Patterns, Patterns and More Patterns

Exploiting Perl's built-in regular expression technology

Pattern Basics

-
- What is a regular expression?

```
/even/


eleven            # matches at end of word
eventually        # matches at start of word
even Stevens      # matches twice: an entire word and within a word

heaven            # 'a' breaks the pattern
Even              # uppercase 'E' breaks the pattern
EVEN              # all uppercase breaks the pattern
eveN              # uppercase 'N' breaks the pattern
leave             # not even close!
Steve not here    # space between 'Steve' and 'not' breaks the pattern
```

What makes regular expressions so special?

```
my $pattern = "even";

my $string = "do the words heaven and eleven match?";

if ( find_it( $pattern, $string ) )
{
    print "A match was found.\n";
}
else
{
    print "No match was found.\n";
}
```

find_it the Perl way

```
my $string = "do the words heaven and eleven match?";

if ( $string =~ /even/ )
{
    print "A match was found.\n";
}
else
{
    print "No match was found.\n";
}
```

Maxim 7.1

Use a regular expression to specify what you want to find, not how to find it

Introducing The Pattern Metacharacters

The + repetition metacharacter

```
/T+/

T
TTTTTT
TT

t
this and that
hello
ttttttttt
```

More repetition

```
/ela+/

elation
elaaaaaaaa

/(ela)+/

elaelaelaela
ela


/\(ela\)+/

(ela))))))

(ela(ela(ela
```

The | alternation metacharacter

```
/0|1|2|3|4|5|6|7|8|9/

0123456789
there's a 0 in here somewhere
My telephone number is: 212-555-1029



/a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z/

/A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z/
```

Metacharacter shorthand and character classes

```
/0|1|2|3|4|5|6|7|8|9/
/[0123456789]/
/[aeiou]/
/a|e|i|o|u/
/[^aeiou]/
/[0123456789]/
/[0-9]/
/[a-z]/
/[A-Z]/
/[-A-Z]/

/[BCFHST][aeiou][mty]/

Bat                 Hog
Hit                 Can
Tot                 May
Cut                 bat
Say

/[BbCcFfHhSsTt][aeiou][mty]/
```

More metacharacter shorthand

```
/[0-9]/
/\d/

/[a-zA-Z0-9_]/
/\w/

/\s/
/[^ \t\n\r\f]/

/\D/


/[0-9][^ \t\n\r\f][a-zA-Z0-9_][a-zA-Z0-9_][^0-9]/

/\d\s\w\w\D/
```

Maxim 7.2

Use regular expression shorthand to reduce the risk of error

More repetition

```
/\w+/

/\d\s\w+\D/

/\d\s\w{2}\D/

/\d\s\w{2,4}\D/

/\d\s\w{2,}\D/
```

The ? and * optional metacharacters

```
/[Bb]art?/

bar
Bar
bart
Bart

/[Bb]art*/

bar
Bart
barttt
Bartttttttttttttttttttt!!!


/p*/
```

The any character metacharacter

```
/[Bb]ar./

barb
bark
barking
embarking
barn
Bart
Barry

/[Bb]ar.?/
```

Anchors

The \b word boundary metacharacter

```
/\bbark\b/

That dog sure has a loud bark, doesn't it?

That dog's barking is driving me crazy!

/\Bbark\B/
```

The ^ start-of-line metacharacter

```
/^Bioinformatics/

Bioinformatics, Biocomputing and Perl is a great book.

For a great introduction to Bioinformatics, see
    Moorhouse, Barry (2004).
```

The $ end-of-line metacharacter

```
/Perl$/

My favourite programming language is Perl

Is Perl your favourite programming language?

/^$/
```

The Binding Operators

```
#! /usr/bin/perl -w

# The 'simplepat' program - simple regular expression example.

while ( <> )
{
    print "Got a blank line.\n" if /^$/;
    print "Line has a curly brace.\n" if /[}{]/;
    print "Line contains 'program'.\n" if /\bprogram\b/;
}
```

Results from simplepat ...

```
$ perl simplepat simplepat

Got a blank line.
Line contains 'program'.
Got a blank line.
Line has a curly brace.
Line has a curly brace.
Line contains 'program'.
Line has a curly brace.
```

To Match or Not To Match ...

```
if ( $line =~ /^$/ )

if ( $line !~ /^$/ )
```

Remembering What Was Matched

```
/(ela)+/
```

```
#! /usr/bin/perl -w

# The 'grouping' program - demonstrates the effect
# of parentheses.

while ( my $line = <> )
{
    $line =~ /\w+ (\w+) \w+ (\w+)/;

    print "Second word: '$1' on line $..\n" if defined $1;
    print "Fourth word: '$2' on line $..\n" if defined $2;
}
```

Results from grouping ...

```
This is a sample file for use with
the grouping program that is included
with the Patterns
Patterns and More Patterns chapter
from Bioinformatics, Biocomputing and Perl.


$ perl grouping test.group.data


Second word: 'is' on line 1.
Fourth word: 'sample' on line 1.
Second word: 'grouping' on line 2.
Fourth word: 'that' on line 2.
Second word: 'and' on line 4.
Fourth word: 'Patterns' on line 4.
```

The grouping2 program

```
#! /usr/bin/perl -w

# The 'grouping2' program - demonstrates the effect of
# more parentheses.

while ( my $line = <> )
{
    $line =~ /\w+ ((\w+) \w+ (\w+))/;

    print "Three words: '$1' on line $..\n" if defined $1;
    print "Second word: '$2' on line $..\n" if defined $2;
    print "Fourth word: '$3' on line $..\n" if defined $3;
}
```

Results from grouping2 ...

```
Three words: 'is a sample' on line 1.
Second word: 'is' on line 1.
Fourth word: 'sample' on line 1.
Three words: 'grouping program that' on line 2.
Second word: 'grouping' on line 2.
Fourth word: 'that' on line 2.
Three words: 'and More Patterns' on line 4.
Second word: 'and' on line 4.
Fourth word: 'Patterns' on line 4.
```

Maxim 7.3

When working with nested parentheses, count the opening parentheses, starting with the leftmost, to determine which parts of the pattern are assigned to which after-match variables

Greedy By Default

```
/(.+), Bart/

Get over here, now, Bart! Do you hear me, Bart?

Get over here, now, Bart! Do you hear me

/(.+?), Bart/

Get over here, now
```

Alternative Pattern Delimiters

```
/usr/bin/perl

//\w+/\w+/\w+/

/\/\w+\/\w+\/\w+/

/\/(\w+)\/(\w+)\/(\w+)/

m#/\w+/\w+/\w+#

m#/(\w+)/(\w+)/(\w+)#

m{ }
m< >
m[ ]
m( )

/even/
m/even/
```

Another Useful Utility

```
sub biodb2mysql {
#
# Given: a date in DD-MMM-YYYY format.
# Return: a date in YYYY-MM-DD format.
#

    my $original = shift;

    $original =~ /(\d\d)-(\w\w\w)-(\d\d\d\d)/;

    my ( $day, $month, $year ) = ( $1, $2, $3 );
```

biodb2mysql subroutine, cont.

```
    $month = '01' if $month eq 'JAN';
    $month = '02' if $month eq 'FEB';
    $month = '03' if $month eq 'MAR';
    $month = '04' if $month eq 'APR';
    $month = '05' if $month eq 'MAY';
    $month = '06' if $month eq 'JUN';
    $month = '07' if $month eq 'JUL';
    $month = '08' if $month eq 'AUG';
    $month = '09' if $month eq 'SEP';
    $month = '10' if $month eq 'OCT';
    $month = '11' if $month eq 'NOV';
    $month = '12' if $month eq 'DEC';

    return $year . '-' . $month . '-' . $day;
}
```

Alternate biodb2mysql patterns

```
/(\d{2})-(\w{3})-(\d{4})/

/(\d+)-(\w+)-(\d+)/
```

Substitutions: Search And Replace

```
s/these/those/

Give me some of these, these, these and these. Thanks.

Give me some of those, these, these and these. Thanks.

s/these/those/g

Give me some of those, those, those and those. Thanks.

s/these/those/gi
```

Substituting for whitespace

```
s/^\s+//

s/\s+$//

s/\s+/ /g
```

Finding A Sequence

```
gccacagatt acaggaagtc atatttttag acctaaatca ctatcctcta tctttcagca   60
agaaaagaac atctacttgg tttcgttccc tatccaagat tcagatggtg aaacgagtga   120
tcatgcacct gatgaacgtg caaaaccaca gtcaagccat gacaaccccg atctacagtt   180
    .
    .
    .
gcatctgtct gtatccgcaa cctaaaatca gtgctttaga agccgtggac attgatttag   6660
gtacgtgtag agcaagactt aaatttgtac gtgaaactaa aagccagttg tatgcattag   6720
ctttttcaat ttgtataacg tataacgtat ataatgttaa ttttagattt tcttacaact   6780
tgatttaaaa gtttaagatt catgtattta tattttatgg ggggacatga atagatct     6838
```

```
if ( $sequence =~ /acttaaatttgtacgtg/ )

s/\s*\d+$//

s/\s*//g
```

The prepare_embl program

```
#! /usr/bin/perl -w

# The 'prepare_embl' program - getting embl.data
# ready for use.

while ( <> )
{
    s/\s*\d+$//;
    s/\s*//g;
    print;
}
```

```
$ perl prepare_embl embl.data > embl.data.out

$ wc embl.data.out
0 1 6838 embl.data.out
```

The match_embl program

```
#! /usr/bin/perl -w

# The 'match_embl' program - check a sequence against
# the EMBL database entry stored in the
# embl.data.out data-file.

use constant TRUE => 1;

open EMBLENTRY, "embl.data.out"
    or die "No data-file: have you executed prepare_embl?\n";

my $sequence = <EMBLENTRY>;

close EMBLENTRY;

print "Length of sequence is: ", length $sequence,
          " characters.\n";

while ( TRUE )
{
```

The match_embl program, cont.

```
    print "\nPlease enter a sequence to check.\n
                Type 'quit' to end: ";

    my $to_check = <>;

    chomp( $to_check );
    $to_check = lc $to_check;

    if ( $to_check =~ /^quit$/ )
    {
        last;
    }
    if ( $sequence =~ /$to_check/ )
    {
        print "The EMBL data extract contains: $to_check.\n";
    }
    else
    {
        print "No match found for: $to_check.\n";
    }
}
```

Results from match_embl ...

```
$ perl match_embl

Length of sequence is: 6838 characters.

Please enter a sequence to check.
Type 'quit' to end: aaatttgggccc
No match found for: aaatttgggccc.
    .
    .
    .
Please enter a sequence to check.
Type 'quit' to end: caGGGGGgg
No match found for: caggggggg.

Please enter a sequence to check.
Type 'quit' to end: tcatgcacctgatgaacgtgcaaaaccacagtcaagccatga
The EMBL data extract contains:
  tcatgcacctgatgaacgtgcaaaaccacagtcaagccatga.

Please enter a sequence to check.
Type 'quit' to end: quit
```

Where To From Here