aufrufenden Browser auf eine neue Adresse umleiten. Die `redirect`-Methode ermöglicht Ihnen, eine Umleitung einzurichten.

```
print $cgivariable->redirect (Adresse) ;
```

- ✓ Die `redirect`-Methode gibt als Ergebnis einen Location-Header zurück.
- ✓ Als Parameter muss eine Adresse angegeben werden, auf die der Browser verzweigen soll.

Sie können auf eine HTML-Seite des gleichen Webservers oder auf einen anderen Webserver umleiten. Auch Umleitungen auf CGI-Programme sind möglich.

Beispiel

```
print $cgivariable->redirect ('/fehler.html');
print $cgivariable->redirect ('http://www.rjs.de/index.html');
print $cgivariable->redirect ('/cgi-bin/form.cgi');
```

14.3 Formulardaten und Umgebungsdaten ermitteln

Auf Formulardaten zugreifen

Das CGI-Modul entbindet Sie vollständig von der Aufgabe, die übergebenen Formulardaten je nach verwendeter Übertragungsmethode auszulesen und zu decodieren. Die `param`-Methode liefert Ihnen die Namen und Werte der Formularelemente.

Die `param`-Methode kann auf verschiedene Weisen aufgerufen werden:

Aufruf der Methode	Erklärung
<code>@namen = \$cgivariablen->param;</code>	Liefert ein Array mit den Namen der Formularelemente
<code>\$wert = \$cgivariablen->param('name');</code>	Liefert den Wert eines Formularelements mit einem bestimmten Namen
<code>@wert = \$cgivariablen->param('name');</code>	Liefert ein Array mit mehreren Werten eines Formularelements, z. B. eines Auswahlfelds mit mehreren auswählbaren Elementen
<code>\$cgivariablen->param('name', 'wert');</code>	Legt einen neuen Wert für ein Formularelement fest



Da der Aufruf der Methode `param()` dem Ausführen eines Unterprogramms entspricht, können Sie diesen nicht innerhalb einer Zeichenkette durchführen, um Formularwerte wie skalare Variablen einzubinden.

Mit der folgenden Anweisung können Sie prüfen, ob Formulardaten an das CGI-Programm übermittelt wurden:

```
if ($cgivariablen->param) {
    ...
}
```

Schneller Zugriff auf Formularelemente

Mit der Methode `import_names` können Sie die Namen aller Formularelemente in einen sogenannten Namespace bzw. Namensraum importieren. Als Namespace können Sie einen oder mehrere Buchstaben festlegen.

Der Zugriff auf die Formulardaten erfolgt danach über Variablen, die mit dem Bezeichner des Namespace beginnen und durch zwei Doppelpunkte `::` getrennt den Namen des Formularelements angeben.

Beispiel

Die Formularelemente werden in den Namespace `F` importiert.

```
$cgivariablen->import_names('F');
print "$F::vorname $F::name";
print "Sie haben die Farben @F::farben ausgewählt. ";
```

Auf Umgebungsdaten zugreifen

Das CGI-Modul verfügt über verschiedene Methoden, die Informationen aus den Umgebungsdaten zurückliefern. Dazu gehören Daten über den verwendeten Browser, die Serversoftware und den Namen des CGI-Programms.

Methode	Erklärung
Accept()	Array mit Dokumenttypen (MIME-Typen), die der Browser akzeptiert
path_info()	Ermittelt die an das CGI-Programm übergebenen Verzeichnisnamen, z. B. /cgi-bin/name.cgi/mein/name liefert /mein/name
referer()	URL der Seite, von der aus das CGI-Programm aufgerufen wurde, enthält keinen Wert, wenn das CGI-Programm direkt aufgerufen wurde
remote_host()	Hostname (oder IP-Adresse) des Besuchers
request_method()	Übertragungsmethode der Formulardaten
script_name()	Verzeichnis und Name des CGI-Programms
server_name()	Hostname des Webservers
url(-full=>1)	Vollständige Adresse des CGI-Programms ohne Query-String
url(-query=>1)	Vollständige Adresse des CGI-Programms mit Query-String
url(-relative=>1)	Relative Adresse des CGI-Programms
user_agent()	Bezeichnung (Name und Versionsnummer) des Browsers

Mit der Methode `script_name()` können Sie sehr einfach Selbstaufrufe des CGI-Programms programmieren, die unabhängig vom tatsächlichen Namen des Programms sind.



Beispiel: vorwahlsuche.html, vorwahlsuche.cgi, vorwahlen.txt

Das CGI-Programm realisiert die Suche nach der Vorwahl für eine eingegebene Stadt. Die Vorwahlen sind in einer Textdatei gespeichert, die in jeder Zeile einen Stadtnamen und getrennt durch ein Komma die Vorwahl enthält. Das Programm verwendet das CGI-Modul.

```
#! "F:/xampp/perl/bin/perl.exe"
use strict;
① use CGI;
use CGI::Carp qw( fatsalsToBrowser );
my $datei="vorwahlen.txt";
my $zeile;
my $vorwahl;

② my $cgi = new CGI;

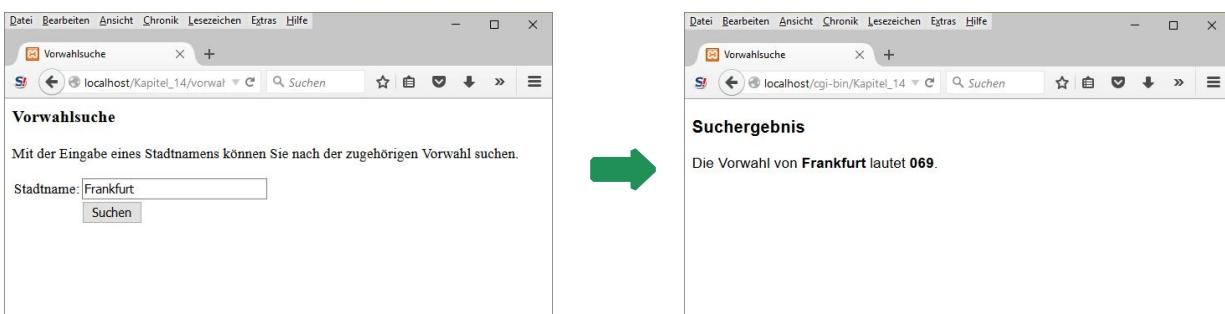
③ if (!$cgi->param) {
    print $cgi->redirect('/vorwahlsuche.html');
} else {
④     print $cgi->header;
⑤     my $stadt=$cgi->param('stadt');
    open(DATEI,<$datei") || die("Datei $datei kann nicht geöffnet werden!");
```

```

while($zeile=<DATEI>) {
⑥    if ($zeile=~/^$stadt; (\d+)/i) {
        $vorwahl=$1;
    }
}
print <>ENDE;
<html><head>
<title>Vorwahlsuche</title>
</head><body>
<font face="Arial">
<h3>Suchergebnis</h3>
ENDE
⑦    if ($vorwahl) {
        print "<p>Die Vorwahl von <b>$stadt</b> lautet <b>$vorwahl</b>.</p>";
    } else {
        print "<p>Die Stadt <b>$stadt</b> ist nicht in der Datenbank.</p>";
    }
print "</font></body></html>";
}

```

- ① Mit der `use`-Anweisung wird das CGI-Modul eingebunden.
Mit `use CGI::Carp qw(fatalsToBrowser);` wird danach ein Feature des CGI-Moduls genutzt, um ggf. Fehlermeldungen in den Browser umzuleiten. Das ist in der Praxis mit Vorsicht anzuwenden, denn nicht in jedem Fall soll ein Anwender Fehlermeldungen des Systems sehen. Aber zu Entwicklungszwecken ist diese Anweisung sinnvoll.
- ② Eine Instanz des CGI-Objekts wird mit der `new`-Anweisung erzeugt und in der Variablen `$cgi` gespeichert.
- ③ An dieser Stelle wird geprüft, ob das CGI-Programm ohne übergebene Formulardaten aufgerufen wurde. In diesem Fall soll auf die Suchseite `vorwahlsuche.html` umgeleitet werden.
- ④ Durch den Aufruf der `header`-Methode ohne Parameter wird eine Header-Zeile für den Inhaltstyp `text/html` erzeugt.
- ⑤ Hier wird ermittelt, welcher Wert in das Formularelement `stadt` eingetragen wurde. Der Wert wird in einer skalaren Variablen gespeichert.
- ⑥ Die Vorwahldatei wird zeilenweise durchlaufen. Mit einem regulären Ausdruck wird für jede Zeile geprüft, ob der Stadtname am Beginn der Zeile vorhanden ist. Die nach dem Komma folgenden Ziffern werden gruppiert, um in der folgenden Zeile auf die gefundene Vorwahl zugreifen zu können.
- ⑦ Die Suche war erfolgreich, wenn in der Variablen `$vorwahl` ein Wert gespeichert wurde. In diesem Fall wird die Vorwahl ausgegeben. Andernfalls erfolgt die Ausgabe einer Fehlermeldung.



Suche nach der Vorwahl für die Stadt Frankfurt

14.4 HTML-Elemente erzeugen

Bisher haben Sie bei der Ausgabe einer HTML-Seite mit einem CGI-Programm alle HTML-Tags direkt ausgegeben. Das CGI-Modul definiert jedoch Methoden für alle wichtigen HTML-Tags, mit denen Sie die Elemente einer HTML-Seite sowie Formulare auf einfache Weise definieren und ausgeben können.

Die entsprechenden Methoden besitzen den Namen des jeweiligen HTML-Tags. Als Argumente können Texte, die in diese Tags eingeschlossen werden sollen, oder Formatierungsoptionen übergeben werden. Der Vorteil bei der Verwendung der Methoden gegenüber der direkten Ausgabe liegt in einem übersichtlicheren Programmcode.

Sie können die direkte Ausgabe von HTML-Quelltext und die Ausgabe über die Methoden des CGI-Moduls auch mischen.



Beispiel

Um eine Überschrift der Größe 1 in einer HTML Seite zu erstellen, wird folgender HTML-Quelltext erstellt:

```
<h1>Das ist die Überschrift</h1>
```

Die gleiche Ausgabe erhalten Sie mit folgendem Aufruf einer Methode des CGI-Moduls:

```
print $cgivariable->h1('Das ist eine Überschrift');
```

Syntax der HTML-Methoden mit einleitenden und schließenden Tags

```
print $cgivariable->htmltag('Text');
```

- ✓ Es wird eine Instanz des CGI-Objekts benötigt. Danach wird der Operator -> angegeben.
- ✓ Der Name der Methode entspricht dem Namen des HTML-Tags in Kleinbuchstaben. Als Argument wird der in die HTML-Tags einzuschließende Text angegeben.
- ✓ Mehrere Argumente müssen durch Kommata getrennt werden. Bei der Ausgabe werden diese aneinander gereiht.
- ✓ Als Resultat wird eine Zeichenkette zurückgegeben, die den in die betreffenden HTML-Tags eingeschlossenen Text enthält.

Die Methodennamen entsprechen den kleingeschriebenen Namen der HTML-Tags. Eine Ausnahme bilden folgende Methoden: Sub(), Tr(), Link(), Select(), da sie mit Schlüsselwörtern von Perl übereinstimmen.

Attribute angeben

Viele der HTML-Methoden akzeptieren weitere Argumente, die den Attributen des HTML-Tags entsprechen und nach folgender Syntax angegeben werden:

```
print $cgivariable->htmltag({attribut => 'wert'}, 'Text');
```

- ✓ Die Attribute werden in geschweifte Klammern eingeschlossen.
- ✓ Zuerst wird der Name des Attributs angegeben. Mit dem Operator => wird der Wert des Attributs angeschlossen. Mehrere Attribute müssen durch Kommata getrennt werden.
- ✓ Nach der Angabe der Attribute können durch Kommata getrennt weitere Argumente angegeben werden.

Beispiel

```
print $cgivariable->img({src='bild.gif', alt='Bild', border='0'});
print $cgivariable->a({href='link.html'}, 'Das ist ein Hyperlink');
print $cgivariable->hr({noshade=undef, size='2'});
```

Der Programmcode ergibt folgenden HTML-Quelltext:

```

<a href="link.html">Das ist ein Hyperlink</a>
<hr noshade size="2">
```

 Wenn Sie einem Attribut den Wert undef zuweisen, taucht es innerhalb des HTML-Tags ohne Wertzuweisung auf.

HTML-Tags öffnen und schließen

Im CGI-Modul werden zusätzlich Methoden definiert, die ein HTML-Tag nur Einleiten oder Schließen. Diese Methoden beginnen stets mit der Bezeichnung `start_` bzw. `end_`. Daran schließt sich der Name des HTML-Tags an.

 Sie sollten diese Methoden vor allem dann verwenden, wenn zwischen dem einleitenden und dem schließenden HTML-Tag sehr viel Text angegeben wird.

Beispiel

```
print $cgivariable->start_p({style=>'color:blue'});
print "Text des Absatzes";
print $cgivariable->end_p;
```

Der Programmcode ergibt folgenden HTML-Quelltext:

```
<p style="color:red">Text des Absatzes</p>
```

HTML-Seite einleiten und beenden

Besonderheiten sind die Methoden `start_html()` zum Einleiten einer HTML-Seite und `end_html()` zum Beenden einer HTML-Seite. Die `start_html`-Methode definiert alle einleitenden HTML-Tags einschließlich des HEAD-Bereichs.

Syntax der `start_html`-Methode

```
print $cgivariable->start_html('Titel');
print $cgivariable->start_html(-title=>'Titel', -meta=>'...', -
script=>'...', ...);
```

- ✓ Es wird eine Instanz des CGI-Objekts benötigt. Nach dem Operator `=>` wird der Methodename `start_html` angegeben.
- ✓ Soll nur der Seitentitel angegeben werden, muss er als einziges Argument übergeben werden.
- ✓ Es können mehrere Optionen angegeben werden, die für einzelne HTML-Tags stehen. Der Optionsname wird mit einem Bindestrich eingeleitet. Danach folgt der Operator `=>` und der Wert der Option bzw. in geschweifte Klammern eingeschlossene Attribute.
- ✓ Mehrere Optionen werden durch Kommata getrennt.

Beispiel

```
print $cgivariable->start_html(-title => 'Titel der Seite',
    -meta => {keywords => 'cgi perl web',
    description=>'Anleitung zum Erstellen von CGI-Programmen'},
    -BGCOLOR=>'gray',
    -TEXTCOLOR=>'black');
print $cgivariable->h1('CGI-Programmierung mit Perl');
print $cgivariable->end_html;
```

Der Programmcode ergibt je nach Version des CGI-Moduls beispielsweise folgenden HTML-Quelltext:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
 "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US">
<head><title>Titel der Seite</title>
<meta name="description" content="Anleitung zum Erstellen von CGI-
 Programmieren" />
<meta name="keywords" content="cgi perl web" />
</head><body textcolor="black" bgcolor="gray">
<h1>CGI-Programmierung mit Perl</h1>
</body></html>
```

Formulare erstellen

Auch für alle Formularelemente gibt es spezielle Methoden des CGI-Objekts, wie z. B. `textfield()` oder `textarea()`. Diese besitzen jedoch meist nicht die Namen der entsprechenden HTML-Tags. Zum Einleiten und Schließen eines Formulars können Sie die Methoden `startform()` und `endform()` verwenden.

Die Tabelle zeigt die Verwendung der wichtigsten Methoden für die Formularerstellung und die möglichen Optionen:

Methoden zur Generierung von Formularelementen	Beschreibung	Optionen
<code>startform('Methode', 'CGI-Programm')</code>	Leitet ein Formular ein und legt die Übertragungsmethode bzw. das aufzurufende CGI-Programm fest	
<code>endform()</code>	Beendet ein Formular	
<code>textfield()</code>	Eingabefeld	<code>name, default, size, maxlength</code>
<code>textarea()</code>	Mehrzeiliges Texteingabefeld	<code>name, default, rows, columns</code>
<code>password_field()</code>	Passwortfeld	<code>name, value, size, maxlength</code>
<code>popup_menu()</code>	Einzeliges Listenfeld	<code>name, value, labels, default</code>

Methoden zur Generierung von Formularelementen	Beschreibung	Optionen
scrolling_list()	Mehrzeiliges Listenfeld	name, values, default, size, multiple, labels
checkbox()	Kontrollfeld	name, value, checked, label
radio_group()	Optionsfelder	name, values, default, linebreak, labels
image_button()	Bildschaltfläche	name, src, align
submit()	Abschicken-Schaltfläche	name, value
hidden()	Verstecktes Feld	name, default



Die Optionen werden jeweils mit einem Bindestrich eingeleitet und durch Kommata getrennt. Den Wert der Optionen geben Sie nach dem Operator => an, beispielsweise -name=>'vorname'.

Beispiel: *gaestebuch.cgi*

Das CGI-Programm zeigt beim Aufruf ohne übergebene Formulardaten eine HTML-Seite mit einem Gästebuchformular an.

```
#! "F:/xampp/perl/bin/perl.exe"
use strict;
use CGI;
my $cgi = new CGI;
① print $cgi->header;
② print $cgi->start_html('Gästebuch');

③ if (!$cgi->param) {
④     print $cgi->h3({style=>'font-family:sans-serif'}, 
                    'Eintrag ins Gästebuch');
⑤     print $cgi->startform('POST',$cgi->script_name);
⑥     print $cgi->b('Ihr Name: ').$cgi->br;
⑦     print $cgi->textfield(-name=>'name',-size=>40).$cgi->br;
⑧     print $cgi->b('Ihre Nachricht: ').$cgi->br;
        print $cgi->textarea(-name=>'text',-rows=>'6',-cols=>'30').$cgi->br;
        print $cgi->br.$cgi->submit;
        print $cgi->endform;
    }
print $cgi->end_html;
```

- ① Mit dem Aufruf der Methode header wird die Headerzeile mit dem Dokumenttyp *text/html* ausgegeben.
- ② Die start_html-Methode erzeugt den Anfang der HTML-Seite. Als Argument wird der Seitentitel übergeben.
- ③ In einer if-Anweisung wird geprüft, dass keine Formulardaten übergeben wurden. Nur in diesem Fall wird das Gästebuchformular ausgegeben.
- ④ Mit der h3-Methode wird eine Überschrift in der Größe 3 mit dem als Argument übergebenen Text erzeugt. Zusätzlich erfolgt die Angabe eines style-Attributs, das die zu verwendende Schrift bestimmt.

- ⑤ An dieser Stelle wird das Formular eingeleitet. Das Formular soll die POST-Übertragungsmethode verwenden. Da das CGI-Programm selbst die Verarbeitung der Formulardaten übernehmen soll, wird mit der Methode `script_name()` ein Verweis auf die Adresse des aktuellen Programms übergeben.
- ⑥ Die Beschriftung des Formularelements wird mit fetter Schrift formatiert. Danach erfolgt die Ausgabe eines Zeilenumbruchs.
- ⑦ Hier erfolgt die Ausgabe eines Eingabefelds für den Namen des Eintragenden.
- ⑧ Die `textarea`-Methode erzeugt ein Eingabefeld mit sechs Zeilen und 30 Spalten für den Gästebucheintrag.

The screenshot shows a web browser window with the title 'Gästebuch'. The address bar displays 'localhost/cgi-bin/Kapitel_14'. The main content area is titled 'Eintrag ins Gästebuch'. It contains two input fields: one for 'Ihr Name:' and another for 'Ihre Nachricht:', which is a multi-line text area. Below these fields is a button labeled 'Daten absenden'.

Das CGI-Programm erzeugt diese HTML-Seite

14.5 Cookies verwenden

Der Aufruf einer HTML-Seite durch einen Browser ist für den Webserver ein einmaliges, abgeschlossenes Ereignis. Auch wenn Sie nacheinander mehrere HTML-Seiten eines Webservers abrufen oder eine Seite jeden Tag besuchen, gibt es für den Webserver keine Möglichkeit zu erkennen, dass Sie die Webseite schon einmal besucht haben.

Es gibt im HTTP-Protokoll keine Möglichkeit, die Daten eines Besuchers über mehrere Besuche einer Webseite zu speichern. In einem Shop-System ist es jedoch beispielsweise sinnvoll, den Warenkorb so lange zu speichern, bis der Besucher die Waren bestellt.

Cookies bieten diese Möglichkeit, indem sie einem Webserver erlauben, Daten auf der Seite des Browsers zu speichern. So können Sie beispielsweise dem Besucher eines Shop-Systems eine Kundensummer zuteilen und als Cookie speichern. Beim nächsten Besuch werden die Daten des Cookies automatisch an den Webserver übermittelt und der Kunde kann identifiziert werden.

Cookies (engl. cookie=Keks) sind Daten, die als Element einer HTTP-Anforderung vom Webserver an den Browser geschickt werden. Der Browser speichert Cookies auf der lokalen Festplatte und sendet sie beim Aufruf einer Webseite an den Webserver zurück. Dabei speichern verschiedene Browser die Cookies intern unterschiedlich, aber das hat keine Auswirkung auf die Anwendung aus Perl.

Cookies werden nur für eine bestimmte Webseite gespeichert. Es ist nicht möglich, durch ein CGI-Programm Cookies auszuwerten, die eine fremde Webseite gesetzt hat. Gleichzeitig besitzen Cookies ein Verfallsdatum und werden gelöscht, wenn dieses überschritten wurde. Grundsätzlich sind Cookies aber vielfältig eingeschränkt und werden aktuell durch neue Techniken aus HTML5 wie Local Storage abgelöst.



Cookies setzen

Mit dem CGI-Modul ist es sehr einfach, Cookies zu verwenden. Zuerst können Sie mit der `cookie`-Methode ein neues Cookie erstellen. Danach können Sie es mit der `header`-Methode an den Browser senden. Cookies werden innerhalb des Headers einer HTTP-Anfrage übertragen.

```
$cookie = $cgivariabile->cookie(-option1=>'...', -option2=>'...', ...);
print $cgivariabile->header(-cookie=>$cookie);
```

- ✓ Mit der `cookie`-Methode erzeugen Sie ein Cookie und speichern es in einer skalaren Variable.
- ✓ Als Argumente werden verschiedene Optionen übergeben, die beispielsweise den Namen und Wert des Cookies festlegen.
- ✓ Beim Aufruf der `header`-Methode wird der Option `-cookie` das erzeugt Cookie zugewiesen.

Folgende Optionen können Sie beim Aufruf der `cookie`-Methode verwenden:

Option	Erklärung	Beispielwert
<code>-name=>'Cookie-Name'</code>	Name des Cookies, dieser wird zum späteren Auslesen der gespeicherten Werte verwendet	BesucherID
<code>-value=>'Cookie-Wert'</code>	Wert, der im Cookie gespeichert werden soll	10023
<code>-path=>'Pfadangabe'</code>	relativer Pfad der Webadresse, für die das Cookie gelten soll	/
<code>-expires=>'Datum'</code>	Datum, an dem das Cookie verfällt.	+3M
<code>-domain=>'Domainname'</code>	Domainname, für den das Cookie gültig ist	meinserver.de

Bei der `-expires`-Option können Sie eine relative Zeitangabe angeben. So steht die Angabe `+3M` für das Datum von heute in drei Monaten. Die folgende Übersicht zeigt einige Beispiele für die Zeitangabe:

Zeitangabe in der <code>-expires</code> -Option	Erklärung
<code>+60s</code>	in 60 Sekunden
<code>+15m</code>	in 15 Minuten
<code>+7d</code>	in 7 Tagen
<code>now</code>	jetzt
<code>+3M</code>	in 3 Monaten
<code>+1y</code>	in einem Jahr

Cookies löschen



Wenn Sie negative Zeitangaben, wie beispielsweise `-1d`, setzen, wird das Cookie gelöscht. Sie können dieses Verhalten gezielt verwenden, um nicht mehr benötigte Cookies aufzuheben.

Cookies lesen

Bei jedem Aufruf einer HTML-Seite sendet der Browser die möglicherweise für diese Seite gespeicherten Cookies automatisch an den Webserver. Das CGI-Modul wertet diese Daten aus und ermöglicht Ihnen den Zugriff ebenfalls über die `cookie`-Methode.

```
$cookiewert = $cgivariabile->cookie('Cookie-Name');
@cookenamen = $cgivariabile->cookie;
```

- ✓ Wird beim Aufruf der `cookie`-Methode nur der Name eines Cookies als Argument übergeben, liefert die Methode den Wert des Cookies.
- ✓ Erfolgt der Aufruf der Methode ohne die Übergabe eines Arguments, liefert sie die Namen aller gesetzten Cookies als Array zurück.

Beispiel: cookie.cgi

Mithilfe eines Cookies soll gezählt werden, wie oft der Besucher das CGI-Programm aufruft.

```
#! "F:/xampp/perl/bin/perl.exe"
use strict;
use CGI;
my $cgi = new CGI;
① my $anzahl_besuche=$cgi->cookie('AnzahlBesuche');
② my $cookie=$cgi->cookie(-name=>'AnzahlBesuche',
                           -value=>$anzahl_besuche+1,
                           -expires=>'+1y');
③ print $cgi->header(-cookie=>$cookie);
print $cgi->start_html('Besuchszähler');
④ print $cgi->p("Sie sind das $anzahl_besuche. Mal auf dieser Seite");
print $cgi->end_html;
```

- ① Hier wird durch einen Aufruf der `cookie`-Methode der Wert des Cookies `AnzahlBesuche` ausgelesen.
- ② Das Cookie wird neu erstellt und dabei die Anzahl der Besuche um den Wert 1 erhöht. Das Cookie wird maximal ein Jahr gespeichert.
- ③ Um das Cookie zu setzen, wird es innerhalb des Headers übermittelt.
- ④ Hier erfolgt die Ausgabe des Cookies.

Die `header`-Methode darf nur einmal in einem CGI-Programm aufgerufen werden. Wenn Sie mehrere Cookies verwenden möchten, müssen Sie diese einzeln mit der `cookie`-Methode erstellen und beim Aufruf der `header`-Methode in folgender Form angeben: `$cgi->header(-cookie=>[$cookie1, $cookie2]);`



14.6 Schnellübersicht

Sie möchten ...	
das CGI-Modul einbinden	<code>use CGI;</code>
eine neue Instanz des CGI-Objekts erzeugen	<code>\$cgi=new CGI;</code>
einen Header für eine HTML-Seite ausgeben	<code>print \$cgi->header;</code>
eine Umleitung auf eine andere Adresse realisieren	<code>print \$cgi->redirect('Adresse');</code>
den Wert eines Formularelements ermitteln	<code>\$wert=\$cgi->param('Name');</code>
die Namen aller Formularelemente ermitteln	<code>@namen=\$cgi->param;</code>
ein HTML-Element ausgeben	<code>print \$cgi->htmltag(...);</code>
ein einleitendes HTML-Tag ausgeben	<code>print \$cgi->start_htmltag(...);</code>
ein schließendes HTML-Tag ausgeben	<code>print \$cgi->end_htmltag(...);</code>
eine HTML-Seite einleiten	<code>print \$cgi->start_html(...);</code>
eine HTML-Seite beenden	<code>print \$cgi->end_html(...);</code>
ein Formular einleiten	<code>print \$cgi->startform(...);</code>
ein Formular beenden	<code>print \$cgi->endhtml(...);</code>
ein Cookie erzeugen	<code>\$cookie=\$cgi->cookie(...);</code>

Sie möchten ...	
ein Cookie auslesen	<code>\$wert=\$cgi->cookie('Cookie-Name');</code>
ein Cookie setzen	<code>print \$cgi->header(-cookie=>\$cookie);</code>

14.7 Übungen

Grundlagen zu CGI-Modulen

Übungsdatei: --

Ergebnisdatei: --

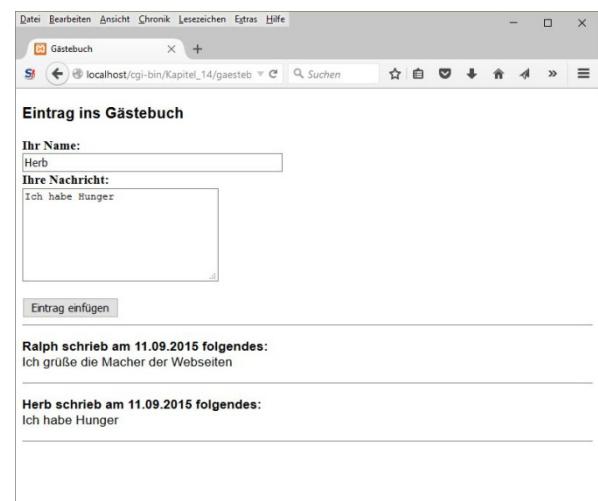
1. Welche Vorteile bringt der Einsatz des CGI-Moduls?
2. Erläutern Sie die Begriffe „Methode“ und „Konstante“.
3. Wie binden Sie das CGI-Modul ein, und wie erfolgt der Zugriff auf Methoden und Konstanten?
4. Wie können Sie HTML-Elemente mithilfe des CGI-Moduls ausgeben?
5. Was sind Cookies, und wozu können Sie diese verwenden?
6. Erläutern Sie das Setzen und Lesen von Cookies mit dem CGI-Modul.

CGI-Module verwenden

Übungsdatei: --

Ergebnisdatei: *gaestebuch.cgi*

1. Es soll ein CGI-Programm für ein Gästebuch erstellt werden. Das Programm soll vorhandene Einträge, ein Formular für neue Einträge anzeigen und neue Einträge in der Datei *gaestebuch.txt* speichern.
2. Binden Sie das CGI-Modul ein, und erstellen Sie eine neue Instanz des CGI-Objekts.
3. Geben Sie den korrekten Header für den verwendeten Dokumententyp aus.
4. Prüfen Sie, ob Formulardaten übergeben wurden. Ist dies der Fall, soll der neue Gästebucheintrag in der Datei gespeichert werden.
5. Ermitteln Sie das aktuelle Datum. Speichern Sie danach die Daten in der Form `name | datum | text` in der Datei. Das Datum soll mit zweistelligen Tages- und Monatsangaben und einer vierstelligen Jahreszahl gespeichert werden.
6. Geben Sie den Beginn der HTML-Seite mit einem geeigneten Seitentitel aus.
7. Zeigen Sie ein Eingabeformular für neue Gästebuch-einträge an. Der Besucher soll einen Namen und den Text für den Eintrag eingeben können. Das Formular soll das eigene CGI-Programm aufrufen.
8. Danach sollen alle vorhandenen Gästebucheinträge angezeigt werden. Öffnen Sie dazu die Datei, und durchlaufen Sie diese zeilenweise.
9. Trennen Sie die Zeilen am Zeichen `\n` auf, und geben Sie die entsprechenden Daten aus (vgl. die obenstehende Abbildung).



Das Gästebuch in Aktion

15 Datenbanken mit MySQL

In diesem Kapitel erfahren Sie

- ✓ wie Sie einen MySQL-Datenbankserver installieren
- ✓ wie Sie in Perl eine Verbindung zum Datenbankserver herstellen
- ✓ wie Sie eine SQL-Abfrage erstellen
- ✓ wie Sie die Ergebnisdatensätze einer Abfrage ermitteln

Voraussetzungen

- ✓ Aufbau und Struktur relationaler Datenbanken, SQL-Anweisungen
- ✓ CGI-Programme mit Perl erstellen
- ✓ Umgang mit Modulen
- ✓ Grundlagen der OOP in Perl

15.1 Datenbankmanagementsysteme und Perl

Datenbankmanagementsysteme sind spezielle Server zum Speichern großer bis sehr großer Datenmengen. Sie werden meist verwendet, um miteinander verknüpfte Daten organisiert in Tabellen und Datenbanken zu speichern. Dabei bieten sie deutlich mehr Möglichkeiten und Leistung, als das Speichern von Daten beispielsweise in Textdateien.

Besonders bei der Entwicklung von Webseiten wird das weit verbreitete Datenbankmanagementsystem MySQL verwendet. Es ist kostenlos für viele Betriebssysteme verfügbar und bietet Unterstützung für das relationale Datenmodell. Deshalb wird die Zusammenarbeit von Perl mit einem Datenbankmanagementsystem beispielhaft mit MySQL gezeigt. Da der Datenbankzugriff über Module erfolgt, können Sie Perl prinzipiell mit jedem Datenbanksystem betreiben, für das ein entsprechendes Modul zur Verfügung steht, wie beispielsweise Oracle, PostgreSQL, ODBC, Ingres oder dBase.

Perl bietet keine direkte Unterstützung für den Zugriff auf Datenbankmanagementsysteme bzw. Datenbanken. Sie können dafür jedoch das Standardmodul **DBI** (DataBase Interface) verwenden. Es stellt unabhängig vom verwendeten Datenbanksystem Befehle für den Datenzugriff zur Verfügung.

Daten mit SQL abfragen und verändern

Mit einem Datenbanksystem kommunizieren Sie über eine spezielle Sprache: SQL. SQL (engl. Structured Query Language=strukturierte Abfragesprache) wurde speziell für das Verwalten von Datenbanken und das Abfragen von Daten entworfen. Mit den SQL-Anweisungen sind Sie in der Lage, eine Datenbank anzulegen, Datensätze einzufügen, abzurufen sowie Tabellen zu erstellen, zu ändern oder zu löschen.



Die Sprache SQL ist relativ einfach zu erlernen, bietet jedoch sehr mächtige Funktionen und eine spezielle Syntax, auf die in diesem Kapitel nicht eingegangen werden kann. Eine ausführliche Einleitung in die Sprache SQL und das Erstellen von Datenbanken finden Sie im HERDT-Buch *SQL - Grundlagen und Datenbankdesign*. Bei den meisten Anwendungen kommen Sie mit den vier Grundanweisungen SELECT, INSERT, DELETE und UPDATE aus.

DBI als universelle Schnittstelle

DBI ist, wie *CGI.pm*, ein Zusatzmodul, das Perl um verschiedene Befehle erweitert. Es stellt dabei eine universelle Schnittstelle für den Zugriff auf verschiedene Datenbanksysteme zur Verfügung. Beim Aufruf des Moduls legen Sie fest, mit welchem Datenbanksystem Sie arbeiten wollen. DBI lädt danach automatisch die passenden Treiber, wenn diese vorhanden sind. Die für das DBI verwendeten Treiber werden **DBD** (engl. DataBase Driver = Datenbanktreiber) genannt. Über die Funktionen des Moduls führen Sie SQL-Abfragen aus und ermitteln die Ergebnisse einer Abfrage.

Das DBI-Modul wird meist standardmäßig bei der Installation von Perl eingerichtet. Dazu gehören auch die wichtigsten Datenbanktreiber. Die aktuelle Version des DBI erhalten Sie im CPAN (Comprehensive Perl Archive Network) unter der Adresse <http://search.cpan.org/search?dist=DBI>. Dort erhalten Sie auch alle verfügbaren Datenbanktreiber. Die Online-Dokumentation des Moduls finden Sie unter der Adresse <http://www.perldoc.com/cpan/DBI.html>.



15.2 Installation von MySQL

Die neusten Versionen von MySQL können unter der URL <http://www.mysql.com/downloads/mysql.html> bezogen werden. Allerdings ist MySQL auch bei XAMPP dabei. Das macht die manuelle Installation und Konfiguration überflüssig. Mit dem XAMPP-Control-Panel können Sie den MySQL-Server bequem starten (vgl. Anhang).

MySQL verwalten

Der MySQL-Server arbeitet unsichtbar im Hintergrund, wenn er etwa über das XAMPP Control Panel gestartet wurde.

The screenshot shows the XAMPP Control Panel interface. At the top, it displays "XAMPP Control Panel v3.2.1". Below this is a table with service information:

Service	Module	PID(s)	Port(s)	Actions
	Apache	6700 6372	80, 443	Stop Admin Config Logs
	MySQL	5408	3306	Stop Admin Config Logs
	FileZilla			Start Admin Config Logs
	Mercury			Start Admin Config Logs
	Tomcat			Start Admin Config Logs

On the right side of the panel, there are several buttons: Config, Netstat, Shell, Explorer, Services, Help, and Quit. Below the table, a log window displays the following text:

```

08:13:10 [main] You are not running with administrator rights! This will work for
08:13:10 [main] most application stuff but whenever you do something with services
08:13:10 [main] there will be a security dialogue or things will break! So think
08:13:10 [main] about running this application with administrator rights!
08:13:10 [main] XAMPP Installation Directory: "F:\xampp\"
08:13:10 [main] Checking for prerequisites
08:13:11 [main] All prerequisites found
08:13:11 [main] Initializing Modules
08:13:11 [main] Starting Check-Timer
08:13:11 [main] Control Panel Ready
08:13:14 [Apache] Attempting to start Apache app...
08:13:14 [Apache] Status change detected: running
08:13:15 [main] Executing "F:\xampp\"
09:27:12 [mysql] Attempting to start MySQL app...
09:27:13 [mysql] Status change detected: running

```

Webserver Apache und MySQL-Server gestartet

Wenn Sie im Control Panel die Schaltfläche **Admin** anklicken, werden Sie zu einer Web-basierten Administrationsoberfläche mit Namen **phpMyAdmin** weitergeleitet, über die Sie MySQL bequem und vollständig administrieren können.

The screenshot shows the phpMyAdmin configuration page for a MySQL server at 127.0.0.1. It includes sections for general settings (character set: utf8mb4_general_ci), display settings (language: Deutsch - German, font size: 82%), and server details (Apache/2.4.10, PHP/5.6.3). The sidebar lists databases like bestell Datenbank, cdc01, and test.

Web-basierende Verwaltung von MySQL

i Die Administrationsoberfläche phpMyAdmin basiert offensichtlich auf PHP. Da sie standardmäßig mit XAMPP ausgeliefert wird, können Sie sie bequem auch bei der Programmierung mit Perl nutzen. Es gibt allerdings auch ein auf Perl basierendes Tool mit Namen **perlMyAdmin** (<http://sourceforge.net/projects/perlmyadmin/>). Dieses Tool müssen Sie allerdings in Ihrem XAMPP-System nachinstallieren. Sie erhalten eine komprimierte Datei, die Sie im CGI-Verzeichnis einfach extrahieren. Ggf. müssen Sie die Shebang anpassen und die Rechte der Datei auf ausführbar setzen.

The screenshot shows the SourceForge project page for PerlMyAdmin. It features a summary section with 4.5 stars, 3 downloads this week, and a last update date of 2013-03-22. A prominent green 'Download' button links to PerlMyAdmin-1.0.0-beta3.zip. The page also includes sections for recommended projects (Flat-File Database for perl, MySQL::Admin, cpan MySQL::Admin, WWWdb) and latest tech jobs.

Download von perlMyAdmin

Wenn Sie im Control Panel die Schaltfläche *Config* anklicken, sehen Sie in einem Editor die Konfigurationsdateien von MySQL.

```
# Example MySQL config file for small systems.
#
# This is for a system with little memory (<= 64M) where MySQL is only used
# from time to time and it's important that the mysqld daemon
# doesn't use much resources.
#
# You can copy this file to
# F:/xampp/mysql/bin/my.cnf to set global options,
# mysql-data-dir/my.cnf to set server-specific options (in this
# installation this directory is F:/xampp/mysql/data) or
# ~/.my.cnf to set user-specific options.
#
# In this file, you can use all long options that a program supports.
# If you want to know which options a program supports, run the program
# with the "--help" option.

# The following options will be passed to all MySQL clients
[client]
# password      = your_password
port            = 3306
socket          = "F:/xampp/mysql/mysql.sock"

# Here follows entries for some specific programs
```

Die Konfigurationsdateien von MySQL

Sie können MySQL aber auch ohne XAMPP verwalten. Mit dem zum Installationspaket gehörenden MySQL-Monitor können Sie eine Verbindung zum Server aufbauen und SQL-Anweisungen ausführen.

Sie können dies verwenden, um Datenbanken und Tabellen zu erstellen. Die Eingabe und Abfrage der Daten wird danach meist durch ein Perl-Programm durchgeführt.

- ▶ Öffnen Sie eine Konsole und geben Sie im Unterverzeichnis *bin* Ihrer MySQL-Installation die folgende Anweisung ein:

mysql

Der MySQL-Monitor meldet sich mit verschiedenen Statusinformationen.

- ▶ Geben Sie die gewünschten SQL-Anweisungen ein. Schließen Sie diese mit einem Semikolon  und  ab.
- ▶ Verlassen Sie den MySQL-Monitor mit dem Befehl *exit* .

```
F:\xampp\mysql\bin>mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.21 MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

Der MySQL-Monitor

Wenn Sie auf einen MySQL-Server zugreifen, für den Sie bereits einen Benutzernamen und ein Passwort besitzen, müssen Sie den MySQL-Monitor mit folgenden Optionen aufrufen:

```
mysql --user=benutzername --password=passwort datenbankname
```

15.3 DBI-Modul installieren

Es kann sein, dass Ihnen das DBI-Modul automatisch in Ihrer Perl-Installation zur Verfügung steht. Dann können Sie direkt Ihre Perl-Programme erstellen und anwenden. Das sollte bei der Verwendung von XAMPP der Fall sein. Es kann aber auch sein, dass Sie das Modul nachinstallieren müssen. Wie das geht, unterscheidet sich je nach Perl-Distribution als auch nach dem Betriebssystem. Die genauen Details sind in der Dokumentation des DBI-Moduls beschrieben, aber im Grunde erfolgt die Installation eines CPAN-Moduls immer gleich. Das gilt damit auch für das DBI-Modul.

- ✓ Die erste Anweisung lautet:
`cpan App::cpanminus`
- ✓ Danach kann das DBI-Modul wie folgt installiert werden:
`cpanm Module::DBI`

Im Folgenden werden noch einige alternative Wege zur Installation von DBI vorgestellt.

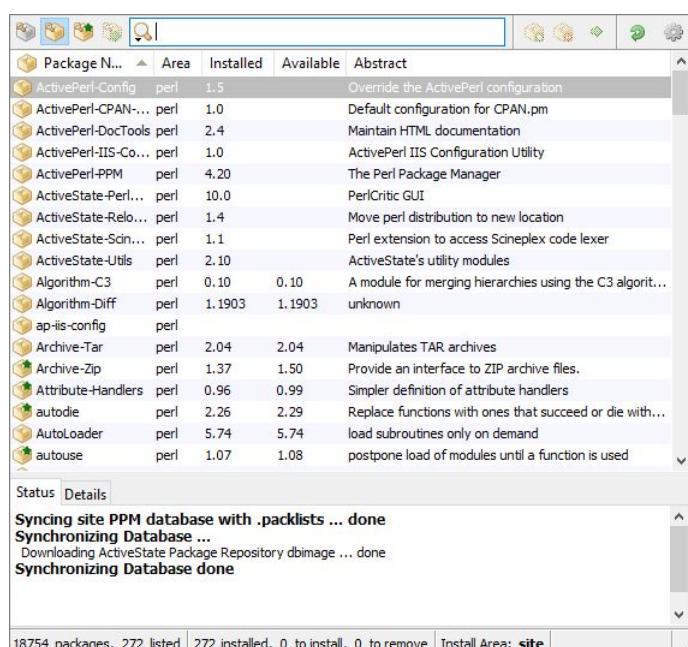
Installation unter Windows mit ActiveState-Perl

Im Buch wird unter anderem die Perl-Distribution von ActiveState verwendet. ActiveState-Perl wird mit dem Programm `ppm` (Programmer's Package Manager) installiert, das Ihnen hilft, fehlende Perl-Module direkt aus dem Internet zu installieren.



Wenn Sie eine Firewall oder einen Proxy-Server verwenden, müssen Sie unter Umständen einige spezielle Umgebungsvariablen einrichten. Mit der Umgebungsvariable `HTTP_proxy` legen Sie die Adresse des Proxy-Servers oder der Firewall in der Form `http://Adresse:Port` fest. Mit den Variablen `HTTP_user` und `HTTP_pass` können Sie zusätzlich einen Benutzernamen und ein Passwort eintragen.

- ▶ Öffnen Sie eine Konsole.
- ▶ Wechseln Sie zum Verzeichnis, in das Sie Perl installiert haben, z. B. `c:\perl`. Wählen Sie dort das Unterverzeichnis `bin`.
- ▶ Geben Sie den Befehl `ppm` ein, um den Package Manager zu starten. Bei neueren Versionen von Perl wird ein bequemes grafisches Interface geöffnet, während bei einigen Distributionen von Perl (insbesondere, aber nicht ausschließlich, bei älteren Versionen) in der Konsole eine Eingabeaufforderung `PPM>` angezeigt wird.



Ein grafisches Interface für ppm

Im grafischen Interface können Sie oben im Suchfeld nach einem Modul suchen und es ggf. installieren.

- ▶ Geben Sie in der Konsole alternativ den folgenden Befehl ein, um das DBI-Modul zu installieren:
`install DBI`
Diese Angabe können Sie aber auch schon beim Aufruf von ppm direkt anhängen:
`ppm install DBI`
- ▶ Bestätigen Sie die Nachfrage, ob Sie das Modul installieren wollen.
Die benötigten Dateien werden automatisch geladen und installiert.
- ▶ Nun müssen Sie wahrscheinlich noch die benötigten Datenbanktreiber laden. Um den MySQL-Treiber zu installieren, geben Sie bei Bedarf folgenden Befehl ein:
`install DBD-Mysql`
- ▶ Bestätigen Sie die Installation.
- ▶ Beenden Sie den Package Manager. Geben Sie dazu auf der Konsole `exit` ein.

Durch die Eingabe des Befehls `search DBD` können Sie in der Konsole nach allen verfügbaren Datenbanktreibern suchen. Mit dem `install`-Befehl können Sie diese dann gezielt installieren.



Installation unter Unix

Unter der URL <http://search.cpan.org/search?dist=DBI> finden Sie das DBI-Modul zum Download. Auch hier können Sie zuerst den Standardweg für CPAN-Module versuchen. Alternativ können Sie wie folgt vorgehen.

- ▶ Melden Sie sich als der Benutzer im System an, unter dem Sie Perl installiert haben.
- ▶ Downloaden Sie die Installationsdatei in ein neues Verzeichnis.
- ▶ Extrahieren Sie die Datei mit dem Befehl
`tar xvfz DBI-[Versionsnummer].tar.gz`
- ▶ Wechseln Sie in das erzeugte Unterverzeichnis `DBI-[Versionsnummer]`.
- ▶ Führen Sie folgende Befehle aus, um die Installation durchzuführen. Schließen Sie jeden Befehl mit `[←]` ab.
`perl Makefile.PL`
`make`
`make test`
`make install`

Bevor Sie das Modul nutzen können, müssen Sie zusätzlich die benötigten Datenbanktreiber installieren. Unter der URL <http://search.cpan.org/search?mode=module&query=DBD> finden Sie eine Liste aller verfügbaren Treiber.

Unter der URL <http://search.cpan.org/search?dist=DBD-mysql> finden Sie den Treiber für den MySQL-Datenbanksystem.

- ▶ Downloaden Sie die Installationsdatei in ein neues Verzeichnis, und extrahieren Sie die Datei mit dem Befehl
`tar xvfz DBD_mysql_[Versionnummer].tar.gz`
- ▶ Wechseln Sie in das beim Extrahieren neu erstellte Verzeichnis, und führen Sie folgende Befehle aus. Schließen Sie jeden Befehl mit `[←]` ab.
`perl Makefile.PL`
`make`
`make test`
`make install`

15.4 Datenbanken und Tabellen erstellen

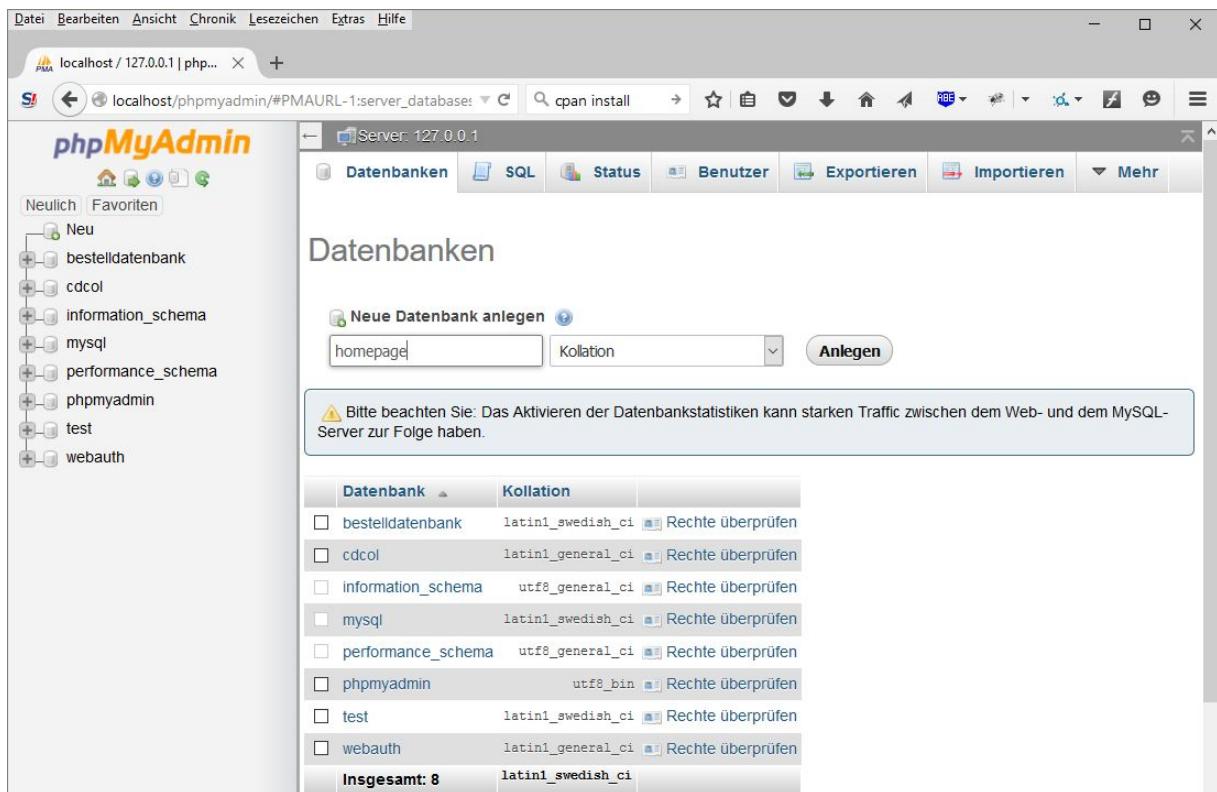
Bevor Sie Daten in einem Datenbanksystem erstellen können, benötigen Sie mindestens eine Datenbank und eine Tabelle. In Tabellen erfolgt die eigentliche Speicherung der Datensätze. Mehrere thematisch zusammengehörende Tabellen fassen Sie in einer Datenbank zusammen.

Datenbanken und Tabellen werden mit SQL-Anweisungen erstellt. Beispielsweise können Sie dies im MySQL-Monitor durchführen.

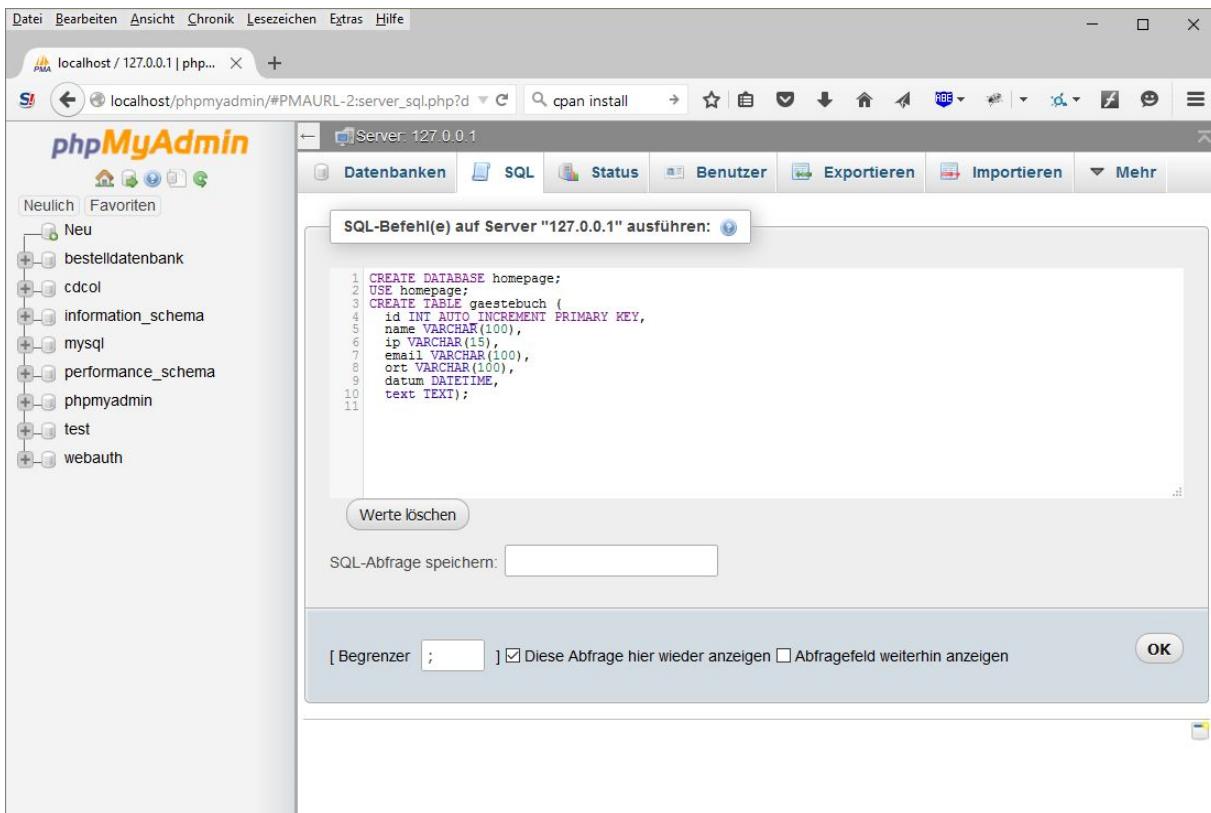
SQL-Anweisung	Erklärung
CREATE DATABASE name;	Datenbank erstellen
CREATE TABLE name (datenfeld1 datentyp, datenfeld2 datentyp, ...);	Tabelle erstellen, in runden Klammern werden die verschiedenen Datenfelder definiert
USE datenbankname;	Aktuelle Datenbank wechseln

 Für alle SQL-Anweisungen benötigen Sie die Ausführungsrechte. Wenn Sie beispielsweise den Datenbankserver eines Internetproviders nutzen, ist oft eine Datenbank fest vorgegeben. Sie können daher keine neuen Datenbanken, sondern ausschließlich Tabellen erstellen.

Bequemer ist die Verwendung von phpMyAdmin. Dort können Sie sowohl SQL-Befehle eingeben, aber auch grafisch die Datenbank erzeugen, ohne einen einzigen SQL-Befehl von Hand eingeben zu müssen.



Anlegen einer Datenbank mit phpMyAdmin über ein Webformular



Auch das Ausführen von SQL-Anweisungen geht in phpMyAdmin über ein Webformular

Beispiel

Das Gästebuch einer Webseite soll in der Datenbank homepage eines MySQL-Datenbankservers gespeichert werden. Für jeden Eintrag sollen Name, Datum, IP-Adresse, E-Mail-Adresse, Ort und ein Text gespeichert werden.

```
CREATE DATABASE homepage;
USE homepage;
CREATE TABLE gaestebuch (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    ip VARCHAR(15),
    email VARCHAR(100),
    ort VARCHAR(100),
    datum DATETIME,
    text TEXT);
```

Im Beispiel werden die Datentypen INT (Ganzzahl), VARCHAR (Zeichenkette mit maximaler Länge), DATETIME (Datum mit Uhrzeit) und TEXT (variabler Text) verwendet. Das Datenfeld `id` dient als so genannter Primärschlüssel, der jeden Datensatz eindeutig kennzeichnet.

Mit den Anweisungen `SHOW TABLES;` erhalten Sie im Monitor eine Übersicht der in einer Datenbank enthaltenen Tabellen. Mit `DESCRIBE tabellenname;` können Sie sich die Struktur einer Tabelle anzeigen lassen.

```
mysql> SHOW TABLES;
+-----+
| Tables_in_homepage |
+-----+
| gaestebuch          |
+-----+
1 row in set (0.00 sec)
```

mysql> DESCRIBE gaestebuch;						
Field	Type	Null	Key	Default	Extra	
id	int(11)		PRI	NULL	auto_increment	
name	varchar(100)	YES		NULL		
ip	varchar(15)	YES		NULL		
email	varchar(100)	YES		NULL		
ort	varchar(100)	YES		NULL		
datum	datetime	YES		NULL		
text	text	YES		NULL		

7 rows in set (0.00 sec)

In phpMyAdmin wird das alternativ alles etwas umfangreicher, aussagekräftiger und vor allen Dingen bequemer angezeigt. Insbesondere können Sie dort die Strukturen direkt im Webformular bearbeiten.

#	Name	Typ	Kollation	Attribute	Null	Standard	Extra	Aktion
1	id	int(11)			Nein	kein(e)	AUTO_INCREMENT	Bearbeiten Löschen
2	name	varchar(100)	latin1_swedish_ci		Ja	NULL		Bearbeiten Löschen
3	ip	varchar(15)	latin1_swedish_ci		Ja	NULL		Bearbeiten Löschen
4	email	varchar(100)	latin1_swedish_ci		Ja	NULL		Bearbeiten Löschen
5	ort	varchar(100)	latin1_swedish_ci		Ja	NULL		Bearbeiten Löschen
6	datum	datetime			Ja	NULL		Bearbeiten Löschen
7	text	text	latin1_swedish_ci		Ja	NULL		Bearbeiten Löschen

Alle auswählen markierte: Anzeigen Bearbeiten Löschen Primärschlüssel

Druckansicht Beziehungsübersicht Tabellenstruktur analysieren Information

Spalte(n) einfügen An das Ende der Tabelle An den Anfang der Tabelle Nach **id**

OK + Indizes

Die gleichen Informationen in phpMyAdmin

15.5 Verbindung zum Datenbankserver herstellen

Um das Datenbankmodul DBI nutzen zu können, müssen Sie es am Beginn des Perl-Programms mit der Anweisung `use DBI;` einbinden.

Wie das CGI-Modul stellt auch das Datenbankmodul ein Objekt zur Verfügung, über das Sie Zugriff auf Methoden und Konstanten für den Datenbankzugriff erhalten. Mit der `connect`-Methode nehmen Sie Verbindung zum Datenbankserver auf und erstellen gleichzeitig eine neue Instanz des DBI-Objekts.

Syntax der `connect`-Methode

```
$datenbankhandle=DBI->connect ("Datenquelle", "Benutzer", "Passwort");
```

- ✓ Die `connect`-Methode wird mithilfe des Objektnamens `DBI` und dem Operator `->` aufgerufen.
- ✓ Als erstes Argument wird die Bezeichnung der Datenquelle angegeben. Diese enthält alle Angaben zum zu verwendenden Treiber und dem Datenbankserver.

- ✓ Als zweites und drittes Argument werden der Benutzername und das zugehörige Passwort für den Datenbank-Benutzer angegeben.
- ✓ Die Methode liefert einen Verweis auf eine geöffnete Datenbankverbindung zurück, der in einer skalaren Variable als so genanntes Datenbankhandle gespeichert werden kann. Über diesen Verweis erfolgen alle weiteren Datenbankzugriffe.

Datenquelle angeben

Als Datenquelle wird eine Zeichenkette verwendet, die den Datenbanktreiber, den Hostnamen des Datenbankservers und eine Portadresse des Servers angibt. Ohne diese Angaben kann kein Zugriff auf die Datenbank erfolgen. Folgende Möglichkeiten für die Angabe der Datenquelle gibt es:

```
dbi:Treibername:Datenbankname
dbi:Treibername:Datenbankname@Hostname
dbi:Treibername:Datenbankname@Hostname:Port
dbi:Treibername:database=Datenbankname;host=Hostname;port=Port
```

- ✓ Zuerst erfolgt stets die Angabe des Bezeichners `dbi`. Mit einem Doppelpunkt `:` wird der Name des Datenbanktreibers angeschlossen.
- ✓ Nach einem Doppelpunkt wird der Name der Datenbank angegeben.
- ✓ Zusätzlich ist in verschiedenen Schreibweisen die Angabe eines Hostnamens und einer zusätzlichen Portadresse möglich. Diese Angaben sind optional.

Wenn der Datenbankserver auf dem gleichen Rechner wie das Perl-Programm ausgeführt wird, kann die Angabe des Hostnamens entfallen. In allen anderen Fällen können Sie den Hostnamen als IP-Adresse oder Domainnamen aufführen. Im Folgenden sehen Sie einige Beispiele für die Angabe einer Datenquelle bei einem MySQL-Server:

Datenquelle	Erklärung
<code>dbi:mysql:homepage</code>	Verbindung zum lokalen MySQL-Server und der Datenbank <code>homepage</code>
<code>dbi:mysql:homepage@192.168.0.100</code>	Verbindung zum MySQL-Server mit der angegebenen IP-Adresse und der Datenbank <code>homepage</code>
<code>dbi:pg:info@rjs.de</code>	Verbindung zu einem PostgreSQL-Server unter der Adresse <code>rjs.de</code> und Zugriff auf die Datenbank <code>info</code>

Verfügbare Treiber und Datenquellen abfragen

Das DBI-Modul stellt Ihnen zwei Methoden zur Verfügung, mit denen Sie feststellen können, welche Datenbanktreiber installiert sind und welche Datenquellen ein Treiber auf Ihrem System finden konnte.

<code>@treiber=DBI->available_drivers;</code>	Die Methode ermittelt ein Array aller verfügbaren Datenbanktreiber des DBI-Moduls
<code>@sources=DBI->data_sources("Treiber");</code>	Die Methode versucht, die Namen aller Datenbanken zu ermitteln, auf die über einen angegebenen Treiber zugegriffen werden kann. Als Resultat erhalten Sie ein Array mit Datenquellen.

Die `data_sources`-Methode wird nicht von allen Treibern unterstützt. Sie erhalten in diesem Fall das Ergebnis `undef`.



Fehler abfangen



Im Fall eines Verbindungsfehlers liefert die Methode `false`. Dies können Sie verwenden, um Fehler abzufangen. Fehlermeldungen des DBI-Objekts rufen Sie mit der Konstante `$DBI::errstr` ab:

```
$datenbankhandle=DBI->connect ("Datenquelle", "Benutzer", "Passwort") ||  
die("Keine Verbindung zum Datenbankserver möglich! Fehler:  
$DBI::errstr\n");
```

Bei einer aktiven Datenbankverbindung können Sie die Fehlermeldung auch mit dem Ausdruck `$datenbankhandle->errstr` abrufen.

Syntax der `disconnect`-Methode

```
$datenbankhandle->disconnect;
```

- ✓ Um eine Verbindung zu einem Datenbankserver zu beenden, wird die `disconnect`-Methode verwendet.
- ✓ Dabei müssen Sie den Verweis auf eine geöffnete Datenbankverbindung und den Operator `->` verwenden.

Die Datenbankverbindung wird am Ende eines Perl-Programms auch automatisch geschlossen. Sie sollten eine Verbindung jedoch stets selbst beenden, um genutzte Ressourcen sicher freizugeben.

Beispiel: *DatenbankVerbindung.pl*

Das Perl-Programm prüft, ob der MySQL-Datenbanktreiber installiert ist und stellt eine Verbindung zum lokalen Datenbankserver und der Datenbank `homepage` her.

```
#!/usr/bin/perl  
use strict;  
  
① use DBI;  
② my $datenquelle="dbi:mysql:homepage";  
my $benutzer="root";  
my $passwort="";  
my $mysql;  
③ my @treiberliste=DBI->available_drivers();  
foreach my $treiber (@treiberliste) {  
④   $mysql=1 if ($treiber eq "mysql");  
}  
⑤ if (!$mysql) {  
  die("MySQL-Datenbanktreiber nicht installiert!\n");  
}  
  
⑥ my $dbh=DBI->connect($datenquelle,$benutzer,$passwort) ||  
  die("Keine Verbindung zum Datenbankserver möglich! Fehler:  
$DBI::errstr\n");  
print "Verbindung aufgebaut...\n";  
⑦ $dbh->disconnect;  
print "Verbindung beendet...\n";
```

- ① Mit der `use`-Anweisung wird das DBI-Modul in das Perl-Programm eingebunden.
- ② Als Datenquelle wird die MySQL-Datenbank `homepage` auf dem lokalen Computer festgelegt. Danach werden der Benutzername und das Passwort für den Zugriff auf die Datenbank festgelegt. Standardmäßig ist dies der Benutzer `root` ohne eine Passwortangabe.

- ③ Mit der `available_drivers`-Methode wird eine Liste der vom System unterstützten Datenbanktreiber ermittelt.
- ④ Die Variable `$mysql` wird nur dann gesetzt, wenn ein Treiber mit dem Namen `mysql` gefunden wurde.
- ⑤ Falls kein MySQL-Treiber existiert, wird das Programm mit einer Fehlermeldung beendet.
- ⑥ Über die `connect`-Methode wird mithilfe der vorher definierten Datenquelle und den Anmelddaten eine Verbindung zum MySQL-Server aufgebaut. Im Erfolgsfall wird eine Instanz des DBI-Objekts in der Variablen `$dbh` gespeichert. Andernfalls wird das Programm mit einer Fehlermeldung beendet.
- ⑦ Am Ende des Programms erfolgt das Trennen der Datenbankverbindung mithilfe der `disconnect`-Methode.

```
F:\PerlProgramme\Kapitel_15>perl DatenbankVerbindung.pl
Verbindung aufgebaut...
Verbindung beendet...
F:\PerlProgramme\Kapitel_15>
```

Wenn keine Fehlermeldungen auftauchen, sollte der Kontakt zum Datenbankmanagementsystem und der Datenbank funktionieren.

15.6 Abfragen erstellen

Mithilfe einer **Abfrage** (engl. query) senden Sie SQL-Anweisungen an den Datenbankserver, um Datensätze in eine Tabelle einzufügen, Datensätze zu löschen, zu ändern oder Daten aus einer Tabelle zu ermitteln. Dafür werden die folgenden SQL-Anweisungen verwendet:

SQL-Anweisung	Erklärung
<code>SELECT Feld1, ... FROM Tabelle WHERE Bedingung</code>	Ermitteln von Daten aus einer oder mehreren Tabellen in Abhängigkeit von einer Bedingung
<code>INSERT INTO Tabelle (Feld1, ...)</code> <code>VALUES (Wert1, ...)</code>	Einfügen von Datensätzen in eine Tabelle
<code>UPDATE Tabelle SET Feld1=Wert1, ...</code> <code>WHERE Bedingung</code>	Ändern von Datensätzen einer Tabelle in Abhängigkeit von einer Bedingung
<code>DELETE FROM Tabelle WHERE Bedingung</code>	Löschen von Datensätzen in Abhängigkeit von einer Bedingung

Auch wenn der Begriff „Abfrage“ etwas anderes vermuten lässt, wird er nicht nur für das Ermitteln von Daten aus einer Datenbank, sondern synonym auch für das Einfügen und Löschen von Datensätzen verwendet.



Mit dem DBI-Modul werden Abfragen in der Regel in drei Schritten ausgeführt, für die jeweils entsprechende Methoden existieren:

Arbeitsschritt	Erklärung	Methode
Vorbereiten	Die Abfrage wird an das DBI-Objekt übergeben und überprüft. Als Ergebnis erhalten Sie ein Abfragehandle, über das die weiteren Schritte durchgeführt werden.	<code>prepare()</code>
Ausführen	Übergebene Argumente werden in die Abfrage eingefügt. Die SQL-Anweisung wird an den Datenbankserver gesendet und ausgeführt.	<code>execute()</code> <code>do()</code>
Abrufen	Falls die Abfrage Datensätze zurückliefert (<code>SELECT</code>), werden diese in diesem Schritt vom Datenbankserver abgerufen.	<code>fetchrow_array()</code>

Syntax der `prepare`-Methode

```
$abfragehandle = $datenbankhandle->prepare ("SQL-Anweisung") ;
```

- ✓ Mit der `prepare`-Methode übergeben Sie eine SQL-Anweisung an das DBI-Objekt. Für den Aufruf wird ein Verweis auf eine geöffnete Datenbankverbindung benötigt.
- ✓ Als Argument der Methode wird eine SQL-Anweisung übergeben. Im Gegensatz zum direkten Ausführen einer SQL-Anweisung wird dabei kein Semikolon ; angegeben.
- ✓ Die Methode liefert als Ergebnis ein Abfragehandle, das im weiteren Verlauf der Abfrage verwendet wird.
- ✓ Im Fall eines Fehlers liefert die Methode den Wert `false`.

Behandlung von Sonderzeichen

Der Datenbankserver erwartet alle Zeichenketten in Anführungszeichen „“ oder Hochkommata „“. Gleichzeitig müssen alle Sonderzeichen durch einen vorangestellten Backslash \ gekennzeichnet werden. Das Ignorieren dieser Festlegungen ist eine häufige Fehlerursache beim Ausführen von SQL-Anweisungen.



Das DBI-Modul stellt die Methode `quote()` zur Verfügung, die alle Sonderzeichen automatisch durch einen Backslash markiert und die Zeichenkette in Hochkommata einschließt. Vor allem wenn Sie Benutzereingaben an die Datenbank übergeben, sollten Sie diese Funktion verwenden.

```
$zeichenkette = $datenbankhandle->quote ("Zeichenkette") ;
```

Platzhalter verwenden

Beim Definieren einer SQL-Anweisung können Sie den Platzhalter ? verwenden, um Werte zu kennzeichnen, die erst beim Ausführen der Anweisung mit der `execute`-Methode übergeben werden. Dies ist vor allem dann sinnvoll, wenn Sie eine SQL-Anweisung mehrfach mit verschiedenen Werten ausführen möchten. In diesem Fall ist die Vorbereitungsphase nur einmal notwendig.

Beispiel

Im Folgenden finden Sie einige Beispiel für die Vorbereitung von SQL-Anweisungen:

```
$dbh->prepare("SELECT id FROM gaestebuch WHERE name=?");
$dbh->prepare("SELECT id,name,? FROM ?");
$sql="SELECT * FROM gaestebuch ORDER BY datum DESC";
$dbh->prepare($sql);
$dbh->prepare("DELETE FROM gaestebuch WHERE name LIKE ?");
```

Syntax der `execute`-Methode

```
$anzahl = $abfragehandle->execute(Wert1, ...) ;
```

- ✓ Für den Aufruf der `execute`-Methode wird ein gültiges Abfragehandle benötigt, das von einer vorher ausgeführten `prepare`-Methode stammt.
- ✓ Die als Argumente übergebenen Werte werden in der angegebenen Reihenfolge für die in der SQL-Anweisung verwendeten Platzhalter ? eingesetzt. Die Werte werden automatisch in Hochkommata gesetzt.
- ✓ Im Fall eines Fehlers liefert die Methode den Wert `false`.
- ✓ Bei SQL-Anfragen, die Daten verändern, liefert die Methode die Anzahl der veränderten Datensätze.

Abfragen direkt ausführen

Falls Sie keine Platzhalter benötigen, da die SQL-Anweisung nur einmal ausgeführt werden soll, können Sie statt der `prepare`- und `execute`-Methode einen einzigen Aufruf der Methode `do()` verwenden.

Diese Methode ist vor allem für SQL-Anweisungen sinnvoll, die Daten verändern (`INSERT`, `DELETE`, `UPDATE`), aber keine Datensätze zurückliefern.



```
$anzahl = $datenbankhandle->do ("SQL-Anweisung");
```

- ✓ Mit der `do`-Methode übergeben Sie eine SQL-Anweisung an den Datenbankserver und führen sie sofort aus. Für den Aufruf wird ein Verweis auf eine geöffnete Datenbankverbindung benötigt.
- ✓ Als Argument der Methode wird eine SQL-Anweisung übergeben. Im Gegensatz zum direkten Ausführen einer SQL-Anweisung wird dabei kein Semikolon `:` angegeben.
- ✓ Die Methode liefert die Anzahl der durch die Anweisung veränderten Datensätze zurück. Im Fehlerfall wird `false` als Ergebnis zurückgegeben.

Beispiel: `gaestebuch_db.cgi`, `gaestebuch.html`

Die HTML-Seite stellt ein Formular zum Eintrag in das Gästebuch einer Webseite zur Verfügung. Die Daten werden danach in einer Datenbank gespeichert.

```
#!/F:/xampp/perl/bin/perl.exe"
use strict;
① use DBI;
use CGI;
use CGI::Carp qw( fatalToBrowser );
② my $datenquelle="dbi:mysql:homepage";
my $benutzer="root";
my $passwort="";
my $mysql;
my $tabelle="gaestebuch";

my $cgi=new CGI;
③ my $dbh=DBI->connect($datenquelle,$benutzer,$passwort) ||
die("Keine Verbindung zum Datenbankserver möglich! Fehler:
$DBI::errstr\n");

if ($cgi->param) {
④ my $text=$cgi->param('text');
$text =~ s/\n/<br>/g;
$text =~ s/\r//g;
⑤ my $sql="INSERT INTO $tabelle (name,ip,email,ort,datum) VALUES (" ;
⑥ $sql.=$dbh->quote($cgi->param('name')).",";
$sql.=$dbh->quote($cgi->remote_addr).",";
$sql.=$dbh->quote($cgi->param('email')).",";
$sql.=$dbh->quote($cgi->param('ort')).",";
$sql.=$dbh->quote($text).",";
⑦ $sql.="now())";
⑧ $dbh->do($sql) ||
die("Fehler beim Speichern des Datensatzes!\n".$dbh->errstr);
```

```

⑨ print $cgi->header;
print $cgi->start_html("Gästebuch");
print $cgi->p(
    "Ihr Eintrag ins Gästebuch wurde gespeichert.");
print $cgi->end_html;
}

⑩ $dbh->disconnect;

```

- ① Mit der `use`-Anweisung werden das CGI- und das DBI-Modul eingebunden.
- ② Zuerst wird die Datenquelle für den Zugriff auf die Datenbank in einer Variablen festgelegt. Hier wird über den MySQL-Treiber auf die Datenbank `homepage` des lokalen Datenbankservers zugegriffen. Danach werden Benutzername, zugehöriges Passwort und Tabellenname in Variablen gespeichert. Diese Daten sind den jeweiligen Gegebenheiten anzupassen.
- ③ Mit den vorher definierten Zugangsdaten wird an dieser Stelle versucht, eine Verbindung zum Datenbankserver aufzubauen. War der Versuch erfolgreich, wird die Verbindungs-kennung in der Variablen `$dbh` gesichert. Andernfalls wird das Programm mit einer Fehlermeldung beendet.

The screenshot shows a web browser window titled "Mein Gästebuch". The URL in the address bar is "localhost/Kapitel_15/gaes". The page content is a form titled "Eintrag ins Gästebuch". It contains four input fields: "Ihr Name:" with the value "Ralph", "Ihre E-Mail-Adresse:" with the value "info@rjs.de", "Ihr Wohnort:" with the value "Bodenheim", and "Ihre Nachricht:" with the value "Ich grüße die Macher der Webseite". Below the form is a "Speichern" button and a link "Einträge lesen".

Das Gästebuchformular

- ④ Im Text des Gästebucheintrags sollen alle Zeilenumbrüche durch das entsprechende HTML-Tag `
` ersetzt werden. Dazu wird der Wert des Formularelements zuerst in einer Variablen gespeichert. Danach werden mit jeweils einem regulären Ausdruck alle Zeilenumbrüche `\n` in das HTML-Tag umgewandelt und die unter Windows dazugehörigen Wagenrückläufe `\r` gelöscht.
- ⑤ Hier beginnt das Definieren der SQL-Anweisung für das Einfügen des neuen Datensatzes (`INSERT`). Zur besseren Übersicht erfolgt dies in einer eigenen Variablen.
- ⑥ Nacheinander werden die Werte der Formularelemente in die Abfrage eingefügt. Mithilfe der Methode `quote()` werden die möglicherweise eingegebenen Sonderzeichen mit einem Backslash gekennzeichnet und der gesamte Wert in Hochkommata eingeschlossen.
- ⑦ Zum Einfügen des aktuellen Datums wird innerhalb der SQL-Anweisung die MySQL-Funktion `now()` verwendet. Das Datum wird daher nicht von Perl, sondern direkt vom Datenbankserver eingefügt.
- ⑧ Mit der `do`-Methode des DBI-Objekts wird die Abfrage zum Datenbankserver gesendet und ausgeführt. Im Falle eines Fehlers wird das Programm mit einem entsprechenden Hinweis beendet.
- ⑨ Mithilfe verschiedener Methoden des CGI-Objekts wird eine HTML-Seite als Bestätigung des Gästebuch-eintrags erstellt.
- ⑩ Am Ende des Programms wird die Verbindung zum Datenbankserver wieder beendet.



Mithilfe einer SELECT-Abfrage können Sie im MySQL-Monitor überprüfen, ob der Gästebucheintrag korrekt gespeichert wurde. Noch bequemer ist es mit phpMyAdmin möglich.

The screenshot shows the phpMyAdmin interface for a MySQL database named 'homepage'. The left sidebar lists databases like 'bestelldatenbank', 'cdcol', and 'homepage'. Under 'homepage', there is a 'Neu' folder containing a 'gaestebuch' table. The main panel displays the contents of the 'gaestebuch' table with one row:

	id	name	ip	email	ort	datum	text
	4	Ralph	::1	ralph@rjs.de	Bodenheim	2015-09-11 12:36:50	Ich grüße die Macher der Webseite

Below the table, there are buttons for 'Bearbeiten' (Edit), 'Löschen' (Delete), and 'Exportieren' (Export).

Die Eingaben wurden in die Datenbank eingefügt

15.7 Abfrageergebnis ermitteln

Beim Ausführen von Abfragen, die Daten verändern (INSERT, UPDATE, DELETE), erhalten Sie als Ergebnis der execute- oder do-Methode nur die Anzahl der modifizierten Datensätze. Das Resultat einer SELECT-Abfrage ist jedoch eine Menge von Datensätzen, die Sie mit verschiedenen Methoden abrufen können.

Die folgende Tabelle gibt die Methoden an, die für das Ermitteln des Ergebnisses einer SELECT-Abfrage von Bedeutung sind:

Methode	Erklärung
\$anzahl=\$abfragehandle->rows;	Anzahl der selektierten Datensätze
@daten=\$abfragehandle->fetchrow_array;	Abrufen eines Datensatzes und Speichern der Werte in einem Array
\$abfragehandle->finish;	Beenden der Abfrage, wenn keine weiteren Datensätze benötigt werden

Um auf die Datensätze einer SELECT-Abfrage zugreifen zu können, müssen Sie die Abfrage stets mit der prepare-Methode vorbereiten und danach mit execute() ausführen. Nur so erhalten Sie ein Abfragehandle, das Sie für den Zugriff auf das Abfrageergebnis benötigen:

```
$datenbankhandle = DBI->connect(...);
$abfragehandle = $datenbankhandle->prepare("SELECT ...");
$abfragehandle->execute(...);
print $abfragehandle->rows;
```

Syntax der `rows`-Methode

```
$anzahl = $abfragehandle->rows;
```

- ✓ Für den Aufruf der `rows`-Methode wird ein gültiges Abfragehandle benötigt.
- ✓ Die Methode liefert die Anzahl der Datensätze, die eine SELECT-Abfrage als Ergebnismenge zurückgeliert hat.
- ✓ Hat die Abfrage keine Datensätze zum Ergebnis oder ist ein Fehler aufgetreten, liefert die Methode den Wert -1.



Die Methode liefert den Wert 0, wenn die SELECT-Abfrage keine Datensätze zum Ergebnis hat. Beachten Sie, dass Perl diesen Wert bereits als `false` betrachtet. Konnte die Methode nicht ermitteln, wie viele Datensätze zurückgegeben wurden oder ist ein Fehler aufgetreten, liefert die Methode den Wert -1.

Syntax der `fetchrow_array`-Methode

```
@daten = $abfragehandle->fetchrow_array;
```

- ✓ Für den Aufruf der `fetchrow_array`-Methode wird ein gültiges Abfragehandle benötigt.
- ✓ Die Methode liefert bei jedem Aufruf einen neuen Datensatz. Nach dem letzten Datensatz wird der Wert `false` zurückgegeben.
- ✓ Das Ergebnis der Methode ist ein Array, in dem jedes Element einem Datenfeld des Datensatzes entspricht.



Um alle Datensätze des Abfrageergebnisses zu durchlaufen, können Sie eine `while`-Schleife verwenden.

Syntax der `finish`-Methode

```
$abfragehandle->finish;
```

- ✓ Mit der Methode `finish()` wird eine Abfrage beendet und der verwendete Speicherplatz freigegeben.
- ✓ Nach dem Aufruf dieser Methode können keine Datensätze mehr abgerufen werden.



Der Aufruf der `finish`-Methode ist nur dann notwendig, wenn Sie nicht alle Datensätze einer SELECT-Abfrage auslesen. In diesem Fall können Sie die Abfrage vorzeitig beenden und Speicherplatz sparen.

Beispiel: `gaestebuch_db2.cgi`, `gaestebuch2.html`

Das CGI-Programm `gaestebuch.cgi` aus dem vorherigen Beispiel soll um eine Anzeige der Gästebucheinträge erweitert werden. Dabei sollen die neuesten Einträge zuerst erscheinen und, mit einer Möglichkeit zum Blättern, pro Seite nur 5 Einträge angezeigt werden.

```

#!/F:/ActivePerl/bin/perl.exe"
use strict;
use DBI;
use CGI;
use CGI::Carp qw( fatalstoBrowser );
my $datenquelle="dbi:mysql:homepage";
my $benutzer="root";
my $passwort="";
my $mysql;
my $tabelle="gaestebuch";
my @daten;
my $cgi=new CGI;
my $anz_pro_seite=5;
my $start_datensatz=1;
my $abfrage;
my $cgi=new CGI;
my $dbh=DBI->connect ($datenquelle,$benutzer,$passwort) ||
    die("Keine Verbindung zum Datenbankserver möglich! Fehler:
$DBI::errstr\n");

print $cgi->header;
print $cgi->start_html ("Gästebuch");

if ($cgi->param('text')) {
    my $text=$cgi->param('text');
    $text =~ s/\n/<br>/g;
    $text =~ s/\r//g;
    my $sql="INSERT INTO $tabelle (name,ip,email,ort,text,datum) VALUES (";
    $sql.=$dbh->quote($cgi->param('name')).",";
    $sql.=$dbh->quote($cgi->remote_addr).",";
    $sql.=$dbh->quote($cgi->param('email')).",";
    $sql.=$dbh->quote($cgi->param('ort')).",";
    $sql.=$dbh->quote($text).",";
    $sql.="now())";
    $dbh->do($sql) ||
        die("Fehler beim Speichern des Datensatzes!\n".$dbh->errstr);
}

① if (!$cgi->param('offset')) {
    $cgi->param(-name=>'offset',-value=>0);
}
② &Anzeige($cgi->param('offset'));
print $cgi->end_html;
$dbh->disconnect;

sub Anzeige {
    my $start_datensatz=shift;
}
③ $abfrage=$dbh->prepare(
    "SELECT name,email,ort,date_format(datum,'%d.%m.%Y'),
    text FROM $tabelle ORDER BY datum DESC
    LIMIT $start_datensatz,$anz_pro_seite");

```

```

④ $abfrage->execute ||  

    die("Fehler beim Ausführen der Abfrage!\n".$dbh->errstr\n");  

⑤ while(@daten=$abfrage->fetchrow_array) {  

    print $cgi->start_p({style=>'font-family:sans-serif'});  

⑥    print $cgi->b($cgi->a({href=>"mailto:$daten[1]"},$daten[0]).  

        " aus $daten[2] schrieb am $daten[3] folgendes:").$cgi->br;  

    print $daten[4];  

    F $cgi->end_p.$cgi->hr({size=>'1'});  

}  

print $cgi->start_p({style=>'font-family:sans-serif'});  

⑦ if ($start_datensatz>=$anz_pro_seite) {  

    print $cgi->a({href=>$cgi->script_name."?offset=".  

        ($start_datensatz-$anz_pro_seite)}, "Vorherige Seite");  

    print " | ";  

}  

⑧ if ($start_datensatz<$abfrage->rows) {  

    print $cgi->a({href=>$cgi->script_name."?offset=".  

        ($start_datensatz+$anz_pro_seite)}, "Nächste Seite");  

    print " | ";  

}  

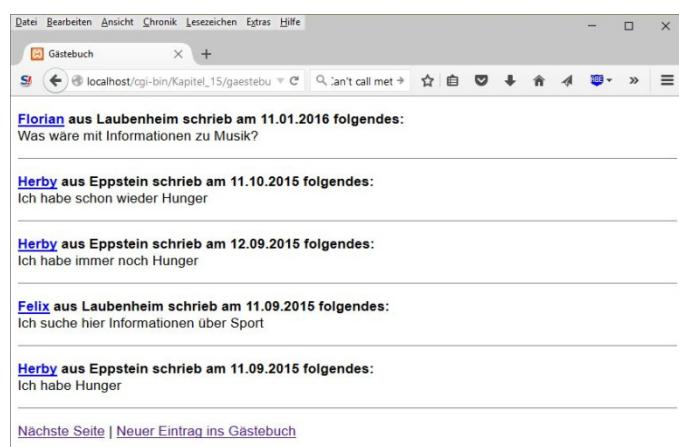
print $cgi->a({href=>"/Kapitel_15/gaestebuch2.html"},  

    "Neuer Eintrag ins Gästebuch");  

print $cgi->end_p;
}

```

- ① Um das Blättern in den Gästebucheinträgen zu ermöglichen, wird an das CGI-Programm der Parameter `offset` übermittelt. Falls er nicht gesetzt ist, wird er mithilfe der Methode `param()` auf den Wert 0 festgelegt.
- ② Hier erfolgt der Aufruf des Unterprogramms `Anzeige()` mit der Nummer des Startdatensatzes als Argument.
- ③ Mit der `prepare`-Methode wird die `SELECT`-Anweisung an das DBI-Objekt übermittelt und vorbereitet. Mithilfe des SQL-Befehls `LIMIT` wird die Anzeige einer bestimmten Anzahl Datensätze ab einer festgelegten Position erreicht. Die `date_format`-Funktion ist eine MySQL-Funktion und wandelt das Datum bereits auf dem Datenbankserver in das gewünschte Format.
- ④ Die `execute`-Methode sendet die Abfrage an den Datenbankserver und führt sie aus.
- ⑤ In einer `while`-Schleife werden die aus der Datenbank gelesenen Gästebucheinträge durch die `fetchrow_array`-Methode abgerufen und im Array `@daten` gespeichert.
- ⑥ An dieser Stelle erfolgt die Ausgabe der Gästebucheinträge. Die Indexwerte des Arrays entsprechen dabei der Reihenfolge der Datenfelder in der `SELECT`-Abfrage.
- ⑦ Wenn der aktuelle Startdatensatz größer ist als die Anzahl der Gästebucheinträge pro Seite, wird ein Hyperlink zum Blättern auf die vorherige Seite erzeugt. Der Link verweist auf das aktuelle CGI-Programm und verwendet den Parameter `offset`.
- ⑧ Der Hyperlink zum Blättern auf die nächste Seite wird nur angezeigt, wenn die Nummer des Startdatensatzes kleiner ist als die Gesamtzahl der Gästebucheinträge.



Die erste Seite des Gästebuchs

15.8 Weitere Methoden des DBI-Objekts

Das DBI-Objekt stellt eine Vielzahl weiterer Methoden und Konstanten für den Zugriff auf Datenbankserver zur Verfügung:

Methode/Konstante	Erklärung
<code>@daten = \$dbh->selectrow_array("SQL-Abfrage");</code>	Kombination der Methoden <code>prepare</code> , <code>execute</code> und <code>fetchrow_array</code>
<code>\$abfrage = \$dbh->prepare_cached("SQL-Abfrage");</code>	Vorbereiten einer Abfrage und Zwischenspeichern für eine weitere Verwendung
<code>\$dbh->commit;</code>	Änderungen an einer Datenbank, die Transaktionen unterstützt durchführen
<code>\$dbh->rollback;</code>	Änderungen an einer Datenbank, die Transaktionen unterstützt zurücknehmen
<code>\$abfrage->bind_param(Platzhalternummer, Wert);</code>	Wert für einen bestimmten Platzhalter in der Abfrage festlegen

Eine vollständige Liste aller Methoden des DBI-Moduls sowie Beispiele zur Anwendung finden Sie in der Online-Dokumentation des Moduls unter <http://www.perldoc.com/cpan/DBI.html>.



15.9 Schnellübersichten

Was bedeutet ...	
Datenbankserver	Serverprogramm, das umfangreiche Datenbestände in Datenbanken und Tabellen organisiert
Datenbank	Container für mehrere, meist thematisch zusammengehörige Tabellen
Tabelle	Ort, um Datensätze in einer festgelegten Struktur aus Datenfeldern zu speichern
Datensatz	Eine Zeile einer Tabelle
Datenfeld	Eine Spalte einer Tabelle
DBI	Engl. DataBase Interface, Perl-Modul für universellen Datenbankzugriff
DBD	Engl. DataBase Driver, Treiber für den Zugriff auf ein bestimmtes Datenbanksystem mit dem DBI-Modul
SQL	Engl. Structured Query Language, Sprache zum Erstellen von Anfragen an einen Datenbankserver
Datenquelle	Bezeichnet im DBI eine Ausdruck, der den Datenbanktreiber, die Datenbank und den Hostnamen des Servers festlegt

Sie möchten ...	
eine Datenquelle festlegen	dbi :Treibername :Datenbankname dbi :Treibername :Datenbankname@Hostname
zu einem Datenbankserver verbinden	\$dbh=DBI->connect (Datenquelle,Benutzer,Passwort);
eine Serververbindung schließen	\$dbh->disconnect;
eine Abfrage vorbereiten	\$abfrage=\$dbh->prepare ("SQL-Anweisung");
eine Abfrage ausführen	\$abfrage->execute; \$dbh->do ("SQL-Anweisung");
die Anzahl der Datensätze einer SELECT-Abfrage ermitteln	\$abfrage->rows;
einen Datensatz einer SELECT-Abfrage ermitteln	\$abfrage->fetchrow_array;

15.10 Übungen

Grundlagen zu MySQL und SQL

Übungsdatei: --

Ergebnisdatei: --

1. Erläutern Sie die Schritte, die bei einer Verbindung zu einem MySQL-Datenbankserver über das DBI-Modul durchgeführt werden müssen.
2. Geben Sie an, auf welche Datenbank und welchen Server bei den folgenden Datenquellen zugriffen werden soll:

```
dbi:mysql:kunden
dbi:mysql:shop@210.124.0.1
dbi:mysql:database=shop
dbi:mysql:database=verkauf;host=meinedb.de
```
3. Mit welcher SQL-Anweisung selektieren Sie Datensätze aus einer Tabelle?
4. Erläutern Sie die Schritte beim Erstellen einer SELECT-Abfrage.

SQL anwenden

Übungsdatei: --

Ergebnisdatei: *newsletter.cgi*

1. Es soll ein Perl-Programm zur Verwaltung eines Newsletters erstellt werden. Über ein Formular sollen sich neue Mitglieder an- und abmelden können.
2. Erstellen Sie eine Tabelle `newsletter` mit folgenden Datenfeldern:


```
id      INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
name    VARCHAR(150) NULL,
email   VARCHAR(150) NULL,
datum   DATETIME
```
3. Beim Aufruf des CGI-Programms ohne Parameter soll mithilfe der Methoden des CGI-Moduls ein Formular zur Eingabe eines Namens und einer E-Mail-Adresse angezeigt werden. Gleichzeitig soll der Besucher mit einem Optionsfeld die Wahl zwischen An- und Abmelden haben (vgl. Abbildung unten).
4. Bauen Sie im CGI-Programm zuerst eine Verbindung zum Datenbankserver auf. Verwenden Sie dabei die korrekte Datenquelle und einen gültigen Benutzernamen mit Passwort.
5. Prüfen Sie zuerst, ob beide Formularfelder ausgefüllt wurden. Ist dies nicht der Fall, soll eine Fehlermeldung angezeigt werden. Erstellen Sie zu diesem Zweck ein Unterprogramm, das eine variable Fehlermeldung anzeigen kann und danach das Programm beendet.
6. Ermitteln Sie mit einer `SELECT`-Abfrage, ob die E-Mail-Adresse bereits gespeichert ist. Speichern Sie für spätere Testzwecke die Anzahl der gefundenen Datensätze.
7. Prüfen Sie die eingegebene E-Mail-Adresse mit einem regulären Ausdruck auf äußerliche Gültigkeit (mindestens ein Zeichen vor dem @, danach mindestens ein Zeichen und nach einem Punkt eine Domainangabe). Geben Sie gegebenenfalls mit Ihrem eigenen Unterprogramm eine Fehlermeldung aus.
8. Testen Sie, ob die Option zum Anmelden gewählt wurde und die E-Mail-Adresse noch nicht in der Datenbank gefunden wurde. Speichern Sie in diesem Fall die Daten in der Tabelle `newsletter`. Achten Sie dabei darauf, Sonderzeichen umzuwandeln. Ist die Adresse bereits gespeichert, soll eine entsprechende Meldung ausgegeben werden.
9. Wenn die Option zum Abmelden gewählt wurde und die E-Mail-Adresse in der Datenbank gefunden wurde, löschen Sie den Eintrag mit einer `DELETE`-Anweisung. Wurde die E-Mail-Adresse nicht gefunden, soll auch hier eine Meldung angezeigt werden.

Das Formular der Newsletter-Anmeldung...

... und eine Fehlermeldung

16 E-Mails senden

In diesem Kapitel erfahren Sie

- ✓ wie Sie mit Perl E-Mails versenden
- ✓ wie Sie das Modul Mail::Sendmail installieren und verwenden
- ✓ wie Sie eine E-Mail-Adresse prüfen

Voraussetzungen

- ✓ CGI-, Datenbank-Programmierung
- ✓ Module installieren

16.1 E-Mail senden

E-Mail mit *sendmail* verschicken

Perl verfügt über keine eigenen Funktionen zum Versenden von E-Mails. Sie sind dazu auf externe Programme oder Module angewiesen. Sehr einfach funktioniert der E-Mail-Versand unter Unix-Betriebssystemen. Dort ist standardmäßig das Programm *sendmail* installiert, das nur von Perl die benötigten Daten erhalten muss.

Unter Windows ist das Vorgehen schwieriger. Doch auch hier können Sie mit etwas mehr Aufwand E-Mails aus einem Programm heraus versenden. Der manuelle Weg soll hier aber nicht vertieft werden. Allerdings sind Sie auch unter Windows in der Lage mit *sendmail* zu arbeiten, wenn Sie XAMPP verwenden.

Perl-Module für den E-Mail-Versand

Neben der Möglichkeit, auf externe Programme zuzugreifen, können Sie auch Perl-Module verwenden, die E-Mails per SMTP (Simple Mail Transfer Protocol) versenden. Die Module arbeiten meist unabhängig vom verwendeten Betriebssystem, erfordern jedoch einen SMTP-Server, über den der Versand erfolgen soll.

Modul	Erklärung, Download
Mail::Sendmail	Modul zum einfachen Versand von E-Mails per SMTP http://search.cpan.org/search?dist=Mail-Sendmail
Mail::Sender	Modul, das ein Objekt zum Versenden von E-Mails per SMTP bereitstellt und Attachments verarbeitet http://search.cpan.org/search?dist=Mail-Sender
Net::SMTP	Modul, das verschiedene Funktionen zum Versand von E-Mails bereitstellt, gehört zum libnet-Paket http://search.cpan.org/search?dist=libnet



Das Modul Net::SMTP arbeitet auf einer sehr niedrigen Ebene des SMTP-Protokolls und erfordert spezielles Fachwissen. Das Objekt des Moduls Mail::Sender ist sehr leistungsfähig, besitzt aber eine Vielzahl von Methoden. Für das einfache Versenden von E-Mails ist besonders das Modul Mail::Sendmail geeignet, das in diesem Kapitel beschrieben wird.

SMTP ist das Standardprotokoll, über das im Internet E-Mails gesendet und zwischen E-Mail-Servern ausgetauscht werden.

16.2 Mailversand unter Unix & Co

Für den Mailversand können Sie in allen Betriebssystemen auf Unix-Basis das Programm **sendmail** verwenden. Sie finden es in der Regel unter `/usr/lib/sendmail` bzw. `/usr/sbin/sendmail`. Das Programm wird in Perl mit der Funktion `open()`, ähnlich wie beim Öffnen einer Datei, gestartet. Die Option `-t` beim Aufruf von `sendmail` bewirkt dabei, dass Sie eine E-Mail mit allen Headerzeilen (Informationen im Kopf der E-Mail) über die Standardeingabe übergeben können.

Syntax der Mailversands über *sendmail*

```
open(MAIL, " | /usr/sbin/sendmail -t");
print MAIL "To: Empfängeradresse\n";
print MAIL "From: Absenderadresse\n";
print MAIL "Subject: Betreffzeile\n\n";
print MAIL "Mailtext";
close(MAIL);
```

- ✓ Mit der `open`-Funktion wird `sendmail` mit der Option `-t` gestartet und ein Datei-Handle definiert. Das Zeichen **[1]** ermöglicht den Aufruf von Programmen wie eine Datei, in die geschrieben wird.
- ✓ Das Schreiben der E-Mail-Daten erfolgt wie das Schreiben in eine Datei mit der `print`-Funktion und einem Datei-Handle.
- ✓ Zuerst werden die Headerzeilen ausgegeben. Diese bestehen immer aus einem Schlüsselwort, gefolgt von einem Doppelpunkt **[2]**, einem Leerzeichen und dem Wert. Mit dem Schlüsselwort `To` wird der Empfänger angegeben. `From` bezeichnet die E-Mail-Adresse des Absenders und `Subject` den Betreff der E-Mail. Jede Headerzeile wird mit einem Zeilenumbruch `\n` beendet.
- ✓ Die letzte Headerzeile muss stets mit zwei Zeilenumbrüchen `\n\n` abgeschlossen werden. Alle folgenden Informationen werden als Text der E-Mail interpretiert.
- ✓ Wenn Sie das Datei-Handle schließen, wird die E-Mail gesendet.

Es gibt eine Vielzahl verschiedener Headerzeilen, die Sie übermitteln können. Die folgende Tabelle listet die wichtigsten auf:

Header	Erklärung
<code>From:</code>	Absender (Autor) der E-Mail
<code>To:</code>	Empfänger der E-Mail
<code>Cc:</code>	Sichtbare Kopien-Empfänger der E-Mail (Carbon Copy)
<code>Bcc:</code>	Unsichtbare Kopien-Empfänger (Blind Carbon Copy)

Alle mit `X-` beginnenden Header-Informationen sind nicht standardisiert und werden von jedem E-Mail-Programm unterschiedlich bzw. gar nicht ausgewertet.



Als Absender und Empfänger wird eine E-Mail-Adresse angegeben. Wenn Sie zusätzlich einen Namen verwenden wollen, muss die Angabe in folgender Form erfolgen:

"Name im Klartext" <E-Mail-Adresse>

Da das Zeichen **[@]** in Perl als Kennzeichen für Arrays reserviert ist, müssen Sie es innerhalb von E-Mail-Adressen durch einen vorangestellten Backslash **[3]** markieren, z. B. `info\@meineseite.de`.



Beispiel: *emailkontakt.cgi*, *emailkontakt.html*

Die HTML-Seite stellt ein Kontaktformular zur Verfügung, mit dem die Besucher eine Nachricht an den Betreiber der Seite schreiben können. Die Nachricht wird mithilfe eines CGI-Programms an eine festgelegte E-Mail-Adresse gesendet.

```
#!"F:/ActivePerl/bin/perl.exe"
use strict
use CGI;
use CGI::Carp qw( fatalitiesToBrowser );
my $cgi = new CGI;
① my $mailprogramm="/usr/sbin/sendmail -t";
my $to="info@meineseite.de";
my $from="info@meineseite.de";

print $cgi->header;

② if (!$cgi->param('from')) {
    $cgi->param(-name=>'from', -value=>$from);
}

③ open(MAIL, "|$mailprogramm") ||
    die("Das E-Mail-Programm konnte nicht gestartet werden.");
④ print MAIL "From: \"". $cgi->param('name') . "\" <". $cgi-
>param('from') . ">\n";
print MAIL "To: $to\n";
⑤ print MAIL $cgi->param('text');
⑥ print MAIL $cgi->param('text');
close(MAIL);

print $cgi->start_html('Kontakt');
print $cgi->h3({style=>'font-family:sans-serif'}, 'Nachricht gesendet');
print $cgi->p({style=>'font-family:sans-serif'},
'Vielen Dank für Ihre Nachricht. Wir werden uns umgehend darum
kümmern');
print $cgi->end_html;
```

- ① Am Anfang des Programms wird der Pfad zum sendmail-Programm festgelegt. Diese Angabe muss den jeweiligen Gegebenheiten angepasst werden. Danach werden die Empfängeradresse und eine standardmäßige Absenderadresse definiert.
- ② Die Absenderadresse ist beim Versand von E-Mails Pflicht. Hat der Besucher im Formular keine Adresse angegeben, wird die Standardadresse verwendet.

- ③ An dieser Stelle wird das Mailprogramm gestartet. Über das Datei-Handle MAIL werden im weiteren Verlauf Daten an das Programm gesendet.
- ④ Zuerst erfolgt die Angabe des Absenders in der Form "Name" <E-Mail-Adresse>. Danach wird die Empfängeradresse übermittelt.
- ⑤ Nach der Ausgabe des Betreffs wird der Kopfbereich der E-Mail mit zwei Zeilenumbrüchen abgeschlossen.
- ⑥ Nun wird der eigentliche Nachrichtentext übermittelt und das Mailprogramm beendet.

The screenshot shows a web browser window with the title 'Kontakt-Formular'. Below the title, there is a message: 'Senden Sie uns eine Nachricht mit Ihrem Anliegen und wir melden uns bei Ihnen.' There are four input fields: 'Name:', 'E-Mail:', 'Betreff:', and 'Nachricht:'. Below the 'Nachricht:' field is a text area. At the bottom right of the form is a grey button labeled 'Senden'.

Die Daten des Kontaktformulars werden per E-Mail übermittelt

16.3 Mailversand mit dem Modul `Mail::Sendmail`

Das Modul `Mail::Sendmail` ermöglicht das einfache Versenden von E-Mails über einen SMTP-Server und arbeitet daher sowohl unter Unix als auch unter Windows. Da es nicht zum Lieferumfang jeder Perl-Distribution gehört, muss es unter Umständen zuerst installiert werden. Wie Module installiert werden, wurde bereits behandelt.

E-Mails mit dem Modul senden

Das Modul muss am Anfang des Programms mit dem Befehl `use Mail::Sendmail;` eingebunden werden. Das Senden erfolgt durch einen Aufruf der Funktion `sendmail` mit einem Hash als Argument. Das Hash muss Einträge für die verschiedenen Headerzeilen, den Nachrichtentext und den SMTP-Server enthalten.

```
use Mail::Sendmail;
$mail = (To      => "Empfänger",
         From    => "Absender",
         Subject => "Betreffzeile",
         Message => "Nachricht",
         smtp    => "SMTP-Server");
sendmail(%mail);
```

- ✓ In einem Hash werden für die verschiedenen Headerzeilen Einträge erzeugt, dabei wird nicht zwischen Groß- und Kleinschreibung unterschieden.
- ✓ Unter dem Schlüsselwort `Message` wird der eigentliche Text der Nachricht angegeben.
- ✓ Mit `smtp` wird der Eintrag für den zu verwendenden SMTP-Server bezeichnet.
- ✓ Die E-Mail wird durch einen Aufruf der Funktion `sendmail()` mit dem Hash als Argument verschickt.

Die Funktion liefert den Wert `false`, wenn das Senden der E-Mail misslungen ist. In diesem Fall können Sie durch Auslesen der Variablen `$Mail::Sendmail::error` die Fehlerursache ermitteln.



Beispiel: `emailkontakt2.cgi`, `emailkontakt2.html`

Das CGI-Programm aus dem vorherigen Beispiel soll statt des Programms `sendmail` das Modul `Mail::Sendmail` verwenden.

```

#!/F:/ActivePerl/bin/perl.exe"
use strict;
use CGI;
use CGI::Carp qw( fatalitiesToBrowser );
① use Mail::Sendmail;
my $cgi = new CGI;

② my $to="ich@absend.er";
my $from="ich@absend.er";
my $smtp_server="smtp.absend.er";

print $cgi->header;
print $cgi->start_html('Kontakt');

if (!$cgi->param('from')) {
    $cgi->param(-name=>'from', -value=>$from);
}

③ my %mail=(To      => " \"" . $cgi->param('name') .
"\" <". $cgi->param('email') .">",
            From    => $from,
            Subject => $cgi->param('betreff'),
            Message => $cgi->param('text'),
            smtp    => $smtp_server);

④ if (sendmail(%mail)) {
    print $cgi->h3({style=>'font-family:sans-serif'}, 'Nachricht
gesendet');
    print $cgi->p({style=>'font-family:sans-serif'},
'Vielen Dank für Ihre Nachricht. Wir werden uns umgehend darum
kümmern');
} else {
    print $cgi->h3({style=>'font-family:sans-serif'}, 'Nachricht nicht
gesendet');
    print $cgi->p({style=>'font-family:sans-serif'},
'Beim Versenden der Nachricht ist folgender Fehler aufgetreten:
$Mail::Sendmail::error');
}
print $cgi->end_html;

```

- ① Mit der `use`-Anweisung wird das Modul `Mail::Sendmail` eingebunden.
- ② Die Variablen für die Absender- und Empfängeradresse müssen den jeweiligen Gegebenheiten angepasst werden. Für das Versenden der E-Mails wird ein SMTP-Server benötigt. Auch diese Angabe muss geändert werden.
- ③ In einem Hash werden die für das Senden der E-Mail notwendigen Informationen gespeichert.
- ④ Durch einen Aufruf der Funktion `sendmail()` wird die E-Mail an den SMTP-Server übermittelt.
- ⑤ Konnte die E-Mail nicht korrekt gesendet werden, erfolgt an dieser Stelle die Ausgabe der betreffenden Fehlermeldung.

16.4 E-Mail-Adresse prüfen

Vor dem Senden einer E-Mail sollten Sie die E-Mail-Adresse auf äußerliche Korrektheit prüfen. Sie können damit zwar nicht ermitteln, ob die E-Mail-Adresse überhaupt existiert, vermeiden aber eine Vielzahl zurückgesendeter E-Mails, bei denen die E-Mail-Adresse bereits sichtbar falsch ist.

Die äußerliche Gültigkeit einer E-Mail-Adresse können Sie mit einem regulären Ausdruck überprüfen. Dazu sollten Sie den grundsätzlichen Aufbau einer E-Mail-Adresse kennen:

Teil der Adresse	Erklärung	Erlaubte Zeichen	Suchmuster
Name	Teil der E-Mail-Adresse vor dem Zeichen @; kann mit einem Punkt . untergliedert sein; Unterstrich _ ist zulässig; mindestens ein Zeichen	a-z 0-9 _ .	[_\.a-zA-Z0-9]+
Trennzeichen	Trennt den Namen von der Domainangabe	@	\@
Domain	Bezeichnet den Domännamen, kann mit einem Punkt . untergliedert sein; Bindestrich - ist zulässig; mindestens ein Zeichen	a-z 0-9 - .	[-\.a-zA-Z0-9]+
Toplevel-Domain	Wird nach dem letzten Punkt angegeben; kann aus zwei bis vier Buchstaben bestehen	a-z	[a-zA-Z]{2,4}

Aus diesen Angaben ergibt sich folgendes Suchmuster, das auf eine gültige E-Mail-Adresse zutrifft:

```
/^[_\.a-zA-Z0-9]+\@[[-\.a-zA-Z0-9]+\.[a-zA-Z]{2,4}]/i
```

Beispiel

Der Ausschnitt aus einem Perl-Programm prüft eine über ein HTML-Formular eingegebene E-Mail-Adresse auf Gültigkeit.

```
$email = $cgi->param('email');
if ($email =~ /^[_\.a-zA-Z0-9]+\@[[-\.a-zA-Z0-9]+\.[a-zA-Z]{2,4}]/i) {
    print "Die E-Mail-Adresse <b>$email</b> scheint gültig zu sein.";
} else {
    print "Die E-Mail-Adresse <b>$email</b> ist nicht gültig.";
```

16.5 Übung

Mit E-Mails arbeiten

Übungsdatei: --

Ergebnisdatei: *newsletter_senden.cgi*

1. Das CGI-Programm soll einen Newsletter an alle in der Tabelle `newsletter` gespeicherten Empfänger senden.
2. Wird das Programm ohne Parameter aufgerufen, soll ein einfaches Formular zum Eingeben von Betreffzeile und Nachrichtentext angezeigt werden.
3. Das Programm soll die E-Mail über einen SMTP-Server senden. Binden Sie das benötigte Modul ein, und definieren Sie per Variable einen SMTP-Server. Legen Sie zusätzlich eine Absenderadresse fest.
4. Stellen Sie im Programm eine Verbindung zur Datenbank `homepage` auf dem lokalen MySQL-Server her.
5. Erstellen Sie eine Abfrage, die Ihnen alle Empfänger des Newsletters mit Name und E-Mail-Adresse liefert.
6. Prüfen Sie jede E-Mail-Adresse vor dem Versand auf äußerliche Gültigkeit.
7. Senden Sie den über das Formular eingegebenen Text mit der Betreffzeile nacheinander an die Newsletter-Empfänger. Geben Sie dabei die Empfängeradresse mit Name und E-Mail an.
8. Reagieren Sie auf mögliche Fehler beim Senden der E-Mails.
9. Zeigen Sie am Ende des Programms an, an wie viele Empfänger der Newsletter gesendet wurde.

Anhang

Im Anhang erfahren Sie

- ✓ wie Sie Perl installieren
- ✓ wie Sie XAMPP installieren
- ✓ wie Sie geeignete Editoren und integrierte Entwicklungsumgebungen für Perl auswählen und installieren

A.1 Perl installieren

Unter Linux/Unix, Mac OS X und anderen Unix-artigen Betriebssystemen ist Perl normalerweise automatisch vorhanden, während Perl unter Windows in der Regel nachinstalliert werden muss. Nachfolgend wird gezeigt, wie Sie Perl unter Windows und Linux/Unix/Mac OS X installieren können. Dabei werden ebenso unterschiedliche Distributionen von Perl vorgestellt. Perl ist in der jeweils aktuellen Version für alle unterstützten Betriebssysteme kostenlos erhältlich, es gibt aber auch kostenpflichtige Distributionen.

Perl als Quellcode oder Binärdistribution verwenden

Für Unix-artige-Betriebssysteme ist Perl sowohl als Quellcode zum manuellen Kompilieren als auch als fertige Installationsversion (Binärdistribution) verfügbar. Wenn Sie sich mit der Übersetzung und Installation unter Linux/Unix gut auskennen, können Sie den Quellcode von Perl verwenden und sich ein angepasstes Perl-System installieren. Der Vorteil ist, dass dieser Quellcode meist ohne zusätzliche Einstellungen übersetzt wird und dabei automatisch die Besonderheiten Ihres Systems integriert. Bei den bereits vorübersetzten binären Standard-distributionen werden häufig erweiterte Features weggelassen, die nicht oft benötigt werden. Allerdings muss zu einer manuellen Übersetzung ein passender C-Compiler und Linux/Unix-Werkzeuge wie die Shell (sh) oder der Programm-Manager make vorhanden sein und Sie müssen sich mit der Befehlszeile unter Linux/Unix auskennen. Windows-Anwender können zur Installation eines Perl-Systems aus Quellcode zwar *cygwin* installieren, um Linux-Programme unter Windows zu nutzen, aber dies ist ein zusätzlicher vorbereitender Schritt, der den Vorgang noch aufwändiger macht.

Die manuelle Übersetzung der Perl-Quellcodes und individuelle Installation ist bei modernen Perl-Varianten selten wirklich notwendig. Für Windows, Mac OS X und Linux/Unix sind vollständig kompilierte Versionen von Perl als Binärdistributionen (ein übersetztes Programm nennt man Binärkode, daher der Name) verfügbar. Diese Distributionen sind vom Umfang her sehr vollständig und auf dem neuesten Stand der Entwicklung. Die Installation einer Binärdistribution ähnelt meist der Installation von anderen Programmen auf einem System und ist daher leicht durchzuführen. Hier sind einige Beispiele für Distributionen bzw. wo Sie diese erhalten:

Betriebssystem	Distribution	Download-Adresse
Windows	Strawberry Perl	http://strawberryperl.com/
Windows/ Linux/Unix/Mac OS X	ActivePerl	https://www.activestate.com/activeperl/downloads
Linux/Unix/Mac OS X	Verschiedene Distributionen	https://www.perl.org/get.html

Sie erhalten auf den Downloadseiten meist Setupdateien, deren Dateiname neben diversen Informationen oft auch die genaue Versionsnummer von Perl selbst enthält. Zum Beispiel ActivePerl-5.20.2.2002-MSWin32-x86-64int-299195.msi unter Windows, ActivePerl-5.20.2.2002-x86_64-linux-glibc-2.15-299195.tar.gz unter Linux/Unix oder ActivePerl-5.20.2.2002-darwin-10.8.0-299195.dmg unter Mac OS X. Diese Versionsnummer kann sich daher im Laufe der Zeit ändern.



Für viele weitere Betriebssysteme, wie z. B. NetBSD, Amiga, Solaris, AIX, HPUX, BeOS, SGI, finden Sie eine Perl-Version unter den Adressen <https://www.perl.org/get.html> oder <http://www.perl.com/CPAN/ports/index.html>.

Installation von Perl-Binärdistributionen unter Windows

Unter Windows werden Sie meist eine Binärdistribution von Perl installieren. Für die verschiedenen Windows-Versionen gibt es mehrere vorkompilierte Perl-Distributionen. Empfehlenswert sind ActiveState oder Strawberry Perl, da diese Distributionen lange schon entwickelt wurden und ausgereift sind, einen großen Funktionsumfang besitzen und einfach zu installieren sind.

Auf der ActiveState-Webseite finden Sie unter <http://www.activestate.com/Products/Download> das jeweils aktuellste Perl-Paket für alle relevanten Windows-Versionen.

Beachten Sie, dass es von ActivePerl auch kommerzielle Varianten gibt (nur die Community Edition ist frei), während Strawberry Perl rein OpenSource ist.



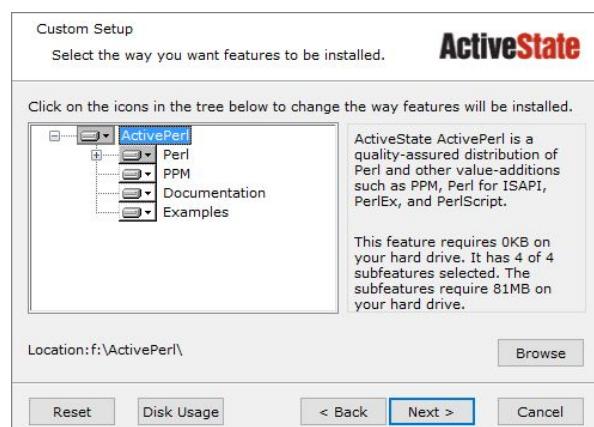
Binärdistribution unter Windows installieren

ActivePerl installieren Sie wie folgt (die Installation bei Strawberry Perl läuft nahezu gleich ab):

- ▶ Führen Sie nach dem Download der Setupdatei einen Doppelklick auf diese Datei aus, um die Installation zu starten.
Es startet ein Assistent, den Sie in der Regel ohne große Änderungen durchspielen können. Die Dialoge sollten wenige Fragen aufwerfen.
- ▶ Behalten Sie im Dialogfenster, in dem Sie die zu installierenden Komponenten festlegen können, die Vorauswahl bei. Allerdings kann es interessant sein, dass Sie das Installationsverzeichnis individuell anpassen (beispielsweise falls ein Verzeichnis mit Leerzeichen im Namen vorgeschlagen wurde, sollten Sie in jedem Fall ein neues Verzeichnis mit einem Namen ohne Leerzeichen erstellen). Klicken Sie *Next*, um zum nächsten Schritt zu gelangen.



Der Start des Assistenten bei ActivePerl



Komponenten festlegen

- Bestätigen Sie mögliche weitere Dialogfenster mit *Next*, bis Sie zum Dialogfenster *Choose Setup Options* gelangen.

Mit der Option ① erreichen Sie, dass der Dateipfad zu Perl im System eingetragen wird. Damit können Sie den Perl-Interpreter aus jedem Verzeichnis ohne Pfadangabe zum Installationsverzeichnis von Perl aufrufen.

Wenn Sie mehrere Perl-Distributionen bzw. -Varianten von Perl auf Ihrem Rechner haben, kann diese Einstellung problematisch sein, da damit immer der Perl-Interpreter verwendet wird, der (zuerst) im Suchpfad des Betriebssystems gefunden wird – sofern Sie beim Aufruf keine absolute Pfadangabe vornehmen.

Mit der Option ② legen Sie fest, dass Perl-Programme durch einen Doppelklick im Explorer gestartet werden können.

- Bestätigen Sie den Dialog mit *Next*.
 - Klicken Sie auf *Install*, um mit der Installation zu beginnen.
- Die Installation kann je nach Leistung Ihres Computers einige Minuten dauern.
- Beenden Sie die Installation im letzten Dialogfenster mit *Finish*.

ActivePerl wird standardmäßig im Verzeichnis *c:\perl* installiert, wobei es oft sinnvoll sein kann, ein individuelles Installationsverzeichnis zu wählen. Der vollständige Pfad zum Perl-Interpreter lautet im Standardfall *c:\perl\bin\perl*.



Die Perl-Installation enthält unter Windows zusätzlich den **Perl Package Manager** (PPM). Dieses Programm ermöglicht Ihnen die einfache Verwaltung von Erweiterungen und Modulen.

Um die erfolgreiche Installation zu testen, können Sie – wie in Kapitel 2 beschrieben – in der Konsole den Perl-Interpreter mit dem Parameter *-v* aufrufen. Als Ergebnis erhalten Sie die Versionsbezeichnung von Perl.

Aus Quellcodes unter Linux/Unix/Mac OS X installieren

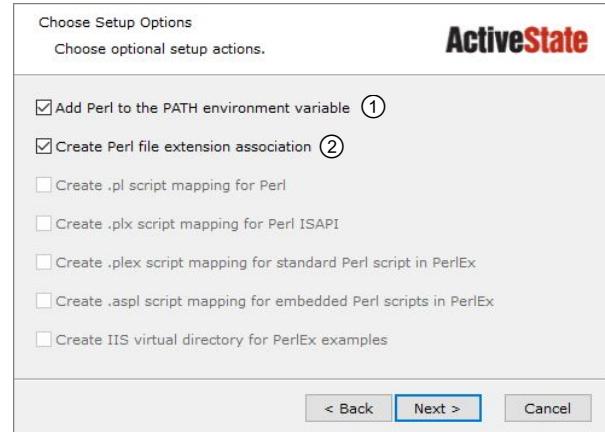
Binärdistribution gibt es auch für alle auf Unix basierenden Betriebssysteme. Selbstverständlich können Sie aber auch heute noch aus den Quelltexten von Perl individuell eine Perl-Version für Ihr System erstellen. Vom Unterbau ist Mac OS X ein BSD-Unix mit einigen Besonderheiten, weshalb viele Schritte gleich oder ähnlich wie unter Linux und Unix ablaufen. Der Vollständigkeit halber soll das hier ausgeführt werden, wobei die nachfolgend noch beschriebene Installation einer Binärdistribution bequemer ist und empfohlen wird – sofern Sie überhaupt die Standardinstallation von Perl auf Ihrem Linux/Unix/Mac OS X ersetzen oder ergänzen wollen.

Die folgenden Schritte erklären die Installation von Perl aus Quellcodes. Früher war das der einzige sinnvolle und recht umständliche Weg. Sie benötigen auf Ihrem Linux/Unix-System dazu den Compiler *cc* sowie diverse Befehlszeilen-Tools (etwa zum Entpacken von Dateien). Diese Tools bringt jede moderne Linux/Unix-Distribution mit.



Die Installation von Perl aus Quellcodes, kann sich auf verschiedenen Linux-Distributionen und auch Unix selbst oder Mac OS X im Detail etwas unterscheiden. Die nachfolgenden Schritte beziehen sich auf ein Ubuntu-System und sollten so auf allen Debian-Systemen funktionieren. Bei anderen Linux/Unix-Distributionen und Mac OS X laufen die Schritte ähnlich ab.

Die Installation aus Quellcode mit folgender Schrittfolge, – wobei Sie Administrator-Rechte benötigen, wenn Sie die Installation nicht in Ihr Home-Verzeichnis vornehmen wollen, – wird nachfolgend im Home-Verzeichnis installiert. Der Name der Perl-Datei ist anzupassen.



Setup-Optionen festlegen

- ▶ Unter <https://www.perl.org/get.html> finden Sie den Perl-Quelltext als so genannte **Source Code Distribution** (Quellcode-Distribution). Führen Sie den Download der *.tar.gz*-Datei mit dem Quellcode durch, der die aktuellste und eine stabile Perl-Version enthält.
- ▶ Entpacken Sie die *.gz*-Datei manuell mit einem Programm wie gzip (etwa mit `gzip -d perl-5.22.0.tar.gz`) oder einem grafischen Tool.
Als Resultat finden Sie im aktuellen Verzeichnis die Datei *perl-5.22.0.tar*.
oder
- ▶ Geben Sie den Befehl

```
wget http://www.cpan.org/src/5.0/perl-5.22.0.tar.gz
```

ein. Das lädt und entpackt gleichzeitig die *.gz*-Datei auf Ihrem Rechner.
- ▶ Die Datei *perl-5.22.0.tar* ist ein Archiv für die einzelnen Quelltextdateien. Zum Umwandeln führen Sie folgenden Befehl aus:

```
tar -xzf perl-5.22.0.tar.gz.
```

Es entsteht ein Verzeichnis mit dem Namen der zu installierenden Perl-Version.
- ▶ Wechseln Sie in das generierte Verzeichnis:

```
cd perl-5.22.0
```
- ▶ Legen Sie einige Optionen für die Installation mit folgender Zeile fest (beachten Sie den führenden Punkt):

```
./Configure -des -Dprefix=$HOME/localperl
```
- ▶ Nach Abschluss der Konfiguration können Sie den nächsten Schritt der Installation starten. Geben Sie dazu den Befehl `make` ein.
- ▶ Testen Sie die Installationsanweisungen mit dem Befehl `make test`.
- ▶ Wenn der Test erfolgreich war, schließen Sie die Installation mit dem Befehl `make install`.

Erst zu diesem Zeitpunkt wird die Installation tatsächlich durchgeführt und die Quelltexte kompiliert. Dieser Vorgang kann je nach Leistung Ihres Computers mehrere Minuten dauern. Möglicherweise werden Ihnen zusätzliche Fragen gestellt, die Sie jedoch meist mit der Standardeinstellung beantworten können.

Nach dem erfolgreichen Abschluss der Installation erscheint die Meldung, dass die Installation beendet ist. Wenn Sie eine Fehlermeldung erhalten, sollten Sie in der Datei *INSTALL* nach Lösungsmöglichkeiten suchen.

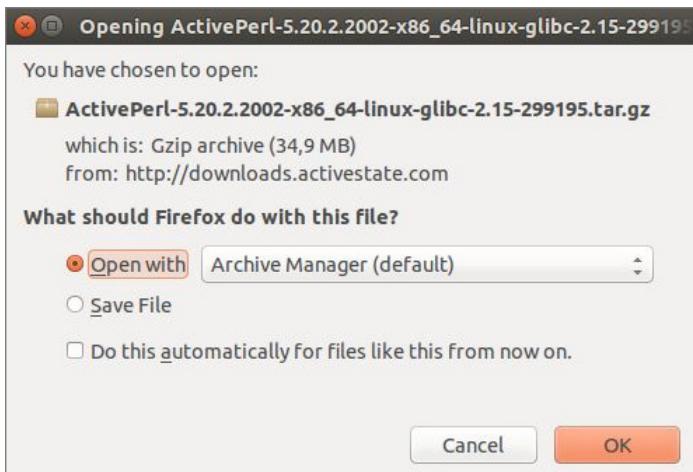
Perl steht unter Linux/Unix/Mac OS X standardmäßig im Verzeichnis */usr/bin/* oder */usr/local/bin/*. Der vollständige Pfad zum Perl-Interpreter lautet daher */usr/bin/perl* oder */usr/bin/local/perl*. Mit der Anweisung `which perl` in einer Konsole können Sie nachsehen, wo sich Ihre Perl-Standardinstallation befindet. Beachten Sie, dass die Beschreibung oben Perl im Home-Verzeichnis installiert hat. Daher kann der Pfad zu dieser Perl-Installation auch */home/ralph/perl-5.22.0* oder ähnlich sein.



Binärdistribution unter Linux/Unix/Mac OS X installieren

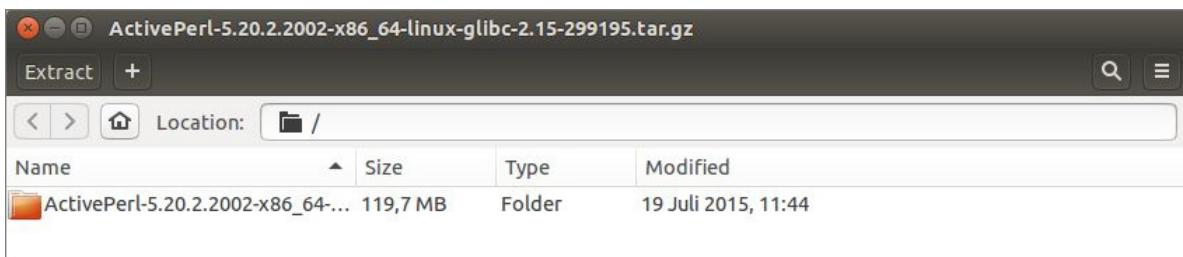
Durch die Verfügbarkeit von Binärdistribution sowie grafischer Tools auch für Linux/Unix ist die Installation von Perl unter Linux/Unix mittlerweile ähnlich einfach wie unter Windows.

- ▶ Laden Sie die passende *.tar.gz*-Datei aus dem Internet.
Meist wird beim Download angeboten, dass die Datei automatisch mit dem Archive Manager geöffnet wird.



Automatisches Öffnen mit dem Archive Manager

- ▶ Wird die Datei nicht automatisch geöffnet, können Sie die Datei speichern und anschließend mit dem Archive Manager öffnen oder mit einem anderen Tool zum Extrahieren.



Nachträgliches Öffnen des geladenen Archivs

- ▶ Extrahieren Sie das Archiv.
Sie können das Verzeichnis dazu auch einfach mit der Maus an einen gewünschten Ort ziehen.

Das Verzeichnis enthält nun ein Installationsskript. Bei ActivePerl lautet der Name dieses Skripts *install.sh*.

- ▶ Rufen Sie das Installationsskript in einer Konsole wie folgt auf:

```
sh install.sh.
```

Unter Umständen benötigen Sie Administrationsrechte. Mit `sudo sh install.sh` können Sie die Installation gleich als root ausführen.

Perl steht standardmäßig auch unter Mac OS X zur Verfügung. Wollen Sie Perl aus einer Binärdistribution selbst installieren wollen, laden Sie die *.dmg*-Datei und führen diese aus.

A.2 XAMPP

Was ein Webserver ist

Wenn Sie mit Perl CGI-Skripte erstellen und ausführen wollen, benötigen Sie einen Webserver, der Perl unterstützt. Ein **Server** ist ein Programm, das für seine **Clients** (Kunden) Anfragen beantwortet. Ein **Webserver** ist ein spezieller Server, der Anfragen seiner Clients (Browser) über das Web beantwortet. Als Protokoll muss der Client http verstehen.

Apache als Webserver

Für die Ausführung von Perl-CGI-Skripten wird als Webserver meist Apache (<http://httpd.apache.org/>) eingesetzt. Nahezu alle Internetprovider verwenden Apache als Webserver.

Wenn Sie bereits ein Hosting-Angebot mit Apache als Webserver verwenden, werden Sie diesen nicht einrichten und administrieren müssen, und Sie können dort Perl-Programme ausführen. Da eine lokale Ausführung von Perl-CGI-Skripten zu Testzwecken zu empfehlen ist, benötigen Sie für Ihre lokale Testumgebung einen Apache-Webserver.

Außerdem benötigen Sie für Datenbankzugriffe mit Perl auch MySQL, eventuell die Möglichkeit von E-Mail-Versand und die Unterstützung für Perl.

XAMPP – Komplettpaket für die Testumgebung

Empfehlenswert ist die Verwendung des Komplettpakets XAMPP (<https://www.apachefriends.org/de/>), das bereits alle Bestandteile enthält.

The screenshot shows a web browser displaying the Apache Friends website at <https://www.apachefriends.org/de/index.html>. The page title is "XAMPP Apache + MySQL + PHP + Perl". The main content area features the XAMPP logo (an orange icon resembling a stylized 'X' or a play button next to the word "XAMPP"). Below the logo, there is a section titled "Was ist XAMPP?" which describes it as the most popular PHP development environment. It states that XAMPP is a completely free, easy-to-install Apache distribution that includes MySQL, PHP, and Perl. A green "Download" button with the text "Hier klicken für weitere Versionen" (Click here for more versions) is visible. To the right, there are three download links for different operating systems: "XAMPP für Windows v5.6.12 (PHP 5.6.12)", "XAMPP für Linux v5.6.12 (PHP 5.6.12)", and "XAMPP für OS X v5.6.12 (PHP 5.6.12)".

Die Homepage des XAMPP-Projekts

Beachten Sie, dass sich der Aufbau der Webseite des XAMPP-Projekts immer wieder ändert. Die Links zum Download der Software finden Sie immer hervorgehoben auf der Webseite.

XAMPP ist eine kostenlose, leicht zu installierende und zu nutzende Apache-Distribution, die MySQL, PHP und Perl enthält. XAMPP steht für die Betriebssysteme Windows, Mac OS X und Linux als Betriebssysteme zur Verfügung.

XAMPP besteht aus:

- ✓ Apache (A)
- ✓ MySQL (M)
- ✓ PHP (P)
- ✓ Perl (P)

Das X ist ein Platzhalter, der historische Ursachen hat. Ursprünglich gab es die Apache-Distribution nur für Linux und ohne Perl, LAMP genannt. Dann kam ein Paket für Windows dazu (WAMP). Das Betriebssystem wurde später durch den Platzhalter X ersetzt und das zweite P für Perl hinzugefügt, als auch dafür Unterstützung integriert wurde. Das standardmäßige Installationsverzeichnis des Systems unter Linux heißt heute noch *lampp*.

XAMPP downloaden

XAMPP können Sie auf Ihrem lokalen Rechner, einem Rechner in Ihrem lokalen Netzwerk oder einem Rechner installieren, der mit dem Internet verbunden und nur darüber erreichbar ist und auf dem Sie einen Webserver betreiben wollen. Im Buch wird nur auf die lokale Installation eingegangen.

- ▶ Laden Sie sich zuerst die neueste Version von XAMPP auf den Rechner, auf dem Sie den Webserver installieren und betreiben wollen.
Sie können entweder direkt die empfohlene Standardvariante von XAMPP zum Download auswählen (wenn diese auf der Einstiegsseite angezeigt wird) oder Sie wählen gezielt eine bestimmte Version für XAMPP aus. Je nach Aufbau der Webseite des XAMPP-Projekts finden Sie dazu eine entsprechend benannte Schaltfläche oder einen Hyperlink. Der Download sollte automatisch starten und Sie werden unter Umständen zusätzlich zu einer Webseite beim SourceForge-Projekt geleitet, wo weitere Versionen von XAMPP bereitgestellt werden.
- ▶ Wenn der Download startet, wählen Sie den Link zum Speichern und speichern Sie die Datei auf Ihrer Festplatte.

Auf der Landingpage des XAMPP-Projekts finden Sie auch den Link *Download - Hier klicken* für weitere Versionen. Wenn Sie keine der Standardzusammenstellungen, die in den parallelen Links auf der Landingpage des Projekts direkt zum Download zu wählen sind, wünschen, klicken Sie auf diesen Link, um zur Seite mit den Download-Varianten zu gelangen. Sie können dort gezielt eine bestimmte Version von XAMPP auswählen.



Wenn Sie eine individuelle Auswahl der XAMPP-Version vornehmen, sollten Sie einen Installer verwenden. Die nachfolgenden Ausführungen beziehen sich auf einen Installer sowohl unter Windows als auch Linux/Unix/Mac OS X.

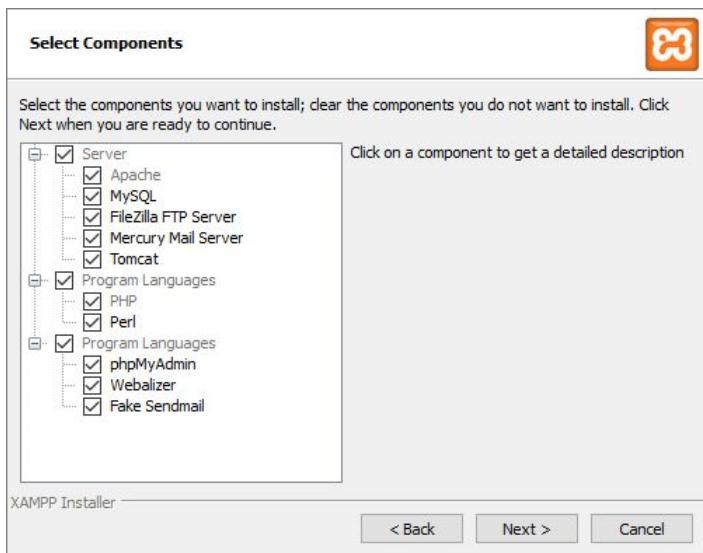
XAMPP unter Windows installieren

Installieren Sie XAMPP unter Windows beispielsweise wie folgt:

- ▶ Öffnen Sie auf dem Installationsrechner den Windows-Explorer oder den Arbeitsplatz.
- ▶ Wechseln Sie in den Ordner, in dem Sie XAMPP beim Download gespeichert haben.
- ▶ Führen Sie einen Doppelklick auf die heruntergeladene XAMPP-Datei aus.

Die Installation von XAMPP sollte nun starten. Wenn Sie Windows 7, 8 oder 10 im Einsatz haben, erscheinen wahrscheinlich vor und während der Installation eine oder mehrere Sicherheitswarnung(en). So können aktivierte VirensScanner zu einer Meldung führen und das Installationsprogramm warnt etwa davor, XAMPP in den Ordner *C:\Program Files (x86)* zu installieren, da es hier wegen fehlender Ordner-Schreibberechtigungen zu einer eingeschränkten Funktionalität von XAMPP kommen kann. Falls Sie solche Meldungen erhalten, ignorieren Sie die Warnungen und bestätigen Sie diese Meldungen alle mit *OK*. Falls Sie die Meldung zu fehlenden Schreibberechtigungen erhalten, sollten Sie XAMPP in ein eigenes Hauptverzeichnis auf Ihrer Festplatte installieren, zum Beispiel *C:\xampp* oder *F:\xampp*.

In den Folgeschritten werden Sie nach verschiedenen Konfigurationen der zu installierenden XAMPP-Version gefragt, zum Beispiel welche Bestandteile von XAMPP Sie installieren möchten.



XAMPP-Bestandteile auswählen

Bleiben Sie bei allen Dialogen des Assistenten bei den Vorgabeeinstellungen, außer ggf. bei dem oben genannten Problem mit dem Installationsverzeichnis. Hierdurch werden alle benötigten Komponenten, wie zum Beispiel Perl und MySQL, installiert. Ein XAMPP-Startmenü-Eintrag sowie eine Verknüpfung zu XAMPP auf dem Desktop wird erstellt.

Der Installationsvorgang kann einige Minuten dauern. Sobald die Installation beendet ist, erscheint eine Meldung.

- ▶ Zum Beenden der Installation klicken Sie auf *Finish* beziehungsweise *Fertig stellen*.
- ▶ Bestätigen Sie die anschließend eingeblendete Frage mit *Ja*, wenn Sie das XAMPP Control Panel starten wollen.

XAMPP unter Linux installieren

Für die Installation von XAMPP unter Linux werden etwas mehr Kenntnisse zum Betriebssystem vorausgesetzt als bei Windows. Arbeiten Sie hier am besten in einer Konsole.

- ▶ Öffnen Sie eine Konsole.
- ▶ Wechseln Sie in das Verzeichnis, in dem Sie die Installationsdatei von XAMPP gespeichert haben (meist das Verzeichnis *Downloads* im Home-Verzeichnis des Anwenders).

```
ralph@ubuntu: ~/Downloads
ralph@ubuntu:~/Downloads$ ls
xampp-linux-5.6.12-0-installer.run
ralph@ubuntu:~/Downloads$ ;5~
```

Setup-Datei im Download-Verzeichnis

- ▶ Ändern Sie die Rechte des XAMPP-Installers in einer Shell mit dem folgenden Befehl und ersetzen Sie den * durch die konkrete Versionsnummer:
- ```
chmod 755 xampp-linux-*installer.run
```

- Führen Sie den Installer in der Shell mit dem folgenden Befehl (\* = konkrete Versionsnummern) aus:  
`sudo ./xampp-linux-*installer.run`

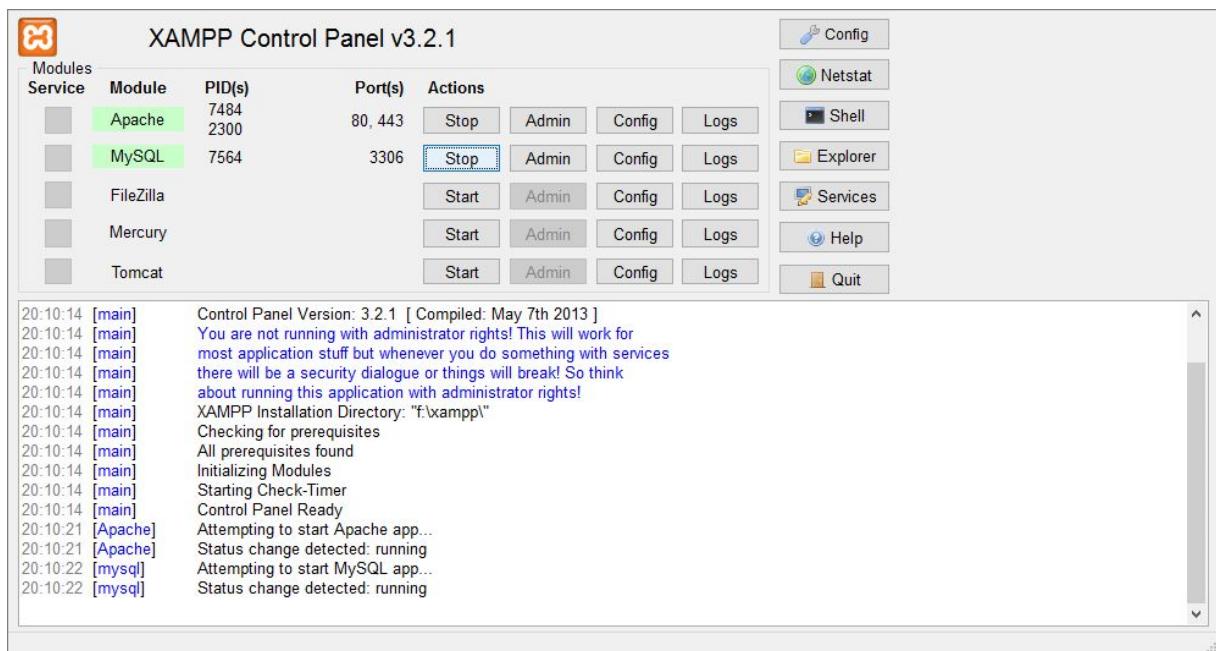
Dabei müssen Sie das Passwort für den root-Anwender eingeben.

XAMPP wird damit standardmäßig im Verzeichnis `/opt/lampp` installiert.

## XAMPP betreiben

Nach der erfolgreichen Installation von XAMPP auf Ihrem Rechner, sollten Sie testen, ob Sie die Webserver-Komponenten problemlos aufrufen können. Dazu muss die Webserver-Software Apache (die auch Perl bereitstellt) gestartet werden. Wenn Sie mit Perl Datenbankzugriffe durchführen wollen, muss auch das Datenbankmanagementsystem MySQL gestartet werden.

- Betätigen Sie neben dem Eintrag *Apache* die Schaltfläche *Starten*, um die Webserver-Software zu starten.
- Klicken Sie neben dem Eintrag *MySQL* auf Starten, um das DBMS zu starten.



Beispiel für ein Control Panel von XAMPP



In einigen Versionen von XAMPP sind die Programme FileZilla (FTP-Server), Mercury (Mailserver) und Tomcat (HTTP-Server mit Java-Laufzeitumgebung) dabei. Diese werden für CGI-Skripte mit Perl nicht benötigt. Sie müssen daher auch nicht gestartet werden. Für den Versand von E-Mails ist wichtig, dass in XAMPP *sendmail* dabei ist.

- Das gesamte XAMPP-System (z.B. Webserver und Datenbank) unter Linux/Unix starten Sie bei einer Standardinstallation mit dem folgenden Befehl in der Konsole:  
`sudo /opt/lampp/lampp start`
- Mit `sudo /opt/lampp/lampp stop` stoppen Sie das gesamte System.

## A.3 Editoren und Entwicklungstools für Perl

Perl-Programme können in einem beliebigen Texteditor erstellt werden. Besonderen Komfort bieten Texteditoren, die eine Syntaxhervorhebung für Perl-Programme besitzen oder spezielle integrierte Entwicklungsumgebungen (IDE) für Perl.

### Editoren für Linux & Co

Bei Linux sind leistungsfähige Editoren mit Unterstützung für Programmiersprachen meist bereits standardmäßig verfügbar.

Folgende Editoren eignen sich besonders gut zur Programmierung, müssen allerdings nachträglich hinzugefügt werden. Sie stehen teilweise auch für andere Betriebssysteme zur Verfügung:

- ✓ gVim
- ✓ gEdit
- ✓ Bluefish
- ✓ Cream
- ✓ Scribes
- ✓ jEdit
- ✓ emacs
- ✓ SciTE
- ✓ UltraEdit

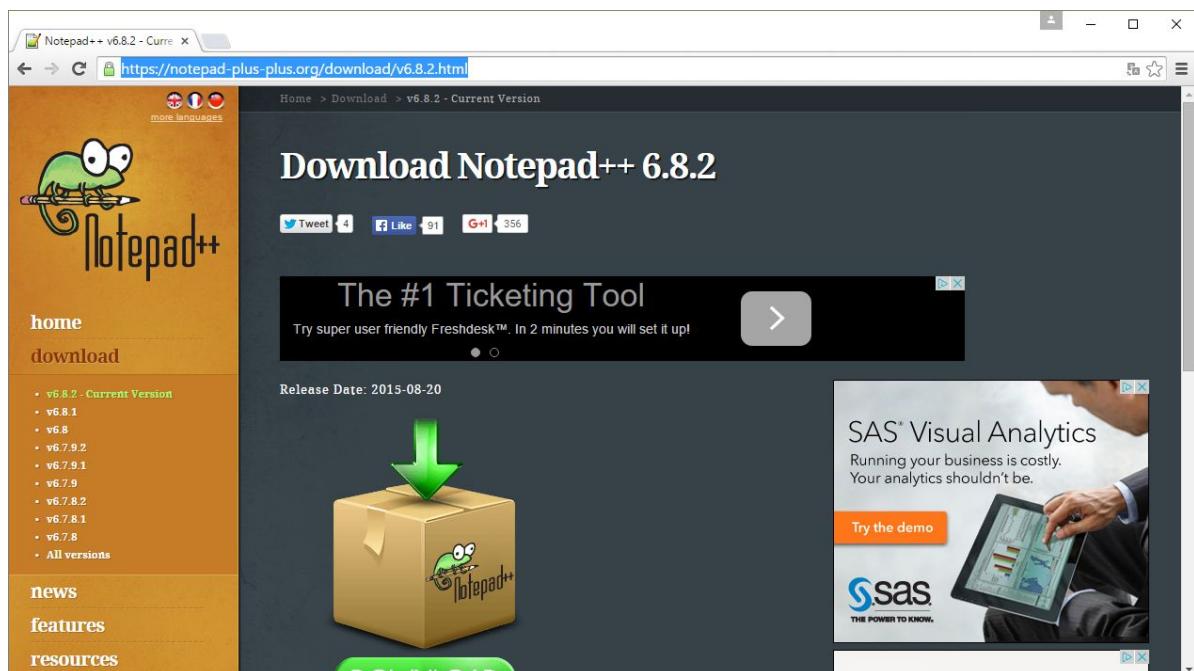
### Der Texteditor Notepad++ für Windows

Notepad++ ist ein sehr mächtiger Texteditor für Windows, der viele Features bereitstellt – von der Unterstützung vieler Programmiersprachen (Syntax-Hervorhebung, Multi-Ansicht, Drag&Drop, Auto-Vervollständigung und vieles mehr) bis hin zu den wichtigsten Features einer vollwertigen Textverarbeitung. Dennoch lädt Notepad++ schnell und braucht wenige Ressourcen. Die Bedienung ist sehr einfach und intuitiv und braucht kaum Eingewöhnungszeit.

Bei Notepad++ (OpenSource) sind mittlerweile einige Plug-ins erschienen, welche die Arbeit weiter erleichtern. Diese können Sie bei der Installation aus- oder abwählen.

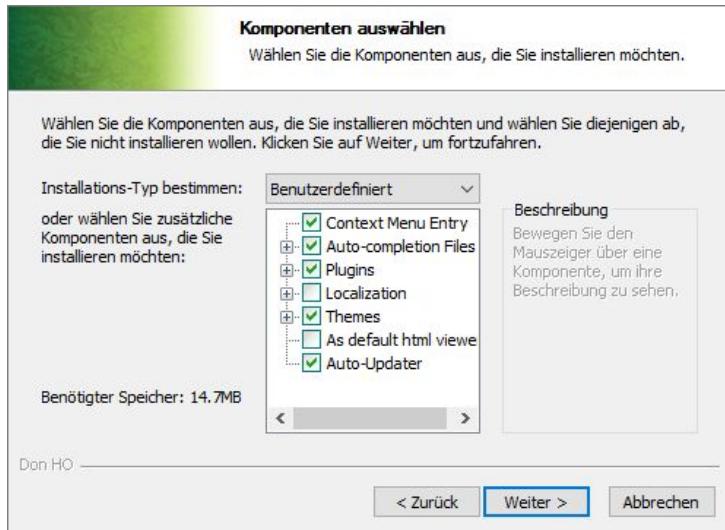


Den Editor erhalten Sie unter <https://notepad-plus-plus.org/download/> in der jeweils aktuellsten Version. Es gibt sowohl Installer, aber auch Versionen, die einfach extrahiert werden können und keine Veränderungen am Betriebssystem vornehmen.



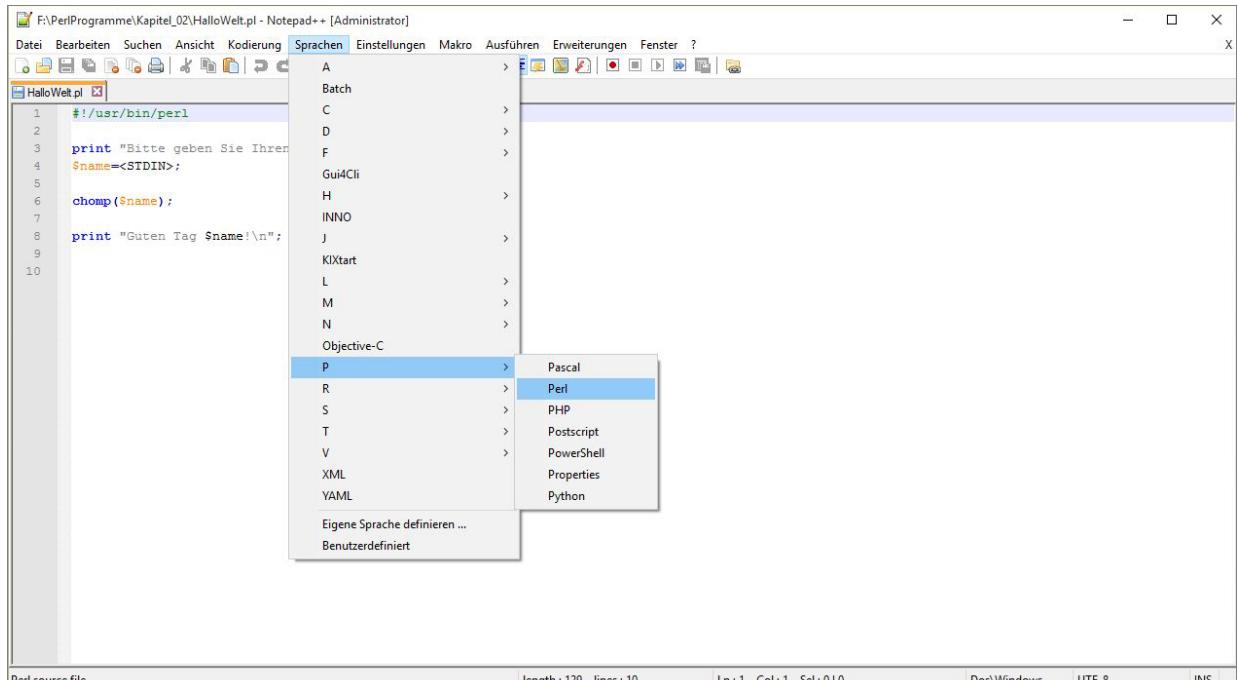
Die Downloadseite von Notepad++

Der Installer ist selbsterklärend. Sie können in der Regel die Vorgabeeinstellung beibehalten. Nur bei den installierten Komponenten nehmen Sie möglicherweise eine individuelle Anpassung vor.



#### *Komponenten bei der Installation auswählen*

Nach der Installation können Sie eine Perl-Datei in den Editor laden und sehen die Syntax-Hervorhebung für Perl. Der Editor entscheidet sich dabei auf Grund der Dateierweiterung, welche Programmiersprache unterstützt werden soll. Über das Menü *Sprachen* können Sie aber auch explizit auswählen, welche Syntax einer Programmiersprache unterstützt wird.

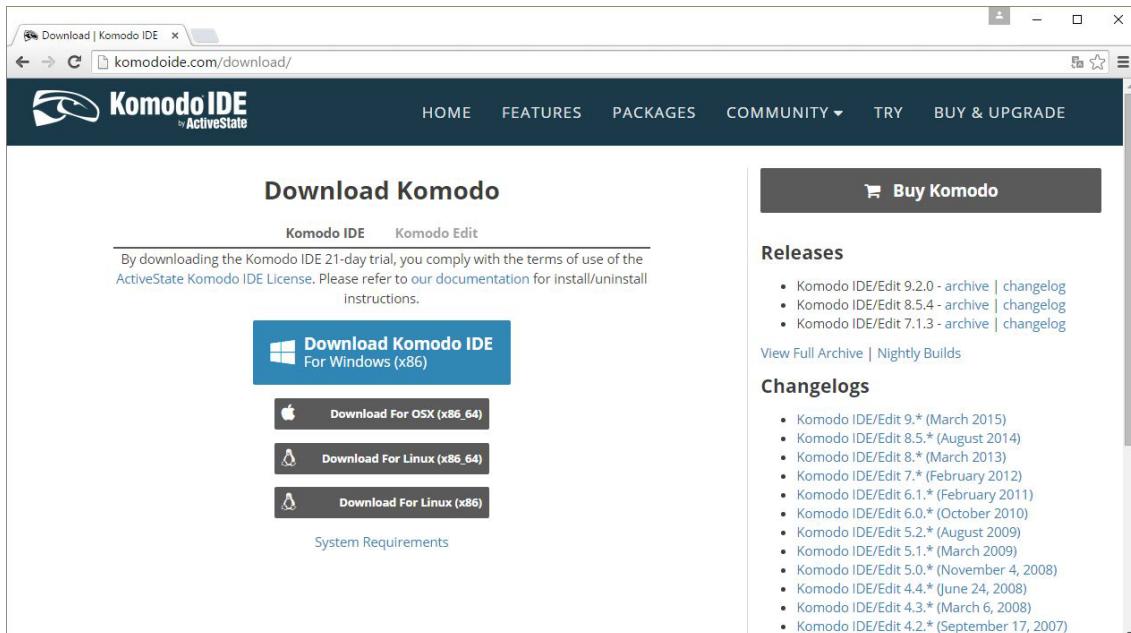


#### *Unterstützte Sprache auswählen*

## Die Entwicklungsumgebung Komodo Edit

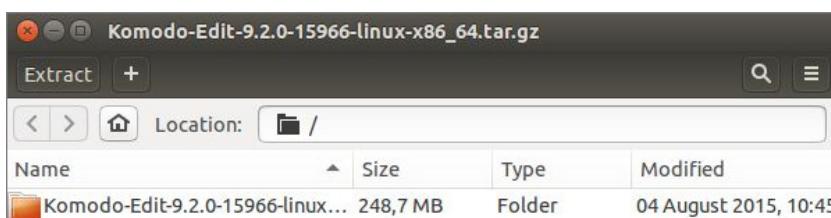
Ein mächtiger Editor wie Notepad++ genügt in vielen Fällen zur Erstellung von Perl-Programmen. Mehr Komfort bieten integrierte Entwicklungsumgebungen. Für Perl ist Komodo Edit (<http://komodoide.com/komodo-edit/>) eine empfehlenswerte Wahl, die für mehrere Betriebssysteme (u.a. Windows, Linux, Mac OS X) verfügbar ist. Komodo Edit bezeichnet die OpenSource-Variante der kommerziellen Komodo-IDE, die noch diverse weitergehende Features bereitstellt. Komodo Edit stellt einen guten Kompromiss zwischen Leistungsfähigkeit und dennoch relativ einfacher und intuitiver Bedienbarkeit dar.

Um Komodo Edit zu installieren, laden Sie die Setupdatei für Ihr Betriebssystem von der Webseite.



### Komodo Edit downloaden

Sie erhalten einen betriebssystemspezifischen Installer, der wie üblich zu installieren ist.



### Der Installer für Linux

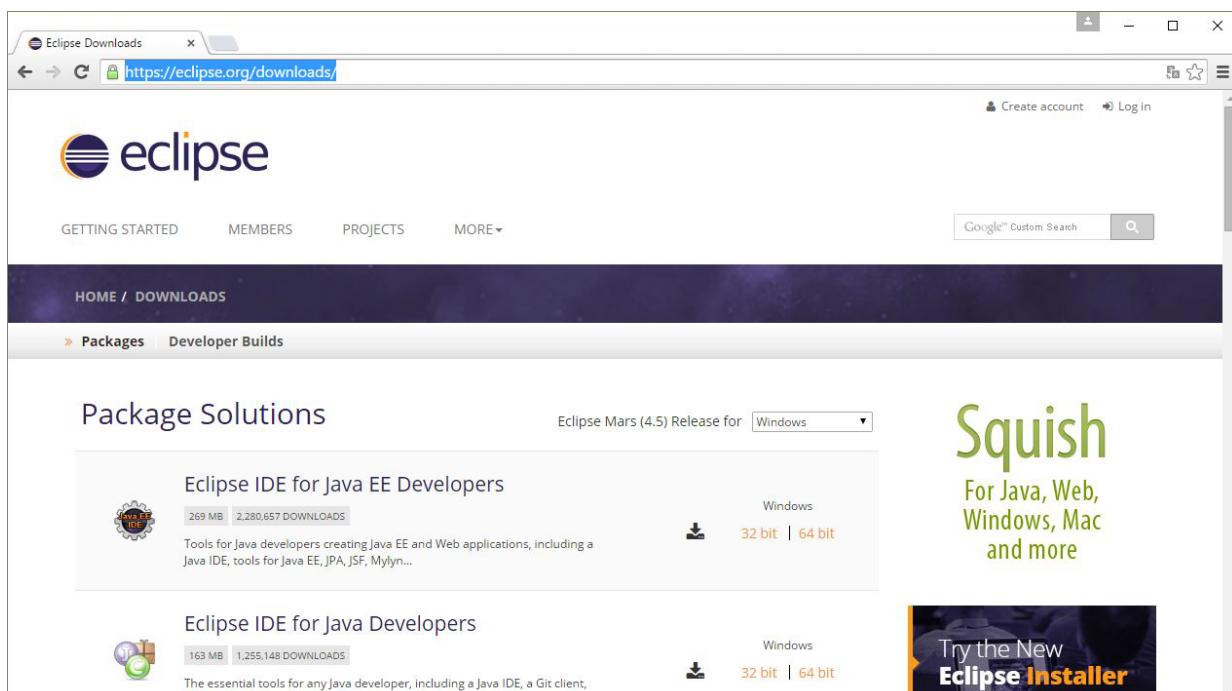
Komodo Edit unterstützt Sie mit folgenden Techniken bei der Programmierung:

- ✓ Syntax Highlighting
- ✓ Vielseitige Einstellungsmöglichkeiten, die Sie auch speichern können
- ✓ Konfigurierbare Tastenbelegung und Menüs
- ✓ Reguläre Ausdrücke beim Suchen und Ersetzen sowie ein RegEx-Tool zum Experimentieren und Fehler suchen
- ✓ Editieren von mehreren Dateien gleichzeitig
- ✓ Umfangreiche Undo- und Redo-Funktionen
- ✓ Ein umfangreiches Hilfesystem inkl. Anzeige aller definierten Tastenkürzel
- ✓ Ein integrierter Debugger
- ✓ Eine Projektverwaltung

- ✓ Eine Versionsverwaltung über CVS
- ✓ Automatisiertes Kommentieren und Auskommentieren
- ✓ Auf Wunsch läuft ein Perl-Interpreter im Hintergrund und markiert alle gefundenen Fehler in Echtzeit
- ✓ Expandieren und Kollabieren von Strukturen im Code
- ✓ Umwandlung von markiertem Text in Groß- oder Kleinschreibung
- ✓ Auto vervollständigung (deaktivierbar)
- ✓ Markieren von Blockstrukturen
- ✓ Vorschau der Programmausgabe bei CGI-Skripten in einem integrierten oder externen Browser

### Die Entwicklungsumgebung Eclipse mit EPIC (Perl Editor and IDE for Eclipse)

Noch leistungsfähiger als Komodo Edit ist die Eclipse-IDE. Eclipse ist OpenSource und Sie erhalten die IDE für mehrere Betriebssysteme (unter anderem Windows, Linux, Mac OS X) auf der Webseite <https://eclipse.org/downloads/>.



*Download von Eclipse*

### Die richtige Eclipse-Version wählen

Eclipse war ursprünglich für die Java-Entwicklung entwickelt, bietet aber mittlerweile Unterstützung für viele Programmiersprachen. Die Fülle der Möglichkeiten von Eclipse macht den Umgang mit der IDE nicht ganz einfach und erfordert eine gewisse Einarbeitung.

Es gibt verschiedene Versionen von Eclipse, aber da standardmäßig Perl noch nicht dabei ist, sollten Sie die Eclipse IDE for Java Developers oder die Eclipse IDE for Java EE Developers laden. Die Unterstützung für Perl wird anschließend hinzugefügt.

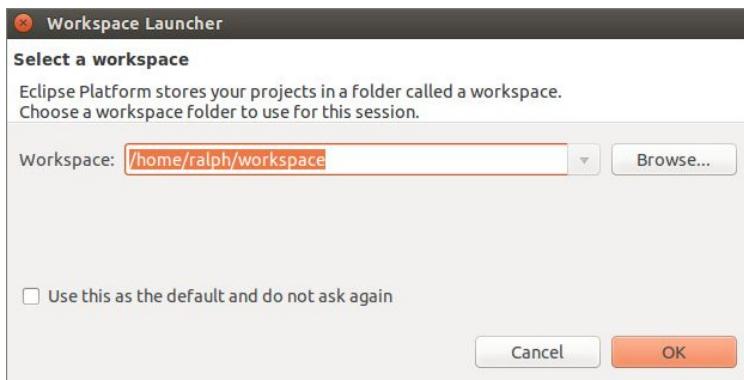
Die Installation von Eclipse ist einfach, Sie erhalten ein Archiv, das Sie bloß extrahieren müssen. Alternativ können Sie etwa auch unter Linux mit den speziellen Installationsmanagern Ihrer Distribution die IDE installieren.

## Die Bedeutung von Plug-ins in Eclipse

Die IDE ist nahezu beliebig erweiterbar und setzt dazu auf sogenannte Plug-ins. Ein Plug-in kann die einfachsten Funktionalitäten bis hin zu komplexesten Anwendungen ergänzen. Plug-ins definieren sogenannte Extension Points, was die formale Definition einer Schnittstelle bedeutet, über die eine Komponente erweitert werden kann. Plug-ins werden in einem eigenen Unterverzeichnis des Eclipse-Verzeichnisses gespeichert. Nach einem Neustart von Eclipse stehen die Plug-ins dann zur Verfügung.

## Workspace und Projekte

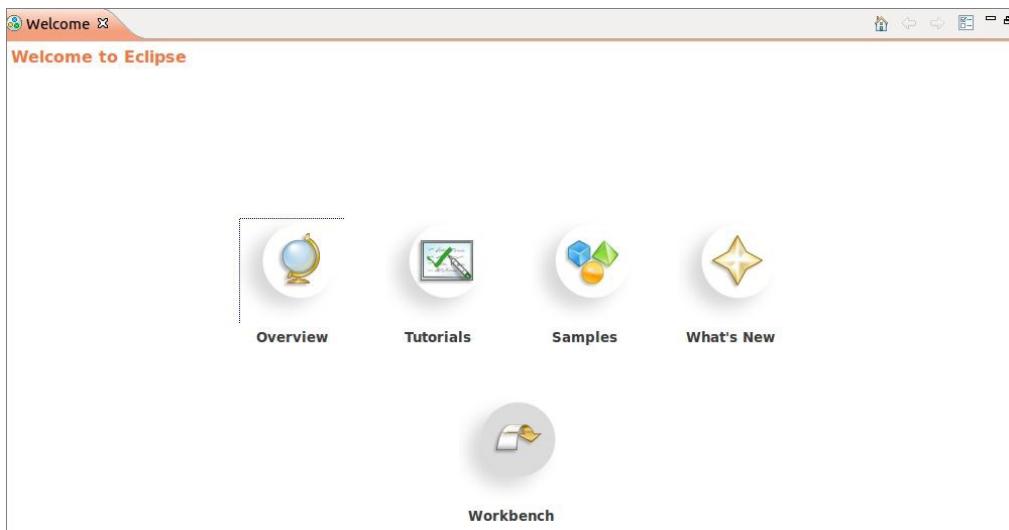
Wenn Sie Eclipse das erste Mal starten, müssen Sie meist beantworten, welchen Workspace Sie verwenden wollen.



*Workspace beim Start von Eclipse auswählen*

In Eclipse arbeiten Sie mit sogenannten **Projekten**, um zusammengehörende Ressourcen in einem verwalteten Verzeichnis zusammenzufassen. Diese Projekte selbst werden in einem Hauptverzeichnis mit Metainformationen zusammengefasst – dem sogenannten Workspace-Verzeichnis.

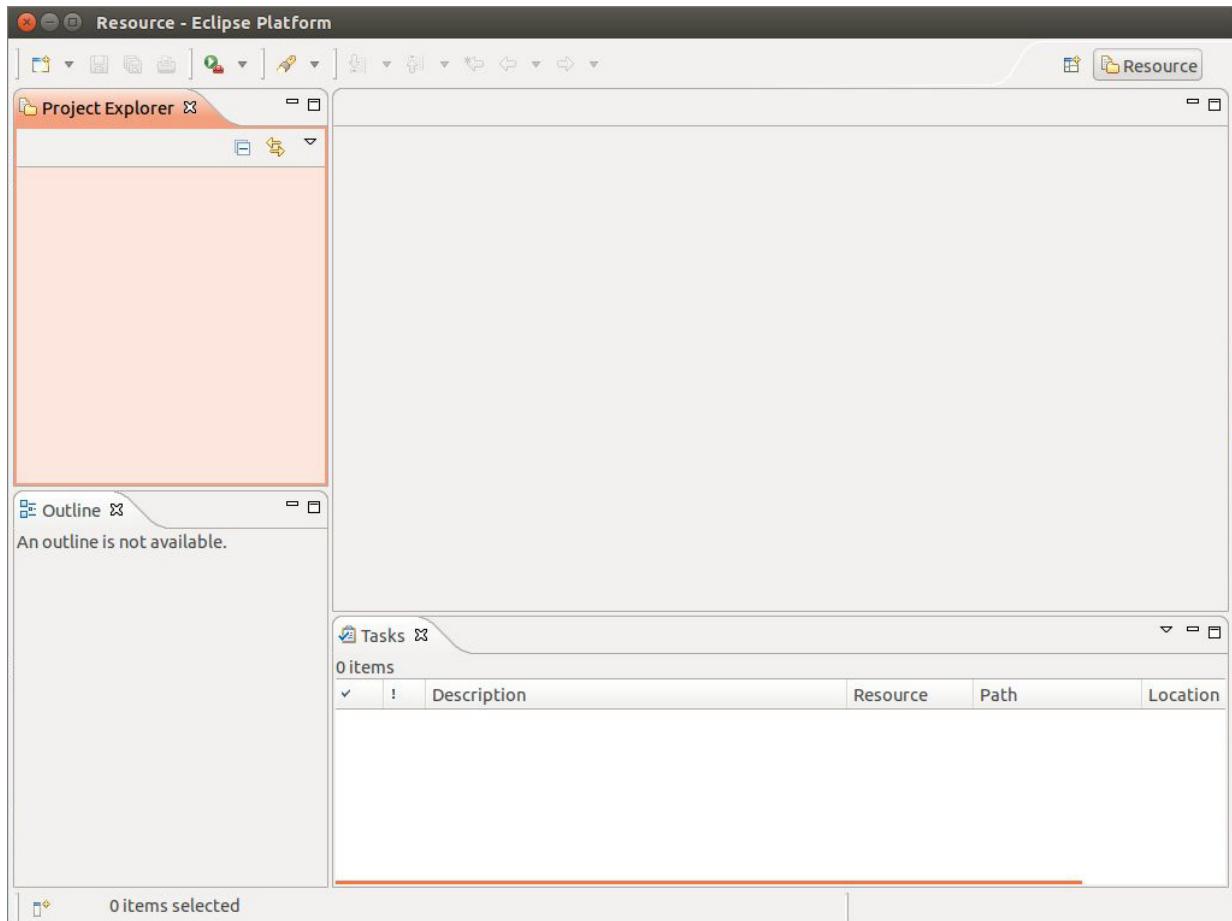
- ▶ Bestätigen Sie die Grundeinstellung.
- ▶ Wird nach dem Start der IDE ein Fenster mit Willkommensmeldungen eingeblendet, schließen Sie das Fenster.



*Willkommensfenster*

## Workbench und Perspektive

Das gesamte Eclipse-Fenster mit allen Fenstersegmenten, Menüs und den dahinter liegenden Funktionalitäten heißt **Workbench**. Die Anordnung gewisser Fenster und Menüs wird eine **Perspektive** genannt.

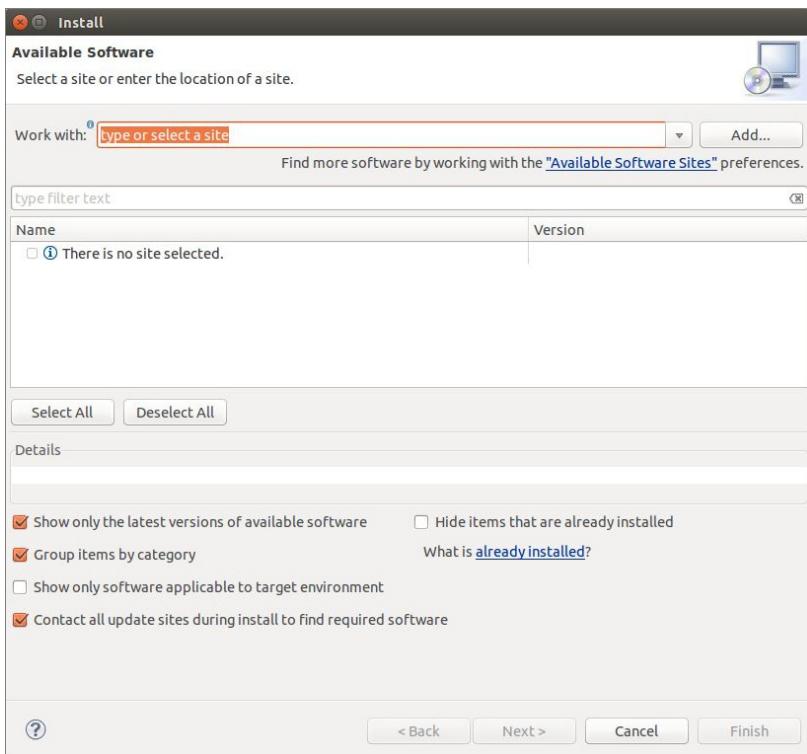


Die Workbench mit einer Perspektive

## Plug-in EPIC installieren

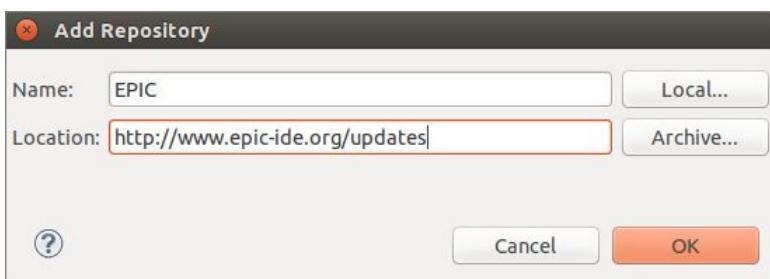
Das Installieren eines Plug-ins erfolgt am einfachsten mit dem Update-Manager von Eclipse.

- Verwenden Sie das Menü *Help - Install new Software*.



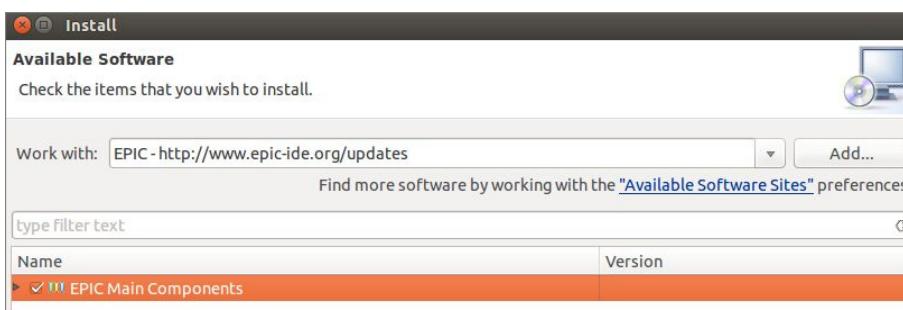
#### *Neue Software in Eclipse installieren*

- ▶ Klicken Sie auf die Schaltfläche *Add...*
- ▶ Im nachfolgenden Dialog tragen Sie einen Namen und den URL der Plug-in-Quelle ein.  
Hier soll das Plug-in EPIC installiert werden. Dazu verwenden Sie den URL <http://www.epic-ide.org/updates>. Der Name ist frei wählbar. Sinnvoll ist etwa EPIC.



#### *URL des Plug-in EPIC*

- ▶ Bestätigen Sie die Auswahl mit *OK*.
- ▶ Aktivieren Sie im folgenden Dialog das Kontrollfeld *EPIC Main Components*. Das Plug-in wird installiert.



- ▶ Bestätigen Sie mit *Next >*.

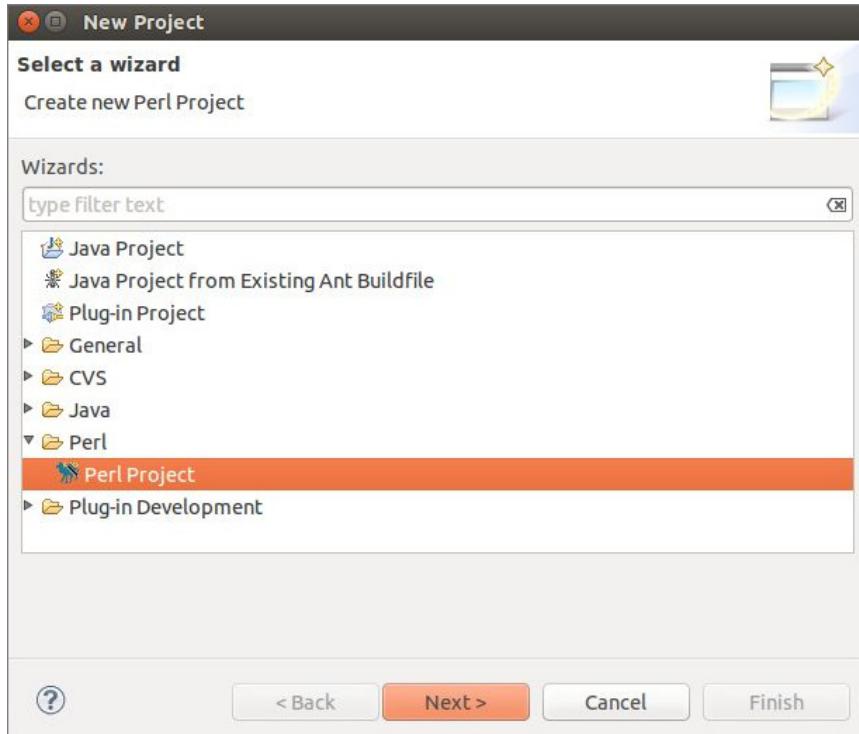
Der Rest der Installation wird vom Update Manager durchgeführt. Sie müssen den Fortschritt beobachten und einige wenige Fragen (etwa nach der Lizenz oder ob wirklich installiert werden soll) bestätigen. Anschließend müssen Sie Eclipse neu starten. Das Plug-in stellt Ihnen in Eclipse speziell auf Perl zugeschnittene Features zur Verfügung, z.B.

- ✓ Syntax Highlighting
- ✓ On-the-fly Syntax-Überprüfung
- ✓ Content assist
- ✓ Quick reference
- ✓ Outline Ansicht der benutzten Module und Subroutinen
- ✓ Source formatter
- ✓ HTML export
- ✓ Perldoc support
- ✓ Templating support

### Projekt erstellen

In Eclipse erstellen Sie in der Regel immer zuerst ein Projekt.

- Klicken Sie auf *File - New - Perl - Perl Project*.  
Eclipse wird Ihnen eine Perspektive einstellen, die auf die Bedürfnisse von Perl zugeschnitten ist.



### Perl-Projekt erzeugen

- Vergeben Sie einen Namen für das Projekt.
- Fügen Sie über *File - New - Other - Perl - Perl File* dem Projekt eine Perl-Datei hinzu.



| \$                                        | B            | D                                  |
|-------------------------------------------|--------------|------------------------------------|
| \$\_                                      | 72, 106      | Data : : Dumper 165                |
| @                                         |              | Dateien lesen, vollständig 92      |
| @ARGV                                     | 106          | Dateien lesen, zeichenweise 96, 97 |
| @EXPORT                                   | 166          | Dateien lesen, zeilenweise 91      |
| @EXPORT_OK                                | 166          | Dateien öffnen 88                  |
| @INC                                      | 161          | Dateien schließen 91               |
| <                                         |              | Dateien sperren 93                 |
| <>                                        | 105          | Dateien umbenennen 100             |
| A                                         |              | Dateien, Binärmodus 98             |
| Abfrageergebnis                           | 201          | Dateien, Eigenschaften 102         |
| Abfragehandle                             | 198          | Dateien, Fehler 90                 |
| Abfragen                                  | 197          | Dateien, schreiben in 92           |
| ActivePerl                                | 5, 13        | Dateien, Schreibweise 88           |
| ActiveState                               | 190, 217     | Dateien, Statusinformationen 95    |
| Aggregationen                             | 167          | Dateien, Zugriffsmodi 89           |
| Alias, lexikalisch                        | 28           | Datei-Handle 88, 91                |
| Anweisungen                               | 17           | Datei-Handle, unbenanntes 90       |
| Anweisungsblöcke                          | 17, 43       | Dateiprüfoperatoren 95             |
| Apache                                    | 221          | Dateisystem 88                     |
| Apache-Webserver, Installation            | 143          | Dateisystem, Funktionen 100        |
| Argumente                                 | 76, 79       | Dateizeiger 97                     |
| Array aufteilen                           | 61           | Daten, Ausgabe 20                  |
| Array ausgeben                            | 61           | Datenbanken erstellen 192          |
| Array, Elemente                           | 62           | Datenbankhandle 195                |
| Array, Funktionen                         | 64           | Datenbankmanagementsysteme 186     |
| Array, letztes Element                    | 62           | Datenbankmodul 194                 |
| Array, mehrdimensional                    | 65           | Datenbankserver, Verbindung 194    |
| Array, sortieren                          | 65           | Datenbanktreiber 187               |
| Array, Tastatur                           | 60           | Datenfelder 58                     |
| Arrays                                    | 59           | Datenfelder, Arten 58              |
| Arrays, Eingabe                           | 60           | Datenfelder, assoziative 67        |
| Arrays, Zeichenketten                     | 60           | Datenfelder, Hashes 67             |
| Array-Scheiben                            | 63           | Datenfelder, initialisieren 58     |
| Assoziationen                             | 167          | Datenquelle 195                    |
| Attribute                                 | 167          | Datentypen 32, 34                  |
| Ausdrücke                                 | 35           | Datentypen, numerische 32          |
| Ausdrücke, reguläre                       | 118          | Datentypkonvertierung 34           |
| Ausführungskontext                        | 81           | Datum ausgeben 135                 |
| Ausführungsrechte, CGI-Programme          | 146          | Datum, Funktionen 85               |
| Ausgabe, lange Texte                      | 21           | Datumsberechnungen 135             |
| Ausschlusszeichen                         | 121          | Datumsfunktionen 134               |
| Auswahl, einseitige                       | 44           | DBD 187                            |
| Auswahl, mehrstufig                       | 48           | DBI 186                            |
| Auswahl, zweiseitige                      | 45, 47       | DBI-Modul 194                      |
| Auswahlanweisung                          | 43           | DBI-Modul, Installation 190        |
| awk                                       | 9            | DBI-Objekt 194                     |
| B                                         |              | Dekrementoperator 36               |
| Basic                                     | 9            | delete 70                          |
| Bedingungen                               | 42           | Dereferenzierung 83                |
| Bedingungsauswahl                         | 43           | Deskriptor 88                      |
| Bezeichner                                | 19           | DESTROY 170                        |
| Binärdistribution                         | 216          | Deutsche Umlaute 25                |
| Binärmodus                                | 98           |                                    |
| binmode                                   | 25, 98       |                                    |
| Bitbearbeitungsoperatoren                 | 37           |                                    |
| bless                                     | 169          |                                    |
| Bytecode                                  | 9            |                                    |
| C                                         |              |                                    |
| CGI                                       | 142          |                                    |
| <i>CGI.pm</i>                             | 172          |                                    |
| CGI : : Fast                              | 165          |                                    |
| <i>cgi-bin</i> , Verzeichnis              | 143          |                                    |
| CGI-Modul                                 | 172          |                                    |
| CGI-Objekt                                | 172          |                                    |
| CGI-Programm, Übertragungs-methode        | 150          |                                    |
| CGI-Programme                             | 8, 142       |                                    |
| CGI-Programme, Ausführungs-rechte         | 146          |                                    |
| CGI-Programme, Datenübergabe              | 148          |                                    |
| CGI-Programme, Perl einrichten            | 146          |                                    |
| CGI-Programme, Umgebungs-variablen        | 155          |                                    |
| CGI-Programme, URL-Codierung              | 152          |                                    |
| CGI-Programmierung                        | 172          |                                    |
| CGI-Umgebung                              | 10           |                                    |
| chdir                                     | 100, 102     |                                    |
| chmod                                     | 100, 101     |                                    |
| close                                     | 91           |                                    |
| closedir                                  | 103          |                                    |
| Common Gateway Interface (CGI)            | 142          |                                    |
| Comprehensive Perl Archive Network (CPAN) | 14, 165, 187 |                                    |
| Config                                    | 165          |                                    |
| connect                                   | 194          |                                    |
| constant, Pragma                          | 166          |                                    |
| Content-Type                              | 146, 173     |                                    |
| cookie                                    | 181          |                                    |
| Cookies lesen                             | 182          |                                    |
| Cookies löschen                           | 182          |                                    |
| Cookies setzen                            | 181          |                                    |
| cp437                                     | 25           |                                    |
| CPAN                                      | 14, 165, 187 |                                    |
| CPAN-Module                               | 158          |                                    |
| CPAN-Module, installieren                 | 158          |                                    |
| Cygwin                                    | 14, 216      |                                    |

|                              |          |                             |                 |                             |          |
|------------------------------|----------|-----------------------------|-----------------|-----------------------------|----------|
| diagnostics, Pragma          | 166      | <b>G</b>                    | integer, Pragma | 166                         |          |
| Diamond-Operator             | 105      | Ganzahlarithmetik           | 32, 166         | Integers                    | 32       |
| DirHandle                    | 165      | Gästebuch                   | 199             | Interpretersprache          | 9        |
| Dokumentation                | 12       | Geltungsbereich             | 28              | Intervalle                  | 59       |
| do-Methode                   | 199      | Generalisieren              | 167             | Intervalloperator           | 59       |
| do-until-Schleife            | 53       | gensym                      | 90              | IO                          | 165      |
| do-while-Schleife            | 53       | getc                        | 97              | <b>J</b>                    |          |
| <b>E</b>                     |          | GET-Übertragungsmethode     | 150             | join                        | 64       |
| each                         | 71       | Gleitkommaarithmetik        | 32              | <b>K</b>                    |          |
| Eclipse                      | 228      | Greedy                      | 122, 127, 128   | keys                        | 69       |
| Editoren                     | 16, 225  | Greenwich-Zeitzone          | 134             | Klassen                     | 167      |
| Eigenschaften                | 167      | grep                        | 118             | Kommentare                  | 17       |
| E-Mail                       | 208      | gzip                        | 219             | Kommentarzeichen            | 18       |
| E-Mail-Adresse               | 209      | <b>H</b>                    |                 | Komodo Edit                 | 227      |
| Encode                       | 165      | Hash                        | 67              | Komplexe Zahlen             | 165      |
| Encoding                     | 25       | Hash, Eingabe               | 67              | Kompositionen               | 167      |
| encoding(cp437)              | 25       | Hash, Elemente              | 68              | Konstanten                  | 172      |
| Entwicklungstools            | 225      | Hash, Funktionen            | 69              | Kontrollstrukturen          | 42       |
| Env                          | 165      | Hash, sortieren             | 70              | Konvertierung, Datentyp     | 34       |
| eof                          | 97       | Hash::Util                  | 165             | <b>L</b>                    |          |
| EPIC                         | 228, 231 | HEAD-Bereich                | 178             | Larry Wall                  | 9        |
| Ergebnisdateien              | 6        | Header                      | 146, 173        | Layoutregeln                | 18       |
| Ersetzen, reguläre Ausdrücke | 128      | Headerzeilen                | 209             | lc                          | 115      |
| Ersetzen, zeichenweise       | 131      | Heredoc                     | 33              | Lernziele                   | 4        |
| Escape-Zeichen               | 21, 118  | Heredoc-Syntax              | 21              | Lexikalische Variablen      | 28       |
| Escape-Zeichenfolge          | 88       | Hexadezimalzahlen           | 32              | lib, Pragma                 | 166      |
| exists                       | 71       | Hilfe                       | 12              | List::Util                  | 165      |
| Exporter-Modul               | 166      | HTML-Elemente, CGI-Modul    | 177             | Listen                      | 59, 65   |
| Exportieren                  | 166      | HTML-Formulare              | 148             | http                        | 221      |
| <b>F</b>                     |          | HTML-Tags                   | 177             | HTTP                        | 142      |
| Fast-CGI                     | 165      | http                        | 221             | localhost                   | 145      |
| fatalstToBrowser             | 176      | HTTP                        | 142             | localtime                   | 134      |
| Felder                       | 24, 58   | HTTP-Anfrage                | 151             | Location-Header             | 173      |
| File                         | 165      | Hypertext Transfer Protocol | 142             | Lokalisierungen             | 165      |
| Flags, Formatierung          | 111      | <b>I</b>                    |                 | Loop Back                   | 145      |
| Floatingpoint                | 33       | if-Anweisung                | 44              | <b>M</b>                    |          |
| flock                        | 93       | if-else-Anweisung           | 45              | Mail::Sender                | 208      |
| Folding                      | 16       | if-elsif-Anweisung          | 48              | Mail::Sendmail              | 208, 211 |
| foreach-Schleife             | 71       | import_names                | 174             | Mailversand, Mail::Sendmail | 211      |
| Formatierung, Zeichenkette   | 110      | Importieren                 | 166             | Mailversand, Unix           | 209      |
| Formatierungsanweisungen     | 110      | index                       | 113, 118        | Mailversand, Windows        | 211      |
| form-Tag                     | 148      | Indexwert                   | 62              | main, Namensraum            | 162      |
| Formulardaten                | 151      | Indexwert, höchster         | 62              | Makro                       | 9        |
| Formulardaten, CGI-Modul     | 174      | Indexwerte, negative        | 62              | Man pages                   | 12       |
| Formulare, CGI-Modul         | 179      | Inkrementoperator           | 36              | MD5                         | 165      |
| Fragezeichen-Operator        | 47       | Installation                | 10              | Methoden                    | 167, 172 |
| FTP                          | 165      | Installation:Perl           | 216             | MIME-Typ                    | 146      |
| Funktionen                   | 76       | Installation testen         | 11              | mkdir                       | 100, 102 |
| Funktionen:trigonometrische  | 165      | Installation, Windows       | 217             | Moduldokumentationen        | 164      |
| Funktionen, vordefinierte    | 84       | Instanz                     | 167             |                             |          |

|                            |          |                                    |                              |          |
|----------------------------|----------|------------------------------------|------------------------------|----------|
| Module                     | 158      | <b>P</b>                           | Referenz                     | 68       |
| Module, einbinden          | 159      | Packages                           | Referenzen                   | 82       |
| Verzeichnisse              | 161      | Pakete                             | Referenzen auflösen          | 83       |
| Monatsnamen                | 136      | param                              | Referenzen erstellen         | 83       |
| Musteranker                | 124      | Parameter                          | Regular Expressions          | 118      |
| Mustererkennung            | 119      | Parameterübergabe                  | Reguläre Ausdrücke           | 118      |
| my                         | 28       | Path, Perl                         | Reguläre Ausdrücke, Ersetzen | 128      |
| MySQL                      | 186      | Pattern                            | Reguläre Ausdrücke, Regeln   | 128      |
| MySQL, Installation        | 187      | Perl, Binärdistributionen          | rename                       | 100, 101 |
| MySQL-Monitor              | 189      | Perl, path                         | require                      | 159      |
| <b>N</b>                   |          | Perl, Quellcode                    | return                       | 79       |
| Namensraum                 | 174      | Perl Editor and IDE for Eclipse    | reverse                      | 64       |
| Namensräume                | 162      | Siehe EPIC                         | rewinddir                    | 103, 105 |
| Namespace                  | 174      | Perl installieren                  | Rhombus-Operators            | 105      |
| Net : : FTP                | 165      | perldoc                            | rmdir                        | 100, 102 |
| Net : : POP3               | 165      | Perl-Funktionen                    | Rückgabewert                 | 79       |
| Net : : SMTP               | 165, 208 | Perl-Interpreter, Optionen         | <b>S</b>                     |          |
| NetBSD                     | 217      | perlMyAdmin                        | Schleife                     | 50       |
| new, Konstruktor           | 168      | MySQL                              | Schleifen, bedingte          | 50       |
| no, Pragma                 | 166      | Perspektive                        | Schleifen, fußgesteuerte     | 53       |
| Non-Greedy-Verhalten       | 127      | Eclipse                            | Schleifen, kopfgesteuerte    | 51       |
| Notepad                    | 10, 16   | Pfad                               | Schleifenkörper              | 51       |
| Notepad++                  | 225      | Pfadangabe                         | Schleifensteuerung           | 50, 55   |
| Numbers                    | 32       | phpMyAdmin                         | Schlüsselwert                | 68       |
| <b>O</b>                   |          | Platzhalter                        | Schlüsselwerte, unbekannte   | 69       |
| Objektvariable             | 172      | Polymorphie                        | sed                          | 9        |
| ODBC                       | 186      | pop                                | seek                         | 97       |
| Oktalzahlen                | 32       | PostgreSQL                         | seekdir                      | 103, 105 |
| open                       | 88       | POST-Übertragungsmethode           | SELECT-Abfrage               | 201      |
| opendir                    | 103      | ppm                                | sendmail                     | 208, 224 |
| OpenSource                 | 9        | Practical Extraction and Reporting | Server                       | 221      |
| Operanden                  | 35       | Language (Perl)                    | She-Bang                     | 18       |
| Operator, ternärer         | 47       | Pragmas                            | Shebang-Zeile                | 146      |
| Operatoren                 | 35       | print                              | shift                        | 64, 169  |
| Operatoren, arithmetische  | 35       | print-Anweisung                    | Skripte                      | 9        |
| Operatoren, Bitbearbeitung | 37       | printf                             | Skriptsprache                | 9        |
| Operatoren, logische       | 37, 42   | Programmsteuerung                  | SMTP                         | 208      |
| Operatoren, Rangfolge      | 39, 43   | push                               | SMTP-Server                  | 208      |
| Operatoren, Vergleich      | 36       | <b>Q</b>                           | Software, verwendete         | 5        |
| Operatoren, Zeichenketten  | 38       | Quantoren                          | Solaris                      | 217      |
| Operatoren, Zuweisung      | 38       | Quellcode-Distribution             | Sonderzeichen                | 25       |
| Optionen                   | 19       | Query                              | sort                         | 64       |
| Oracle                     | 186      | Query-String                       | Source Code Distribution     | 219      |
| Ordner auslesen            | 103      | quote-Methode                      | Spezialisieren               | 167      |
| Ordner erstellen           | 102      | qw                                 | splice                       | 64       |
| Ordner öffnen              | 103      | qw-Funktion                        | split                        | 64       |
| Ordner schließen           | 103      | <b>R</b>                           | sprintf                      | 110      |
| Ordner wechseln            | 102      | Rangfolge                          | SQL                          | 186      |
| Ordner-Handle              | 103      | read                               | Standardmodule               | 158, 165 |
| Ordnerzeiger               | 105      | readdir                            | start_html-Methode           | 178      |
| our                        | 28       | redirect                           | startform-Methode            | 179      |
|                            |          |                                    | stat                         | 100, 102 |
|                            |          |                                    | Statisch gebundene Variablen | 28       |

|                                            |          |                                          |          |                                       |             |
|--------------------------------------------|----------|------------------------------------------|----------|---------------------------------------|-------------|
| STDERR                                     | 91       | unless-Anweisung                         | 47       | Wiederholungsschleifen                | 50          |
| STDIN                                      | 91       | unlink                                   | 100, 101 | Wikipedia                             | 14          |
| STDOUT                                     | 91       | unshift                                  | 64       | Wochentag                             | 136         |
| Strawberry Perl                            | 5, 217   | Unterprogramme                           | 76       | Workbench:Eclipse                     | 230         |
| strict, Pragma                             | 166      | Unterprogramme, Argumente                | 79       | Workspace:Eclipse                     | 229         |
| String                                     | 33       | Unterprogramme, Arrays                   | 82       | World Wide Web                        | 142         |
| Subroutinen                                | 76       | Unterprogramme, Ausführungs-<br>kontext  | 81       | WWW                                   | 8, 142      |
| Substitution                               | 128      | Unterprogramme, Rückgabewert             | 79       | <b>X</b>                              |             |
| Substitutionsausdrücke                     | 130      | Unterprogramme, Variablen                | 78       | XAMPP                                 | 5, 143, 221 |
| substr                                     | 113      | Unterprogramme, Vorteile                 | 80       | XAMPP Control Panel:MySQL             | 187         |
| Suchmuster                                 | 118      | until-Anweisung                          | 52       | <b>Z</b>                              |             |
| Suchmuster definieren                      | 119      | Update-Manager Eclipse                   | 230      | Zahlen                                | 32          |
| Suchmuster planen                          | 118      | URL                                      | 150      | Zahlen, komplexe                      | 165         |
| Suchmuster verwenden                       | 119      | URL-Codierung                            | 152      | Zeichenbereiche                       | 120         |
| Suchmuster, Alternativen                   | 125      | use                                      | 159      | Zeichenkette beschneiden              | 113         |
| Suchmuster, Gruppieren                     | 127      | use CGI::Carp<br>qw( fatsalsToBrowser ); | 176      | Zeichenkette durchsuchen              | 113         |
| Suchmuster, Platzhalter                    | 119      | use integer                              | 32       | Zeichenkette, zerlegen                | 64          |
| Suchmuster, Übereinstimmungen              | 131      | use lib                                  | 161      | Zeichenketten                         | 33          |
| Suchmuster, Übereinstimmungen<br>speichern | 125      | use strict;                              | 26       | Zeichenketten modifizieren            | 115         |
| Suchmuster, Verankerung                    | 124      | <b>V</b>                                 |          | Zeichenketten, ausgeben               | 20          |
| Suchmuster, Wiederholungen                 | 122      | values                                   | 69       | Zeichenketten, formatiert<br>ausgeben | 110         |
| Suchmuster, Zeichenklassen                 | 120      | Variable, skalare                        | 58       | Zeichenketten, Schreibweise           |             |
| Suchpfad, Perl                             | 218      | Variablen                                | 24       | ändern                                | 115         |
| Symbol-Modul                               | 90       | Variablen, dynamisch gebundene           | 28       | Zeichenkettenoperatoren               | 38          |
| Syntax-Highlighting                        | 16       | Variablen, lexikalische                  | 28       | Zeichenklassen                        | 120         |
| <b>T</b>                                   |          | Variablen, statisch gebundene            | 28       | Zeige-Operator                        | 68          |
| Tabellen erstellen                         | 192      | Variablen, Ausgeben                      | 20       | Zeilenvorschub                        | 21          |
| tell                                       | 97       | Variablen, Geltungsbereich               | 28       | Zeit ausgeben                         | 135         |
| telldir                                    | 103, 105 | Variablen, globale                       | 28       | Zeitberechnungen                      | 135         |
| Textdateien lesen                          | 91       | Variablen, lokale                        | 29       | Zeitfunktionen                        | 134         |
| time                                       | 134      | Variablen, Typen                         | 24       | Zielgruppe                            | 4           |
| timegm                                     | 138      | Variablen, undefinierte                  | 27       | Zugriffsrechte                        | 101         |
| timelocal                                  | 138      | Verankerung                              | 124      | Zuweisungsoperatoren                  | 38          |
| Triadischer Operator                       | 47       | Vererbung                                | 167      |                                       |             |
| Trigonometrische Funktionen                | 165      | Vergleich, numerischer                   | 36       |                                       |             |
| Typisierung, automatische                  | 34       | Vergleich, Zeichenketten                 | 37       |                                       |             |
| Typkonvertierung                           | 26, 34   | Vergleichsoperator, ternärer             | 47       |                                       |             |
| <b>U</b>                                   |          | Vergleichsoperatoren                     | 36       |                                       |             |
| Übersetzungsfunktion                       | 131      | Verzeichnistrenner:Module                | 162      |                                       |             |
| Übertragungsmethode                        | 150      | Verzeichniszugriffe                      | 103      |                                       |             |
| uc                                         | 115      | vi                                       | 10       |                                       |             |
| Umgebungsdaten                             | 175      | Vorkenntnisse, empfohlene                | 4        |                                       |             |
| Umgebungsvariablen                         | 155      | <b>W</b>                                 |          |                                       |             |
| Umlaute, deutsche                          | 25       | wantarray                                | 81       |                                       |             |
| Umleitung                                  | 173      | Warning-Modus                            | 18       |                                       |             |
| undef                                      | 28       | Webserver                                | 142      |                                       |             |
| Unix                                       | 8        | Webservers Apache                        | 143      |                                       |             |
| Unix-Shell                                 | 9        | Wertzuweisung                            | 25       |                                       |             |
| Unix-Zeitangabe                            | 134      | wget                                     | 219      |                                       |             |
| Unix-Zeitangabe, Datum<br>umwandeln        | 138      | while-Anweisung                          | 51       |                                       |             |
|                                            |          | Wiederholung, zählergesteuerte           | 54       |                                       |             |

---

# Impressum

Matchcode: PRL5

Autoren: Ralph Steyer, Peter Teich

Redaktion: Andrea Weikert

Produziert im HERDT-Digitaldruck

2. Ausgabe, Juli 2016

HERDT-Verlag für Bildungsmedien GmbH  
Am Kümmerling 21-25  
55294 Bodenheim  
Internet: [www.herdt.com](http://www.herdt.com)  
E-Mail: [info@herdt.com](mailto:info@herdt.com)

© HERDT-Verlag für Bildungsmedien GmbH, Bodenheim

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlags reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Dieses Buch wurde mit großer Sorgfalt erstellt und geprüft. Trotzdem können Fehler nicht vollkommen ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Wenn nicht explizit an anderer Stelle des Werkes aufgeführt, liegen die Copyrights an allen Screenshots beim HERDT-Verlag. Sollte es trotz intensiver Recherche nicht gelungen sein, alle weiteren Rechteinhaber der verwendeten Quellen und Abbildungen zu finden, bitten wir um kurze Nachricht an die Redaktion.

Die in diesem Buch und in den abgebildeten bzw. zum Download angebotenen Dateien genannten Personen und Organisationen, Adress- und Telekommunikationsangaben, Bankverbindungen etc. sind frei erfunden. Eventuelle Übereinstimmungen oder Ähnlichkeiten sind unbeabsichtigt und rein zufällig.

Die Bildungsmedien des HERDT-Verlags enthalten Verweise auf Webseiten Dritter. Diese Webseiten unterliegen der Haftung der jeweiligen Betreiber, wir haben keinerlei Einfluss auf die Gestaltung und die Inhalte dieser Webseiten. Bei der Bucherstellung haben wir die fremden Inhalte daraufhin überprüft, ob etwaige Rechtsverstöße bestehen. Zu diesem Zeitpunkt waren keine Rechtsverstöße ersichtlich. Wir werden bei Kenntnis von Rechtsverstößen jedoch umgehend die entsprechenden Internetadressen aus dem Buch entfernen.

Die in den Bildungsmedien des HERDT-Verlags vorhandenen Internetadressen, Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen waren zum Zeitpunkt der Erstellung der jeweiligen Produkte aktuell und gültig. Sollten Sie die Webseiten nicht mehr unter den angegebenen Adressen finden, sind diese eventuell inzwischen komplett aus dem Internet genommen worden oder unter einer neuen Adresse zu finden. Sollten im vorliegenden Produkt vorhandene Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen nicht mehr der beschriebenen Software entsprechen, hat der Hersteller der jeweiligen Software nach Drucklegung Änderungen vorgenommen oder vorhandene Funktionen geändert oder entfernt.