

Perl-Training

Agenda

- Einführung
- strict
- Subroutinen
- Exkurs: Referenzen
- Chained if's versus Dispatcher
- map
- grep
- sprintf
- Exporter
- Kleinere Optimierungen (Performance, Wartbarkeit)
- viel, viel mehr...

Was ist strict?

- strict ist ein Pragma – „use“
- Lexikalischer Scope
- Variablen müssen deklariert werden

- Erleichtert Fehlersuche
- Schont den Speicher

strict

```
#!/usr/bin/perl
```

```
$line = "Hello";
```

```
$zahl = 15;
```

```
if(($zahl*10) < 200){
```

```
    $test = 34;
```

```
    $line = "Test";
```

```
}
```

```
$zahl = 444.23 if($zahl==150);
```

```
print "$zahl: $line wurde eingegeben\n";
```

strict

```

C:\Dokumente und Einstellungen\Renee>perl
$line = "Hello";
$zahl  = 15;

if(<($zahl*10) < 200)<
    $test = 34;
    $line = "Test";
}

$zahl = 444.23 if($zahl==150);
print "$zahl: $lin wurde eingegeben\n";

^D
15: wurde eingegeben

```

strict

```
#!/usr/bin/perl
```

```
use strict;
```

```
$line = "Hello";
```

```
$zahl = 15;
```

```
if(($zahl*10) < 200){
```

```
    $test = 34;
```

```
    $line = "Test";
```

```
}
```

```
$zahl = 444.23 if($zahl==150);
```

```
print "$zahl: $line wurde eingegeben\n";
```

strict

```
C:\ Eingabeaufforderung
C:\Dokumente und Einstellungen\Renee>perl
use strict;

$line = "Hello";
$zahl = 15;

if(<<$zahl*10> < 200)<
    $test = 34;
    $line = "Test";
}

$zahl = 444.23 if($zahl==150);
print "$zahl: $lin wurde eingegeben\n";
^D
Global symbol "$line" requires explicit package name at - line 3.
Global symbol "$zahl" requires explicit package name at - line 4.
Global symbol "$zahl" requires explicit package name at - line 6.
Global symbol "$test" requires explicit package name at - line 7.
Global symbol "$line" requires explicit package name at - line 8.
Global symbol "$zahl" requires explicit package name at - line 11.
Global symbol "$zahl" requires explicit package name at - line 11.
Global symbol "$zahl" requires explicit package name at - line 12.
Global symbol "$lin" requires explicit package name at - line 12.
Execution of - aborted due to compilation errors.
```

strict

```
#!/usr/bin/perl
```

```
use strict;
```

```
my $line = "Hello";
```

```
my $zahl = 15;
```

```
if(($zahl*10) < 200){  
    my $test = 34;  
    $line = "Test";  
}
```

```
$zahl = 444.23 if($zahl==150);
```

```
print "$zahl: $line wurde eingegeben\n";
```

strict

```
my @array;  
{  
    use strict;  
    my $test = 'Test';  
    print "$test\n";  
}  
  
$hallo = "Hallo";  
print "$hallo\n";
```

strict

```
C:\Dokumente und Einstellungen\Renee>perl
my @array;
{ use strict; my $test = 'Test'; print "$test\n";}
$hallo = "Hallo";
print "$hallo\n";
^D
Test
Hallo

C:\Dokumente und Einstellungen\Renee>
```

```
C:\Dokumente und Einstellungen\Renee>perl -Mstrict
my @array;
{ use strict; my $test = 'Test'; print "$test\n";}
$hallo = "Hallo";
print "$hallo\n";
^D
Global symbol "$hallo" requires explicit package name at - line 3.
Global symbol "$hallo" requires explicit package name at - line 4.
Execution of - aborted due to compilation errors.

C:\Dokumente und Einstellungen\Renee>
```

strict

```
# some hashes for nicer output
my %anrede = (1 => 'Herr', 2 => 'Frau');
my %titel   = (1 => 'Dr.', 2 => 'Prof.', 3 => 'Prof. Dr.');
```

```
# replace parameters
$template->param(
    name          => $loc_felder{name}->getValue,
    vorname       => $loc_felder{vorname}->getValue,
    hausnum       => $loc_felder{hausnum}->getValue,
    abonnement    => $loc_felder{abonnement}->getValue,
    anrede        => $anrede{ $loc_felder{anrede}->getValue },
    titel         => $titel{ $loc_felder{titel}->getValue },
    strasse       => $loc_felder{strasse}->getValue,
    plz          => $loc_felder{plz}->getValue,
    ort           => $loc_felder{ort}->getValue,
```

Subroutinen-Parameterliste

```
my @array = ( 1..10 );  
my $returnvalue = subroutine( @array, 23.55 );  
  
sub subroutine{  
    my (@array, $value) = @_;  
    print $value;  
}
```

- Alle Parameter werden in eine Liste zusammengefasst

Subroutinen-Parameterliste

- Mehrere Arrays können so nicht übergeben werden...

```
C:\Dokumente und Einstellungen\Renee>perl -Mwarnings
my @array = ( 1..10 );
my $returnvalue = subroutine( @array, 23.55 );

sub subroutine(
    my ( @array, $value ) = @_;
    print $value;
}
^D
Use of uninitialized value in print at - line 6.
```

Subroutinen – Parameterliste – Die Lösung

- Bei mehreren Parameter wie Arrays und/oder Hashes → Referenzen

```
my @array = ( 1..10 );  
my $returnvalue = subroutine( \@array, 23.55 );  
  
sub subroutine{  
    my ($arrayref, $value) = @_;  
    print $value;  
}
```

Subroutinen – Parameterliste – Die Lösung

Auswählen Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee>perl -Mwarnings
my @array = ( 1..10 );
my $returnvalue = subroutine( \@array, 23.55 );

sub subroutine{
    my ($arrayref, $value) = @_;
    print $value;
}
^D
23.55
C:\Dokumente und Einstellungen\Renee>
```

Exkurs: Referenzen

- Für komplexe Datenstrukturen wichtig
- Werden erzeugt mit:
 - \ ...
 - [] -- Arrayreferenz
 - {} -- Hashreferenz
- Ergebnis: ein Skalar
- Nützliches Modul: Data::Dumper

Exkurs: Referenzen

```
my @array = ( 1..10 );  
my $arref = \@array;  
my $arref2 = [1,2,3,4];  
  
my %hash = ( Test => 1 );  
my $hashref = \%hash;  
my $hashref2 = { Test => 1 };  
  
my $scalarref = \'Test';  
my $funcref = \&subroutine;
```

Exkurs: Referenzen

- Wofür brauchen wir Referenzen?
 - „mehrdimensionale“ Arrays
 - Mehrere Werte für einen Schlüssel
 - Dispatch-Tables
 - Teilweise Parameterübergabe an Subroutinen
 - ...
- Wie wird dereferenziert?
 - `%{ $hashref }`
 - `@{ $arrayref }`
 - `$$skalarref`
 - `$ref->[$index], $ref->{ $key }`

Exkurs: Referenzen

```
my @array = ( 1..10 );  
my $arref = \@array;  
  
my @tmp = @{ $arref };  
  
print $arref->[2];
```

Exkurs: Referenzen

```
use Data::Dumper;
my $komplexe_datenstruktur = [
    {
        key1 => 'value1',
        key2 => 'value2',
    },
    {
        key1 => 'value3',
        key2 => 'value4',
    },
];

print Dumper $komplexe_datenstruktur;
```

Exkurs: Referenzen

cmd Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee>perl
use Data::Dumper;
my $komplexe_datenstruktur = [
    {
        key1 => 'value1',
        key2 => 'value2',
    },
    {
        key1 => 'value3',
        key2 => 'value4',
    },
];

print Dumper $komplexe_datenstruktur;
^D
$VAR1 = [
    {
        'key2' => 'value2',
        'key1' => 'value1'
    },
    {
        'key2' => 'value4',
        'key1' => 'value3'
    }
];
```

Exkurs: Referenzen

```
use Data::Dumper;
my $komplexe_datenstruktur = [
    {
        key1 => 'value1',
        key2 => 'value2',
    },
    {
        key1 => 'value3',
        key2 => 'value4',
    },
];

print $komplexe_datenstruktur->[0]->{key1};
```

Chained if's

```
if($form eq "form1")  
{  
    %{$felder} = setFelder_form1(\%{$felder},$form);  
}
```

```
if($form eq "form1a")  
{  
    %{$felder} = setFelder_form1a(\%{$felder},$form);  
}
```

```
if($form eq "form2")  
{  
    %{$felder} = setFelder_form2(\%{$felder},$form);  
}
```

```
if($form eq "form3")  
{  
    %{$felder} = setFelder_form3(\%{$felder},$form);  
}
```

```
...  
...  
...
```

Verwenden von Dispatch-Hashes

```
my %hash = (  
    form1 => \&setFelder_form1,  
    form1a => \&setFelder_form1a,  
    form2 => \&setFelder_form2,  
    form3 => \&setFelder_form3,  
);  
  
if( exists $hash{$form} )  
{  
    %{$felder} = $hash{$form}->(\%{$felder}, $form);  
}
```

Referenzen auf Subroutinen

Holt die Referenz auf die Sub...

... und führt sie aus

Dispatch-Hash - Beispiel

```
my %hash = (  
    form1 => \&form1,  
);  
  
if( exists $hash{form1} ){  
    $hash{form1}->( `form1` );  
}  
  
sub form1{  
    my ($text) = @_;  
    print $text;  
}
```

Dispatch-Hash - Beispiel

```
C:\> Eingabeaufforderung
C:\Dokumente und Einstellungen\Renee>perl -Mwarnings
my %hash = (
    form1 => \&form1,
);

if( exists $hash{form1} ){
    $hash{form1}->('form1');
}

sub form1{
    my ($text) = @_;
    print $text;
}

^D
form1
C:\Dokumente und Einstellungen\Renee>
```

Bisheriger Code

```
delete $hash{ "key1" };  
delete $hash{ "key2" };  
delete $hash{ "key3" };  
delete $hash{ "key4" };  
delete $hash{ "key5" };  
delete $hash{ "key6" };
```

“Bessere” Variante

```
my @del_keys = qw(  
    key1  
    key2  
    key3  
    key4  
    key5  
);  
  
delete @hash{@del_keys};
```

Bisheriger Code

```
if($hash{"Test1"}) {
    if( $form{"Test1"} eq "1" ){$note = "++";}
    elsif( $form{"Test1"} eq "2" ){$note = "+";}
    elsif( $form{"Test1"} eq "3" ){$note = "-";}
    $form{"Test1"} = qq|Test1: $note| ;
}
if($form{"Test2"}) {
    if( $hash{"Test2"} eq "1" ){$note = "++";}
    elsif( $hash{"Test2"} eq "2" ){$note = "+";}
    elsif( $hash{"Test2"} eq "3" ){$note = "-";}
    $hash{"Test2"} = qq|Test2: $note| ;
}
if($form{"Test3"}) {
    if( $hash{"Test3"} eq "1" ){$note = "++";}
    elsif( $hash{"Test3"} eq "2" ){$note = "+";}
    elsif( $hash{"Test3"} eq "3" ){$note = "-";}
    $hash{"Test3"} = qq|Test3: $note| ;
}
```

Übersichtlicher

```
my %marks = (  
    1 => '++',  
    2 => '+',  
    3 => '-',  
);  
  
my @tests = (1..3)  
  
for my $test ( @tests ){  
    $hash{ 'Test'.$test } = $marks{ $hash { 'Test'.$test } };  
}
```

Im Code

```
my %hash1 = funktion(\%hash, "form1");
my %hash2 = funktion(\%hash, "form2");

my %hash;

##hashe 1 bis 3 dem hash %hash zuweisen
foreach my $key ( keys %hash1)
{
    $hash{$key} = $hash1{$key};
}

foreach my $key ( keys %hash2)
{
    $hash{$key} = $hash2{$key};
}
```

Übersichtlicher

```
my %tmp_hash;  
my %hash = (  
    funktion(\%tmp_hash,"form1"),  
    funktion(\%tmp_hash,"form2"),  
);
```

Im Code

```
( $sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst) =  
    localtime(time() + 7 * 24 * 60 * 60);  
  
$params{"DATUM"} =  
    sprintf("%02d.%02d.%04d", $mday, $mon+1, $year+1900);};
```

Im Code

```
($mday,$mon,$year) =  
    ( localtime(time() + 7 * 24 * 60 * 60) )[3..5];  
  
$params{"DATUME"} =  
    sprintf("%02d.%02d.%04d", $mday, $mon+1, $year+1900);}
```

map

- map „läuft“ über eine Liste und führt für jedes Element eine AKTION aus.
- Liefert eine Liste zurück
- **map{ AKTION } LISTE**
- Aufgabe: Ich möchte eine Liste mit den „Quadratzahlen“ der Elemente

„normale“ Lösung

```
my @array = ( 1 .. 10 );  
  
for my $elem ( @array ){  
    $elem = $elem * $elem;  
}  
  
print $_, "\n" for @array;
```

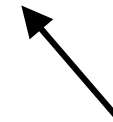
map-Lösung

AKTION



```
my @array = map{ $_ * $_ } ( 1 .. 10 );
```

```
print $_, "\n" for @array;
```



Liste

Möglichkeit ausrealem Code

```
# Adressdaten
$params{"STRASSE"} = $hash{"strasse"}->getValue();
$params{"PLZ"} = $hash{"plz"}->getValue();
$params{"ORT"} = $hash{"ort"}->getValue();
$params{"Land"} = $hash{"land"}->getValue();
```

Möglichkeit aus realem Code

```
my @fields = qw(STRASSE PLZ ORT Land);  
  
# Adressdaten  
@params{ @fields } = map{  
    $hash{ lc($_) }->getValue();  
}@fields;
```

sort

- sort sortiert eine Liste
- Liefert eine Liste zurück
- Verwendet die “Spezialvariablen” \$a und \$b
- **sort{ BEDINGUNG } LISTE**
- Aufgabe: Ich möchte eine Liste mit Zahlen sortieren

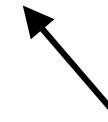
Lösung

Bedingung



```
my @array = sort{ $a <=> $b }( 1,3,6,2,7,10,4,5 );
```

```
print $_, "\n" for @array;
```



Liste

Lösung

Bedingung



```
my @array = sort{ $b <=> $a }( 1,3,6,2,7,10,4,5 );
```

```
print $_, "\n" for @array;
```



Liste

sort

- Aufgabe: Ich möchte eine Liste mit Wörtern sortieren

Lösung

Bedingung

```
my @liste = qw( Dies ist ein Test );  
my @array = sort{ $a cmp $b }@liste;  
  
print $_, "\n" for @array;
```

Liste

Lösung

```
my @liste = qw( Dies ist ein Test );  
my @array = sort{ $b cmp $a }@liste;  
  
print $_, "\n" for @array;
```

Lösung

 **Auswählen Eingabeaufforderung**

```
C:\Dokumente und Einstellungen\Renee>perl
my @liste = qw< Dies ist ein Test >;
my @array = sort< $b cmp $a >@liste;

print $_, "\n" for @array;
^D
ist
ein
Test
Dies

C:\Dokumente und Einstellungen\Renee>
```

grep

- grep „läuft“ über eine Liste und prüft für jedes Element eine BEDINGUNG.
- Liefert eine Liste zurück mit allen Elementen für die diese Bedingung „wahr“ ist
- **grep{ BEDINGUNG } LISTE**
- Aufgabe: Ich möchte eine Liste mit den Elementen, die „Perl“ beinhalten

„normale“ Lösung

```
my @array = qw(
    Perl-Programm
    Programmierer
    Perl-Kamel
);

my @tmp;
for my $elem ( @array ){
    if( $elem =~ /Perl/ ){
        push @tmp, $elem;
    }
}

print $_, "\n" for @tmp;
```

grep-Lösung

```
my @array = qw(
    Perl-Programm
    Programmierer
    Perl-Kamel
);
@array = grep[ $_ =~ /Perl/ ]@array;

print $_, "\n" for @array;
```

Bedingung

Liste

sprintf

- Formatierung von Zahlen oder Strings
- **sprintf FORMAT, LISTE**
- Wichtigste Formatierungsregeln
 - %s → Strings
 - %d → Dezimalzahlen
 - %.#f → Kommazahlen

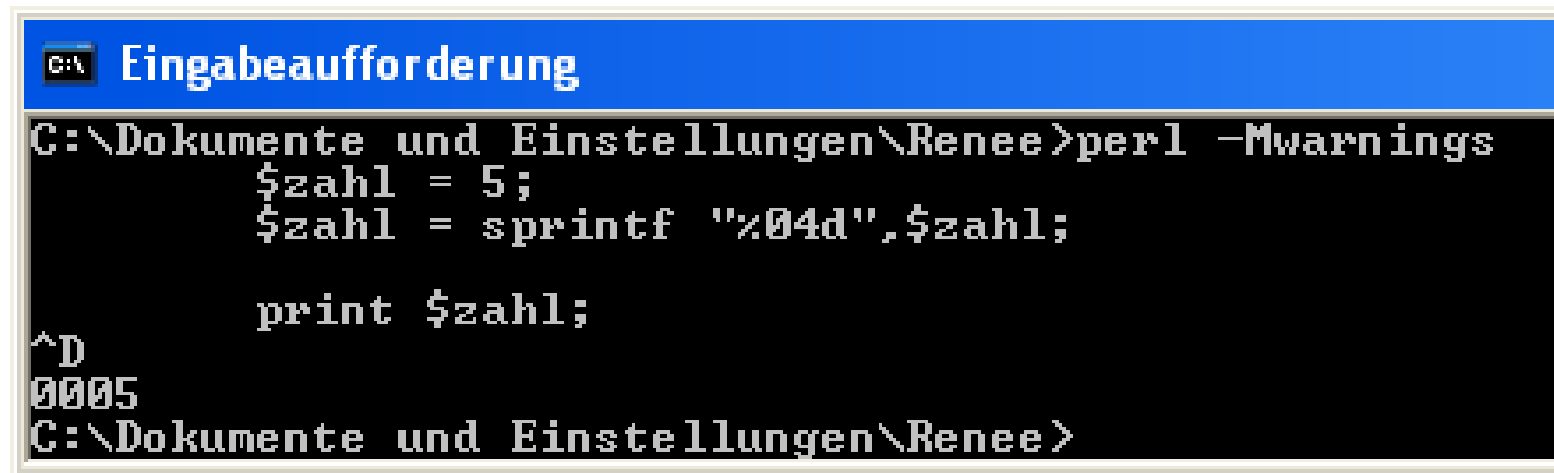
sprintf

```
if($zahl < 1000){
    $zahl = "0" . $zahl;
}
elseif($zahl < 100){
    $zahl = "00" . $zahl;
}
elseif($zahl < 10){
    $zahl = "000" . $zahl;
}

print $zahl;
```

sprintf - Beispiel

```
$zahl = 5;  
$zahl = sprintf "%04d", $zahl;  
  
print $zahl;
```



```
C:\Dokumente und Einstellungen\Renee>perl -Mwarnings  
    $zahl = 5;  
    $zahl = sprintf "%04d", $zahl;  
  
    print $zahl;  
^D  
0005  
C:\Dokumente und Einstellungen\Renee>
```

sprintf

```
my $tag = 3;
my $monat = 5;
my $jahr = 2007;

if($tag < 10){
    $tag = "0" . $tag;
}

if($monat < 10){
    $monat = "0" . $monat;
}

my $date = $tag . "." . $monat . "." . $jahr;
print $date;
```

sprintf

```
my $tag = 3;  
my $monat = 5;  
my $jahr = 2007;
```

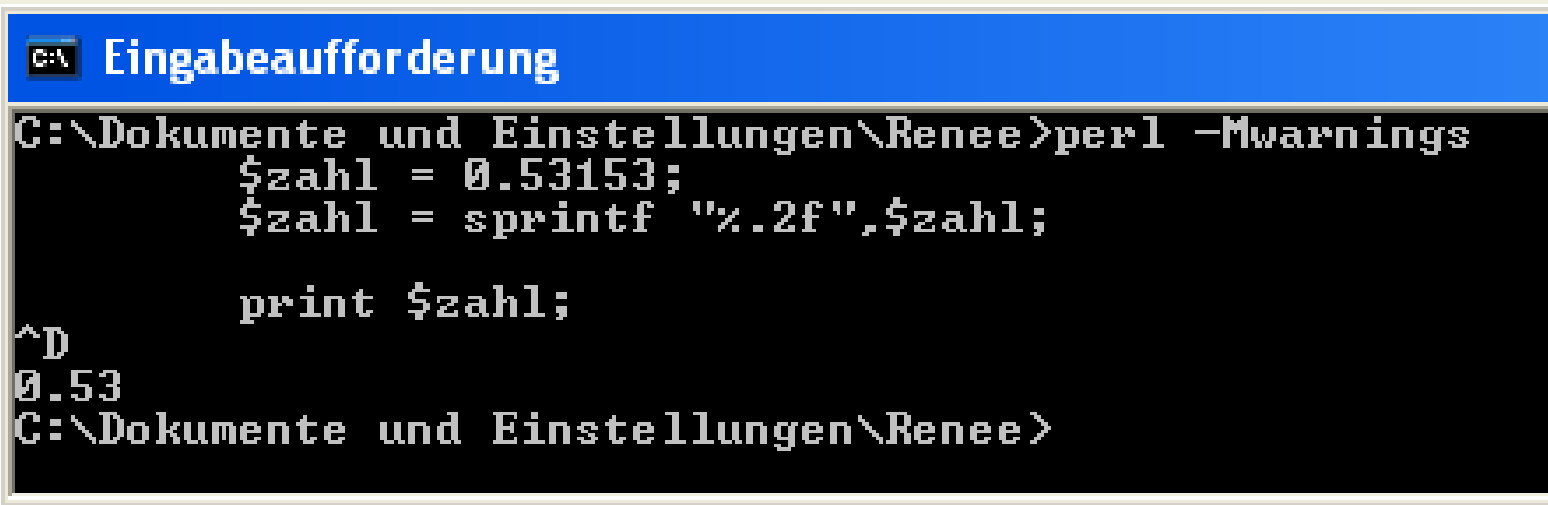
```
my $date = sprintf "%02d.%02d.%04d", $tag, $monat, $jahr;  
print $date;
```

☞ Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee>perl -Mwarnings  
my $tag = 3;  
my $monat = 5;  
my $jahr = 2007;  
  
my $date = sprintf "%02d.%02d.%04d", $tag, $monat, $jahr;  
print $date;  
^D  
03.05.2007  
C:\Dokumente und Einstellungen\Renee>
```

sprintf

```
$zahl = 0.53153;  
$zahl = sprintf "%.2f", $zahl;  
  
print $zahl;
```



```
C:\ Eingabeaufforderung  
C:\Dokumente und Einstellungen\Renee>perl -Mwarnings  
    $zahl = 0.53153;  
    $zahl = sprintf "%.2f", $zahl;  
  
    print $zahl;  
^D  
0.53  
C:\Dokumente und Einstellungen\Renee>
```

„komplexe“ map-sort-Anwendung

- Die Worte „Dies“, „ist“, „ein“, „Test“ sollen alphabetisch sortiert werden
- ohne Beachtung von Groß-/Kleinschreibung
- ➔ Schwarz'sche Transformation

„komplexe“ map-sort-Anwendung

```
my @liste = qw( Dies ist ein Test );
```

Ziel:

Dies

ein

ist

Test

Schritt 1

- Man braucht zu allen Werten auch die Kleinschreibweise oder nur Großbuchstaben
- (dies, ist, ein, test)
- Ursprüngliche Werte dürfen nicht verloren gehen

Schritt 1

```
my @liste = qw( Dies ist ein Test );  
my @array = map { [$_, lc($_)] } @liste;  
  
print Dumper \@array;
```

Schritt 1

C:\ Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee>perl -MData::Dumper
my @liste = qw( Dies ist ein Test );
my @array = map { [$_, lc($_)] } @liste;

print Dumper \@array;
^D
$VAR1 = [
    [
        'Dies',
        'dies'
    ],
    [
        'ist',
        'ist'
    ],
    [
        'ein',
        'ein'
    ],
    [
        'Test',
        'test'
    ]
];

C:\Dokumente und Einstellungen\Renee>
```

Schritt 2

- Man muss die Liste sortieren (nach kleingeschriebenen Wörtern)

Schritt 2

```
my @liste = qw( Dies ist ein Test );  
my @array = sort{ $a->[1] cmp $b->[1] }  
              map { [$_, lc($_)] } @liste;  
  
print Dumper \@array;
```

Schritt 2

🖥️ Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee>perl -MData::Dumper
my @liste = qw( Dies ist ein Test );
my @array = sort< $a->[1] cmp $b->[1] >
             map < [$_, lc<$_>]             >@liste;

print Dumper \@array;
^D
$VAR1 = [
    [
        'Dies',
        'dies'
    ],
    [
        'ein',
        'ein'
    ],
    [
        'ist',
        'ist'
    ],
    [
        'Test',
        'test'
    ]
];

C:\Dokumente und Einstellungen\Renee>
```

Schritt 3

- Man braucht in der sortierten Liste nur noch die “Originalversion”
- (Dies, ist, ein, Test)

Schritt 3

```
my @liste = qw( Dies ist ein Test );
my @array = map { $_->[0] }
              sort{ $a->[1] cmp $b->[1] }
              map { [$_, lc($_)] }@liste;

print $_, "\n" for @array;
```

Schritt 3

C:\ Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee>perl -MData::Dumper
my @liste = qw( Dies ist ein Test );
my @array = map { $_->[0] }
             sort{ $a->[1] cmp $b->[1] }
             map { [$_, lc($_)] } @liste;

print $_, "\n" for @array;^D
Dies
ein
ist
Test
C:\Dokumente und Einstellungen\Renee>
```

@INC verändern

- Eigene Module sollen verwendet werden, die nicht im Standardpfad liegen

In Skripten

```
push(@INC, "/mein/pfad/1", "/mein/pfad/2", "/mein/pfad/3")
    if ($system == $test);
push(@INC, "/unser/pfad/1", "/unser/pfad/2")
    if ($system == $test2);
push(@INC, "/dein/pfad/1", "/dein/pfad/2")
    if ($system == $test3);
```

Beispielskript

```
#!/usr/bin/perl

use strict;
use warnings;

BEGIN{
    push @INC, './perllib';
    use Data::Dumper;
}

my @array = (1..3);
print Dumper \@array;
```

Uuups...

```
C:\> Auswählen Eingabeaufforderung
C:\Dokumente und Einstellungen\Renee\Eigene D
_Training>perl inc_push.pl
$VAR1 = [
    1,
    2,
    3
];
C:\Dokumente und Einstellungen\Renee\Eigene D
_Training>
```

```
package Data::Dumper;
```

```
use Exporter;
```

```
our @ISA = qw(Exporter);
```

```
our @EXPORT = qw(Dumper);
```

```
sub Dumper{
    my ($arref) = @_;
    print "Element: $_\n" for @$arref;
}
```

Probleme mit „push @INC“

- Pfade werden hinten angefügt
 - Es wird immer nur erster Fund geladen
 - Häufig werden eigene Module nicht geladen
- Kompilierte Komponenten werden nicht gefunden
- Doppelte Einträge

besser

```
BEGIN{
  my @libs = ();
  @libs = ("/mein/pfad/1", "/mein/pfad/2")
    if ( $system == $test);
  @libs = ("/unser/pfad/1", "/unser/pfad/2")
    if ( $system == $test1 );
  @libs = ("/dein/pfad/1", "/dein/pfad/2")
    if ( $system == $test2 );

  use lib @libs;
}
```

...weil...

- Pfade werden **vorne** angefügt
- Kompilierte Komponenten werden nicht gefunden, da *lib* nach „arch“-Verzeichnissen sucht
- Doppelte Einträge werden eliminiert
- Automatische Prüfung, ob Verzeichnis existiert

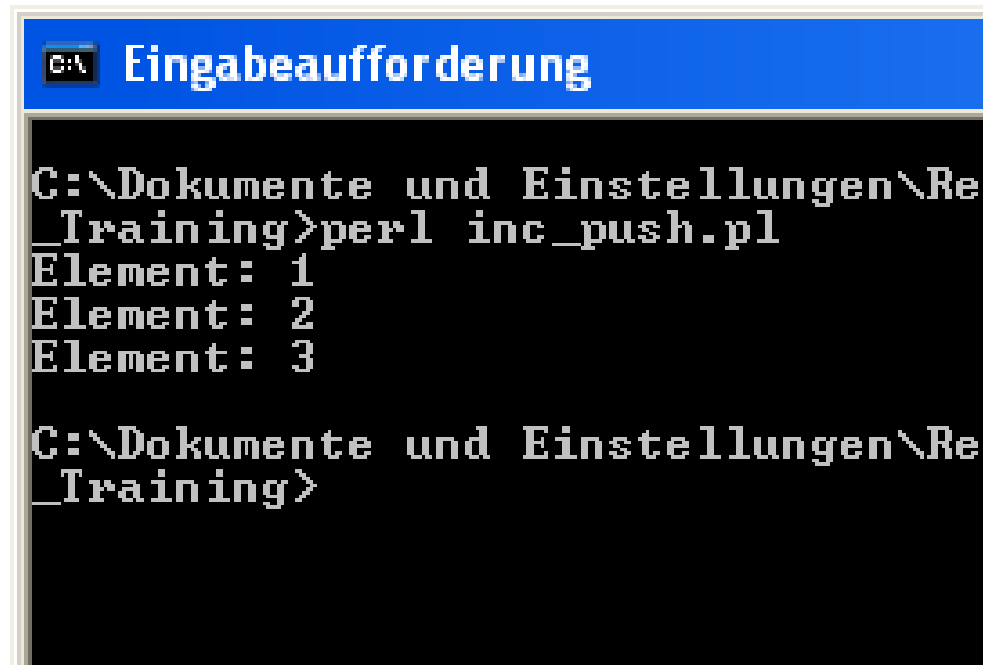
Beispiel mit „use lib“

```
#!/usr/bin/perl

use strict;
use warnings;
use lib './perllib';
use Data::Dumper;

my @array = (1..3);
print Dumper \@array;
```

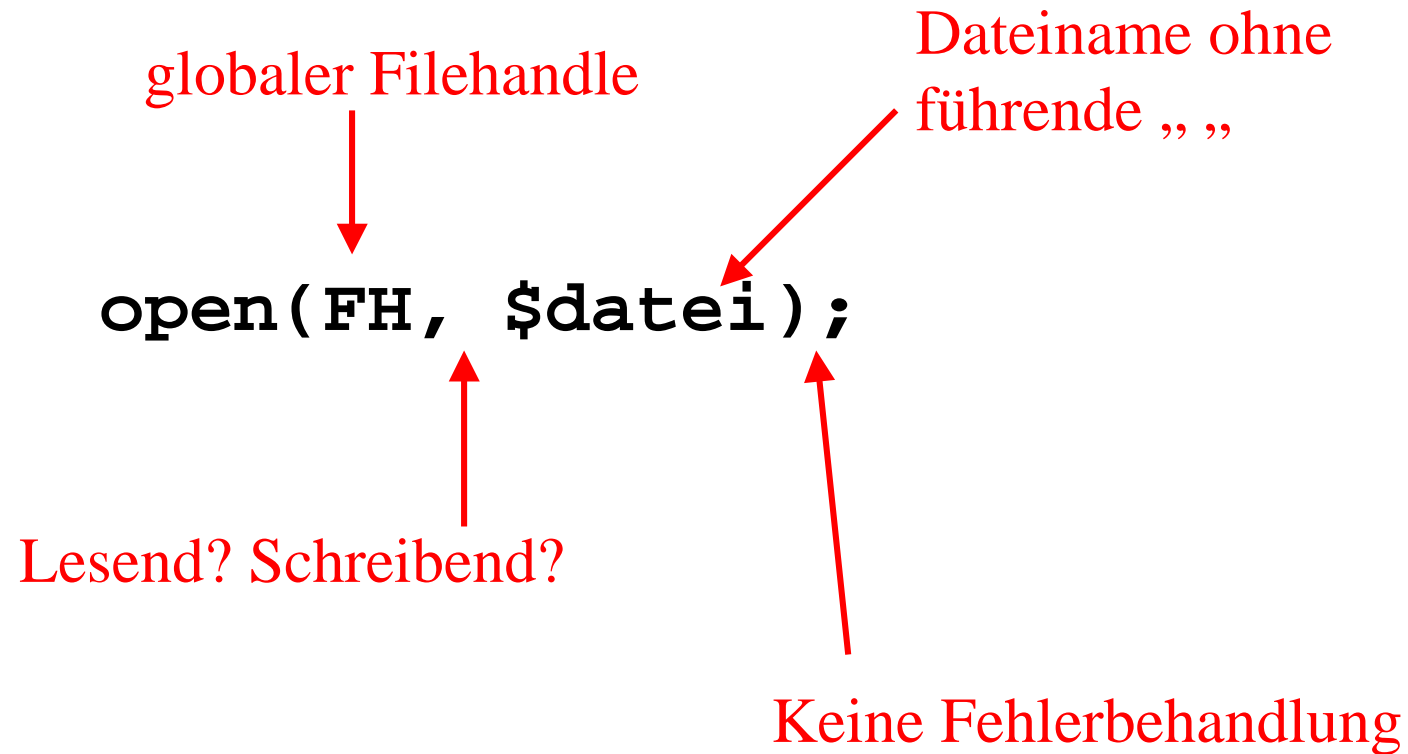
Ausgabe



```
C:\Dokumente und Einstellungen\Re
_Training>perl inc_push.pl
Element: 1
Element: 2
Element: 3

C:\Dokumente und Einstellungen\Re
_Training>
```

Typische Probleme bei „open“



Problem mit globalem Filehandle

```
#!/usr/bin/perl

use strict;
use warnings;

my $file = 'test.txt';
readFile($file);

sub readFile{
    my ($file) = @_;
    if(open(FH,$file)){
        while( <FH> ){
            chomp;
            readFile( $_ );
        }
        close FH;
    }
}
```

```
open_1.pl
test3.txt
```

Problem mit globalem Filehandle

C:\ Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\Auftraege\etect
_Training>perl open_2.pl
readline() on closed filehandle FH at open_2.pl line 15, <FH> line 2.
readline() on closed filehandle FH at open_2.pl line 15.

C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\Auftraege\etect
_Training>
```

Problem mit globalem Filehandle

```
#!/usr/bin/perl

use strict;
use warnings;

my $file = 'test.txt';
readFile($file);

sub readFile{
    my ($file) = @_;
    if(open(FH,$file)){
        while( <FH> ){
            chomp;
            readFile( $_ );
        }
        close FH;
    }
}
```

Problem mit globalem Filehandle

```
sub readFile{
  my ($file) = @_ ;
  if(open(my $fh,$file)){
    while( <$fh> ){
      chomp;
      readFile( $_ );
    }
    close $fh;
  }
}
```

Lesend? Schreibend?

```
#!/usr/bin/perl

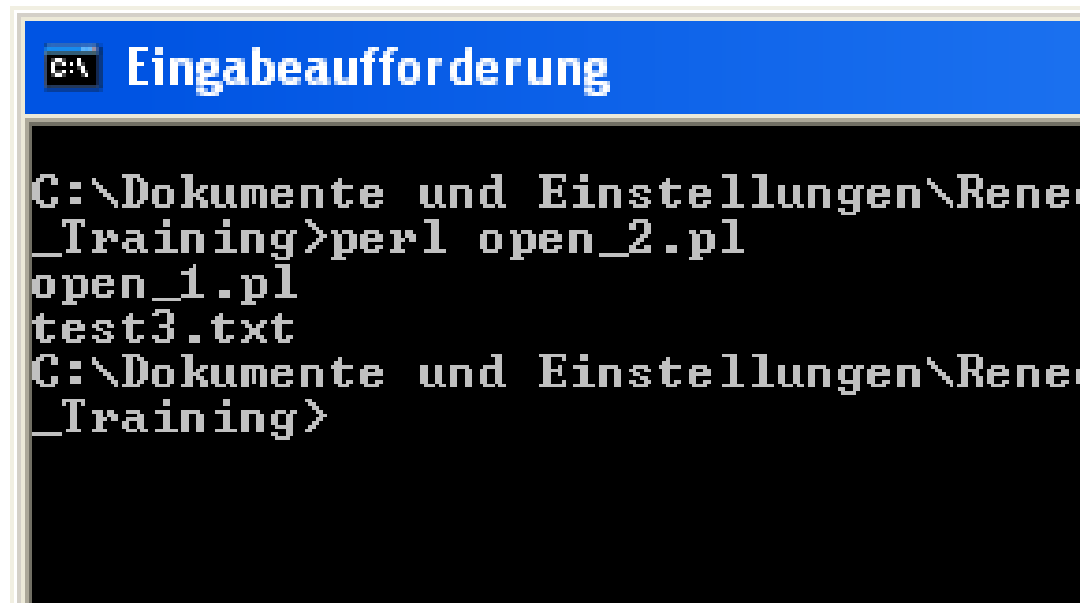
use strict;
use warnings;

my $file = 'test.txt';
readFile($file);

sub readFile{
    my ($file) = @_;

    if(open(my $fh,$file)){
        while( <$fh> ){
            print;
        }
        close $fh;
    }
}
```

Lesend? Schreibend?



```
C:\Dokumente und Einstellungen\Ren...
_Training>perl open_2.pl
open_1.pl
test3.txt
C:\Dokumente und Einstellungen\Ren...
_Training>
```

Lesend? Schreibend?

```
#!/usr/bin/perl

use strict;
use warnings;

my $file = '> test.txt';
readFile($file);

sub readFile{
    my ($file) = @_;

    if(open(my $fh,$file)){
        while( <$fh> ){
            print;
        }
        close $fh;
    }
}
```

Lesend? Schreibend?

Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\Auftr  
_Training>perl open_2.pl  
Filehandle $fh opened only for output at open_2.pl line 13.  
  
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\Auftr  
_Training>more test.txt  
  
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\Auftr  
_Training>
```

Lesend? Schreibend?


```
_Training>perl open_2.pl  
Filehandle $fh opened only for output at open_2.pl line 13.
```

Lesend? Schreibend?

```
Training>more test.txt
```

Lesend? Schreibend?

```
if(open(my $fh, '<', $file)){  
    while( <$fh> ){  
        print;  
    }  
    close $fh;  
}
```



In der 3-Arg-Form von open
aund zusätzlich führende und
angehängte Leerzeichen erlaubt

Fehler abfangen

```
#!/usr/bin/perl

use strict;
use warnings;

my $file = 'test5.txt';
readFile($file);

sub readFile{
    my ($file) = @_;

    open(FH,$file) or warn $!;
    my @lines = <FH>;
    close FH;
}
```

Fehler abfangen

cmd Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl
_Training>open_1.pl
No such file or directory at C:\Dokumente und Einstellu
\Perl\Auftraege\etecture\Perl_Training\open_1.pl line 1
readline() on closed filehandle FH at C:\Dokumente und
e Dateien\Perl\Auftraege\etecture\Perl_Training\open_1
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl
_Training>
```

Fehler abfangen

```
Perl Training>open_1.pl  
No such file or directory at C:\Dokumente und Einstellungen\...  
\Perl\Auftraege\etecture\Perl Training\open_1.pl line 1
```

Fehler abfangen

```
\Perl\Auftraege\etecture\Perl_Training\open_1.pl line 1  
readline() on closed filehandle FH at C:\Dokumente und  
e Dateien\Perl\Auftraege\etecture\Perl_Training\open_1
```

Sonstiges zur Dateibearbeitung

- Vorsicht vor Race-Conditions
 - Lesender Zugriff während anderer Prozess schreibt
 - „flock“
 - „sysopen“

Ternary Operator

- if-else-“Ersatz“

Vorteil

- Kürzer / Übersichtlicher
- Zweck schneller erkennbar

Nachteil

- Schlechter wenn erweitert werden muss

Beispiel

```
my $anredeText;  
  
if ($anrede eq "1") {  
    $anredeText = "Herrn";  
}  
else {  
    $anredeText = "Frau";  
}
```

```
my $anredeText2;  
  
if ($anrede2 eq "1") {  
    $anredeText2 = "Herrn";  
}  
else {  
    $anredeText2 = "Frau";  
}
```

Beispiel

```
my $anredeText = ($anrede == 1) ? „Herrn“ : "Frau";  
my $anredeText2 = ($anrede2 == 1) ? „Herrn“ : "Frau";
```

Quote Operatoren

- Quoting ohne “ und ‘, Aufruf von Programmen,...
- Kann Strings übersichtlicher machen
- Verschiedene Operatoren
 - q
 - qq
 - qx
 - qw
 - ...

Quote Operatoren - q

- „Ersetzt“ ‘string‘
- Keine Interpolation
- ‘ muss nicht „escaped“ werden

Quote Operatoren - q

```
my $string = `Hello World - $test`;  
my $string = `Hello World - O\`Reilly`;
```

```
my $string = q~Hello World - $test~;
```

```
my $string = q!Hello World - O`Reilly!;
```

Quote Operatoren - qq

- „Ersetzt“ `“string“`
- Interpolation
- `“` muss nicht `„escaped“` werden

Quote Operatoren - qq

```
my $string = "<img src=$test>";  
my $string = "<img src=\"\$test\">";
```

```
my $string = qq~<img src=$test>~;
```

```
my $string = qq!!;
```

Quote Operatoren - qx

- Führt ein Kommando aus (``)
- Interpolation (bedingt)

Quote Operatoren - qx

```
my $string = `ls -l`;
```

```
my $string = qx{ls -l};
```

Quote Operatoren - qw

- Erzeugt eine Liste
- Leerzeichen als Wort-Trenner
- Keine Interpolation

Quote Operatoren - qw

```
my @list = ('test', 'hallo', 'welt');
```

```
my @list = qw(test hallo welt);
```

RegEx-Vermeidung

- langsam
- „schlecht lesbar“
- Fehleranfällig
- nicht immer ersetzbar (dynamischer Content)

RegEx-Vermeidung

- Häufig Ersatz möglich durch
 - index
 - substr

RegEx-Vermeidung

```
if ($excpetion =~ m/STRING_5/)

if (($key eq '_test') and ($matchedKey =~ /^Bitte/)) {

if ($buttonClicked =~ /zurueck/)

if ($file =~ /\.jpg/)
```

RegEx-Vermeidung

```
if ( index( $excpetion, "STRING_5" ) != -1 )
```

```
if ( $key eq '_test' and  
    index( $matchedKey, 'Bitte' ) == 0 ) {
```

```
if ( index ( $buttonClicked, 'zurueck' ) != -1)
```

```
if ( substr( $file, -4 ) eq '.jpg' )
```

Code-Review

```
for(my $i=$beginNr;$i < $endNr; $i++)
{
    $resultString = $resultString."<tr class=\"fieldset\">\n";
    $resultString = $resultString."\t".<td>".${$liste}[$i]->getName().
        "<input type=\"hidden\" name=\"name_$$i\" value=\"\".
        {$liste}[$i]->getName().\"></td>\n";
    $resultString = $resultString."\t".<td>".
        {$liste}[$i]->getVorname().
        "<input type=\"hidden\" name=\",,vorname_$$i\" value=\"\".
        {$liste}[$i]->getVorname().\"></td>\n";
    $resultString = $resultString."\t".<td>".${$liste}[$i]->getOrt().
        "<input type=\"hidden\" name=\"ort_$$i\" value=\"\".
        {$liste}[$i]->getOrt().\"></td>\n";
    $resultString = $resultString."</tr>\n";
}
```

Code-Review

```
for(my $i=$beginNr;$i < $endNr; $i++)
{
    $resultString = $resultString."<tr class=\"fieldset\">\n";
    $resultString = $resultString."\t".<td>".${$liste}[$i]->getName().
        "<input type=\"hidden\" name=\"name_$$i\" value=\"".
        "{$liste}[$i]->getName().\"></td>\n";
    $resultString = $resultString."\t".<td>".
        "{$liste}[$i]->getVorname().
        "<input type=\"hidden\" name=\"vorname_$$i\" value=\"".
        "{$liste}[$i]->getVorname().\"></td>\n";
    $resultString = $resultString."\t".<td>".${$liste}[$i]->getOrt().
        "<input type=\"hidden\" name=\"ort_$$i\" value=\"".
        "{$liste}[$i]->getOrt().\"></td>\n";
    $resultString = $resultString."</tr>\n";
}
```

Code-Review

```
for(my $i=$beginNr;$i < $endNr; $i++)
{
    $resultString = $resultString."<tr class=\"fieldset\">\n";
    $resultString = $resultString."\t".<td>".${$liste}[$i]->getName().
        "<input type=\"hidden\" name=\"name_$$i\" value=\"".
            "{$liste}[$i]->getName().\"></td>\n";
    $resultString = $resultString."\t".<td>".
        "{$liste}[$i]->getVorname().
            "<input type=\"hidden\" name=\"vorname_$$i\" value=\"".
                "{$liste}[$i]->getVorname().\"></td>\n";
    $resultString = $resultString."\t".<td>".${$liste}[$i]->getOrt().
        "<input type=\"hidden\" name=\"ort_$$i\" value=\"".
            "{$liste}[$i]->getOrt().\"></td>\n";
    $resultString = $resultString."</tr>\n";
}
```

Code-Review

```
for(my $i=$beginNr;$i < $endNr; $i++)
{
    $resultString = $resultString."<tr class=\"fieldset\">\n";
    $resultString = $resultString."\t".<td>".${$liste}[$i]->getName().
        "<input type=\"hidden\" name=\"name_{$i}\" value=\"\".
        {$liste}[$i]->getName().\"></td>\n";
    $resultString = $resultString."\t".<td>".
        {$liste}[$i]->getVorname().
        "<input type=\"hidden\" name=\"vorname_{$i}\" value=\"\".
        {$liste}[$i]->getVorname().\"></td>\n";
    $resultString = $resultString."\t".<td>".${$liste}[$i]->getOrt().
        "<input type=\"hidden\" name=\"ort_{$i}\" value=\"\".
        {$liste}[$i]->getOrt().\"></td>\n";
    $resultString = $resultString."</tr>\n";
}
```

Code-Review

```
for(my $i=$beginNr;$i < $endNr; $i++)
{
    $resultString = $resultString."<tr class=\"fieldset\">\n";
    $resultString = $resultString."\t".<td>".${$liste}[$i]->getName().
        "<input type=\"hidden\" name=\"name_$$i\" value=\"".
        "{$liste}[$i]->getName().\"></td>\n";
    $resultString = $resultString."\t".<td>".
        "{$liste}[$i]->getVorname().
        "<input type=\"hidden\" name=\"vorname_$$i\" value=\"".
        "{$liste}[$i]->getVorname().\"></td>\n";
    $resultString = $resultString."\t".<td>".${$liste}[$i]->getOrt().
        "<input type=\"hidden\" name=\"ort_$$i\" value=\"".
        "{$liste}[$i]->getOrt().\"></td>\n";
    $resultString = $resultString."</tr>\n";
}
```

Code-Review

```
for my $i ( $beginNr .. $endNr-1 )
{
    my $elem = $liste->[$i];
    my ($name, $vorname, $ort) = (
        $elem->getName,
        $elem->getVorname,
        $elem->getOrt,
    );

    $resultString .= qq~<tr class="fieldset">
\t<td>$name <input type="hidden" name="name_$i" value="$name"></td>
\t<td>$vorname <input type="hidden" name="vorname_$i" value="$plz"></td>
\t<td>$ort <input type="hidden" name="ort_$i" value="$ort"></td>
</tr>
~;
}
```

defined

```
#!/usr/bin/perl

use strict;
use warnings;

chomp( my $test = <STDIN> );

if( $test ){
    print "Sie haben $test eingegeben\n";
}
else{
    print "Sie haben nichts eingegeben\n";
}
```

Beispiel

 Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>d  
hallo  
Sie haben hallo eingegeben  
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>
```

Beispiel

 Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>d  
␣  
Sie haben nichts eingegeben  
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>
```

Beispiel

```
#!/usr/bin/perl
```

```
use strict;  
use warnings;
```

```
chomp( my $test = <STDIN> );
```

```
if( $test ){  
    print "Sie haben $test eingegeben\n";  
}  
else{  
    print "Sie haben nichts eingegeben\n";  
}
```

undef

0

""



Beispiel

```
#!/usr/bin/perl

use strict;
use warnings;

chomp( my $test = <STDIN> );

if( defined $test ){
    print "Sie haben $test eingegeben\n";
}
else{
    print "Sie haben nichts eingegeben\n";
}
```

Beispiel

C: Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>d
```

```
Ø
```

```
Sie haben Ø eingegeben
```

```
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>
```

newforms2

```
if($show_header eq "")  
{  
    $show_header = "true";  
}
```

Realer Code

```
[Thu Jul 5 11:27:46 2007] test.cgi: Use of uninitialized value in string eq  
at /pfad/Modul.pm line 275.
```

```
[Thu Jul 5 11:27:46 2007] test.cgi: Use of uninitialized value in string eq  
at /pfad/Modul.pm line 275.
```

```
[Thu Jul 5 11:29:17 2007] test.cgi: Use of uninitialized value in string eq  
at /pfad/Modul.pm line 275.
```

```
[Thu Jul 5 11:29:17 2007] test.cgi: Use of uninitialized value in string eq  
at /pfad/Modul.pm line 275.
```

```
[Thu Jul 5 11:35:46 2007] test.cgi: Use of uninitialized value in string eq  
at /pfad/Modul.pm line 275.
```

```
[Thu Jul 5 11:35:46 2007] test.cgi: Use of uninitialized value in string eq  
at /pfad/Modul.pm line 275.
```

```
[Thu Jul 5 11:35:46 2007] test.cgi: Use of uninitialized value in string eq  
at /pfad/Modul.pm line 275.
```

Realer Code

```
if( defined $show_header and $show_header eq "" )  
{  
    $show_header = "true";  
}
```

Realer Code

```
##falls normalisierung erfolgte,  
# wird normalisierter wert geprüft  
if($feld->getNormaltyp() ne "")  
{  
    $value = $feld->getNormalvalue();  
}
```

Realer Code

```
[Thu Jul 5 15:02:44 2007] test.cgi: Use of uninitialized value in string eq at /pfad/Modul.pm line 169.
```

```
[Thu Jul 5 15:02:44 2007] test.cgi: Use of uninitialized value in length at /pfad/Modul.pm line 175.
```

```
[Thu Jul 5 15:02:44 2007] test.cgi: Use of uninitialized value in string eq at /pfad/Modul.pm line 169.
```

Realer Code

```
##falls normalisierung erfolgte,  
# wird normalisierter wert geprüft  
my $normalvalue = $ feld->getNormaltyp()  
if( defined $normalvalue and $normalvalue ne "" )  
{  
    $value = $normalvalue;  
}
```

Autovivification

- Automatische Erstellung von Array/Hash-Elementen
- Führt manchmal zu unerwünschten Ergebnissen

Autovivification - Beispiel

```
#!/usr/bin/perl

use strict;
use warnings;
use Data::Dumper;

my %hash;
print Dumper \%hash;

if( $hash{test}->{test2} ){
    #...
}

print Dumper \%hash;
```

Autovivification - Beispiel

C:\ Markieren Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>auto  
ion.pl  
$UAR1 = {};  
$UAR1 = {  
    'test' => {}  
};  
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>
```

Autovivification – ein Problem?

- Ja, wenn die Schlüssel/die Elemente eines Hashs/Arrays wichtig sind
- Nein, wenn diese unwichtig sind

Autovivification – Ein Problem!

```
#!/usr/bin/perl

use strict;
use warnings;
use Data::Dumper;

my @array;
print Dumper \@array;

if( defined $array[8]->[0] ){
    #...
}

if( scalar @array ){
    print "im if-Block weil das Array ",
        scalar(@array),
        " Elemente hat\n";
}
```

Autovivification – ein Problem!

Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>autovivification.pl
$UAR1 = [];
im if-Block weil das Array 9 Elemente hat

C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>
```

exists

- Kann autovivification vermeiden
- Prüft nur die Existenz
- Führt zu mehr „Vorhersagbarkeit“

exists – der Problemlöser

```
#!/usr/bin/perl

use strict;
use warnings;
use Data::Dumper;

my @array;
print Dumper \@array;

if( exists $array[8] and
    defined $array[8]->[0] ){
    #...
}

if( scalar @array ){
    print "im if-Block weil das Array ",
        scalar(@array),
        " Elemente hat\n";
}
```

exists – der Problemlöser

C:\ Eingabeaufforderung

```
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>au  
ion_array.pl  
$VAR1 = [];  
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>_
```

exists – der Problemlöser

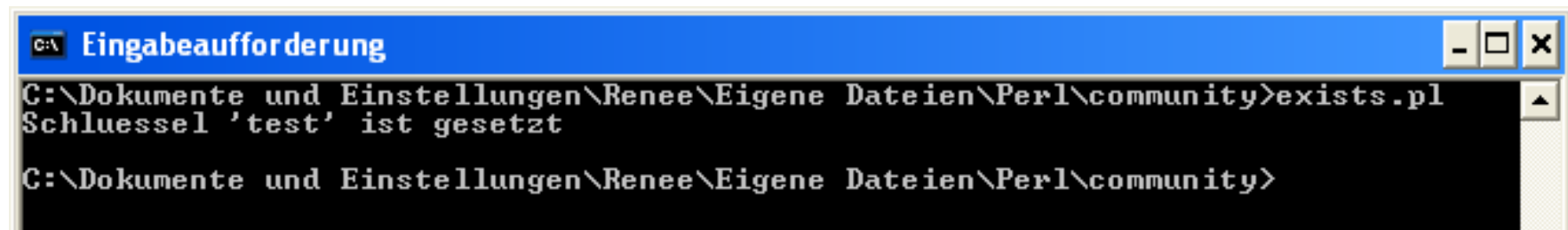
```
#!/usr/bin/perl

use strict;
use warnings;

my %hash = (
    test => 1,
    hallo => 1,
);

if( $hash{test} ){
    print "Schluessel 'test' ist gesetzt\n";
}
```

exists – der Problemlöser



```
C:\> Eingabeaufforderung
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>exists.pl
Schluessel 'test' ist gesetzt
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>
```

exists – der Problemlöser

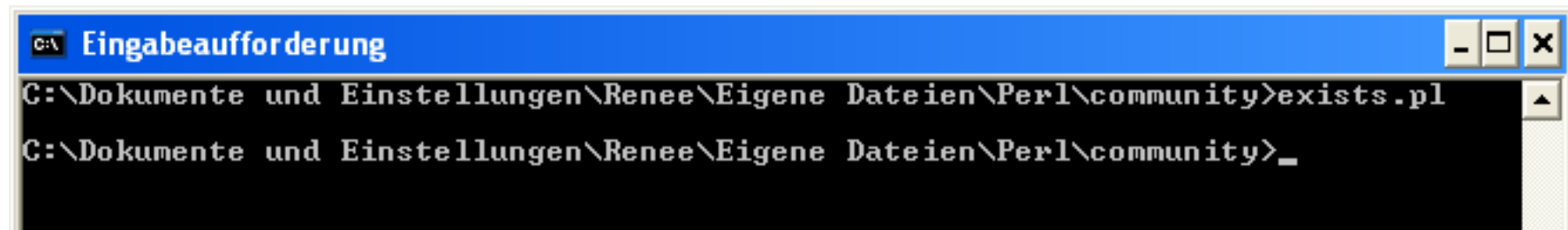
```
#!/usr/bin/perl

use strict;
use warnings;

my %hash = (
    test => 0,
    hallo => 1,
);

if( $hash{test} ){
    print "Schluessel 'test' ist gesetzt\n";
}
```

exists – der Problemlöser



```
C:\> Eingabeaufforderung
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>exists.pl
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>_
```

exists – der Problemlöser

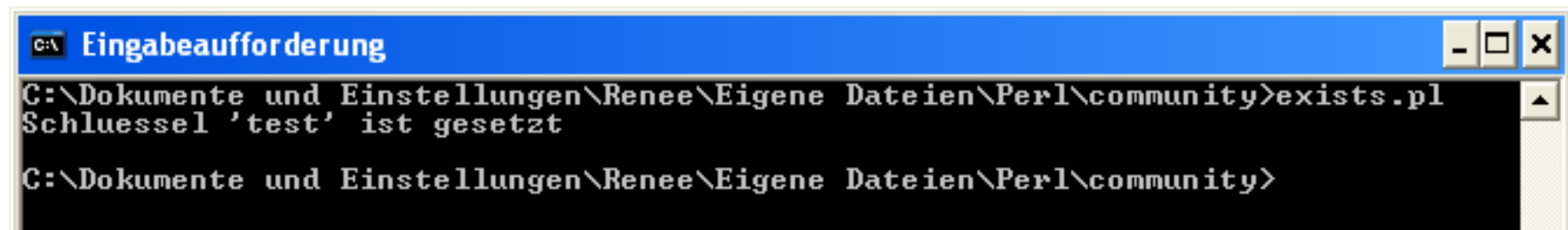
```
#!/usr/bin/perl

use strict;
use warnings;

my %hash = (
    test => 0,
    hallo => 1,
);

if( exists $hash{test} ){
    print "Schluessel 'test' ist gesetzt\n";
}
```

exists – der Problemlöser



```
C:\> Eingabeaufforderung
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>exists.pl
Schlüssel 'test' ist gesetzt
C:\Dokumente und Einstellungen\Renee\Eigene Dateien\Perl\community>
```