Übersicht PERL

!!!! Wichtig: Nach dem Befehl einem Strichpunkt setzen !!!!

Erste Zeile eines PERL-Scripts:

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

Variablen in PERL:

Normale Variablen beginnen mit einem $-Zeichen
Array-Variablen beginnen mit einem @-Zeichen
Die Variablennamen können groß und klein oder gemischt geschrieben werden.
Die Schreibweise ist ausschlaggebend !!

Inhalt:

Normale Variablen können Strings oder Zahlen enthalten
Array-Variablen können Strings oder Zahlen oder beides enthalten.
Die Zahlen können als ganze Zahlen oder als Gleitkommazahlen dargestellt werden.
Sie werden jedoch von PERL intern wie Gleitkommazahlen von Typ *double* behandelt.

Zuweisung:

Normale Variablen:

```
$variablenname = Wert;          bzw.      $variablenname = ”String”;
```

Array-Variablen mit Inhalten vorbelegen:

```
@arrayname = (1,2,3,4);
```

oder

```
@arrayname = (“Stefan“,“Hans“,“Peter“);
```

oder

```
@arrayname = (1,“Text“);
```

Array-Variablen gezielt beschreiben:

```
$arrayname [0] = “1.Feld im Array“;
```

Hier ist zu beachten, daß vor der Array-Variable ein $-Zeichen steht
und der Index des Arrays bei 0 beginnt.

Letzten Index der Array-Variablen ermitteln:

```
$letzter_index = $#arrayname;
```

Das #-Zeichen zwischen dem $-Zeichen und dem Arraynamen führt zu einer Rückgabe des letzten Index des Arrays. Hieraus kann die Länge des Arrays ermittelt werden, indem man zum letzten Index noch 1 dazuzählt.

Übersicht PERL

Befehle aus UNIX (oder LINUX) in PERL nutzen:

Beispiel:

```
use Shell qw( date ps cp tail );
```

Mit dieser Anweisung werden nun die Befehle “date”, “ps”, “cp” und “tail“ in PERL zur Verwendung bereitgestellt. Eventuelle Parameter der Befehle werden bei der Verwendung des Befehl in Klammern angegeben (siehe Beispiel)

Beispiel zur Parameterverwendung:

```
$letzte_Zeile = tail (“--lines=1”,”/var/log/httpd/access_log”);
```

Kommentare:

Kommentare werden in PERL mit einem vorangestellten „#“ gekennzeichnet.
Der Text hinter diesem Zeichen wird als Kommentar gewertet.

Beispiel:

```
$variable = 12;          # Integer-Wert zuweisen – Kommentar hinter der Anweisung
# Ganze Zeile als Kommentar
$string = “Text“;
```

Auswertung der Parameteruebergabe:

Die Daten vom HTML-Formular werden codiert an das PERL-Script gesendet.
Diese Anweisungen decodieren die Daten und legen sie im Array $FORM ab.

```
# Daten von STDIN einlesen und in $buffer speichern
read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
# Daten anhand des Trennzeichens ‘&’ zerlegen und in Array “pairs” schreiben
@pairs = split(/&/, $buffer);
# Für alle Elemente im Array pairs ausführen.
# Aktueller Wert des Elementes steht in $pair
foreach $pair (@pairs)
 {
 # Name und Wert der Variable trennen und $name und $value zuweisen
 ($name, $value) = split(/=/, $pair);
 # Wert decodieren
 $value =~ tr/+/ /;
 $value =~ s/%([a-fA-F0-9][a-fA-f0-9])/pack("C",hex($1))/eg;
 $value =~ s/<!--(.|\n)*-->//g;
 # Wert der Variable in entsprechenden Platz im Array FORM schreiben
 $FORM{$name} = $value;
 }
```

Zugriff auf decodierte Daten aus dem HTML-Form:

```
$wert = $FORM{ ’Feldname’ };
```

Übersicht PERL

Decodieranweisung an den Browser schicken:

Diese Anweisung ist wichtig, wenn auf dem Browser der, von PERL ausgegebene Text, als HTML-Code verstanden werden soll !!!
Sie muß vor der ersten Ausgabe von Text gesendet werden.

Beispiel:

```
print “CONTENT-type: text/html\n\n“;
```

Ausgabe:

```
print “Hallo, ich bin ein Text“;        # Gibt eine Zeichenkette aus
print “\n“;                             # Gibt eine Zeilenschaltung aus
print “\\“;                             # Gibt einen Backslash aus
print “\““;                             # Gibt eine Anführungszeichen aus
print $name;                            # Gibt den Inhalt der Variable „name“ aus
print @feld;                            # Gibt den Inhalt des Arrays „feld“ komplett aus
print $feld[0];                         # Gibt den ersten Wert des Arrays „feld“ aus
```

Einlesen:

```
read $name;     # Liest von der Eingabeaufforderung ein und speichert die Eingabe
                # in der Variable $name
read $feld[1];  # Liest von der Eingabeaufforderung ein und speichert die Eingabe
                # im 2. Element des Arrays „feld“
```

Übersicht PERL

Mathematik in PERL:

Addieren:

```
$erg = 20 + 10;    oder    $erg = $zahl1 + 20;    oder    $erg = $zahl1 + $zahl2;
```

Subtrahieren:

```
$erg = 50 - 40;    oder    $erg = $zahl1 - 3;    oder    $erg = $zahl1 - $zahl2;
```

Multiplizieren:

```
$erg = 3 * 6;    oder    $erg = $zahl1 * 10;    oder    $erg = $zahl1 * $zahl2;
```

Dividieren:

```
$erg = 21 / 7;    oder    $erg = $zahl1 / 5;    oder    $erg = $zahl1 / $zahl2;
```

Rest einer Division (Modulo):

```
$erg = 18 % 5;    oder    $erg = $zahl1 % 3;    oder    $erg = $zahl1 % $zahl2;
```

Potenz:

```
$erg = 18 ** 2;    oder    $erg = $zahl1 ** 3;    oder    $erg = $zahl1 ** $zahl2;
```

Inkrement (++):

```
$zahl=1;
$erg1 = ++$zahl;          # Inhalt von $erg1 = 2, Inhalt von $zahl = 2
$zahl=1;
$erg1 = $zahl++;          # Inhalt von $erg1 = 1, Inhalt von $zahl = 2
```

Dekrement (--):

```
$zahl=5;
$erg1 = --$zahl;    # Inhalt von $erg1 = 4, Inhalt von $zahl = 4
$zahl=5;
$erg1 = $zahl--;    # Inhalt von $erg1 = 5, Inhalt von $zahl = 4
```

Übersicht PERL

Stringoperationen in PERL:

Zusammenfügen: (mit dem Operator „ . „)

```
$alles = “1. Teil“.$teil2;    oder    $alles = $teil1.“2. Teil“;    oder    $alles = $teil1.$teil2;
```

Vervielfachen: (mit dem Operator „x“)

```
$teil = “z“;
$oft = $teil x 10     # Inhalt der Variable $teil wird 10 mal hintereinander in Variable $oft
                      # geschrieben
```

Länge eines Strings bestimmen:

```
$laenge = length($string);          # Gibt die Länge des String $string zurück
```

Bestimmten Ausdruck in String suchen: (mit dem Operator „ =~ „)

Mit dem Operator „ =~ /Ausdruck/“ kann ein einem String nach Ausdruck gesucht werden. Ist der Ausdruck vorhanden, wird true zurückgeliefert, sonst false

Beispiel 1:

```
$zeichenkette = "Demo";
if($t =~ /mo/)
  { print "true\n" }
else
  { print "false\n" }
```

Ausgabe: “true” da ‘mo’ enhalten

Beispiel 2:

```
$zeichenkette = "Tom";
if($t =~ /mo/)
  { print "true\n" }
else
  { print "false\n" }
```

Ausgabe: “false“ da ’mo’ nicht enthalten

String aufteilen:

Mit dem Befehl „@array = split(/Suchmuster/,Variable);“ wird die Variable bei jedem Auftreten von „Suchmuster“ abgeschnitten und in ein Element des Arrays gespeichert.

Mit dem Befehl „$anzahl = split(/Suchmuster/,Variable);“ wird die Anzahl der erzeugten Teilstrings zurückgegeben.

Beispiel:

```
$zeile = “a-b-c-d“;
$anz_elemente = split(/-/,$zeile);      # Anzahl der Teilstrings wird zurückgegeben
@elemente = split(/-/,$zeile);          # Array wird mit den einzelnen Teilstrings gefüllt
```

String mit bestimmten Trennzeichen zusammensetzten:

Mit dem Befehl „$string = join(’Trennzeichen’,’String1’,’String2’,...,’StringX’);“ wird ein String aus String1 bis StringX erzeugt, der nach jedem Teilstring das Trennzeichen enthält.

Übersicht PERL

Kontrollstrukturen:

```
if ( $a == 0 )
  {
    Anweisungen
    ...
  }
elseif ( $b == 2 )
  {
    Anweisungen
    ...
  }
else
  {
    Anweisungen
    ...
  }
```

```
while ( $a < 20 )
  {
    Anweisungen
    ...
  }
```

(elseif kann entfallen !!)

```
for ( $a = 1; $a > 10; $a++)
  {
    Anweisungen
    ...
  }
```

Vergleichsoperatoren:

Vergleiche von Zahlen	
Operator	**Vergleich**
==	gleich
!=	ungleich
>	größer
>=	größer gleich
<	kleiner
<=	kleiner gleich

Vergleich von Strings	
Operator	**Vergleich**
eq	gleich
ne	ungleich
gt	größer
ge	größer gleich
lt	kleiner
le	kleiner gleich

Sonstige Operatoren:

Logik-Operatoren (für Verknüpfung von Bedingungen)	
Operator	**Vergleich**
!	logisches NICHT
&&	logisches UND
||	logisches ODER
not	logisches NICHT
and	logisches UND
or	logisches ODER
xor	logisches XOR

Bit-Operatoren	
Operator	**Vergleich**
&	bitweises UND
|	bitweises ODER
^	bitweises XOR
~	bitweises Komplement
<<	bitweises schieben links
>>	bitweises schieben rechts

Übersicht PERL

Dateioperationen:

Datei zum lesen öffnen:

```
open(FILEHANDLE,“Dateiname“);    oder    open(FILEHANDLE,“< Dateiname“);
```

Mit diesem Befehl wird die Datei mit dem Namen „Dateinamen“ zum lesen geöffnet und FILEHANDLE zugeordnet. Über FILEHANDLE kann auf die Datei zugegriffen werden.

Datei zum neu schreiben öffnen:

```
open ($filehandle,“> Dateiname“);
```

Mit diesem Befehl wird die Datei mit dem Namen „Dateinamen“ zum neu schreiben geöffnet und FILEHANDLE zugeordnet. Über FILEHANDLE kann auf die Datei zugegriffen werden. **Ist die Datei vorhanden, wird Sie überschrieben !!!**

Datei zum anhängenden schreiben öffnen:

```
open (FILEHANDLE,“>> Dateiname“);
```

Mit diesem Befehl wird die Datei mit dem Namen „Dateinamen“ zum anhängend schreiben geöffnet und FILEHANDLE zugeordnet. **Neue Inhalte werden hinten angehängt. Wenn die Datei nicht vorhanden ist, wird sie angelegt.**

In Datei schreiben:

```
print FILEHANDLE „Neue Zeile in der Datei“;    oder    print FILEHANDLE $neuer_text;
```

Mit diesem Befehl wird eine neue Zeile auf FILEHANDLE und somit in die Datei geschrieben.

Aus Datei lesen:

```
while( defined($datei_zeile = <FILEHANDLE>) )
  {
    Anweisungen
    ...
  }
```

Mit diesen Befehlen wird aus FILEHANDLE und damit aus der Datei gelesen.
Wurde eine Zeile eingelesen, so gibt die Funktion „defined()“ true zurück und die Anweisungen werden ausgeführt.

Datei schließen:

```
close (FILEHANDLE);
```

Dateiname aus Directory auslesen:

```
opendir(DIR,“Verzeichnis“);              # Directory „Verzeichnis“ öffnen
while( $dateiname = readdir(DIR) )       # Dateinamen auslesen
  { print $dateiname.“\n“; }             # Dateinamen ausgeben
closedir(DIR);                           # Directory schließen
```