

# Reguläre Ausdrücke

Silke Trißl

Wissensmanagement in der Bioinformatik



## Sinn und Ziel

- *Reguläre Ausdrücke* sind eine Möglichkeit eine Menge von Strings aufgrund von gemeinsamen Merkmalen zu beschreiben.
  - Suche von Strings/Texten/Daten
  - Manipulation von Strings/Texten/Daten
- Bibliothek: `java.util.regex`
  - Pattern
  - Matcher
  - `PatternSyntaxException`

# Zwei Schritte

## 1. Definieren eines regulären Ausdrucks

- Ein Pattern, das beschreibt was gemacht werden soll

```
Pattern p = Pattern.compile("ist");
```

## 2. Anwenden des regulären Ausdrucks auf einen String

- Pattern wird auf den einen String angewendet

```
Matcher m = p.matcher("bist");
```

# Anwendung

```
Pattern p = Pattern.compile("ist", RegexOptions);  
Matcher m = p.matcher("bist");  
  
while(m.find()) {  
    System.out.println("Found: '" + m.group() +  
        "' at (" + m.start() + ", "  
        + m.end() + ")");  
}
```

Findet das nächste Match in der Sequenz  
returns boolean

Gibt die Sequenz zurück, die gematcht hat  
returns String

Start- und Endposition des Matches  
returns int

## ■ m.matches()

- Versucht den ganzen String zu matchen

# Pattern

- Zeichen wie angegeben

```
Regex = ist
Input = bist
Found 'ist' at (1,4)
```

- '.' bedeutet Wildcard = jedes beliebige Zeichen

```
Regex = i.t          Regex = i\\.t
Input = bist es gilt Input = bist es gilt
Found 'ist' at (1,4) No Match found
Found 'ilt' at (9,12)
```

- durch '\\' bedeutet es wieder '.'

- Andere Meta-characters in Java: ( [ { \ ^ \$ | ) ? \* + .

# Pattern – cont. –

- Zeichen können in [...] gruppiert werden

- einfach [abc] a, b, or c
- Negierung [^abc] Jedes Zeichen, außer a, b, oder c
- Intervall [a-zA-Z] a bis z oder A bis Z
- Union [a-d[m-p]] a bis d, oder m bis p
- Intersection [a-z&&[def]] d, e, oder f
- Subtraction [a-z&&[^bc]] a bis z, außer b,c
- Beispiel [a-z&&[^m-p]] a bis z, und nicht m bis p

## Beispiele

- einfach

```
Regex = [abc][isa]st
Input = bist isst
Found 'bist' at (0,4)
```

- Negierung

```
Regex = [^0123459][isa]st
Input = bist isst
Found 'bist' at (0,4)
Found 'isst' at (5,9)
```

- Intervall

```
Regex = [^0-9].st
Input = bist isst
Found 'bist' at (0,4)
Found 'isst' at (5,9)
```

- Subtraction

```
Regex = [a-z&&[^aeiou]]st
Input = bist isst
Found 'sst' at (6,9)
```

## Vordefinierte Characterklassen

- . Irgendein Zeichen
- \d eine Zahl = [0-9]
- \D keine Zahl = [^0-9]
- \s whitespace = [ \t\n\r\f] (Note: original image has \x0B)
- \S kein whitespace = [^\s]
- \w ein character = [a-zA-Z\_0-9]
- \W kein character = [^\w]

# Quantifier

- X? X 0 oder 1 Vorkommen
- X\* X 0 oder n
- X+ X 1 oder n
- X{n} X genau n mal
- X{n,} X mindestens n mal
- X{n,m} X zwischen n und m mal

- Reguläre Quantifier sind 'greedy'
  - matchen so viele Zeichen wie möglich
  - '?' mit einfügen
    - gibt nur noch die kürzeste Möglichkeit zurück

# Beispiele

- Vordefinierte Klassen

```
Regex = A\\d\\d\\d\\dA
Input = A123A456A789
Found 'A123A' at (0,5)
```

```
Regex = A\\d+\\S4
Input = A123A456A789
Found 'A123A4' at (0,5)
```

- greedy

```
Regex = A.*A
Input = A123A456A789
Found 'A123A456A' at (0,9)
```

- nicht greedy

```
Regex = A.*?A
Input = A123A456A789
Found 'A123A' at (0,5)
```

## Weitere Optionen

- ^ Anfang einer Zeile
- \$ Ende einer Zeile
- \b Wortgrenze
- \B keine Wortgrenze

```
Regex = A\d+$  
Input = A123A456A789  
Found 'A789' at (8,12)
```

```
Regex = \\bbis\\b  
Input = bist bis morgen  
Found 'bis' at (5,8)
```

```
Regex = \\bbis\\B  
Input = bist bis morgen  
Found 'bis' at (0,3)
```

## Aufgabe



- Schreibt einen regulären Ausdruck, mit dem ihr verschiedene Datumsformate finden könnt
  - Beispiele:
    - 23.02.2005
    - 23-02-05
    - 23/2/05
    - ...

## OR Operator '|'

- Alternative Terme können durch '|' getrennt werden

```
RegEx = d|c|e  
Input = abc  
Found 'c' at (2,3)
```

```
RegEx = cd|bc|de  
Input = abc  
Found 'bc' at (1,3)
```

## Gruppierung

- ( ) wird benutzt um Gruppen zusammenzufassen
  - Jede Gruppe hat einen Index-Wert (startet bei 1)
  - können durch '\X' angesprochen werden, wobei X die Gruppennummer ist

```
RegEx = abc*  
Input = abcabcc  
Found 'abc' at (0,3)  
Found 'abcc' at (4,7)
```

```
RegEx = (\\w+)\\s\\1  
Input = dem dem  
Found 'dem dem' at (0,7)
```

```
RegEx = (abc)*  
Input = abcabcc  
Found 'abcabc' at (0,6)
```

```
RegEx = (\\d+)\\.\\.\\1\\.\\.\\d{2,4}  
Input = 03.03.05  
Found '03.03.05' at (0,8)
```

# Aufgabe

- ... und noch ein Pattern, um alle paarweisen HTML tags zu finden
  - Beispiel
    - `<head><title>My first web page</title></head>`
    - `<body><p>My name is ... </p><br>`
    - `<p>I work on ... </p> ... </body>`
  - gebt jeweils das entsprechende Paar aus
  - Zusatzaufgabe: gebt alles in XML-Style aus

```
<head>
  <title>My first web page</title>
</head>
<body>
  <p> ...</p>
  <p> ...</p>
...
</body>
```

# Anwendung

- Parsen von Texten
  - finde das Datum, daraus Tag, Monat, Jahr
  - 'wenn eine Zeile so oder so ähnlich anfängt, dann ...'
- Pattern in der Biologie
  - Homologe oder paraloge Proteine haben meist eine oder mehrere Regionen in der sie sich sehr ähnlich sind
  - Diese Regionen werden Pattern genannt
  - Prosite enthält Proteinfamilien und deren Pattern
    - <http://us.expasy.org/prosite/>



# Aufgabe

---

- Kazal serine protease inhibitors family signature
    - C-x(4)-{C}-x(2)-C-x-{A}-x(4)-Y-x(3)-C-x(2,3)-C
  - Sugar transport proteins signatures
    - [LIVMF]-x-G-[LIVMFA]-x(2)-G-x(8)-[LIFY]-x(2)-[EQ]-x(6)-[RK]
  - Beta-amylase active sites
    - G-x-[SA]-G-E-[LIVM]-R-Y-P-S-Y
  - Sucht in den 5 gegebenen Proteinsequenzen nach den Pattern aus Prosite
    - gebt dabei jeweils die gematchten Sequenzen aus
-