

## TP de programmation

### Premiers contacts avec PERL

Tapez le programme suivant, et sauvegardez-le sous le nom `bienvenue.pl`

```
#!/usr/bin/perl
print "Bonjour ", $ARGV[0], "\n";
print "Tu viens d'exécuter ton premier programme perl.\n";
```

Pour l'exécuter, il suffit de taper `perl bienvenue.pl` suivi de l'argument du programme. Ici, comme argument, mettez votre prénom. Quelle est la signification de chacune des instructions ?

### Croissance microbienne

On étudie la croissance d'une population microbienne idéale, qui se reproduit sans contrainte d'environnement. Chaque microbe suit la règle suivante :

- à la première seconde, le microbe est très jeune,
- à la deuxième seconde, le microbe est jeune,
- à partir de la troisième seconde, le microbe est adulte,
- dès qu'un microbe est adulte, il produit un nouveau microbe par seconde.

Pour cet exercice, on suppose qu'à la seconde 1 la population est constituée d'un unique microbe, très jeune.

**Question 1.** Représentez la croissance de la population lors des 6 premières secondes. A chaque seconde, quelle est la taille de la population ?

On peut exprimer le nombre de microbes à la seconde  $i$  grâce à la suite de Fibonacci :

$$\begin{cases} \text{fibonacci}(1) = 1 \\ \text{fibonacci}(2) = 1 \\ \text{fibonacci}(i) = \text{fibonacci}(i-1) + \text{fibonacci}(i-2) \end{cases}$$

**Question 2.** Ecrivez un programme `perl` qui pour un entier  $n$  permet de calculer les valeurs de la suite de Fibonacci de 1 à  $n$ .

Indication: Pour cela, il faut utiliser un tableau `$fibo` comprenant  $n$  cases, la case d'indice  $i$  correspondant à `fibonacci(i)`. Les cases d'indice 1 et 2 sont donc égales à 1. Pour les indices plus grands, il faut utiliser une boucle, qui fait varier  $i$  de 3 à  $n$  (qui correspond à `$ARGV[0]`):

```
for ($i=3;$i<=$ARGV[0];$i=$i+1){ ... }
```

La valeur de `$fibo[$i]` est alors obtenue en faisant la somme de `$fibo[$i-1]` et `$fibo[$i-2]`. Pour afficher les éléments de `$fibo`, il faut avoir recours à une seconde boucle, et utiliser l'instruction `print`.

**Question 3.** Utilisez votre programme pour trouver au bout de combien de secondes la population microbienne compte 1 million d'individus.

**Question 4.** Quelle est la taille de la population après une minute ?

On appelle ce type de croissance une **croissance exponentielle**. On peut montrer que

$$\lim_{i \rightarrow +\infty} \frac{\text{fibonacci}(i)}{\text{fibonacci}(i-1)} = \phi,$$

où  $\phi = \frac{1+\sqrt{5}}{2}$  est connu comme le nombre d'or.  $\phi$  vaut approximativement 1,6. Cela signifie que d'une seconde à l'autre, la taille de la population est multipliée par environ 1,6. Autrement dit, à la seconde  $i$ , il y a environ  $1,6^i$  microbes, ce qui explique l'appellation *croissance exponentielle*.

## Traduction virtuelle

Lors du premier TP de bioinformatique, vous aviez utilisé le programme ORFfinder, qui faisait la traduction d'une séquence d'ARN suivant les 6 cadres de lecture possibles. Le but de cet exercice est d'écrire votre propre programme de traduction.

Par rapport à l'exercice précédent, le programme ne prend plus en entrée un entier, mais une *chaîne de caractères*, qui contient le fragment d'ARN à traduire.

**Question 1.** Allez chercher le fichier `traduction.pl`. Ce fichier propose un squelette pour le programme. A quoi sert la table `%code` ?

**Question 2.** Complétez le programme de manière à ce que celui-ci affiche la traduction d'une séquence d'ARN entrée en argument. On suppose que la séquence est bien codante et dans la bonne phase de lecture. Ce dont vous avez besoin :

`length($ARGV[0])` est égal à la longueur de la chaîne `$ARGV[0]`.

`substr` (abréviation de *substring*, sous-chaîne) permet d'extraire une sous-chaîne dans une chaîne complète. Par exemple, `substr($ARGV[0], $i, 3)` contient les trois lettres de `$ARGV[0]` à partir de la position `$i`.

`$code{...}` permet d'obtenir l'acide aminé correspondant au triplet mentionné entre les accolades.

**Question 3.** Modifiez votre programme de manière à n'afficher la traduction que pour les cadres ouverts de lecture. Il faut donc tenir compte des codons START (AUG) et STOP (UAA, UAG et UGA).

Comment faire cela ? On utilise une variable `$ORF`, qui permet de repérer si l'on se trouve dans un cadre ouvert de lecture. Au départ, `$ORF` a pour valeur 0. Quand on rencontre un codon START, `$ORF` passe à 1. Il vaut de nouveau 0 dès que l'on rencontre un codon STOP. Intuitivement, 0 correspond à faux (on n'est pas dans une ORF), et 1 à vrai (on est dans une ORF).

Le test se fait avec une conditionnelle :

```
if (substr($ARGV[0], $i, 3) eq "AUG") {$ORF=1;}
```

permet de tester si le codon lu est un codon START. Si oui, on met `$ORF` à 1.

```
if ($ORF==1) {print $code{substr($ARGV[0], $i, 3)}} {...};
```

permet de tester si `$ORF` est à 1. Si oui, on affiche l'acide aminé codé par le triplet de nucléotides.