Perl

- Skriptsprache
- Wenig Code für einfache Aufgaben
- Sehr viele String-Funktionen
- Keine Typsicherheit, keine Deklarationen notwendig
- Fast immer mehr als ein Weg, eine Aufgabe zu erledigen

Perl-Skript

```
#!/usr/bin/perl

# die folgende Zeile gibt "Hallo Welt" aus
print "Hallo Welt\n";
```

Skript starten

```
perl versuch.pl

perl -w versuch.pl

./versuch.pl
```

Datentypen

Skalar: $scalar
Array: @array
Hash: %hash
Für Arrays und Hashes keine vorherige Initialisierung mit fester (oder variabler) Größe notwendig
my $var: Geltungsbereich auf aktuellen Block beschränken

Skalare

```
$hallo = "Welt";

$wert = 1;

$doublewert = 6.023e23;

$wert2 = $wert1;
```

Arrays

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

@array = ("Hallo", " ", "Welt");

@array = (1, "Hallo", 1.5);


$array[0] = "Hallo";

$array[10] = "Welt";


$wert = $array[7]
```

Hashes

```
%hash = ("Hallo" => 1, "Welt" => 2);

%hash = ("Hallo" => "Welt", "Welt" => "weit");


$hash{"Hallo"} = "Welt";

$hash{"Welt"} = 5;


$hash{"Hallo"} = $hash{"Welt"} * 0.43;
```

Besonderheiten

Besondere Variablen:

```
$0 # Name des Programms
```

```
@ARGV # Argumente des Programms
```

```
$_ # oft aktueller Wert einer (Schleifen-) Anweisung
```

Länge eines Arrays:

```
$length = @array;
```

Index des letzten Elements eines Arrays:

```
$last = $#array;
```

Operatoren

Wie üblich:

+, *, /, -, %

Potenz: **
Konkatenation: .
String-Multiplikation: x
Mit Zuweisung:
+=, *=, /=, -=, .=, x=
Inkrement/Dekrement:
++$a, $a++, --$a, $a--

Operatoren (2)

Logische Operatoren:
&&, ||, !
and, or, not, xor
Vergleich:
==, !=, <, >, <=, >=
$a <=> $b

$a < $b: −1
$a > $b: 1
$a == $b: 0

Bereichsoperator (..)

```
@array = @array[1 .. 5];

@array = @array[-5 .. -1] # letzte 5 Elemente

@array = @array[10 .. $#array] # 10. bis letztes Element
```

„magisches“ Autoinkrement:

```
@alph = ("A" .. "Z");

@twosym = ("AA" .. "ZZ");
```

Kontrollstrukturen

```
if (...){
    ...
}elsif (...){
...
}else{
...
}
```

```
unless (...){
...
}
```

Kontrollstrukturen (2)

```
while(...){
...
}

until(...){
...
}

for($i=0;$i<100;$i++){
...
}

foreach $value (@array){

}
```

Dateihandles

STDIN, STDOUT, STDERR

```
open(SESAM, "<file") # lesen

open(SESAM, ">file") # schreiben

open(SESAM, ">>file") # anhaengen
```

Testen ob Datei existiert:

```
if( -e /pfad/zur/datei ){ ... }

open(SESAM, ">file") or die "Could not open file!\n";
```

Lesen und Schreiben

```
while(<SESAM>){
    print $_;
}

while(<SESAM>){
    print;
}

while(my $line = <SESAM>){
print $line;
}


if( defined( $line = <SESAM> ) ){
    print $line;
}
```

Pattern matching

Konkatenation

```
"Hallo"."␣"."Welt"

$a.$b

"$a␣$b"

"The␣content␣of␣a␣is␣$a"
```

Substrings

```
substr($str, 10, 15) # ab Pos. 10, L"ange 15
substr($str, 10) # ab Pos. 10, bis Ende
substr($str, -10) # letzte 10 Zeichen

substr($str, 0, 0) = "Hallo" # an Anfang
substr($str, 0, 1) = "Hallo" # erstes Zeichen ersetzt
substr($str, -1) = "Hallo" # letztes Zeichen ersetzt


chomp($str) # entfernt newlines am Ende
```

Reguläre Ausdrücke

\	Escape	\n
|	Alternative	a|b
( )	Gruppierung	(a|b)c
–	Bereiche	A–Za–z
[ ]	Zeichenklasse	[ACGT]
^	Anfang	^(Hallo | Welt)
$	Ende	(the | end)$
.	beliebiges Zeichen	(^.$)

Quantifikatoren

$*$	0 - n mal
$+$	1 - n mal
?	0 - 1 mal
{n}	n mal
{n,}	mindestens n mal
{n,k}	n bis k mal

Normalerweise maximal gematcht, mit nachgestellten ? minimal

Match

```
m/pattern/
/pattern/
```

Skalerer Kontext:

```
$str =~ m/pattern/
# gibt wahr (1) oder falsch ("") zur"uck

if($str =~ m/pattern/){ ... }
```

Listenkontext:

```
@array = $str =~ m/pattern/g # alle matches in Liste
```

Modifier:
i: case-insensitive
g: globale Suche, progressive Suche

Match (2)

Progressive Suche:

```
while( $str =~ m/pattern/ig ){ ... }
```

Besondere Funktionen und Variablen:
pos($str): Position des Zeichens nach aktuellem match
length($str): Länge von $str
$&: aktueller match
$‘: Zeichen davor
$’: Zeichen dahinter
pos($str) == **length**($‘) + **length**($&)

Substitution

```
s///

$str =~ s/pattern/replacement/;

$str =~ s/pattern/replacement/gi;


$str =~ s/pattern/\u$&/g;
($strtemp = $str) =~ s/pattern/replacement/g;

for $_ (@lines) { $_ =~ s/pattern/replacement/g }
for (@lines) { s/pattern/replacement/g }
```

Capturing

```
while( $line =~ m/((.*):(.*)) ){
print "$1 => $2, $3";
}

$line =~ s/(.*)-(.*)/-$2+$1/g;

$line = 1234567890;
while ($line =~ s/([0-9]+)([0-9]{3})($|)/$1 $2$3/g){pri

1234567 890
1234 567 890
1 234 567 890

while ($line =~ s/([0-9]+)([0-9]{3})($|)/$1 $2$3/g){}

1 while ($line =~ s/([0-9]+)([0-9]{3})($|)/$1 $2$3/g);
```

Übersetzung

```
tr///
$str =~ tr/list/list/
$str =~ tr/A–Z/a–z/;
```

Achtung: Keine Regulären Ausdrücke, keine Muster, sondern Listen!

Beispiele

```
$str = "ACATGATAGGCGTATA";

if( $str =~ m/((TA){2}|(TA(C|G)+))/){
        print $&."\n";
}
```

Beispiele

```
$str = "ACATGATAGGCGTATA";

if( $str =~ m/((TA){2}|(TA(C|G)+))/){
        print $&."\n";
}
```

Ausgabe:
TAGGCG

Beispiele (2)

```
$str = "ACATGATAGGCGTATA";

if( $str =~ m/((TA){2}|(TA(C|G)+?))/){
        print $&."\n";
}
```

Beispiele (2)

```
$str = "ACATGATAGGCGTATA";

if( $str =~ m/((TA){2}|(TA(C|G)+?))/){
        print $&."\n";
}
```

Ausgabe:
TAG

Beispiele (3)

```
$str = "ACATGATATATA";

if( $str =~ m/((TA){2}|(TA(C|G)+?))/){
        print $&."\n";
}
```

Beispiele (3)

```
$str = "ACATGATATATA";

if( $str =~ m/((TA){2}|(TA(C|G)+?))/){
        print $&."\n";
}
```

Ausgabe:
TATA

Beispiele (4)

```
$str = "ACATGATAGGCGTATA";

while( $str =~ m/((TA){2}|(TA(C|G)+))/g){
        print $&."\n";
}
```

Beispiele (4)

```
$str = "ACATGATAGGCGTATA";

while( $str =~ m/((TA){2}|(TA(C|G)+))/g){
        print $&."\n";
}
```

Ausgabe:
TAGGCG
TATA

Beispiele (5)

```
$str = "ACATGATAGGCGTATA";

$str =~ s/A/T/g;
$str =~ s/C/G/g;
$str =~ s/G/C/g;
$str =~ s/T/A/g;

print $str."\n";
```

Beispiele (5)

```
$str = "ACATGATAGGCGTATA";

$str =~ s/A/T/g;
$str =~ s/C/G/g;
$str =~ s/G/C/g;
$str =~ s/T/A/g;

print $str."\n";
```

Ausgabe:
ACAACAAACCCCAAAA

Beispiele (6)

```
$str = "ACATGATAGGCGTATA";

$str =~ tr/ACGT/TGCA/;

print $str."\n";
```

Beispiele (6)

```
$str = "ACATGATAGGCGTATA";

$str =~ tr/ACGT/TGCA/;

print $str."\n";
```

Ausgabe:
TGTACTATCCGCATAT