Programming in bioinformatics: BioPerl

Dobrica Pavlinušić
http://www.rot13.org/~dpavlin/

PBF, 10.05.2007.

Programming and biology
Basic algorithm structures

Programming for biology

- Cultural divide between biologists and computer science
  - use programs, don't write them
  - write programs when there's nothing to use
  - programming takes time
- Focus on interesting, unsolved, problems
- Open Source tools comes as part of the rescue

Reasons for programming

- Scientific
  - Quantity of existing data
  - Dealing with new data
  - Automating the automation
  - Evaluating many targets
- Economic

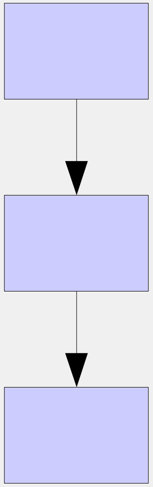
> **…** programmers going into biology often have the harder time of it **…** biology is subtle, and it can take lots of work to begin to get a handle on the variety of living organisms. **Programmers new to the field sometimes write a perfectly good program for what turns out to be the wrong problem!** -- James Tisdall

Biology

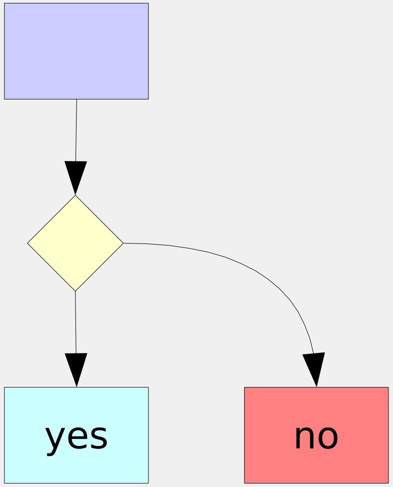
- Science in different mediums
  - in vitro – in glass
  - in vivo – in life
  - in silico – in computer algorithms
- Huge amount of experimental data
  - collected, shared, analyzed
  - biologists forced to relay on computers

Basic programming

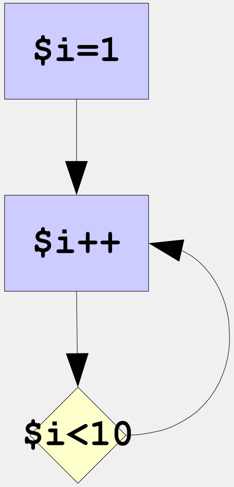
- Simple basic building blocks which enable us to describe desired behavior (algorithm) to computer



sequence



condition



loop

Why perl?

- well suited to text manipulation tasks
- easy to learn
- CPAN modules, including BioPerl
- rapid prototyping
  - duct tape of Internet
- available on multiple platforms
  - Unix, Linux, Windows, VMS...
- TIMTOWTDI
  - **T**here **I**s **M**ore **T**han **O**ne **W**ay **T**o **D**o **I**t

rot13 example

```
        program rot
        character*1 in(52),out(52)
        integer i,j
        integer*2 length

        byte bin(52),bout(52)
        equivalence(bin,in)
        equivalence(bout,out)
        character*16384 test
        logical found
        do i=1,26
           bin(i)=ichar('A')-1 +i
           bin(i+26) = ichar('a') -1 +i
        end do
        do i=1,13
           bout(i)=ichar('N')-1 +i
           bout(i+13) = ichar('A')-1+i
           bout(i+26)=ichar('n')-1 +i
           bout(i+39)=ichar('a')-1+i
        end do
        read (5,'(a)')test
        do i=len(test),1,-1
          if (test(i:i) .ne. ' ') then
             length=i
             goto 101
          end if
        end do
101     continue ! :)
        do i=1,length
           found = .false.
           do j=1,52
                if (test(i:i) .eq. in(j)) then
                   write(6,'(a,$)')out(j)
                   found = .true.
                end if
           end do
           if (.not. found) write(6,'(a,$)')test(i:i)
        end do
        write(6,'(1x)')
        end
```

```
int main ()
{
  register char byte, cap;
  for(;read (0, &byte, 1);)
    {
      cap = byte & 32;
      byte &= ~cap;
      byte = ((byte >= 'A') && (byte <= 'Z') ?
       ((byte - 'A' + 13) % 26 + 'A') : byte) | cap;
      write (1, &byte, 1);
    }
}
```

```
import java.io.*;
public class rot13 {
  public static void main (String args[]) {
    int abyte = 0;
    try { while((abyte = System.in.read())>=0) {
      int cap = abyte & 32;
      abyte &= ~cap;
      abyte = ((abyte >= 'A') && (abyte <= 'Z') ?
        ((abyte - 'A' + 13) % 26 + 'A') : abyte) | cap;
      System.out.print(String.valueOf((char)abyte));
    } } catch (IOException e) { }
    System.out.flush();
  }
}
```

```
#!/usr/bin/perl -p
y/A-Za-z/N-ZA-Mn-za-m/;
```

Art of programming

- Different approaches
  - take a class
  - read a tutorial book
  - get programming manual and plunge in
  - be tutored by a programmer
  - identify a program you need
  - try all of above until you've managed to write the program

Programming process

- identify inputs
  - data from file or user input
- make overall design
  - algorithm by which program generate output
- decide how to output results
  - files, graphic
- refine design by specifying details
- write perl code

IUB/IUPAC codes

Code	Nucleic Acid(s)
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
U	Uracil
M	A or C (amino)
R	A or G (purine)
W	A or T (weak)
S	C or G (strong)
Y	C or T (pyrimidine)
K	G or T (keto)
V	A or C or G
H	A or C or T
D	A or G or T
B	C or G or T
N	A or G or C or T (any)

Code	Amino acid	TLC
A	Alanine	Ala
B	Aspartic acid or Asparagine	Asx
C	Cysteine	Cys
D	Aspartic acid	Asp
E	Glutamic acid	Glu
F	Phenylalanine	Phe
G	Glycine	Gly
H	Histidine	His
I	Isoleucine	Ile
K	Lysine	Lys
L	Leucine	Leu
M	Methionine	Met
N	Asparagine	Asn
P	Proline	Pro
Q	Glutamine	Gln
R	Arginine	Arg
S	Serine	Ser
T	Threonine	Thr
V	Valine	Val
W	Tryptophan	Trp
X	Unknown	Xxx
Y	Tyrosine	Tyr
Z	Glutamic acid or Glutamine	Glx

Variables to store data

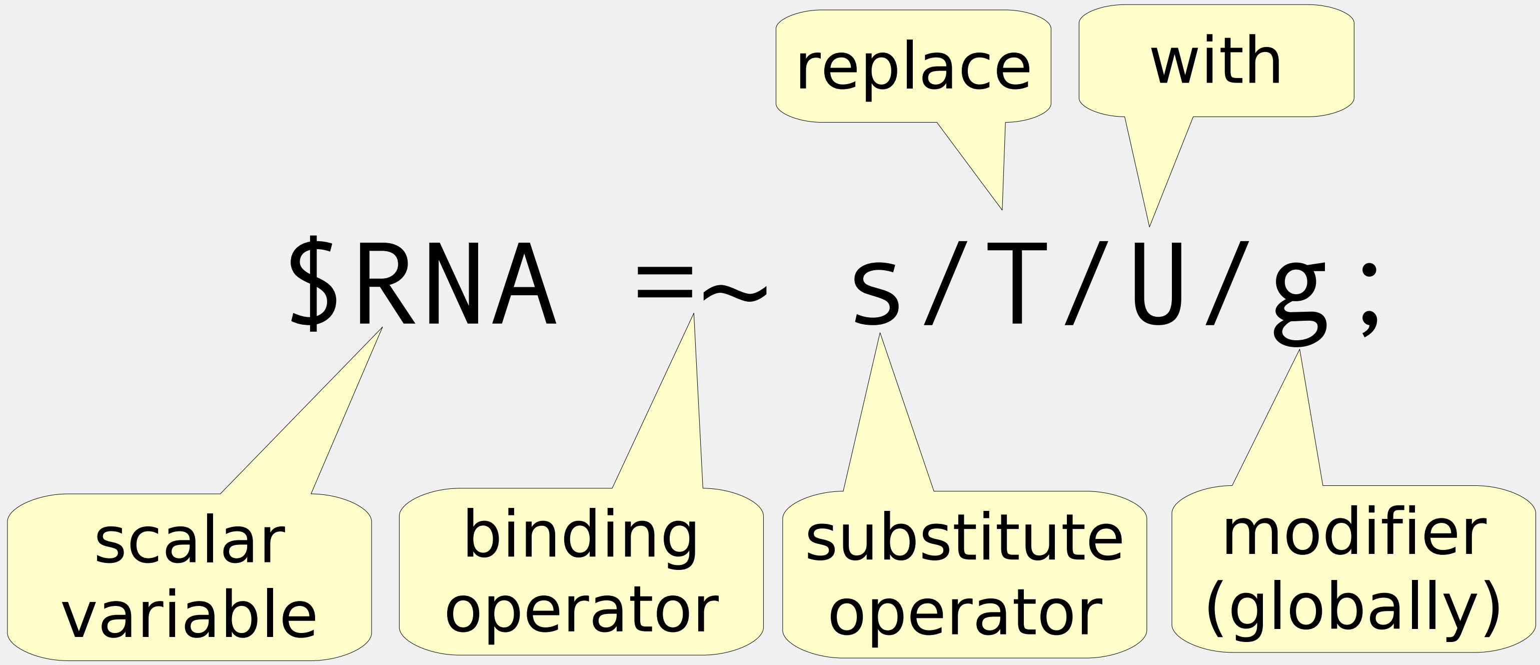
- Scalars
  - denoted by $sigil
  - store sequence of chars
  - join, substr, translate, reverse
- characters used
  - A, C, G, T – DNA nucleic acid
  - A, C, G, U – RNA
  - N – unknown
- `$DNA='ATAGTGCCGAGTGATGTAGTA';`

Transcribing DNA to RNA

```
#!/usr/bin/perl -w
# Transcribing DNA into RNA

# The DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';

# Print the DNA onto the screen
print "Here is the starting DNA:\n\n";

print "$DNA\n\n";

# Transcribe the DNA to RNA by substituting all T's with U's.
$RNA = $DNA;

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

# Print the RNA onto the screen
print "Here is the result of transcribing the DNA to RNA:\n\n";

print "$RNA\n";

# Exit the program.
exit;
```

String substitution

```
Here is the starting DNA:

ACGGGAGGACGGGAAAATTACTACGGCATTAGC

Here is the result of transcribing the DNA to RNA:

ACGGGAGGACGGGAAAAUUACUACGGCAUUAGC
```



```
$RNA =~ s/T/U/g;
```

Reverse complement

```perl
#!/usr/bin/perl -w
# Calculating the reverse complement of a strand of DNA

# The DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';

# Print the DNA onto the screen
print "Here is the starting DNA:\n\n";

print "$DNA\n\n";

# Make a new (reverse) copy of the DNA
$revcom = reverse $DNA;

print "Reverse copy of DNA:\n\n$revcom\n\n";

# Translate A->T, C->G, G->C, T->A, s/// won't work!
$revcom =~ tr/ACGT/TGCA/;

# Print the reverse complement DNA onto the screen
print "Here is the reverse complement DNA:\n\n$revcom\n";

exit;
```

Data in files and loop

```
#!/usr/bin/perl -w
# Calculating the reverse complement of a strand of DNA

# read lines from file or STDIN
while ( $DNA = <> ) {

    # remove line ending
    chomp( $DNA );

    # Make a new (reverse) copy of the DNA
    $revcom = reverse $DNA;

    # Translate A->T, C->G, G->C, T->A
    $revcom =~ tr/ACGT/TGCA/;

    # Print the reverse complement DNA onto the screen
    print "$revcom\n";
}
```

```
$ cat dna.txt
ACGGGAGGACGGGAAAATTACTACGGCATTAGC
$ ./03-complement-file.pl dna.txt
GCTAATGCCGTAGTAATTTTCCCGTCCTCCCGT
```

Introducing @array

- list of ordered elements
  - direct access to element by offset

    ```
    $first_element = $array[0];
    ```

  - can be created from scalars using split

    ```
    @array = split( //, 'ABCD' );
    @array = ( 'A', 'B', 'C', 'D' );
    ```

  - can be iterated, extended and consumed at both ends

    ```
    $first = shift @array; # ('B','C','D')
    $last = pop @array;    # ('B','C')
    unshift @array, 'X';   # ('X','B','C')
    push @array, 'Y';  # ('X','B','C','Y')
    ```

How about mutations?

- perl provides random number generator
- we want to mutate 10% of nucleotides
  - length of DNA divided by 10
- store mutated DNA in array
- for each mutation
  - find $mutation_position
  - select new $random_nucleotide
  - modify @mutated_DNA
- print out @mutated_DNA as string

Random mutations

```
#!/usr/bin/perl -w
use strict;
# randomize 10% of nucleotides

my @nucleotides = ( 'A', 'C', 'G', 'T' );

while ( my $DNA = <> ) {
    chomp( $DNA );
    my $DNA_length = length( $DNA );
    warn "DNA has $DNA_length nucleotides\n";
    my $mutations = int( $DNA_length / 10 );
    warn "We will perform $mutations mutations\n";
    my @mutated_DNA = split( //, $DNA );
    for ( 1 .. $mutations ) {
        my $mutation_position = int( rand( $DNA_length ) );
        my $random_position = int( rand( $#nucleotides ) );
        my $random_nucleotide = $nucleotides[ $random_position ];
        $mutated_DNA[ $mutation_position ] = $random_nucleotide;
        warn "mutation on $mutation_position to $random_nucleotide\n";
    }
    warn "$DNA\n";
    print join('', @mutated_DNA),"\n";
}
```

Evolution at work...

```
$ ./05-random.pl dna2.txt | tee dna3.txt
DNA has 33 nucleotides
We will perform 3 mutations
mutation on 16 to A
mutation on 21 to A
mutation on 8 to A
ACGGGAGGACGGGAAAATTACTACGGCATTAGC
ACGGGAGGACGGGAAAATTACAACGGCATTAGC
DNA has 33 nucleotides
We will perform 3 mutations
mutation on 9 to G
mutation on 24 to A
mutation on 12 to A
GCTAATGCCGTAGTAATTTTCCCGTCCTCCCGT
GCTAATGCCGTAATAATTTTCCCGACCTCCCGT
```

Introducing %hash

- unordered list of pair elements
  - stores key => value pairs

    ```
    %hash = ( foo => 42, bar => 'baz' );
    ```

  - can fetch all key values or pairs

    ```
    @all_keys = keys %hash;
    while (($key, $value) = each %hash) {
          print "$key=$value\n";
    }
    ```

- Examples
  - counters
  - lookup tables (mappings)

Let's count nucleotides!

- read input file for DNA line by line
- split DNA into @nucleotides array
- for each $nucleotide increment %count
  - **key** will be nucleotide code
  - **value** will be number of nucleotides
  - we don't care about order :-)
- iterate through %count and print number of occurrences for each nucleotide
- same as counting letters in string

Counting nucleotides

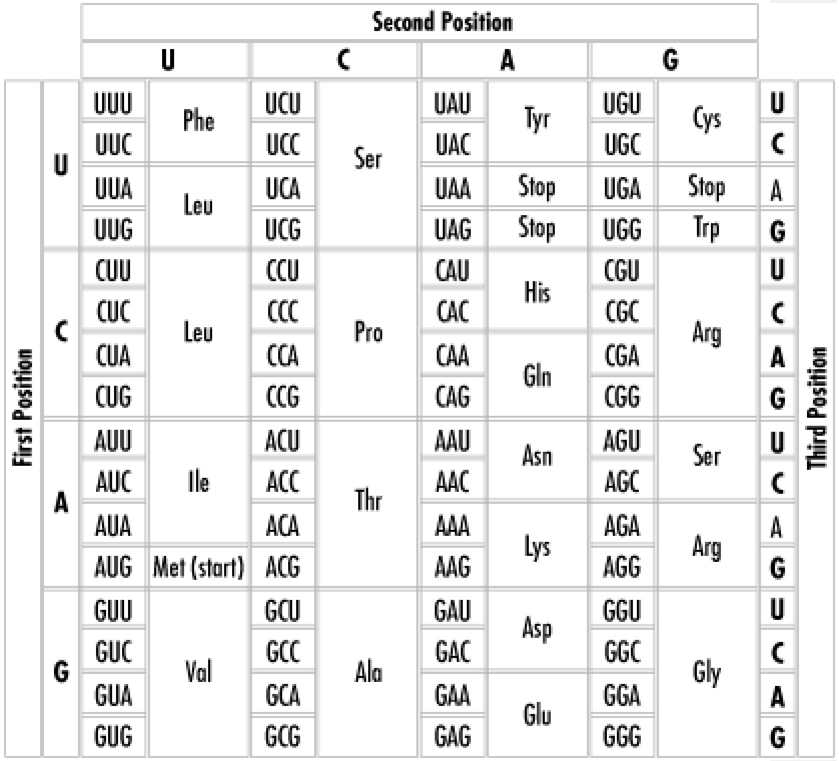
```
#!/usr/bin/perl -w
use strict;
# Count nucleotides in input file

my %count;

while ( my $DNA = <> ) {
     chomp( $DNA );
     # $DNA = “ACGGGAGGACGGGAAATTACTACGGCATTAGC”

     my @nucleotides = split( //, $DNA );
     # ("A","C","G","G","G","A","G","G","A","C","G",G","G","A","A"...)

     foreach my $nucleotide ( @nucleotides ) {
         $count{$nucleotide}++;  # increment by one
     }
}

# %count = ( A => 11, C => 6, G => 11, T => 5 )
while ( my ($nucleotide,$total_number) = each %count ) {
     print "$nucleotide = $total_number\n";
}
```

Unix file handling

```
$ cat dna.txt
ACGGGAGGACGGGAAAATTACTACGGCATTAGC
# make new copy
$ cp dna.txt dna2.txt
# append complement of DNA from dna.txt to dna2.txt
$ ./03-complement-file.pl dna.txt >> dna2.txt
# examine current content of file dna2.txt
$ cat dna2.txt
ACGGGAGGACGGGAAAATTACTACGGCATTAGC
GCTAATGCCGTAGTAATTTTCCCGTCCTCCCGT
# count nucleotides in dna.txt
$ ./04-count.pl dna.txt
A = 11
T = 5
C = 6
G = 11
# and again in dna2.txt - do numbers look OK?
$ ./04-count.pl dna2.txt
A = 16
T = 16
C = 17
G = 17
```

Translating Codons to Amino Acids

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



First Position	Second Position								Third Position
	U		C		A		G		
U	UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys	U
	UUC		UCC		UAC		UGC		C
	UUA	Leu	UCA		UAA	Stop	UGA	Stop	A
	UUG		UCG		UAG	Stop	UGG	Trp	G
C	CUU	Leu	CCU	Pro	CAU	His	CGU	Arg	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	Gln	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser	U
	AUC		ACC		AAC		AGC		C
	AUA		ACA		AAA	Lys	AGA	Arg	A
	AUG	Met (start)	ACG		AAG		AGG		G
G	GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	Glu	GGA		A
	GUG		GCG		GAG		GGG		G

```
# Picture is based on RNA so uracil appears instead of thymine
# we are going directly from DNA to amino acids, So codons use
# thymine instead of uracil
```

Modules and subroutines

```
# define subroutine (in separate file together with %genetic_code)
# and store it in module GeneticCode.pm to be reusable

sub codon2aa {
     my ( $codon ) = @_;

     # check does mapping for codon exists
     if ( exists $genetic_code{ $codon } ) {
          # if it does, return amino acid
          return $genetic_code{ $codon };
     } else {
          # if it doesn't exit with error
          die "bad codon: $codon";
     }
}

# now we can use module directly from command line;
$ perl -MGeneticCode -e "print codon2aa( 'ACG' )"
T
```

Using module

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

# load module (*.pm)
use GeneticCode;

while ( my $DNA = <> ) {
     chomp($DNA);

     my $protein = '';

     # start at beginning and move by three places through DNA
     for ( my $i = 0; $i <= (length($DNA) - 2); $i += 3 ) {
         # extract single codon starting at position $i
         my $codon = substr( $DNA, $i, 3 );
         # call subroutine from GeneticCode module
         $protein .= codon2aa( $codon );
     }

     print "$protein\n";
}
```

Decoding DNA proteins

```
$ cat dna2.txt dna3.txt
ACGGGAGGACGGGAAAATTACTACGGCATTAGC
GCTAATGCCGTAGTAATTTTCCCGTCCTCCCGT
ACGGGAGGACGGGAAAATTACAACGGCATTAGC
GCTAATGCCGTAATAATTTTCCCGACCTCCCGT
$ ./06-dna2protein.pl dna2.txt dna3.txt
TGGRENYYGIS
ANAVVIFPSSR
TGGRENYNGIS
ANAVIIFPTSR
```

Reading frames

```perl
# let's improve our GeneticCode.pm by extending it to DNA2protein.pm

sub DNA2protein {
    my ( $DNA, $offset ) = @_;
    my $protein = '';

    # start at $offset and move by three places through DNA
    for ( my $i=$offset; $i<=(length($DNA)-2-$offset); $i+=3 ) {
        # extract single codon starting at position $i
        my $codon = substr( $DNA, $i, 3 );
        # decode codon to amino acid
        $protein .= codon2aa( $codon );
    }
    # return created protein
    return $protein;
}

sub revcom {
    my ( $DNA ) = @_;
    my $revcom = reverse $DNA;
    $revcom =~ tr/ACGT/TGCA/;
    return $revcom;
}
```

Decoding all reading frames

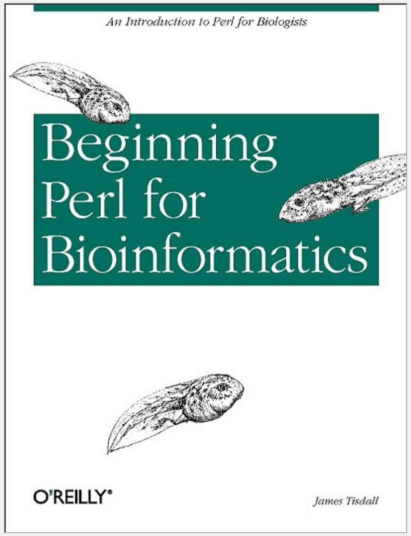
```
#!/usr/bin/perl -w
use strict;

# use module DNA2protein to implement reading frames
use DNA2protein;

while ( my $DNA = <> ) {
     chomp($DNA);

     foreach my $offset ( 0 .. 2 ) {
         print DNA2protein( $DNA, $offset ), "\n";
         print DNA2protein( revcom($DNA), $offset ), "\n";
     }
}

$ ./07-reading-frames.pl dna.txt
TGGRENYYGIS
ANAVVIFPSSR
REDGKITTAL
LMP__FSRPP
GRTGKLLRH_
_CRSNFPVLP
```

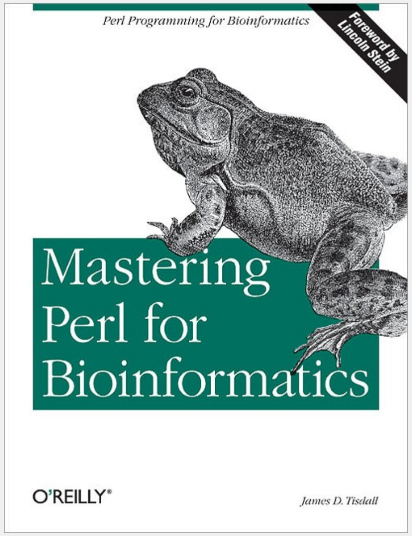
Review

- Why to pursue biology programming?
- Algorithmic way of thinking
- $Scalars, @arrays and %hashes
- Modules as reusable components made of subroutines
- Combination of small tools with pipes (*the Unix way*)

Find out more...

- James Tisdall: **”Beginning Perl for Bioinformatics”**, O'Reilly, 2001
- Lincoln Stein: **"How Perl Saved the Human Genome Project"**, http://www.ddj.com/184410424
- James D. Tisdall: **"Parsing Protein Domains with Perl"**, http://www.perl.com/pub/a/2001/11/16/perlbio2.html
- James Tisdall: **"Why Biologists Want to Program Computers"**, http://www.oreilly.com/news/perlbio_1001.html


An Introduction to Perl for Biologists
Beginning Perl for Bioinformatics
O'REILLY®
James Tisdall


Perl Programming for Bioinformatics
Foreword by Lincoln Stein
Mastering Perl for Bioinformatics
O'REILLY®
James D. Tisdall

Questions?

3*7*2

##