PERL EXERCISES

by Gilabert Navarro, Joan Francesc

0. INSTRUCTIONS

You have to submit this MarkDown file with your answers to the different exercises. First, download the template from this link (Surname_Name_PERL1516_exercises.md), then rename it using your *surname* and *name*. You must change the "`SURNAME, Name`" string within this document too (see line 3). Each exercise has a section with a special comment inside that says "`# ... YOUR ANSWER HERE ...`", just replace that string with your `Perl` code. You can add some comments outside the code bloc defined by the triple backticks lines. You must submit the modified MarkDown text file through this Aula Global link, you do not need to compile it; if you want to try anyway, see code below that requires `pandoc` and some `LaTeX` packages installed in your system.

Document compiled with:

```
pandoc -f markdown_github -t latex      \
       --variable papersize:a4paper     \
       --variable geometry:margin=1.5cm \
       --variable fontsize=11pt         \
       --highlight-style pygments       \
       -o Surname_Name_PERL1516_exercises.pdf \
          Surname_Name_PERL1516_exercises.md
```

1. PERL Basic Exercises

Exercise 1.A

What happens if we set the condition of a `while` to `1`, like in the following code:

```
while (1) {
    print $i++,"\n";
};

# If we run the code given we will have an infinite loop, since any number different than zero is
```

Exercise 1.B

Can you write a short script that will print a countdown on the terminal ?
A tip: you can use `sleep(10);` within the countdown loop to get it readable. You can also use the "`\r`" character instead of "`\n`", and see what happens...

"'{.perl .numberLines}

!/usr/bin/perl

use strict;
use warnings;

for (my $i = 10 ; $i >= 0 ; $i = $i - 1){
print STDERR $i;

print STDERR " “;
print STDERR”\r“;
sleep 10;
}

In this exercise the output is printed through the standard error, since it is unbuffered. The standard output is buffered and the numbers will not be seen in the #screen if we don't use a newline because of the small size of the numbers, that don't fill the buffer

“‘

Exercise 1.C

Write a script that will keep reading your input from the terminal. It should print back "`Tell me more...`" every time the user ends a line with `<RETURN>`.

“‘{.perl .numberLines}

!/usr/bin/perl

use strict;
use warnings;

while(<>){
print “Tell me more. . . \n”;
}
“‘

2. PERL Scalar Vars Exercises

Exercise 2.A

Using the "`$line = <STDIN>;`" instruction, write a short script that computes the factorial of a number, `f(x) = x!`, provided by the user.

“‘{.perl .numberLines}

!/usr/bin/perl

use strict;
use warnings;

print “Please, insert a number: \n”;
my $line=;
chomp($line);
my $fact=factorial($line);
print “$line! = $fact\n”;

sub factorial {
my $n=$_[0];
my $fact;

if ($n < 2) {
$fact = 1;
}
else{
$fact = $n*factorial($n-1);
}
return $fact;
}
“‘

Exercise 2.B

Can you modify the previous section script, which counts the frequencies of nucleotides for a given sequence, in order to count the observed frequency of a given dinucleotide, for instance “TT”.

“‘{.perl .numberLines}

!/usr/bin/perl

use strict;
use warnings;

declaring some variables

my ($dna_seq,$subseq);

initializing variables

$dna_seq = “ATGCATTGGGGAACCCTGTGCGGATTCTTGTGGCTTTGGCCCTATCTTTTCTAT-GTCCAAGCTG”.
“TGCCCATCCAAAAAGTCCAAGATGACACCAAAACCCTCATCAAGACAATTGTCACCAGGATCAA”;

Looping throught the sequence

my $seqlen = length($dna_seq);
my $sub_string = “TT”;
my @seq_arr=split(//,$dna_seq);
$subseq=0;

for(my $i=0;$i<($seqlen-1);$i++){
($seq_arr[$i].$seq_arr[$i+1] eq $sub_string) && ($subseq++);
}
my $freq = $subseq/$seqlen;
my $freq_str= sprintf(“%.2f”,$freq);

Printing results

print “The frequency of $sub_string in the sequence = $freq_str\n”;
“‘

Exercise 2.C

The exercise above can be extended easily to print all the nucleotide positions where the given dinucleotide was found, remember that we are talking about "TT".

"'{.perl .numberLines}

!/usr/bin/perl

use strict;
use warnings;

declaring some variables

my ($dna_seq,$subseq);

initializing variables

$dna_seq = "ATGCATTGGGGAACCCTGTGCGGATTCTTGTGGCTTTGGCCCTATCTTTTCTAT-GTCCAAGCTG".
"TGCCCATCCAAAAAGTCCAAGATGACACCAAAACCCTCATCAAGACAATTGTCACCAGGATCAA";

Looping throught the sequence

my $seqlen = length($dna_seq);
my $sub_string = "TT";
my @seq_arr=split(//,$dna_seq);
$subseq=0;
my @index_array;

for(my $i=0;$i<($seqlen-1);$i++){
if ($seq_arr[$i].$seq_arr[$i+1] eq $sub_string){
$subseq++;
push @index_array, $i+1;
}
}

my $index_string = join ',',@index_array;

Printing results

print STDOUT "The list of the positions of $sub_string in the sequence = $index_string\n";
"'

3. PERL Array Vars Exercises

Exercise 3.A

Write a short script that reads a set of numbers from the terminal, pushes them into an array, then it has to compute the median for that set, that is the middle value when a set of data is arranged in order of increasing magnitude (second quartile, 50%).

“‘{.perl .numberLines}

!/usr/bin/perl

use strict;
use warnings;
my @array;
my $num;
my $median=0;

print “Please introduce as many numbers as you wish separated by an space:\n”;

$num=<>;

chomp($num);
@array=split(/ /,$num);

@array = sort { $a <=> $b } @array;
my $n_elem= $#array + 1;

if ($n_elem % 2 == 0){
$median=(@array[int($n_elem/2)-1]+@array[int($n_elem/2)])/2;
}
else{
\protect\T1\textdollarmedian=@array[int($n_elem/2)];
}
print “The median of the numbers is: $median\n”;

`## Exercise 3.B`
`Encode the following matrix into a Perl variable:`

1 2 3
5 7 12
10 12 13
‘`then calculate a new matrix where each`X’$\{i,j\} = X\{j,i\}$‘`. Print the resulting matrix and its diagonal vector (**do NOT use`Data::Dumper‘ module**).

“‘{.perl .numberLines}

!/usr/bin/perl

use strict;
use warnings;

my @matrix=([1,2,3],[5,7,12],[10,12,13]);

my $nrows=scalar @matrix;
my $ncols=scalar @{$matrix[0]};
my @tmatrix;

for (my $i = 0 ; $i < $nrows ; $i++){
for (my $j = 0 ; $j < $ncols ; $j++){
$tmatrix[$i][$j]=$matrix[$j][$i];
}
}

```
print “The transposed matrix is:\n”;
for (my $ii = 0 ; $ii < $nrows ; $ii++){
for (my $jj = 0; $jj < $ncols ; $jj++){
print $tmatrix[$ii][$jj] . " “;
}
print”\n“;
}
print”And its diagonal vector:\n“;

for (my $iii=0;$iii<$nrows;$iii++){
print $tmatrix[$iii][$iii] . " “;
}
print”\n“;
“‘
```

Exercise 3.C

Take a set of integers from the terminal and push them into a Perl array. For every pair of elements, say here `x_n` and `x_{n+1}`, on that array, calculate the aritmetic mean, `(x_n + x_{n+1}) / 2`, and insert the corresponding value between the two elements of that array (**hint:** you can use `splice` function or an auxiliary array). Print out the resulting list of numbers in a single line, using single whitespaces as output field separator.

“‘{.perl .numberLines}

```
!/usr/bin/perl

use strict;
use warnings;

my @array;
my $num;

print “Please introduce as many numbers as you wish separated by an space:\n”;

$num=<>;

chomp($num);
@array=split(/ /,$num);
my $nelem=scalar @array - 1;
my @final_array=($array[0]);
for (my $i = 0 ; $i < $nelem ; $i++){
push @final_array,($array[$i]+$array[$i+1])/2;
push @final_array,$array[$i+1];
}
my $print_stg=join(" “,@final_array);
print $print_stg .”\n“;
“‘
```

4. PERL Hash Vars Exercises

Exercise 4.A

Gene features can be easily grouped in GFF format thanks to the ninth column, which often contains a gene identifier. From the GFF file available from this link, write a script to count every feature found for each gene. **As a hint**, you can use the gene identifier (the ninth column) as a primary key, and the feature

field (the third column) as a secondary key; then you can increment the value assigned to the secondary key, remember to initialize to 0, each time you find that feature.

“‘{.perl .numberLines}

!/usr/bin/perl

```
use strict;
use warnings;

my $file=‘feats.gff’;

open(my $fh,‘<’,$file)
or die $!;

my $line;
my @aux_array;
my @genes_array;
my %features;
while(<$fh>){
$line=$_;
chomp($line);
@aux_array = split(/\s/, $line);
if (exists($features{$aux_array[8]})){
if (exists($features{$aux_array[8]}{$aux_array[2]})){
$features{$aux_array[8]}{$aux_array[2]} += 1;
}
else{
$features{$aux_array[8]}{$aux_array[2]} = 1;
}
}
else{
$features{$aux_array[8]}={$aux_array[2] => 1};
push @genes_array,$aux_array[8];
}
}
my $outfile=“output4_1.txt”;
open (my $fh2,“>”,$outfile)
or die $!;
foreach my $key (@genes_array){
print $fh2 $key . “\t”;
foreach my $seckey (keys %{$features{$key}}){
print $fh2 $seckey . “\t” . $features{$key}{$seckey} . “\t”;
}
print $fh2 “\n”;
}
close $fh2;
```

“‘

Exercise 4.B

Imagine you have two files in tabular format; both having the same kind of data: just an identifier on the first column, and a numeric value on the second. Write a script that merges the two files into a single output having three columns: the identifier, the value for that id from file 1, and the corresponding value from file 2. Where no value is present on one of the two files, just return the “`NA`” string (for instance, check if the corresponding key is defined, then return the value, otherwise return the *Not-Available* value). Here you can find the links for two example files: fileA.tbl and fileB.tbl

“‘{.perl .numberLines}

!/usr/bin/perl

```
use strict;
use warnings;

my $file1=‘fileA.tbl’;
open (my $fh,‘<’,$file1)
or die $!;
my $line;
my %output;
my @aux_array;
while(<$fh>){
$line = $_;
chomp $line;
@aux_array = split(/\s/, $line);
if (not exists($output{$aux_array[0]})){
$output{$aux_array[0]}=[$aux_array[1]," NA “];

}


}
close($fh);

my $file2=‘fileB.tbl’;
open (my $fh2,‘<’,$file2)
or die $!;

while(<$fh2>){
$line = $_;
chomp $line;
@aux_array = split(/\s/, $line);
if (not exists($output{$aux_array[0]})){
$output{$aux_array[0]}=[" NA “,$aux_array[1]];
}
else{
@{$output{$aux_array[0]}}[1]=$aux_array[1];

}


}
close($fh2);

my $out_file=‘output_4_2.txt’;
open(my $fh3,‘>’,$out_file)
or die $!;
my @key_arr= sort {$a <=> $b} keys %output;
print $fh3 “Id\tValueA\tValueB\n”;
foreach my $k (@key_arr){
print $fh3 join(“\t”,$k,@{$output{$k}}) . “\n”;
}
close($fh3);
```

“‘