Perl TD1
Listes et Tableaux

Résumé

N'oubliez pas la documentation perl avec la commande *perldoc -f cmd.*

Une *liste* représente des données scalaires ordonnées, un tableau est une variable qui contient une liste.

```
# Assigner une liste a une variable tableau:
my @tab = (9,14,8);
my @cop = @tab; # recopie de tableau
# Acces a un element du tableau
my $b = $tab[0]; # 1ere case
$tab[1] = 5;
my $a = $tab[$#tab]; # derniere case, i.e. taille-1 du tableau
# Parcours du tableau:
for (my $i=0; $i<=$#tab; $i++){    OU   for my $elem (@tab) {
    print $tab[$i], " ";                   print $elem, " ";
}                                      }
```

Fonctions **split** et **join** : on peut utiliser des caractères (ou des expressions régulières) pour diviser une chaîne en champs. La fonction **split** remplit cette tâche, tandis que la fonction join recolle les morceaux :

```
my @fields = split; # decoupe $_ selon les blancs
                # equivalent a split(" ", $_)
@fields = split(":", $line); # decoupe $line selon les ':'
my $result = join("+",@fields); # "$fields[0]+$fields[1]+..."
$result = join("+","x",@fields); # "x+$fields[0]+$fields[1]+..."
$result = join("+",@fields,"x"); # "$fields[0]+$fields[1]+...+x"
```

Fonctions **push** et **pop** : pour ajouter et enlever des éléments du côté droit de la liste :

```
my @tab = (1,2,3);
push(@tab,4) # 1 2 3 4
my $b = pop(@tab); # 1 2 3 et $b=4
push(@tab,5,6,7) # 1 2 3 5 6 7
```

Fonctions **shift** et **unshift** : idem que push et pop mais côté gauche.

Lecture de l'entrée (entrée standard ou fichier donné à l'appel du script) ligne par ligne :

```
while (<>) {
      chomp; #supprime le retour chariot à la fin de chaque ligne
      print ‘‘I saw $_ in input. \n’’;
}
```

1 Excercice 1

Ecrivez un script Perl qui renverse l'ordre des éléments d'un tableau.
Indices :on pourra utiliser les fonctions **push**, **pop**, **shift**, **unshift**.

2 Excercice 2

Ecrivez un script Perl qui lit une séquence de nucléotides et pour chaque base **A**, **C**, **G** et **T** affiche combien il y en a dans la séquence donnée.

3 Excercice 3

Ecrivez un script Perl qui réalise les choses suivantes :

1. Définissez un tableau de séquences d'oligonucléotides :

   ```
   @seq = ("tcgtgccca", "tgtt", "cccga", "ttcatcag",
   "ggcaag", "ctg", "ggtgtaccggtgatcac",
   "ccaccta", "cctgaattat" );
   # OU (pour eviter les apostrophes)
   # @seq = qw(tcgtgccca tgtt cccga ttcatcag ggcaag ctg \
   # ggtgtaccggtgatcac ccaccta cctgaattat );
   ```

2. Repartissez les éléments du tableau @seq en 3 tableaux : @small_seq contiendra les oligos ayant entre 0 et 5 bases, @medium_seq ceux ayant entre 6 et 10 bases, et @large_seq ceux ayant 11 bases et plus.
3. Affichez chacun des tableaux résultants.
4. Trouvez le plus long oligo dans @seq et affichez-le.
   Indices : l'opérateur **length** retourne la longueur d'une chaîne, l'opérateur **push** permet d'ajouter un élément à un tableau, et l'opérateur **join** est très utile pour afficher simplement (en une seule instruction) le contenu complet d'un tableau.

4 Exercice 4

Supposons maintenant que votre script Perl, au lieu de définir un tableau de séquences d'oligonucléotides, les lise dans un fichier. Le format du fichier est simple : un oligo par ligne. Modifiez votre script pour qu'il soit capable de lire un nombre quelconque de séquence d'oligo dans un fichier et de faire la même analyse que dans l'exercice 3.

5 Exercice 5

Ecrivez un script Perl qui reconnait les numéros de téléphone de la forme 05-40-00-66-69 ou +33-5-40-00-66-69. Votre programme doit être capable de lire un fichier texte quelconque et afficher tous les numéros de téléphone qu'il contient. Le fichier peut bien entendu contenir zéro, un ou plusieurs numéros de téléphone par ligne.

Indice : découpez chaque ligne avec l'opérateur **split** pour la stocker dans un tableau, puis traitez chacun des éléments du tableau.