PERL

Perl ist

- imperativ (angelehnt an "C"),
- interpretiert,
- optimiert auf Verarbeitung von Text,
- ungetypt
- (weitgehend) portabel,
- als Ersatz für Shell-Scripte geeignet.

Beispiel:

```
#!/usr/bin/perl -w
print "Hello, World! \n";
```

Variablen:

Es gibt drei Arten von Variablen:

- **Skalare Variablen:** Für String, Integer, Real
  Bezeichnung: $<name>

- **Arrays (= Listen):** Für Listen beliebiger Werte
  Bezeichnung: @<name>

- **Hashes:** Für ”Mengen” mit assoziativem Zugriff
  Bezeichnung: %<name>

<u>Form der Namen:</u> Ein Buchstabe oder Underscore (_), gefolgt von Buchstaben, Ziffern und Underscores.

Skalare Werte:

- Ganze Zahlen:
  - Dezimal-Notation: `$a = 123; $a = 1_23;`
  - Hex-Notation: `$b = 0xff;`
  - Oktal-Notation: `$c = 0377;`
- Gleitkommazahlen: `$d = 123.4; $e = 5E-10;`

Skalare Werte:

- Strings:

`'abcd'`	Hier wird `\'` in `'` , und `\\` in `\` umgewandelt.
`"abcd"`	Hier werden Variablen und Escape-Sequenzen eingesetzt.
	Beispiel: `$a = "xyz";`
	`$b = "12$a 34";`
	entspricht: `$b = "12xyz 34";`
	Gebräuchliche Escape-Sequenzen:
	`\n` (newline), `\u` (nächster Buchstabe groß),
	`\l` (nächster Buchstabe klein), `\f` (formfeed)
`` `COMMAND` ``	Systemaufruf
	Beispiel: ``$zeit = `date`;``
	`` `ls` `` liefert z.B. `"text.tex\ndata.dat\n"`

Arrays / Listen:

- Literale:

`()`	leere Liste
`(1,2,3)`	
`(1,"ab",17,'d$e',2.7)`	
`[1,'wer',2.4]`	Array-<u>Referenz</u>

- Benutzung: Sei `@a = ("1","2","3")`.

	`$#a` $\widehat{=}$ `2`	letzter Index
	`$a[0]` $\widehat{=}$ `"1"`	Element an Position 0
	`$a[-1]` $\widehat{=}$ `"3"`	letztes Element
Operatoren:	`pop @a` $\widehat{=}$ `"3"`	und nachher `@a = ("1","2")`
(Auswahl)	`push @a,"4"`	nachher `@a=("1","2","3","4")`
	`shift, unshift`	entsprechend für den Anfang von `$a`
	`sort LIST`	sortiert die Liste.
	`splice @ARRAY,OFFSET,LENGTH,LIST`	ersetzt Teile

Beispiel:

```
#!/usr/bin/perl -w

@a = (1,2);

push @a, 3, 4;                     # jetzt: @a = (1,2,3,4)

unshift @a, 0;                     # jetzt: @a = (0,1,2,3,4)

splice @a, 1, 2, "a", "b", "c";    # jetzt:  @a = (0,"a","b","c",3,4)

pop @a;                            # jetzt:  @a = (0,"a","b","c",3)
```

Operatoren:

Operatoren entsprechen oft "C", z.B. `==,!=,<,>`. Ausnahmen sind:

Stringop's:	`.`	String-concat. Z.B. `"ab"."c"` $\widehat{=}$ `"abc"`
	`x`	Multiplikator. Z.B. `"ab"x3` $\widehat{=}$ `"ababab"`
String-Vergleich:	`eq, ne`	Gleichheit, Ungleichheit. (**EQ**ual, **N**ot**E**qual)
	`lt, gt`	$<, >$ für Strings
	`le, ge`	$\leq, \geq$ für Strings
	`cmp`	Vergleich. Ergebnis $\in \{-1, 0, 1\}$.
Numerisch:	`<=>`	Vergleich. Ergebnis $\in \{-1, 0, 1\}$. "Signum(a-b)".
Logisch:	`!, &&, ||`	Normale log. Operatoren.
	`~, &, |, ^`	Normale Bit-Operatoren.
	`not, and, or, xor`	Operatoren mit geringstem Vorrang.

Kontrollstrukturen (Auswahl):

```
    if (<ausdruck>)     { ... }
[[  elseif (<ausdruck>) { ... }  ]
    else                { ... }   ]

    while (<ausdruck>)  { ... }

    for (<ausdr>,<ausdr>,<ausdr>) { ... }

    foreach <variable> (<array>) { ... }
```

Mit `last` kann eine Schleife beendet werden, mit `next` wird mit der nächsten Iteration fortgefahren.

Beispiel:

```
#!/usr/bin/perl -w
# Ausgeben von @a mit Nummerierung ab 100.
# Ende wenn Zeichenkette "ende" erreicht.

@a = ("abcd","xyz","12345","ende","daef");

$n = 100;
foreach $z (@a) {
    if ($z eq "ende") { last; }
    print $n.": ".$z."\n";
    $n++;
}
```

Ausgabe dieses Programms ist:

```
100: abcd
101: xyz
102: 12345
```

Dateioperationen:

<table>
<tr><td>Öffnen:</td><td><code>open IN, "in";</code></td><td>Öffnen zum Lesen.</td></tr>
<tr><td></td><td><code>open OUT,">out";</code></td><td>Öffnen zum Überschreiben.</td></tr>
<tr><td></td><td><code>open OUT,"+<out";</code></td><td>Öffnen zum Lesen und Schreiben.</td></tr>
<tr><td></td><td><code>open OUT,">>out";</code></td><td>Öffnen zum anhängen.</td></tr>
<tr><td></td><td><code>open COUT,"|cmd";</code></td><td>Startet cmd und verbindet dessen Standard-Eingabe mit <code>COUT</code>.</td></tr>
<tr><td></td><td><code>open CIN,"cmd|";</code></td><td>Startet cmd und verbindet dessen Standard-Ausgabe mit <code>CIN</code>.</td></tr>
<tr><td>Lesen:</td><td><code>$a = <IN>;</code></td><td>Eine Zeile nach <code>$a</code> lesen.</td></tr>
<tr><td></td><td><code>@a = <IN>;</code></td><td>Ganze Datei zeilenweise nach <code>@a</code> lesen.</td></tr>
<tr><td>Schreiben:</td><td><code>print OUT $a;</code></td><td>Inhalt von $a in Datei schreiben.</td></tr>
<tr><td></td><td><code>print OUT @a;</code></td><td>Inhalt von @a in Datei schreiben.</td></tr>
<tr><td>Schließen:</td><td><code>close IN;</code></td><td></td></tr>
</table>

Beispiel:

```
#!/usr/bin/perl -w
# Suchen nach der ersten Zeile "xyz\n" in der Datei "datei.txt"
# und Ersetzen dieser Zeile durch "Hier war das gesuchte.\n"

open(IN,"datei.txt");
@a = <IN>;
close(IN);
foreach $z (@a) {
   if ($z eq "xyz\n") {
      print "gefunden! \n";
      $z = "Hier war das gesuchte.\n";
      last;
   }
}
open(OUT,">datei.txt");
print OUT @a;
close(OUT);
```

Beispiel:

```
#!/usr/bin/perl -w
# Verschicken der Datei "mail" als email

open TEXT,"mail";
@text = <TEXT>;
close TEXT;

open MAIL,"|mail wagner\@ti.uni-trier.de";
print MAIL @text;
close MAIL;
```

Reguläre Ausdrücke:

$var =~ m/<muster>/	Ist wahr gdw. <muster> in $var enthalten
$var =~ s/<muster>/<ersatz>/	Ersetzt erstes <muster> in $var durch <ersatz>
$var =~ s/<muster>/<ersatz>/g	Ersetzt alle <muster>.

Sondersymbole (Auswahl):

```
+ ? ^ * . ( ) [ ] \ \n \s \S \d \D
```

Beispiele: Sei $var = "abcdee"

$var =~ m/bcdee+/	ist wahr.
$var =~ m/a+c+/	ist falsch.
$var =~ m/b.*ee	ist wahr.
$var =~ s/cd/xy/	Nacher: $var="abxyee"
$var =~ s/c(.*)/$1x/	Nacher: $var="abdeex"

Beispiel:

```
#!/usr/bin/perl -w
# Ermitteln der pid der Programms "junkbuster"
open(PS,"ps -x |");
@erg = <PS>;
close PS;

foreach $z (@erg) {
   if ($z =~ m/junkbuster/) {
      @komp = split(/\s+/,$z);
      last;
   }
}
if (!defined @komp) { die "not found!\n";}
do {
   $pid = $komp[0];
   shift @komp;
} while (length ($pid) == 0);
print $pid."\n";
```

Hashes:

- Literale: `()` — leeres Hash

  `("aaa" => 1,"bbb" => 3)`

  `("aaa" => 1, 17 => 2, 'd$e' => 2.7)`

  `{'wer' => 2.4, "was" => 1.2}` — Hash-<u>Referenz</u>

- Benutzung: Sei `%a = (0 => "a", 1 => "b", 2 => "c")`.

  `$a{1}` $\widehat{=}$ `"b"` — Element zum Schlüssel 1

  `$a{3}` = `"d"` — fügt Paar `(3 => "d")` ein

Operatoren: `delete $a{1}` — ergibt `%a = (0 => "a", 2 => "c")`

(Auswahl) `keys %a` $\widehat{=}$ `(2,0,1)` (Reihenfolge nur z.B.)

`values %a` $\widehat{=}$ `("c","a","b")` (Reihenfolge nur z.B.)

`each %a` — Iterator

Beispiel für Hashes:

```
#!/usr/bin/perl -w
%a = ( "Au" => "Gold", "Ag" => "Silber");

# Ausgabe des gesamten Hashes:
@aa = %a; print "@aa\n";
# Gibt aus: "Ag Silber Au Gold"

# Zugriff "uber Schl"ussel:
$e = $a{"Ag"};            # Ergebnis:  $e = "Silber"

# Einf"ugen von Elementen:
$a{"Pu"} = "Plutonium";
# Ergebnis: %a = ("Ag"=>"Silber", "Pu"=>"Plutonium", "Au"=>"Gold")

# L"oschen von Elementen:
delete $a{"Au"};  # Ergebnis: %a= ("Ag"=>"Silber", "Pu"=>"Plutonium")
```