Bioinformatics Programming 2013

Perl – Regular Expression

Chia-Lang Hsu (許家郎)
Department of Life Science,
National Taiwan University

Regular Expression (regex)

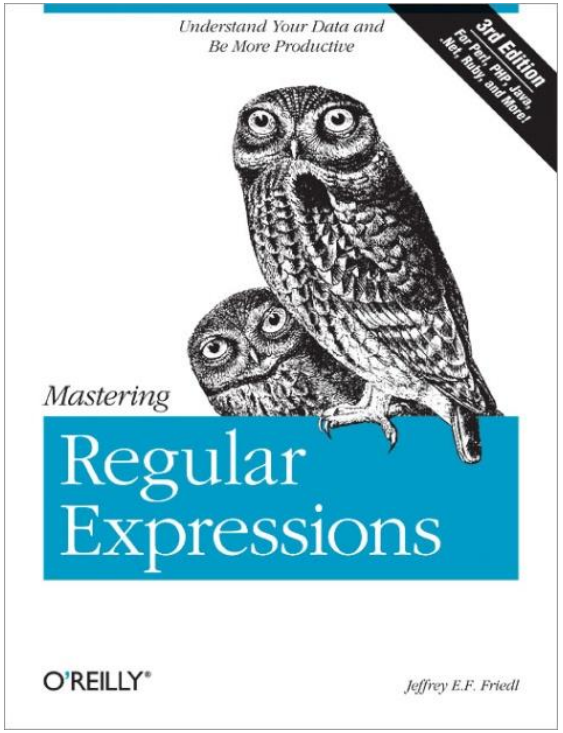
- A regular expression is a string of characters that define the pattern or patterns you are viewing.
  - For example: check if a valid email address
    - ^[A-Z0-9._%+-]+\@[A-Z0-9.-]+\.[A-Z]{2,4}$
- Applications:
  - Web application
  - Text mining
  - Bioinformatics: Motif match

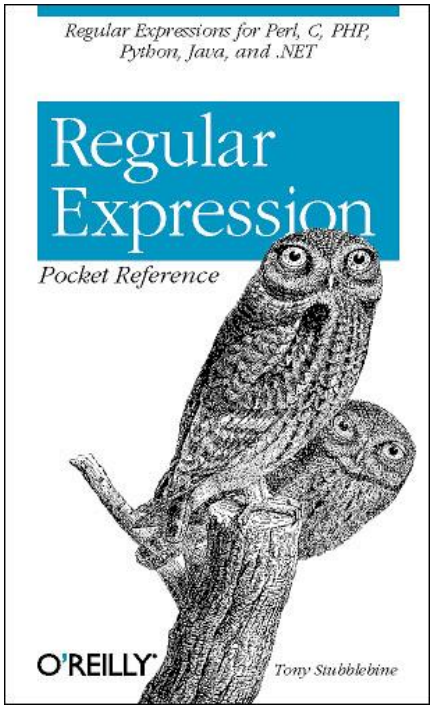
PROSITE patterns

- Rules:
  - Each position is separated by a hyphen
  - One character denotes residuum at a given position
  - [...] denoted a set of allowed amino acids
  - (n) denotes repeat of n times
  - (n,m) denoted repeat between n and m inclusive
  - X – any character

- For example:
  - ATP/GTP binding motive [SG]-X(4)-G-K-[DT]
    - SGMVQGKT, GAKASGKD, or GUCDEGKT ...

Regex references





Regex in Perl

expression	matches...
abc	abc (that exact character sequence, but anywhere in the string)
^abc	abc at the *beginning* of the string
abc$	abc at the *end* of the string
a|b	either of a and b
^abc|abc$	the string abc at the beginning or at the end of the string
ab{2,4}c	an a followed by two, three or four b's followed by a c
ab{2,}c	an a followed by at least two b's followed by a c
ab*c	an a followed by any number (zero or more) of b's followed by a c
ab+c	an a followed by one or more b's followed by a c
ab?c	an a followed by an optional b followed by a c; that is, either abc or ac
a.c	an a followed by any single character (not newline) followed by a c
a\.c	a.c exactly
[abc]	any one of a, b and c
[Aa]bc	either of Abc and abc
[abc]+	any (nonempty) string of a's, b's and c's (such as a, abba, acbabcacaa)
[^abc]+	any (nonempty) string which does *not* contain any of a, b and c (such as defg)
\d\d	any two decimal digits, such as 42; same as \d{2}
\w+	a "word": a nonempty sequence of alphanumeric characters and low lines (underscores), such as foo and 12bar8 and foo_1
100\s*mk	the strings 100 and mk optionally separated by any amount of white space (spaces, tabs, newlines)
abc\b	abc when followed by a word boundary (e.g. in abc! but not in abcd)
perl\B	perl when *not* followed by a word boundary (e.g. in perlert but not in perl stuff)

Metacharacters

Modifier	Description
\	Quote next character
^	Match beginning-of-string
$	Match end-of-string
.	Match any character except newline
|	Alternation
()	Grouping and save subpattern
[]	Character class

Metacharacters

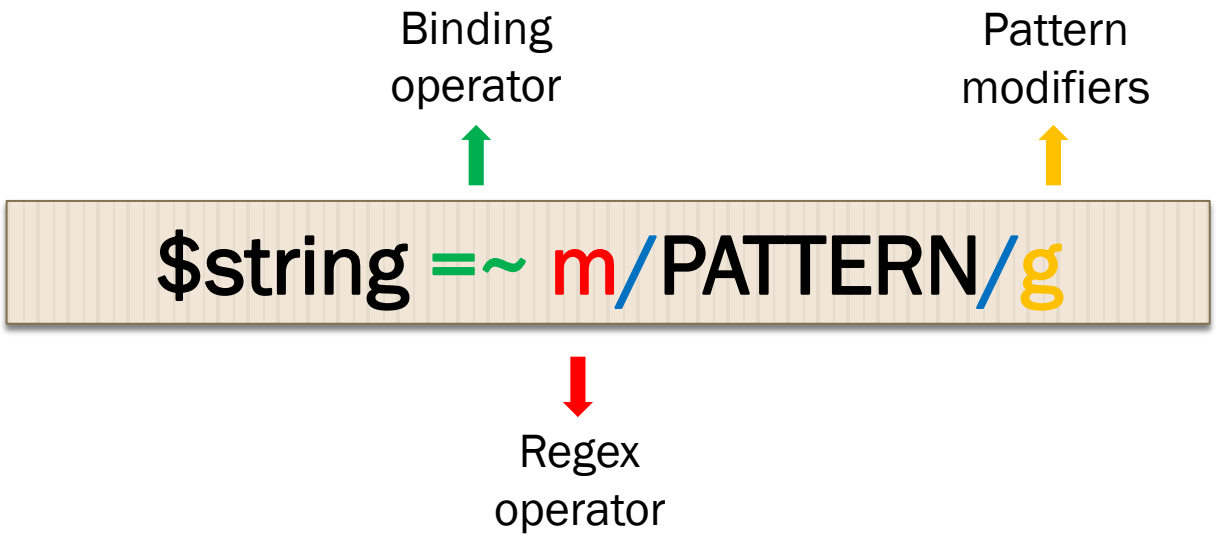
Modifier	Description
\w	matches any word character or alphanumeric character, including the underscore.
\W	matches any non-word character or nonalphanumeric character, and excludes the underscore.
\d	matches a digit character that is equivalent to [0-9]
\D	matches a non-digit character that is equivalent to [^0-9].
\s	matches any white space character, including space, tab, form feed, and so on, and is equivalent to [\f\n\r\t\v].
\S	matches any character that is not a white space character and is equivalent to [^\f\n\r\t\v].
\b	matches a word boundary: 1. "er\b" matches the "er" in "never“ 2. "er\b" does not match the "er" in "verb"
\B	matches a non-word boundary: 1. "er\B" matches the "er" in "verb„ 2. "er\B" does not match the "er" in "never"

Quantifier

- Quantifiers can be used to specify how many of the previous thing you want to match on.

Character	Description
*	Matches the previous atom zero or more times
+	Matches the previous atom one or more times
?	Matches the previous atom zero or one times
{n}	Matches the previous atom exact n times
{n,}	Matches the previous atom n or more times
{n,m}	Matches the previous atom between n and m times

Syntax of regex in Perl



- m/PATTERN/ : match operator
- s/PATTERN/ : substitution operator
- tr/PATTERN/ : translation operator

Match operator

- The match operator, m//, is used to match a string or statement to a regular expression.

```
my $text = "Here is a text";
if ($text =~ m/apple/)
{
        print "Found the text\n";
}
else
{
        print "Not found\n";
}
```

檢查數字

```
## 檢查是否為整數
my $text = "123456789";
if ($text =~ m/^\d+$/){print "It's a number.\n";}

## 檢查是否為浮點數
my $text = "3.1415926";
if ($text =~ m/^\d+\.\d*$/){print "It's a number\n";}

## 檢查數字前的正負號
my $text = "-3.14159";
if ($text =~ m/^[+-]\d+\.\d*$/){print "It's a number\n";}
```

① **^**: 從字串的開頭開始比對。
② **\d**: 符合一個數字。
③ **+**：符合一次或是多次。
④ **$**：從字串的結果開始比對。
⑤ *****：符合0次或是多次。
⑥ **[]**: 代表一群字元。

Match operator modifiers

Modifier	Description
i	Makes the match case insensitive
m	Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of a string boundary
o	Evaluates the expression only once
s	Allows use of . to match a newline character
x	Allows you to use white space in the expression for clarity
g	Globally finds all matches
cg	Allows the search to continue even after a global match fails

Example – using modifiers

```perl
use strict;

my $text = "Fool feel tOok hoOk dump football BOOK";

my $count = 1;
while ($text =~ m/oo/g)   #搜尋所有的符合
{
        print "Found $count oo.\n";
        $count ++;
}

print "----------------------------------------\n";

my $count2 = 1;
while ($text =~ m/oo/gi)  #搜尋所有的符合並忽略大小寫
{
        print "Found $count2 oo.\n";
        $count2 ++;
}
```

```
D:\Dwimperl\perl\bin\perl.exe  test.pl
Found 1 oo.
Found 2 oo.
----------------------------------------
Found 1 oo.
Found 2 oo.
Found 3 oo.
Found 4 oo.
Found 5 oo.
Press any key to continue . . .
```

Capture matched patterns

```
my $text = "there are 20 male and 30 female in 1 bus";

while ($text =~ m/(\d+)\s(\w+)/g)
{
    print "$1\t$2\n"
}
```

```
20      male
30      female
1       bus
```

① **\d**: 符合一個數字。
② **\s**: 符合任何whitespace字元(空白、tab等)。
③ **\w**: 符合任何英數字元。
④ **+** : 符合一次或是多次。
⑤ **()**: 將表示式組為群組。

Return matched patterns

```
my $text = "20.5 adds 40.15 eqauls 60.55";

my @a = ($text =~ m/([\d\.]+)\D+/g);

print "@a\n";
```

```
20.5 40.15 60
```

① **\d**: 符合一個數字字元。
② **\D**: 符合一個非數字字元。
③ **+** : 符合一次或是多次。
④ **[]**: 代表一群字元。
⑤ **()**: 將表示式組為群組。

Position of matched pattern

- Positions of what was matched with the **@-** and **@+** arrays
  - **$-[0]** is the position of the start of the entire match and **$+[0]** is the position of the end
  - **$-[n]** is the position of the start of the **$n** match and **$+[n]** is the position of the end

```
use strict;

my $seq = "ACGTTGTCAGGACGGGACAGCGCGGCGTATGCGCG";

while ($seq =~ m/CG/g)
{
    print "$-[0]\t$+[0]\n";
}
```

```
1       3
12      14
20      22
22      24
25      27
31      33
33      35
```

Search for AT-rich motifs

```
my $DNA = "ATTATACAATTGCGTACTATATATACCGTATAACGTTTTAAAAAG";

while ($DNA =~ m/([AT]{4,8})/g)
{
    my $ATmatch = $1;
    my $ATlength = length($ATmatch);

    my $end = pos($DNA); ## pos() return the end-position
                         ## of matched pattern

    my $start = $end - $ATlength + 1;

    print "AT-rich motif: $ATmatch ";
    print "of length $ATlength from $start to $end\n";
}
```

```
AT-rich motif: ATTATA of length 6 from 1 to 6
AT-rich motif: AATT of length 4 from 8 to 11
AT-rich motif: TATATATA of length 8 from 18 to 25
AT-rich motif: TATAA of length 5 from 29 to 33
AT-rich motif: TTTTAAAA of length 8 from 36 to 43
```

Search for TATA-box

```
my $promoter = "GCGACCACCTTGGTTCAGCAGTATAAAAACGCGCTTGGCG";

print "TATA-box search on $promoter\n";
print "==================================================\n";

if ($promoter =~ m/(TATA[AT]A[AT][AG])/)
{
    my $TATAbox = $1;
    my $TATAlen = length($TATAbox);
    my $TATAlocation = index($promoter, $TATAbox);

    print "Found a TATA-box: $TATAbox \n";
    print "at location $TATAlocation \n\n";
}
else
{
    print "No TATA box was found. \n";
}
```

```
TATA-box search on GCGACCACCTTGGTTCAGCAGTATAAAAACGCGCTTGGCG
==================================================
Found a TATA-box: TATAAAAA
at location 21
```

Search for Mirror Repeats (MR)

MR MR

```
$dna = ATACGTCATGCACTTCTACGTATCGGTGCA
```

$1

```
$dna =~ m/((.)(.)(.)(.)(.*)\5\4\3\2)/g;
```

$2 $3 $4 $5

Search for Mirror Repeats (MR)

MR MR

ATACGTCATGCACTTCTACGTATCGGTGCA

```
my $dna = "ATACGTCATGCACTTCTACGTATCGGTGCA";

while ($dna =~ m/
            (                # capture entire match
            (.)(.)(.)(.)     # any 4 nucleotides
            (.*)             # GREEDY! 0 or more of any nucleotide
            \5\4\3\2         # recall in mirror-order
            )                # end of entire match
            /xg
        )          # x: 忽略樣式內的空白來允許註解。
{
    my $mirrorRepeat = $1;
    my $foundBP = pos($dna)-length($mirrorRepeat);
    print "Found MR: $mirrorRepeat at bp $foundBP \n";
}
```

```
Found MR: ACGTCATGCACTTCTACGTATCGGTGCA at bp 2
```

Search for Mirror Repeats (MR)

MR MR

ATACGTCATGCACTTCTACGTATCGGTGCA

```
my $dna = "ATACGTCATGCACTTCTACGTATCGGTGCA";

while ($dna =~ m/
            (                # capture entire match
            (.)(.)(.)(.)     # any 4 nucleotides
            (.*?)            # NOT GREEDY! 0 or more minimally
            \5\4\3\2         # recall in mirror-order
            )                # end of entire match
            /xg
        )
{
    my $mirrorRepeat = $1;
    my $foundBP = pos($dna)-length($mirrorRepeat);
    print "Found MR: $mirrorRepeat at bp $foundBP \n";
}
```

```
Found MR: ACGTCATGCA at bp 2
Found MR: ACGTATCGGTGCA at bp 17
```

Match a minimal piece of string

- a?? = match 'a' 0 or 1 times. Try 0 first, then 1.
- a*? = match 'a' 0 or more times, i.e., any number of times, but as few times as possible.
- a+? = match 'a' 1 or more times, i.e., at least once, but as few times as possible.
- a{n,m}? = match at least n times, not more than m times, as few times as possible.
- a{n,}? = match at least n times, but as few times as possible.
- a{n}? = match exactly n times. Because we match exactly n times, a{n}? is equivalent to a{n} and is just there for notational consistency.

Search for open reading frame (ORF)

Start codon

Stop codon
(TAG, TGA, & TAA)

```
$dna = ACTGACATGCGCCGTGACGATAAATTTGGCCGATAATGGACCGAG
```

3的倍數

```
$dna =~ m/(?:ATG)(?:...)*?(?:TAG|TGA|TAA)/;
```

Search for open reading frame (ORF)

```
my $seq = "ACTGACATGCGCCGTGACGATAAATTTGGCCGATAATGGACCGAG";

if ($seq =~ m/
        (?: ATG)          # start codon
        (?: ...)          # 3-bp codon
        *?                # zero or more times, non-greedy
        (?: TAG|TGA|TAA) # stop codon
        /x
    )
{
    print "Open reading frame found.\n";
}
else
{
    print "No open reading frame found.\n";
}
```

Review

regex	meaning
TATA	match four consecutive letters, TATA
TAG|TGA|TAA	match TAG or TGA or TAA
.	match any character but not a newline character
..	match any two characters (independently, not necessarily the same character)
(.)	capture (remember) and match any character
.*	match any character 0 or more times (each is independent of others)
(.*)	capture and match any character 0 or more times
.+	match any character 1 or more times (each is independent of others)
(.+)	capture and match any character 1 or more times
\1	recall the first captured (parenthesized) group
\2	recall the second captured group
n	recall the *n*th captured group
.?	optional, match any character 0 or 1 time
T?	optional, match a T or nothing
(CAAT)?	Optional, match CAAT or nothing
A{3,7}	match between 3 and 7 As
A{3,}	match of 3 or more As
[CG]	match any *one* of the characters in the set, a C or a G
TATA[AT]	match TATA followed by an A or a T
[^CG]	match any *one* character that is *not* in the set, not a C and not a G
[CG]{5,10}	match a C or a G between 5 and 10 times
^ATG	string begins with ATG
TAG$	string ends with TAG
\s	match any whitespace character (tab, space, newline)
\S	match any character that is not whitespace
\d	match any character that is a digit, same as [0123456789]
\D	match any character that is not a digit
\w	match any one "word" character (includes alphanumeric, plus '_')
\W	match any one nonword character

Substitution operator

- The match operator, s//, is really just an extension of the match operator that allows you to replace the text matched with some new text.
- The basic form of the operator is:
  - s/PATTERN/REPLACEMENT/imosxge

Modifiers

```
my $text = "This is a DOG";

print "Origin: $text\n";

$text =~ s/DOG/CAT/;

print "Next: $text\n";
```

```
Origin: This is a DOG
Next: This is a CAT
```

Substitution operator modifiers

Modifier	Description
i	Makes the match case insensitive
m	Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of a string boundary
o	Evaluates the expression only once
s	Allows use of . to match a newline character
x	Allows you to use white space in the expression for clarity
g	Replaces all occurrences of the found expression with the replacement text
e	Evaluates the replacement as if it were a Perl statement, and uses its return value as the replacement text

“e” modifier for substitution operator

```perl
my $text = "Here is a house";

print "Origin: $text\n";

$text =~ s/(\w+)/uc($1)/g;

print "Next: $text\n";
```

```
Origin: Here is a house
Next: uc(Here) uc(is) uc(a) uc(house)
```

```perl
my $text = "Here is a house";

print "Origin: $text\n";

$text =~ s/(\w+)/uc($1)/ge;

print "Next: $text\n";
```

```
Origin: Here is a house
Next: HERE IS A HOUSE
```

Transforming format

```
## 日期格式"08/20/2013", 改成"2013-08-20"
my $date = '08/20/2013';

$date =~ s/(\d+)\/(\d+)\/(\d+)/$3-$1-$2/;
```

$1 $2 $3

```
print "$date\n";
```

Transcribing DNA into RNA

Example:
DNA → ATCGGCTTGGAGAA
RNA → AUCGGCUUGGAGAA

```
use strict;

my $dna = "ATCGGCTTGGAGAA";

my $rna = $dna;

$rna =~ s/T/U/g;

print "DNA: ". $dna. "\n";
print "RNA: ". $rna. "\n";
```

Count CpG

Example:

Sequence → AT**CG**GG**CGCG**C**CG**GGTTATAG**CG**GATAGG**CG**AG

```
use strict;

my $seq = "ATCGGGCGCGCCGGGTTATAGCGGATAGGCGAG";

my $count = $seq =~ s/CG/CG/g;

print "$count\n";
```

Translation operator

- Translation is similar, but not identical, to the principles of substitution, but unlike substitution, translation (or transliteration) does not use regular expressions for its search on replacement values.
- The basic form of the operator is:
  - tr/PATTERN/REPLACEMENT/cds
  - y/PATTERN/REPLACEMENT/cds Modifiers

```
my $string = 'The cat sat on the mat';

print "Origin: $string\n";

$string =~ tr/a/o/;

print "Translated: $string\n";
```

```
Origin: The cat sat on the mat
Translated: The cot sot on the mot
```

Translation operator modifiers

Modifier	Description
c	Complement SEARCHLIST.
d	Delete found but unreplaced characters.
s	Squash duplicate replaced characters.

Example: using modifiers for translation operator

The /d modifier deletes the characters matching SEARCHLIST that do not have a corresponding entry in REPLACEMENTLIST. For example:

```
#!/usr/bin/perl

$string = 'the cat sat on the mat.';
$string =~ tr/a-z/b/d;

print "$string\n";

This will produce following result
b b   b.
```

只有a有相對應的取代字元。

The last modifier, /s, removes the duplicate sequences of characters that were replaced, so:

```
#!/usr/bin/perl

$string = 'food';
$string = 'food';
$string =~ tr/a-z/a-z/s;

print $string;

This will produce following result
fod
```

Complementary strand of a DNA

Example:

```
5'-ATCGGCTTGGAGAA-3'
   |||||||||||||||
3'-TAGCCGAACCTCTT-5'
```

```perl
use strict;

my $dna = "ATCGGCTTGGAGAA";

my $revcom = reverse $dna;

$revcom =~ tr/ACGTacgt/TGCAtgca/;

print "5'-" . $dna . "-3'\n";
print "5'-" . $revcom . "-3'\n";
```