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$3/g){print "$line\n"; }

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