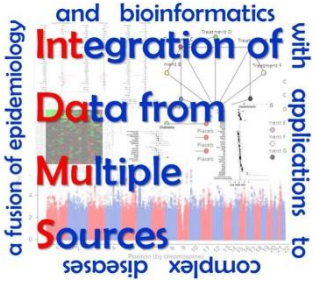


Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Workshop III: Bioinformatics with Perl

October 29, 2015

Instructors: Pantelis Bagos, Panagiota Kontou
University of Thessaly

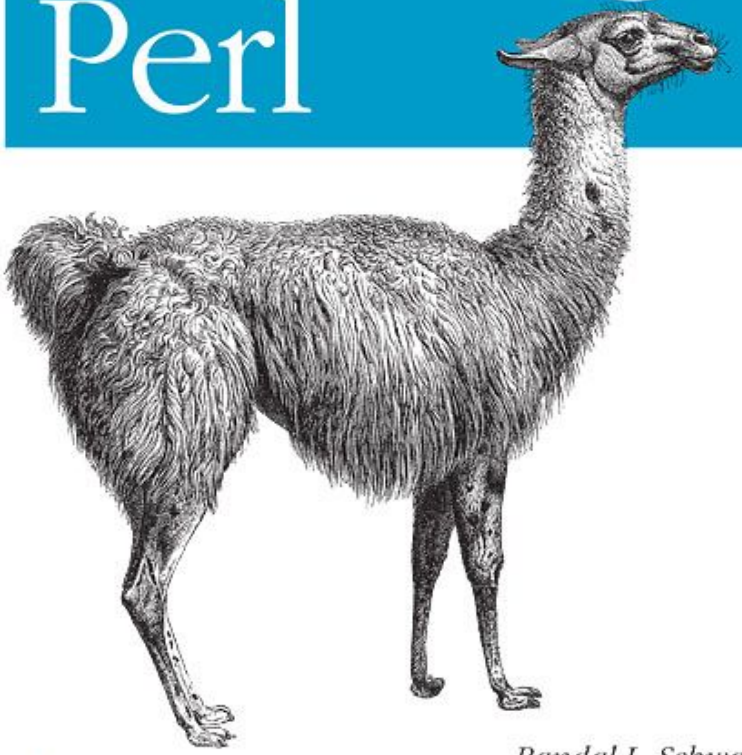
What is Perl?

- Scripting language by Larry Wall
- Practical Extraction and Reporting Language
- Easy language to use
- Fast cross platform text processing.
- Good pattern matching.
- Many extensions for Life Sciences data types

Making Easy Things Easy
& Hard Things Possible

6th Edition
Covers Perl 5.14

Learning Perl



O'REILLY®

Randal L. Schwartz,
brian d foy & Tom Phoenix

Unmatched Power for
Text Processing and Scripting

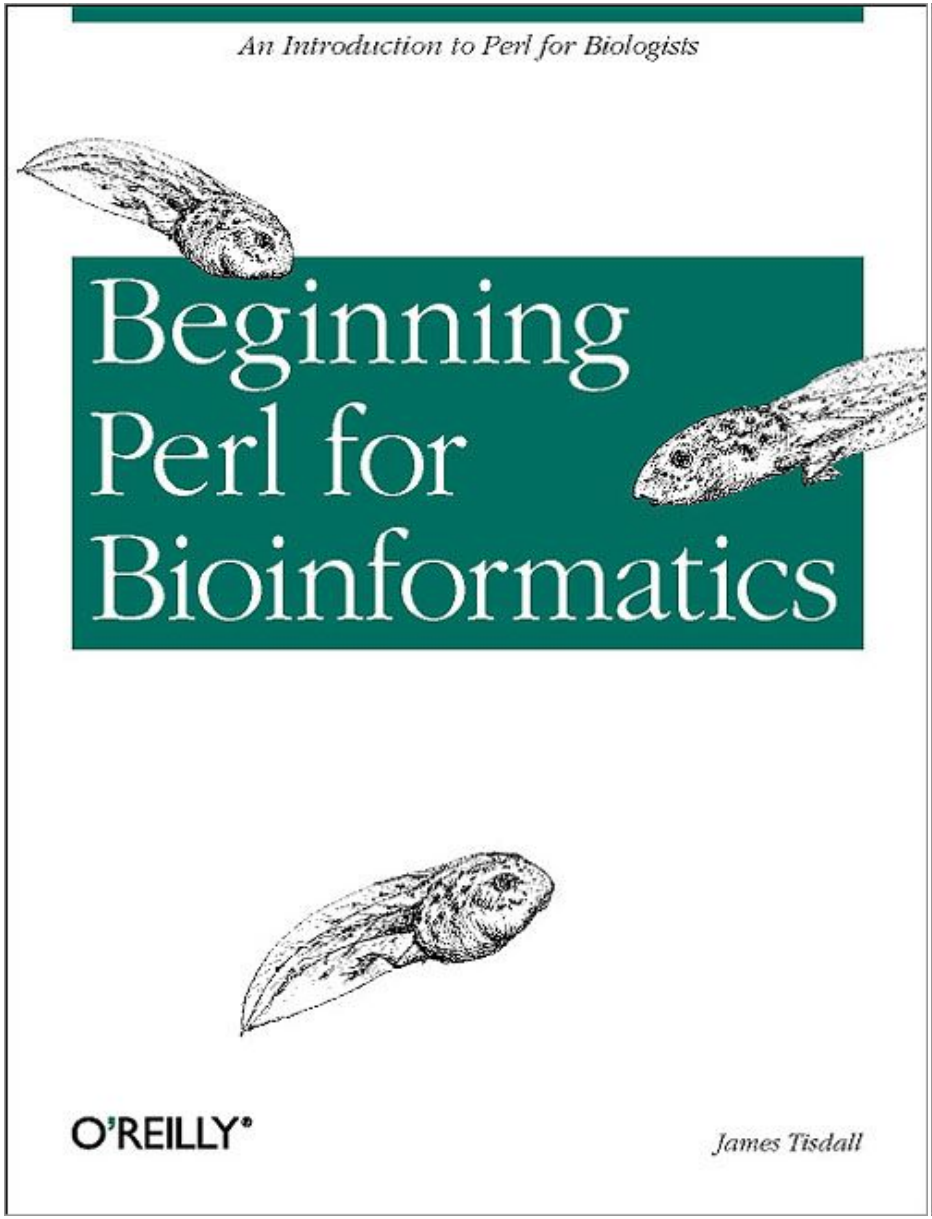
4th Edition
Covers Version 5.14

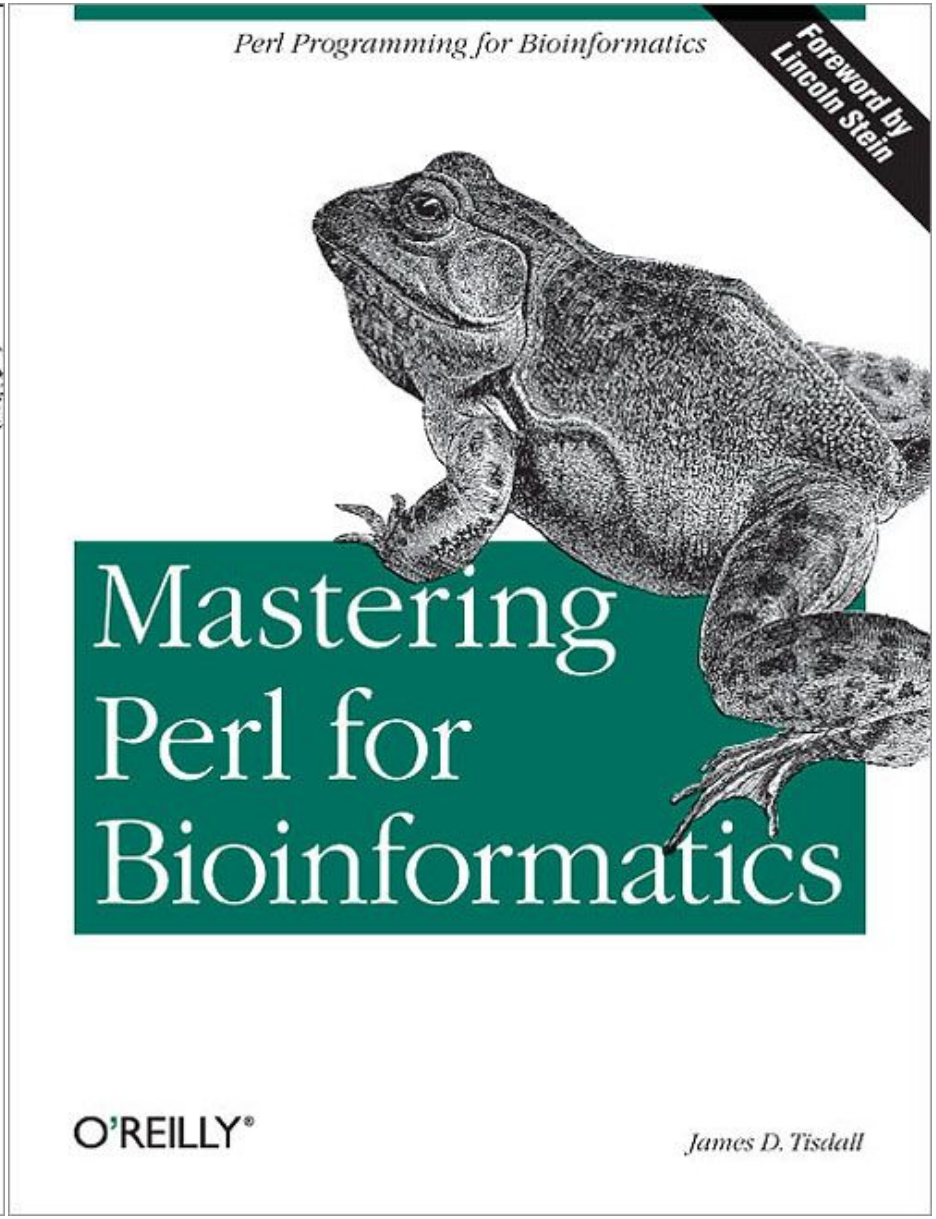
Programming

Perl

O'REILLY®

Tom Christiansen, brian d foy,
Larry Wall & Jon Orwant


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

Getting Started

```
#!/usr/bin/perl
# First Program
print “Hello World\n”;
```

Getting Started

- perl program.pl
- ./program.pl
- perl –e ‘…perl program…’

<STDIN>

```
#!/usr/bin/perl
# Second Program

print “What is your name\?\n”;
$name =<STDIN>;
chomp $name;
print “Hello $name\n”;
```

Scalars και Variables

There are two basic data types in Perl: numbers and strings.

Numbers

- 5
- -245
- 1.25
- 7.25e45
- -12e44
- -1.2e-25

Strings

- Double quotes or single quotes may be used around literal strings:

```
print "Hello, world";
print 'Hello, world';
```

- However, only double quotes "interpolate" variables and special characters such as newlines (\n):

```
print "Hello, $name\n";      # works fine
print 'Hello, $name\n';      # prints $name\n literally
```

- Escape (\t, \n, \U, \L)

printf, sprintf

```
$x=3.7009;
print $x;
printf (“%.3f\n”, $x);
$y=sprintf (“%.3f”, $x);
print “$y\n”;
```

Operators

- Numbers

  +, -, *, /, **, %, <, >, ==, !=

- Strings

  ., x, eq, ne, lt, gt, le, ge

Variables

$variable_name=String or number or expression;

- Special variables ($_, $/, $1,etc)
- $number=5;
- $name=“George”;
- $exp=3*$number+($number+1);
- $a+=5;
- $b*=3;
- $c .=“ ”;
- ++$a; or $a++;
- $a--;

substr, index

- $name=“Takis”;
- $x=substr($name,0,1);
- $y=substr($name,0,1,“L”);
- print “$name\t$x\t$y\n”;
- $position=index($name, “k”);
- $position=rindex($name, “k”);

chomp, chop

- chomp ($x)
- chop ($x)

Interpolation

```
$test=‘bla bla bla’;
print “this is a $test\n”;
print ‘this is a $test\n’;
print ‘this is a \$test\n’;
print “this is a ${test}bla bla\n”;
```

Upper Case, Lower Case

```
$bigname="\Uname";
$name="name";
$bigname="\U$name";
$capname="\u$name"
```

Lists-Arrays

List:
(1, 2, 3)
(“perl”, 3, 15)
($x, 3, $x+2, “$y$x”)
(1..10)

Array
@name=(1..10);

Arrays

- @name=(1..10);
- @table=1;
- @table=@name;
- @name=(“John”, “George”, ”Mike”);
- @name=qw(John George Mike);
- @table=(1, 2, @name, 7);
- ($a, $b, $c)=(1,2,3);
- ($a,$b)=($b,$a);
- ($a, @table)=($c, $d, $e);
- $table=(1,2,3);
  $x=@table;
  ($y)=@table;

Index number of the elements

- $table=(1,2,3);
- $x=$table[0]
- $table[1]++;
- ($table[0], $table[1])= ($table[1], $table[0]);
- @table[0,1]=@table[1,0];
- @table[0,1,2]=@table[1,1,1];
- @table=(1,2,3);$x=$table[3];
- $table[5]=12;
- print $#table;
- print $table[$#table];

push, pop

- push @table,$scalar;
- @table=(@table, $scalar);
- $last=pop(@table);

shift, unshift

- unshift(@table, $scalar);
- @table=($scalar, @table);
- $x=shift(@table);
- ($x, @table)=@table;

reverse, sort, splice

- @table=(1,2,3);
- @new_table=reverse(@table);
- @name=(“John”, “George”, ”Mike”);
- @sort_names=sort(@name);
- splice(@table,1,1);

Hashes

- A hash is a collection of zero or more pairs of scalar values, called keys and values
- An array variable begins with the % sign followed by a legal variable name

```
%genes=('gene1'=>'AACCCGGTTAACCG',
        'gene2'=>'CCAAATTCCCCTTG');
```

Hashes

- %password=(“pbagos”, 123, “pkontou”, 321);
- %table=@table;
- @table=%table;
- %names=reverse %password;

- %password=(“pbagos”, 123, “pkontou”,321);
- @list=keys(%password);
- @list2=values(%password);

Hashes

%password=(“pbagos”=> “123”, “pkontou”=>”321”);

@password{“pbagos”, “pkontou”}=(123, 321);

Hashes

$hash{“key”}=“value”

- $password{“pbagos”}=“123”;
- $password{“pkontou”}=“321”;
- print $password{“pbagos”};
- $name=“pbagos”;
- print $password{$name};

Conditions and Loops

- if/unless
- while/until
- do{}while/until
- for
- foreach
- last, next, redo

if

```
if (some condition)
{
}
elsif
{
}
else
{
}
```

while/until

```
while (condition)
{
…
}
until (condition)
{
…
}
```

do {}while/until

```
do
{
…
} while (condition)
```

```
do
{
…
} until (condition)
```

for

```
for (initialization; condition; update)
{
…
}

for ($i=1; $i<=10;$i++)
{
print “$i\n”;
}
```

foreach

```
foreach $i(@list)
{
…
}

@a=(1,2,3,4,5);
foreach $i(@a)
{
print “$i\n”;
}
```

last

```
while (condition 1)
{
…
if(condition 2)
{
…
last;
}
}
```

Can be used only in: for, foreach, while, until

next

```
while (condition 1)
{
…
if(condition 2)
{
…
next;
}
```

redo

```
while (condition 1)
{
#
…
if(condition 2)
{
…
redo;
}
}
```

<>

```
while(<>)
{
print $_;
}
```

Run:
Perl program.pl file

Input-Output

- open FILEHANDLE, “filename”;
- open IN, “/etc/passwd”;
- $x=<IN>;
- print $x;
- close IN;
- open OUT, “>(>)tempfile”;
- print OUT “bla bla bla\n”;

@ARGV

- perl program.pl file1 file2 …
- file1: $ARGV[0]
- file2: $ARGV[1]

…

Regular Expressions

- A *regular expression* is a simple way of matching a series of symbols to a pattern you have in mind

```
$pattern=~/abc/;

if(/abc/){
print “I found it”;
}
```

Regular Expressions Syntax

char	meaning
^	beginning of string
$	end of string
.	any character except newline
*	match 0 or more times
+	match 1 or more times
?	match 0 or 1 times; or: shortest match
|	alternative
( )	grouping; “storing”
[ ]	set of characters
{ }	repetition modifier
\	quote or special

Regular Expressions Syntax

Matching

\w matches any single character classified as a “word” character
\W matches any non-“word” character
\s matches any space character
\S matches any non-space character
\d matches any digit character, equiv. to [0-9]
\D matches any non-digit character

Examples

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, such as foo and 12bar8 and foo_1

Doing Substitutions

Replace every “Hello” with a “Hi"

$string =~ s/Hello/Hi/;

Doing Translations

Translations are like substitutions, except they happen on a letter by letter basis instead of substituting a single phrase for another single phrase

```
$string =~ tr/[a,e,i,o,u,y]/[A,E,I,O,U,Y]/;
$string =~ tr/[A,E,I,O,U,Y]/[1,2,3,4,5]/;
$string =~ tr/[a-z]/[A-Z]/;
```

split

```
#!/usr/bin/perl
$data = 'Becky Alcorn,25,female,Melbourne';
@values = split(',', $data);
  foreach $val (@values) {
    print "$val\n";
  }
```

This program produces the following output:

```
Becky Alcorn
25
female
Melbourne
```

join

```
#!/usr/bin/perl

$string = join( "-", "one", "two", "three" );
Print "Joined String is $string\n";
```

This program produces the following output:

```
Joined String is one-two-three
```

Application in Bioinformatics

Genetic code

You have to write a Perl program that:

1) Takes as input a DNA sequence

2) Finds the complementary strand and the mRNA that is produced

3) Translates the DNA using all 6 possible reading frames and outputs the possible ORFs and the putative protein sequences.

```
%genetic_code = (
'GCA'=>'A', #Alanine
'GCC'=>'A', #Alanine
'GCG'=>'A', #Alanine
'GCT'=>'A', #Alanine
'AGA'=>'R', #Arginine
'AGG'=>'R', #Arginine
'CGA'=>'R', #Arginine
'CGC'=>'R', #Arginine
'CGG'=>'R', #Arginine
'CGT'=>'R', #Arginine
'AAC'=>'N', #Asparagine
'AAT'=>'N', #Asparagine
'GAC'=>'D', #Aspartic acid
'GAT'=>'D', #Aspartic acid
'TGC'=>'C', #Cysteine
'TGT'=>'C', #Cysteine
'GAA'=>'E', #GlTtamic acid
'GAG'=>'E', #GlTtamic acid
'CAA'=>'Q', #GlTtamine
'CAG'=>'Q', #GlTtamine
'GGA'=>'G', #Glycine
'GGC'=>'G', #Glycine
'GGG'=>'G', #Glycine
'GGT'=>'G', #Glycine
'CAC'=>'H', #Histidine
'CAT'=>'H', #Histidine
'ATA'=>'I', #IsoleTcine
'ATC'=>'I', #IsoleTcine
'ATT'=>'I', #IsoleTcine
'TTA'=>'L', #LeTcine
'TTG'=>'L', #LeTcine
'CTA'=>'L', #LeTcine
'CTC'=>'L', #LeTcine
'CTG'=>'L', #LeTcine
'CTT'=>'L', #LeTcine
'AAA'=>'K', #Lysine
'AAG'=>'K', #Lysine
'ATG'=>'M', #Methionine
'TTC'=>'F', #Phenylalanine
'TTT'=>'F', #Phenylalanine
'CCA'=>'P', #Proline
'CCC'=>'P', #Proline
'CCG'=>'P', #Proline
'CCT'=>'P', #Proline
'AGC'=>'S', #Serine
'AGT'=>'S', #Serine
'TCA'=>'S', #Serine
'TCC'=>'S', #Serine
'TCG'=>'S', #Serine
'TCT'=>'S', #Serine
'ACA'=>'T', #Threonine
'ACC'=>'T', #Threonine
'ACG'=>'T', #Threonine
'ACT'=>'T', #Threonine
'TGG'=>'W', #Tryptophan
'TAC'=>'Y', #Tyrosine
'TAT'=>'Y', #Tyrosine
'GTA'=>'V', #Valine
'GTC'=>'V', #Valine
'GTG'=>'V', #Valine
'GTT'=>'V', #Valine
'TAA'=>'-', #STOP
'TAG'=>'-', #STOP
'TGA'=>'-', #STOP
);
```

```
$seq="AAAAAAATTAATAGATGAACATATATATAGATTTTCTATATAGACCCTCTACCCGATAAGGCTAC";
$seq2=$seq;
$seq2=~tr/ATCG/TAGC/;
$seq2=reverse($seq2);

for($i=0;$i<=length($seq)-3;$i++)
{
    $x=substr($seq,$i,3);

    if ($x  eq 'ATG')
    {
            for($j=$i;$j<=length($seq)-3;$j=$j+3)
            {
                $y=substr($seq,$j,3);
                $k=$genetic_code{$y};
                if($k eq '-')
                {

                print"\n";
                last;
                    }

                print "$k";

            }
    }
}
```

```
for($l=0;$l<=length($seq2)-3;$l++)
{
     $m=substr($seq2,$l,3);

     if ($m  eq 'ATG')
     {
              for($n=$l;$n<=length($seq2)-3;$n=$n+3)
              {
                   $o=substr($seq2,$n,3);
                   $p=$genetic_code{$o};
                   if($p eq '-')
                   {

                   print"\n";
                   last;
                       }

                   print "$p";

              }
     }
}
```

Uniprot to FASTA

In this practical you will have to write a simple Perl script to convert the Uniprot Format to Fasta Format

```
$/="\/\/\n";
while (<>)
{
    if ($_=~/^AC\s{3}(.*?)\;/m)
    {
        print ">$1\n";
    }
    while ($_=~/^      (.*)/mg)
    {
        $line=$1;
        $line=~s/\s//g;
        print $line;
    }
    print "\n";
}
$/="\n";
```

Shuffle sequences

In this practical you will have to write a Perl script to shuffle sequences within a protein dataset

```
while(<>)
{
 $c=0;
 if ($_=~/^>/)
 {
     push @id, $_;
     $seq=<>;
     push @seq, $seq;
     push @c, $c;
     $c++;
 }
}
for ($x=$#c; $x>=0; $x--){
 $rnd=int(rand($x));
 #print $rnd;
 print $id[$rnd].$seq[$rnd];
 splice(@id,$rnd,1);
 splice(@seq,$rnd,1);
 splice(@c,$rnd,1);
}
```

Random Sequences

In this practical you will have to write a Perl script to generate random protein sequences in Fasta format

```
@BASES = ( 'A', 'T', 'C', 'G','D', 'E', 'F','H', 'I','K', 'L', 'M',
'N', 'P', 'Q', 'R','V' ,'W', 'Y', 'S' );

for ( $i=0; $i<500; $i++ ) {
  print '>Random', "$i\n";
  for( $j=0; $j<200; $j++ ) {
    $r = $BASES[ int (rand 20)];
    print $r;
    print "\n" if ($j+1)%60 == 0 and $j;
  }
  print "\n";
}
```

Amino acid Composition

In this practical you will have to write a Perl script to calculate the amino acid composition of protein sequences.

```
$arxeio_sequence = $ARGV[0];
open IN, $arxeio_sequence;
while (<IN>)
{
    if ($_=~/^>/)
    {
        $id=$_;
        chomp $id;
        print $id."\t";
        $seq=<IN>;
        chomp $seq;
    }
        $counter=0;
        @split_seq = split(//, $seq);
        foreach $a(@split_seq)
        { $counter++; }

    print $counter." AA\n";
    @amino_acids = (A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y);

    foreach $z(@amino_acids)
        {
        $count = $seq =~s/$z//g;
        $diairesi = $count/$counter;
        $pososto=sprintf( "%.3f", $diairesi );
        }
    print "\n";
}

close IN;
```

Lipoprotein signals

- The purpose of this practical is to find the presence of lipoprotein signal peptide in bacterial proteins using regular expressions
- You will have to download the 63 sequences from Gram-negative Bacteria
- In the first place you will have to use the scripts that you wrote in the previous practicals and convert the sequences to Fasta format.
- For making the subsequently calculations easier you could choose a single-line fasta format.

Lipoprotein signals

- Afterwards, you should write a simple program to test the existence of the lipo-box.
- The regular expression patterns that could be used are:

- LA[GA]C
- [LVI][ASTG][GA]C
- [^DERK]{6}[LIVMFWSTAG]{2}[LIVMFYSTAGCQ][AGS]C
- [MV].{0,13}[RK][^DERK]{6,20}[LIVMFESTAG][LVIAM][IVMSTAFG][AG]C

References
•Juncker AS, Willenbrock H, Heijne GV, et al. **Prediction of lipoprotein signal peptides in Gram-negative bacteria**. 2003;12:1652-1662. [PDF]
•Sutcliffe IC, Harrington DJ. **Pattern searches for the identification of putative lipoprotein genes in Gram-positive bacterial genomes**. *Microbiology (Reading, England)*. 2002;148(Pt 7):2065-77. [PDF]
•Bagos PG, Tsirigos KD, Liakopoulos TD, Hamodrakas SJ. **Prediction of lipoprotein signal peptides in Gram-positive bacteria with Hidden a Markov Model**, 2008, *J Proteome Research*, 7(12):5082-93 [PDF] [Pubmed] [Google Scholar]

Lipoprotein signals

- Check which of the proposed patterns performs better
- Afterwards, you should remove the mature part of the protein and keep only the sequence of the signal peptide (including the Cysteine) and write a program for aligning the sequences to the right.
- For instance if there were two sequences with lengths of signal peptide equal to 25 and 30 respectively, the former should have five gaps (-) preceeding the initial Methionine.
- This special form of a multiple alignment should be used for performing analyses of the aminoacid frequencies in the lipobox and an easy way to perform such an analysis is using the WebLogo sever (http://weblogo.berkeley.edu/)

LA[GA]C

```
while (<>){
   if    ($_=~/^>(.*)/)
{
        $name=$1;
        $seq=<>;

        if($seq=~/(.*LA[GA]C)/)
        {
            $x=length($1);

            print "$name\t LIPOPROTEIN \t $x\t $1\n";

            $a=$a+1;     }
     else
     {
         print "$name\t NO LIPOPROTEIN\n";      }
}
}
print "$a LIPOPROTEINS FOUND";
```

[LVI][ASTG][GA]C

```
while (<>){
   if    ($_=~/^>(.*)/)
{
        $name=$1;
        $seq=<>;

        if($seq=~/(.*[LVI][ASTG][GA]C)/)
        {
            $x=length($1);

            print "$1\n";
            $a=$a+1;
        }
     else
     {
          print "$name\t NO LIPOPROTEIN\n";      }
}
}
print "$a LIPOPROTEINS FOUND";
```

[^DERK]{6}[LIVMFWSTAG]{2}[LIVMFYSTAGCQ][AGS]C

```
while (<>){
   if    ($_=~/^>(.*)/)
{
        $name=$1;
        $seq=<>;

        if($seq=~/(.*[^DERK]{6}[LIVMFWSTAG]{2}[LIVMFYSTAGCQ][AGS]C)/)
{
            $x=length($1);

         print "$name\t LIPOPROTEIN \t $x\t $1\n";
         $a=$a+1; #YPOLOGIZEI POSES LIPOPROTEINES VRISKEI
        }
     else
     {
          print "$name\t NO LIPOPROTEIN\n";
     }
}
}
print "$a LIPOPROTEINS FOUND";
```

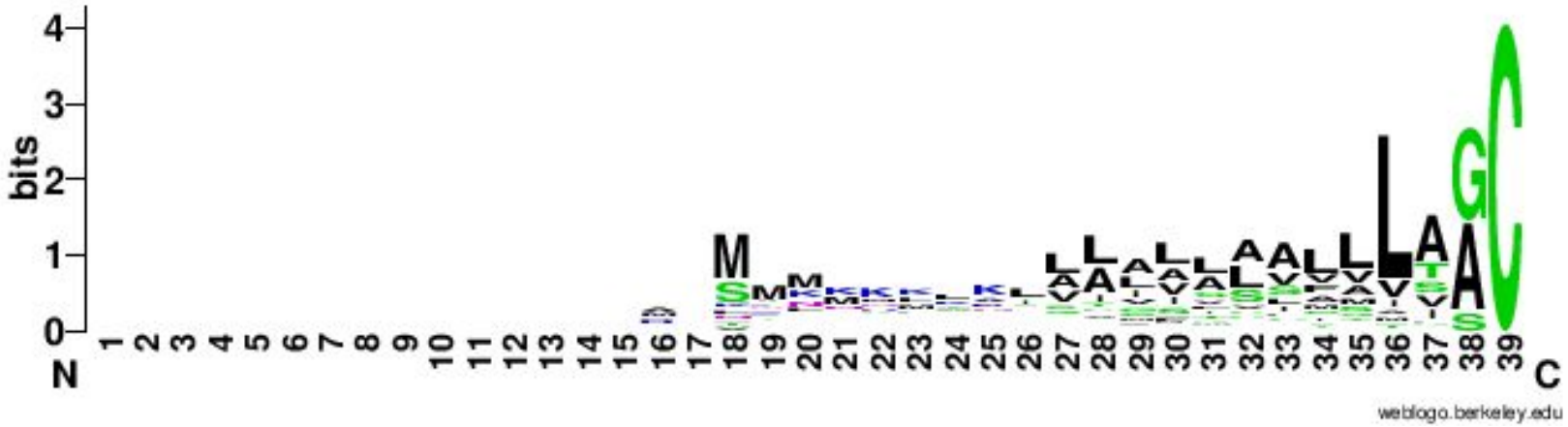
[MV].{0,13}[RK][^DERK]{6,20}[LIVMFESTAG][LVIAM][IVMSTAFG][AG]C

```
while (<>){
if   ($_=~/^>(.*)/)
{
        $name=$1;
        $seq=<>;
            if($seq=~/^([MV].{0,13}[RK][^DERK]{6,20}[LIVMFESTAG][LVIAM]
[IVMSTAFG][AG]C)/)
{
            $x=length($1);

            print "$name\t LIPOPROTEIN \t $x\t $1\n";           $a=$a+1;
}
else
{
print "$name\t NO LIPOPROTEIN\n";
}

}
}
print "$a LIPOPROTEINS FOUND";
```

```
open out,(">>Out.txt");
while (<>){
  if   ($_=~/^>(.*)/)
  {
     $name=$1;
        $seq=<>;

        if($seq=~/^([MV].{0,13}[RK][^DERK]{6,20}[LIVMFESTAG][LVIAM][IVMSTAFG][AG]C)/)
        {
           $x=length($1);

        }

       if ($max<$x)
       {

              $max=$x;
       }

  }
}
continue {
    if (eof) {    #an eisai sth teleutaia  grammi tou arxeiou
        seek ARGV, 0, 0; #gurizei to filehandler sthn 1h grammh tou arxeiou mas
        last;
    }
}
while (<>){
  if   ($_=~/^>(.*)/)
  {
     $name=$1;
        $seq=<>;
        if($seq=~/^([MV].{0,13}[RK][^DERK]{6,20}[LIVMFESTAG][LVIAM][IVMSTAFG][AG]C)/)
        {
           $x=length($1);

              #print out ">$name\n ";
              for($i=0; $i<=$max-$x; $i++)
              {
                     print out "-";
              }
              print out "$1\n";
           $a=$a+1;
       }
}
}
```

```
------------------MRRCMPLVAASVAALMLAGC
-----------------MKLKQLFAITAIASALVLTGC
------------------MKLLSKIMIIALAASMLQAC
--------------MNKNRGFTPLAVVLMLSGSLALTGC
-------------------MKRQALAAMIASLFALAAC
--------------------MRLLPLVAAATAAFLVVAC
-----------------------MRIVIFILGILLTSC
---------------------MFKRFIFITLSLLVFAC
---------------------MLKKVYYFLIFLFIVAC
---------------------MKKILLTVSLGLALSAC
-----------------MVKKAIVTAMAVISLFTLMGC
-----------------MKQLIVNSVATVALASLVAGC
-------------------MKLKTLALSLLAAGVLAGC
--------------------MKAYLALISAAVIGLAAC
--------------------MKLKATLTLAAATLVLAAC
------------------MQKTPKKLTALCHQQSTASC
----------------MPLPDFRLIRLLPLAALVLTAC
----------------MKNQVKKILGMSVVAAMVIVGC
---------------------MKKFLPLSISITVLAAC
-----------------MKRLFLSFVALALLAGSIAAC
---------------------MCGKILLILFFIMTLSAC
---------------------MSKRLLSLASLALLFGC
-------------------MFKRRYVTLLPLFVLLAAC
-------------------MKKIIKLSLLSLSIAGLASC
-----------------MGRSKIVLGAVVLASALLAGC
-----------------MKAKIVLGAVILASGLLAGC
-----------------MNNVLKFSALALAAVLATGC
--------------MKLTTHHLRTGAALLLAGILLAGC
-------------MAYSVQKSRLAKVAGVSLVLLLAAC
------------MSAGSPKFTVRRIAALSLVSLWLAGC
```

```
-MDKGEGLRLAATLRQWTRLYGGCHLLLGAVVCSLLAAC
-------------------MKPFLRWCFVATALTLAGC
-----------------MNIATKLMASLVASVVLTAC
----------------MQNAKLMLTCLAFAGLAALAGC
----------------------MKKYLLGIGLILALIAC
----------------------MRLLIGFALALALIGC
--------------MFVTSKKMTAAVLAITLAMSLSAC
-----------------MNKNMAGILSAAAVLTMLAGC
-----------------MHVSSLKVVLFGVCCLSLAAC
----------------MYKNGFFKNYLSLFLIFLVIAC
-------------------MNKFVKSLLVAGSVAALAAC
-------------------MKKTNMALALLVAFSVTGC
----------------MSLTHYSGLAAAVSMSLILTAC
-------------------MLRYTRNALVLGSLVLLSGC
---------------------MRNFILFPMMAVVLLSGC
---------------------MRKQWLGICIAAGMLAAC
-------------------MRYLATLLLSLAVLITAGC
-------------------MNMTKGALILSLSFLLAAC
----------------MNKKIFTLFLVVAASAIFAVSC
---------------------MVKRGRFALCLAVLLGAC
-----------------MKVKYALLSAGALQLLVVGC
-----------------MNNPLVNQAAMVLPVFLLSAC
----------------MNAHTLVYSGVALACAAMLGSC
----------------MKLKSLVFSLSALFLVLGFTGC
-----------------MREKWVRAFAGVFCAMLLIGC
---------------MKHNVKLMAMTAVLSSVLVLSGC
---------------------MKLRLSALALGTTLLVGC
-----------MRKRISAIINKLNISIIIMTVVLMIGC
-------------------MRKRISAIIMTLFMVLVSC
-----------MRKRISAIINKLNISIMMMIVVLMIGC
```


bits
4
3
2
1
0
N
C
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
weblogo.berkeley.edu