

Introduction to Perl Programming –Input/Output, Regular Expressions, String Manipulation

Beginning Perl, Chap 4–6

Example 1

```
#!/usr/bin/perl -w                                # version 4: with compositional bias
use strict;                                       my %comp = ( A => 0.20,
                                                    C => 0.35,
                                                    G => 0.25,
                                                    T => 0.20
                                                    );
# version 1:                                       my @nt = keys %comp;
my @nt = ('A', 'C', 'G', 'T');                 my @bucket;
my $dna;
for (1 .. 100) {
    $dna .= $nt[int(rand 4)];
}
# version 2:                                       # fill the bucket to draw from:
my @nt = qw(A C G T);                           for my $nt (@nt) {
my $dna;                                         push @bucket,
for (1 .. 100) {                                ($nt) x ($comp{$nt} * 100);
    $dna .= $nt[int(rand 4)];
}
# version 3: most Perl-ish                       my $dna;
my @nt = qw(A C G T);                           for (0 .. 99) {
my $dna;                                         $dna .= $bucket[int(rand scalar @bucket)];
for (1 .. 100) {
    $dna .= $nt[int(rand(scalar @nt))];
}
print $dna, "\n";
```

Example 2

```
#!/usr/bin/perl -w
use strict;

# version 1;
# same as nt construction
my @aa = qw(A C D E F G H I
            K L M N P Q R S T V W Y);
my $protein;
for (1 .. 200) {
    $protein .= $aa[int(rand scalar @aa)];
}
print $protein, "\n";

# version 2: very Perl-ish;
my @letters = ( A .. Z );
my %comp;
# hash slice:
@comp{@letters} = ( 1 / 20 ) x @letters;
# remove letter that aren't aa's
$comp{'B'} = $comp{'J'} =
$comp{'O'} = $comp{'X'} =
$comp{'U'} = $comp{'Z'} = 0;

my @bucket;
for my $l (@letters) {
    push @bucket ($l) x ($comp{$l} * 100);
}

my $length = 200 + rand(100);
my $protein;
for (1 .. $length) {
    push $protein, $bucket[rand @bucket];
}
}
```

Example 3

```
#!/usr/bin/perl -w
use strict;

my %codontable = (
    'AAA' => 'K', 'AAC' => 'N', 'AAG' => 'K', 'AAT' => 'N',
    'ACA' => 'T', 'ACC' => 'T', 'ACG' => 'T', 'ACT' => 'T',
    'AGA' => 'R', 'AGC' => 'S', 'AGG' => 'R', 'AGT' => 'S',
    'ATA' => 'I', 'ATC' => 'I', 'ATG' => 'M', 'ATT' => 'I',
    'CAA' => 'Q', 'CAC' => 'H', 'CAG' => 'Q', 'CAT' => 'H',
    'CCA' => 'P', 'CCC' => 'P', 'CCG' => 'P', 'CCT' => 'P',
    'CGA' => 'R', 'CGC' => 'R', 'CGG' => 'R', 'CGT' => 'R',
    'CTA' => 'L', 'CTC' => 'L', 'CTG' => 'L', 'CTT' => 'L',
    'GAA' => 'E', 'GAC' => 'D', 'GAG' => 'E', 'GAT' => 'D',
    'GCA' => 'A', 'GCC' => 'A', 'GCG' => 'A', 'GCT' => 'A',
    'GGA' => 'G', 'GGC' => 'G', 'GGG' => 'G', 'GGT' => 'G',
    'GTA' => 'V', 'GTC' => 'V', 'GTG' => 'V', 'GTT' => 'V',
    'TAA' => '*', 'TAC' => 'Y', 'TAG' => '*', 'TAT' => 'Y',
    'TCA' => 'S', 'TCC' => 'S', 'TCG' => 'S', 'TCT' => 'S',
    'TGA' => '*', 'TGC' => 'C', 'TGG' => 'W', 'TGT' => 'C',
    'TTA' => 'L', 'TTC' => 'F', 'TTG' => 'L', 'TTT' => 'F'
);

# make DNA the boring way:
my @nt = qw(A C G T);
my $dna;
my $length = 500 + int(rand(500));
for (1 .. $length) {
    $dna .= $nt[int(rand @nt)];
}

# OK, now translate:
my $protein;
my @seq = split(' ', $dna);

my $codon = '';
for my $i (0 .. $#seq) {
    $codon .= $seq[$i];
    if (($i + 1) % 3 == 0) {
        $protein .= $codontable{$codon};
        $codon = '';
    }
}

# another way, using "length()":
for my $nt (@seq) {
    $codon .= $nt;
    if (length($codon) == 3) {
        $protein .= $codontable{$codon};
        $codon = '';
    }
}

# yet another variation on the theme:
while (@seq) {
    $codon .= shift @seq;
    if (length $codon == 3) {
        $protein .= $codontable{$codon};
        $codon = '';
    }
}

# even better:
while (@seq) {
    $protein .= $codontable{ shift @seq . shift
@seq . shift @seq };
}
}
```

Input/Output

- `open (INFILE, "< gtm1_human.aa");` – open file for reading
- `$file = $ARGV[0];`
`open (INFILE, "< $file")||`
`die ("Cannot open $file");`
- `open (OUTFILE, "> $file");` – open file for writing
- `$line = <FILE>;` – read a line
- `chomp($line);` – remove “\n”
- `while ($line=<FILE>) { ... }`
- `print FILE "$line\n";`
- `close(FILE);`

Regular expressions

```
>gi|121694|sp|P20432|GTT1_DROME Glutathione S-transferase 1-1
```

- used for string matching, substitution, pattern extraction
- `/^>gi\|\/` matches `>gi|121694|sp|P20432|GTT1_DROME ...`
- `if ($line =~ m/^>gi/) { ... } #match`
- `$line =~ /^>gi\|(\d+)\|\/; # extract gi#`
`$gi = $1;`
- `($gi) = $line =~ /^>gi\|(\d+)\|\/; #same`
- `$line =~ s/^>(.*?)$/>>$1/; # substitution`

Regular expressions (cont.)

```
>gi|121694|sp|P20432|GTT1_DROME Glutathione S-transferase 1-1
```

- `m/plaintext/`
`m/one|two/; # alternation`
`m/(one|two)|(three)/; # grouping with # parenthesis`
- `/^>gi\|(\d+)/ #beginning of line`
`/.+ (\d+) aa$/ # end of line`
- `/a*bc/ # bc,abc,aabc, ... # repetitions`
`/a?bc/ # abc, bc`
`/a+bc/ # abc, aabc, ...`

Regular Expressions, III

```
>gi|121694|sp|P20432|GTT1_DROME Glutathione S-transferase 1-1
```

- Matching classes:
 - `/^>gi\|[0-9]+\|[a-z]+\|[A-Z][0-9a-z]+\|/`
 - `[a-z]` `[0-9]` -> class
 - `[^a-z]` -> negated class
 - `/^>gi\|\d+\|[a-z]\|\w+\|/`
 - `\d` -> number `[0-9]` `\D` -> not a number
 - `\w` -> word `[0-9A-Za-z_]` `\W` -> not a word char
 - `\s` -> space `[\t\n\r]` `\S` -> not a space
- Capturing matches:
 - `/^>gi\|(\d+)\|([a-z])\|(\w+)\|/`
`$1 $2 $3`
 - `($gi,$db,$db_acc) =`
`$line =~ /^>gi\|(\d+)\|([a-z])\|(\w+)\|/;`

Regular expressions - modifiers

- `m/That/i` # ignore case
- `s/this/that/g` # global replacement
- `m/>gi\\(\\d+)\\|([a-z]{2,3}|(\\w+))` # {range}
- `s//m` # treat string as multiple lines
- `s//s` # span over \\n
- `s/\\n//gs` # remove \\n in multiline entry
- `s/GAATTC/$1\\n/g` # break lines at EcoRI site

```
{ local $/ = "\\n"; # change the line separator to "\\n"
while ($entry = <INFILE>) {
  chomp($entry);
  $entry =~ s/\\A?>/>; # replace missing >, if necessary
  # \\A is like ^ (beginning of string)
  open(OUT, ">file$num.fa") or die $!;
  $num++;
  print OUT "$entry\\n";
  close(OUT);
}
```

String expressions (with regular expressions)

- `if (/^>gi\\|/) { ... }`
- `if ($line =~ m/^>gi\\|/) { ... }`
- `while ($line !~ m/^>gi\\|/) { ... }`
- Substitution:
`$new_line =~ s/\\|/:/g;`
- Pattern extraction:
`($gi,$db,$db_acc) =
$line = /^>gi\\(\\d+)\\|([a-z])\\| (\\w+)\\|/;`
- **split** (`/|/`, \$line)
- **join** ("`:`", @fields);
- `substr($string,$start,$length);` # rarely used
- `index($string,$query);` # rarely used
- Comparison: "`eq`" "`ne`" "`cmp`" "`lt`" "`gt`" "`le`" "`ge`"

Perl mrtrans.pl

```
use vars qw(%proteins %dnas
            $line $name $scodon $aa);
my ($pfile, $nfile) = @ARGV;
open(PROTEIN, "<$pfile") or die $!;
open(DNA, "<$nfile") or die $!;

while(my $line = <PROTEIN>) {
    chomp($line);
    if($line =~ m/^\>/) {
        $name = shift split(' ', $line);
        next;
    }
    $proteins{$name} .= $line if $name;
}
close(PROTEIN);

for my $name (keys %proteins) {
    $proteins{$name} =~ s/\s+//sg;
}
while($line = <DNA>) {
    chomp($line);
    if($line =~ m/^\>/) {
        $name = shift split(' ', $line);
        next;
    }
    $dnas{$name} .= $line if $name;
}
close(DNA);

for $name (keys %dna) {
    unless (exists $protein{$name}) {
        warn "DNA sequence $name has no
            matching protein sequence!\n";
        next;
    }
    $dna{$name} =~ s/\s+//sg;
    print "$name\n";
    my @protein=split(' ', $proteins{$name});
    my @dna = split(' ', $dnas{$name});
    for $aa (@protein) {
        if($aa eq '-') {
            print "----";
        } else {
            $scodon = shift @dna
                . shift @dna . shift @dna;
            # check that $scodon codes for $aa
            print $scodon;
        }
    }
    print "\n";
}
```

Exercises

1. Take a set of FASTA format files and make a FASTA library, combining those files
2. Take a FASTA library, and break it into individual FASTA files
3. Write an extraction program to report
 - (a) the most significant library hit
 - (b) all significant library hitsthat result from a FASTA search. Several example search results are available at watson.achs:/r0/seqlib/bioch508/perl/fasta.result*
4. Write the same program for blast output. Sample results are in the same directory
5. Write a perl equivalent to "mrtrans" that does not care about the order of sequence input. Given two files, a FASTA protein library (with '-' for gaps) and a FASTA DNA library with the same sequence names, generate a FASTA DNA alignment with '---' inserted into the DNA sequence for each '-' in the protein sequence