Regular Expressions: Searching strings for patterns

April 24, 2008

Introduction

Programming tools that can search (long) strings for specific characters and patterns

- Unix: grep, awk, sed
- Perl, a language (originally) devoted to text and file manipulation
- Matlab: regexp, regexpi, regexprep

Terminology:

Regular expression: a formula for matching strings that follow a specified pattern, made up with the following constructs

- Character expression: formula to match a single character with prescribed properties
- Modifier: the number of times the character expression is applied
- Quantifier: greedy or lazy
- LookAround operator: specify patterns that must occur before/after the match
- Positional operators: "or"
- Grouping operators: 
- Tokens:

Character expression patterns

Single character classes

- ce = '[WxYz6]' % any one of the 5 listed chars
- ce = '[m-y]' % any one of m, n, o, ... x, y
- ce = '[2-8]' % any one of 2, 3, 4, ... 7, 8
- ce = 'a' % any single character
- ce = '[ctwH]' % any character except c, t, w, H
- ce = '.' % any whitespace character
- ce = '[a-zA-Z_0-9]' % any "word" char, [a-zA-Z_0-9]
- ce = 't' % horizontal tab character
- ce = '\n' % newline character

>> ce = 'and' % the 3 character sequence
>> ce = 'f or'; % the 4 character: f-space-o-r
>> ce = 'Mr.' % the 3 character sequence 'Mr.'
>> ce = 'g.[tp]' % g, any 3 characters, t or p

Greedy/Lazy

str = 'I wished I was living in Mississippi';

cel = 'i[^i]+' % i followed by one or more not-i
regexp(str,cel,'match')

Note: behavior is greedy. Tries to match as much of str that is possible (i, then many not-i, until it encounters an i)

str = 'I wished I was living in Mississippi';

cel2 = 'i[^i]+?' % ? quantifies the + as "lazy"
regexp(str,cel2,'match')

Note: now behavior is lazy – each match as short as possible

LookAround Operators

LookAhead

ce = 'Andy(\?= Packard)'
match 'Andy' only if followed by 'space-Packard'

Negative LookAhead

ce = 'UC(?!LA)'
match 'UC' only if not followed by 'LA'

LookBehind

ce = '(\?=Hewlett)?Packard)'
match 'Packard' only if preceded by 'Hewlett' or 'Hewlet', with or without a space at end

Negative LookBehind

ce = '(\?<flexible )deadline'
match 'deadline' only if not preceded by 'flexible '

Character modifiers

<table>
<thead>
<tr>
<th>ce</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ce = 'z?'</td>
<td>% zero or one z</td>
</tr>
<tr>
<td>ce = 'Y*'</td>
<td>% one or more Y</td>
</tr>
<tr>
<td>ce = 'D+'</td>
<td>% one or more D</td>
</tr>
<tr>
<td>ce = 'p{2}'</td>
<td>% two p (ie., pp)</td>
</tr>
<tr>
<td>ce = 'r{3,}'</td>
<td>% at least 3 r (rrr, or rrrr, or...)</td>
</tr>
<tr>
<td>ce = 'H{2,4}'</td>
<td>% HH or HHH or HHHH</td>
</tr>
</tbody>
</table>

Multiple character patterns

<table>
<thead>
<tr>
<th>ce</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ce = 's'</td>
<td>% any character</td>
</tr>
<tr>
<td>ce = 'f'</td>
<td>% any character besides 'f'</td>
</tr>
<tr>
<td>ce = 'd'</td>
<td>% any numeric digit</td>
</tr>
<tr>
<td>ce = 'w'</td>
<td>% any &quot;word&quot; char, [a-zA-Z_0-9]</td>
</tr>
<tr>
<td>ce = '\t'</td>
<td>% horizontal tab character</td>
</tr>
<tr>
<td>ce = '\n'</td>
<td>% newline character</td>
</tr>
</tbody>
</table>

Greedy/Lazy

str = 'I wished I was living in Mississippi';

cel = 'i[^i]+' % i followed by one or more not-i
regexp(str,cel,'match')

Note: behavior is greedy. Tries to match as much of str that is possible (i, then many not-i, until it encounters an i)

str = 'I wished I was living in Mississippi';

cel2 = 'i[^i]+?' % ? quantifies the + as "lazy"
regexp(str,cel2,'match')

Note: now behavior is lazy – each match as short as possible

LookAround Operators

LookAhead

ce = 'Andy(\?= Packard)'
match 'Andy' only if followed by 'space-Packard'

Negative LookAhead

ce = 'UC(?!LA)'
match 'UC' only if not followed by 'LA'

LookBehind

ce = '(\?=Hewlett)?Packard)'
match 'Packard' only if preceded by 'Hewlett' or 'Hewlet', with or without a space at end

Negative LookBehind

ce = '(\?<flexible )deadline'
match 'deadline' only if not preceded by 'flexible '
Positional Operators

Beginning of the string:
  \ce = 'To be'
  match 'To be' only if it is at the beginning of the string

End of the string:
  \ce = ' (\d+)\$
  match one or more digits at the end of the string

Beginning of a word:
  \ce = '\<ard'
  match 'ard' only if it is at the beginning of a word

End of a word:
  \ce = '\ard> '
  Match 'ard' only if it is at the end of a word

Exact word:
  \ce = '\<part> '
  Match 'part' only if exactly matches a word

Logical Operator

'OR' operator:
  \ce = '\Jan|\Feb'
  match 'Jan' at the beginning of a word, or 'Feb' at the beginning a word

Tokens

We have seen that parenthesis can be used to group patterns, and establish precedence.

Parenthesis also implicitly define variables (called tokens) which can be used in the character expression.

Match two 4-digit strings separated by characters
  \ce1 = '\d{4}.\d{4}'

Match 4 digit string that occurs twice w/ characters in between
  \ce2 = '\(\d{4}\)\ \d{4}\1'

str = 'the year 1999Feb had Feb1999 stuff';
[m,t] = regexp(str,\ce1,'match','tokens');
m -> {'1999Feb had Feb1999'}
t -> {'1999' 'Feb'}