

Perl-Praxis  
**Reguläre Ausdrücke**

Jörn Clausen, Jan Krüger  
Jens Reeder, Alex Sczyrba

AG Praktische Informatik  
Technische Fakultät  
Universität Bielefeld

`{asczyrba, jkrueger}@TechFak.Uni-Bielefeld.DE`

# Übersicht

- Reguläre Ausdrücke
- Muster suchen
- Muster finden

## zur Erinnerung

- Perl := Practical Extraction and Report Language
- Text-Dateien zeilenweise einlesen
- Zeilen
  - nach Mustern durchsuchen
  - in Worte zerlegen
- Textstücke ersetzen

## Reguläre Ausdrücke

- beschreiben einfache Sprachen
- „schwächste“ Chomsky-Grammatik
- in Perl: *regular expressions* (RE)
- REs in Perl deutlich mächtiger als Reguläre Sprachen

Die Chomsky-Hierarchie:

Typ-0-Sprachen	allgemeine Regelsprachen
Typ-1-Sprachen	kontextsensitive Sprachen
Typ-2-Sprachen	kontextfreie Sprachen
Typ-3-Sprachen	reguläre Sprachen

## regular expressions

- Suchen:

```
m/Gold/
```

- Kurzform:

```
/Gold/
```

- Ersetzen:

```
s/Blei/Gold/
```

- RE an Ausdruck binden: =~ (*match operator*)

```
$story = 'In a hole in the ground there lived a boggit.';
if ($story =~ /ground/) { ... }
$story =~ s/boggit/hobbit/;
```

Wenn der Suchausdruck mit `m` eingeleitet wird, kann (fast) jedes beliebige andere Zeichen anstatt des *slash* als „Klammer“ verwendet werden. Dies sind alles äquivalente Ausdrücke:

```
m#Gold#
m|Gold|
m!Gold!
```

Dies ist vor allem dann Hilfreich, wenn der slash Teil des Suchmusters ist, z.B. wenn Verzeichnispfade oder URLs verarbeitet werden sollen. Sonst müßte ein slash, der nicht Begrenzer des Suchmusters ist, als `\` maskiert werden. Analog gilt dies für die Ersetzung mit `s///`.

## modifier

- beeinflussen Verhalten von `m//` und `s///`
- nur die zwei wichtigsten:

```
m/gold/i      # case insensitive  
s/Blei/Gold/g # global
```

## Aufgaben

- Wir verwenden im weiteren Verlauf die GenBank Datei NC\_000913.gbk um Texte zu suchen und zu manipulieren. Es handelt sich hierbei um den GenBank-Eintrag zum *E. coli* K12 Genom. Die Datei liegt unter `/vol/lehre/PerlPraxis/2005`
- Sieh Dir nochmal den Aufbau eines GenBank Files an.
- Wieviele Features *gene* und wieviele *CDSes* sind in NC\_000913.gbk enthalten? (Wieso gibt es hier einen Unterschied?)
- Ersetze Feature *gene* durch *Gen*.

```
open(GENBANK, 'NC_000913.gbk') || die "can't open file: $\n";
while ($line = <GENBANK>) {
    $line =~ s/ gene / gen /g;
    print $line;
}
close(GENBANK);
```

- Namen ersetzen (Vorsicht: nur den Namen des Features ändern!):

Die Datei enthält 4395 genes und 4279 CDSes.

```
open(GENBANK, 'NC_000913.gbk') || die "can't open file: $\n";
$count = 0;
while ($line = <GENBANK>) {
    $count_gene++ if $line =~ / gene /;
    $count_cds++ if $line =~ / CDS /;
}
close(GENBANK);
print "found $count_gene genes and $count_cds CDSes\n";
```

- Nach Gold suchen:

## regular expressions, cont.

- Alternativen:

```
m/Huey|Dewey|Louie/
```

- Gruppierung:

```
m/(Hu|Dew)ey|Louie/
```

- Quantoren

m/ab?a/	# aa, aba
m/ab*a/	# aa, aba, abba, abbba, abbbba, ...
m/ab+a/	# aba, abba, abbba, abbbba, ...
m/ab{3,6}a/	# abbbba, abbbba, abbbba, abbbba
m/a(bab)+a/	# ababa, ababbaba, ababbabbaba, ...

## regular expressions, cont.

- Zeichenklassen

```
m/hello\s+world/ # whitespace
m/es ist \d+ Uhr/ # digits
m/name: \w+/     # letters (words)
```

- „Gegenteile“: \S, \D, \W

- selbstgemachte Zeichenklassen

```
m/M[ea][iy]er/ # Meier, Meyer, Maier, Mayer
m/[a-z]{2,8}/  # Account-Namen
m/[A-Z][^0-9]+/
```

- paßt auf alles: .

## Aufgaben

- Lies die Datei „NC\_000913.gbk“ zeilenweise ein.
  - Gib alle Zeilen aus, die ein *gene*-Feature auf dem Gegenstrang (complement) annotieren.
  - Wieviele *Aminopeptidasen* sind in *E. coli* annotiert? (Ezymklasse: 3.4.11.-)

```
while ($line = <GENBANK>) {  
  print $line if $line =~ #/ECnumber="3\.4\.11\.11\.\d+"#;  
}
```

- Enzymklasse 3.4.11.- herausfiltern:

```
while ($line = <GENBANK>) {  
  print $line if $line =~ /\s+gene\s+complement/;  
}
```

- *gene*-Features:

## Anker

- Muster an bestimmte Position binden
- Zeilenanfang, Zeilenende:

```
m/^LOCUS.+/      # LOCUS Zeile aus GenBank Datei  
s/\s+$/ /        # remove trailing whitespace  
m/^\d+ \d+ \d+$/ # 3d coords
```

- Wortzwischenraum:

```
m/\bmit\b/       # "nicht mit mir", "Kommen Sie mit!"  
m/\bmit\B/       # "mittendrin", nicht "vermitteln"
```

## pattern capturing

- bis jetzt: Muster kommt in Text vor!
- aber: Wie sah der Treffer aus?
- interessanten Bereich markieren:

```
m/^LOCUS\s+(\S+)/  
m/^VERSION\s+(\S+)\.(\d+)\s+GI:(\d+)\$/
```

- Treffer landen in \$1, \$2, ...

```
$line =~ m/^VERSION\s+(\S+)\.(\d+)\s+GI:(\d+)\$/  
print "Accession: $1, Version: $2, GI: $3\n";
```

- Quantoren richtig setzen: (\w)+ vs. (\w+)

# Aufgaben

- In NC\_000913.gbk finden sich Gen-Annotationen der Form

```
gene          5234..5530
gene          complement(1182840..1183667)
```

Extrahiere die Positionen aller Gene.

- Genpositionen:

```
while ($line = <GENBANK>) {
  if ($line =~ /\s+gene[\^0-9]+(\d+)\.\.\d+(\d+)//) {
    $start = $1;
    $stop = $2;
    print "Found gene from $start to $stop\n";
  }
}
```

## pattern capturing, cont.

- etwas eleganter: mit Zuweisung

```
($x, $y, $z) = ($line =~ m/^(\\d+) (\\d+) (\\d+)$/);
```

- Muster muß vollständig passen

```
if (defined($x)) {  
    print "point at $x,$y,$z\\n";  
}
```

- Unterschied zwischen grouping und capturing:

```
($dir, $who) = ($header =~ m/^(From|To): (.+)/);  
($who) = ($header =~ m/^(?:From|To): (.+)/);
```

### Der Ausdruck

```
($dir, $who) = ($header =~ m/^(From|To): (.+)/);
```

captured auch `From\To`, der folgende nicht:

```
($who) = ($header =~ m/^(?:From|To): (.+)/);
```

## greedy matches

- Was passiert, wenn pattern nicht eindeutig ist?

```
$text = "aaaaaaaaa";  
($w1, $w2) = ($text =~ /(a+)(a+)/);
```

- ausprobieren:

```
/(a+)(a*)/  
/(a*)(a+)/  
/(a*)(a*)/  
/(a?)(a*)/  
/(a{2,4})(a*)/
```

- Setze ? hinter den ersten quantifier:

```
+? *? ?? {2,4}?
```

- greedy:

```
+ +: >>aaaaaaaa<< >>a<<  
+ *: >>aaaaaaaa<< >><<  
* +: >>aaaaaaaa<< >>a<<  
* *: >>aaaaaaaa<< >><<  
? *: >>a<< >>aaaaaaaa<<  
{2,4} *: >>aaaa<< >>aaaa<<
```

- non-greedy:

```
+? +: >>a<< >>aaaaaaaa<<  
+? *: >>a<< >>aaaaaaaa<<  
*? +: >><< >>aaaaaaaa<<  
*? *: >><< >>aaaaaaaa<<  
?? *: >><< >>aaaaaaaa<<  
{2,4}? *: >>aa<< >>aaaaaaaa<<
```

## Text zerteilen

- `join`: Array-Elemente zu String vereinen
- entgegengesetzte Operation: `split`
- Trennung durch pattern:

```
$story = "In a hole in the ground there lived a hobbit.";  
@words = split(/\s/, $story);
```

# Aufgaben

- Trenne den Satz

**„In a hole in the ground there lived a hobbit.“**

mit den folgenden Mustern. Welche „Worte“ entstehen dabei?

```
//  
//  
/\s*/  
/\b/  
/\B/
```

Wie „groß“ sind die patterns, an denen getrennt wird?

- verschiedene Trennmuster:

```
// In a hole in the ground there lived a hobbit.  
// In|a|h|o|l|e| |i|n| |t|h|e| |g|r|o|u|n|d| |t|h|e|r|e|  
// |t|h|e|r|e| |l|i|v|e|d| |a| |h|o|b|b|i|t|.|  
/\s*/ In a hole in the ground there lived a hobbit.  
/\b/ In|a|h|o|l|e| |i|n| |t|h|e|r|e| |l|i|v|e|d| |a| |h|o|b|b|i|t|.|  
/\B/ In a hole in the ground there lived a hobbit.  
In|a|h|o|l|e| |i|n| |t|h|e|r|e| |l|i|v|e|d| |a| |h|o|b|b|i|t|.|
```

Das pattern / / ist ein Zeichen groß. Die patterns //, /\b/ und /\B/ haben keine Ausdehnung. Das pattern \s\* hat variable Größe.