**Petit mémo Perl pour débuter**

Première ligne du programme doit toujours indiquer au système d'exploitation le chemin de l'exécutable à lancer pour interpréter le programme

#!/usr/bin/perl -w

(# pour indiquer à l'interpreteur Perl que c'est un commentaire, puis le après ! le chemin absolu vers l'exécutable perl qui doit être adapté à votre installation  (pour le savoir faire which perl ou type perl) , -w indique à l'interpréteur d'afficher les warning (comme par exemple une variable déclarée jamais utilisée), cela n'empêche pas le programme d'être exécuté mais indique où le code peut être nettoyé.

Les instructions en perl se terminent par ;

**Déclaration des variables**

En Perl il n'est pas obligatoire de déclarer les variables, par défaut l'usage d'une variable la crée. Cependant il est fortement conseillé de le faire  pour faciliter la relecture du programme mais aussi pour éviter des erreurs. Le nom de la variable doit être précédé de my.
Liste ou tableau : @nom
Tableau associatif : %nom
Si la variable est déclarée en début de programme →variable globale
On peut cependant déclarer une variable dans une boucle ou un test →variable locale (n'existe plus à la sortie de la boucle ou du test)

Exemples :

use strict; (impose que toutes les variables soient déclarées)

Déclaration plus initialisation :

my $seq\_name = 'BS09819'; #variable chaîne de caractères initialisée à BS09819
my $version = 0; #variable de type nombre. Perl ne fait pas de différence entre le type nombre et le type chaîne de caractère tous deux considérés comme un scalaire

my @liste =(); #initialisation à vide d'une liste ou d'un tableau

my %tab =(); #initialisation à vide d'un tableau associatif

Déclaration sans initialisation :

my ($strand, $start, @colonne, $file, $outfile1, $outfile2)

Un tableau spécial : le tableau @ARGV

Ce tableau est préinstallé par l'interpréteur Perl comme liste d'arguments de la commande ligne.
Exemples :
instructions testant si le nombre d'arguments tapés lors du lancement du programme est bon. Si ce nombre n'est pas bon, on envoie un message d'erreur et on quitte le programme, sinon on récupère les informations dans les variables correspondantes (ici trois nom de fichiers, un d'entrée et deux de sortie)

if (@ARGV < 3 || @ARGV > 3) {
    print "\n Usage : parse\_gm <infile\_name> <outfile\_name\_CDS> <outfile\_name\_start>";
    exit;
}else{
    $file = $ARGV[0];
    $outfile1 = $ARGV[1];
    $outfile2 = $ARGV[2];
}

**Quelques instructions.**

Ouverture d'un fichier en lecture avec message d'erreur en cas d'échec :
open(FILE,"$file") or die("Error, cannot open: $file$!");
Ouverture d'un fichier en écriture :
open(OUTFILE,">$outfile1");

fermeture d'un fichier
close(FILE);

Les structures de contrôle :

Un bloc d'instruction est un ensemble d'instructions compris entre deux accolades :

{
    instruction 1;
    instruction 2;
    ...
    instruction i;
    ...
    dernière instruction;
}

**Les instructions de test :**if ( condition 1) {
        instruction 1;
        instruction 2;
        ...
        instruction i;
} elsif (condition 2) {
        instruction 1;
        instruction 2;
        ...
        instruction i;

}else {
        instruction 1;
        instruction 2;
}

Exemple :

if ( $end != $save ) {
       $nb\_orf++ ; # équivallent à nb\_orf = nb\_orf +1
       $version = 1;
} else {
       $version++;
}

**Les boucles :**

Elles permettent de répéter plusieurs fois les mêmes instructions et s'arrête quand la condition d'arrêt est vraie

**La boucle pour (ou for) :**
La boucle pour comprend trois expressions : initialisation de la variable de la boucle, condition pour continuer la boucle, incrémentation de la valeur de la boucle.
Syntaxe :

for (initialisation; condition; incrémentation) {
            instruction 1;
            instruction 2;
            ....
            dernière instruction;
}

Exemple : Affiche les entiers de 0 et 20.

for (my $i = 0; $i <= 20; $i++) {
    print "$i\n";
}

**La boucle tant que (while) :**while( condition ) {
        instructions;
}Les instructions sont exécutées tant que la condition est vraie. La partie initialisation doit être effectuée avant la boucle. La variable doit être modifiée dans le bloc d'instruction de la boucle.

Exemple : comme pour la boucle for, affiche les entiers de 0 à 20. La différence réside dans le fait que la variable $i existera en dehors de la boucle. Donc à la sortie dans l'exemple, elle aura la valeur 20 et pourra être utilisée dans la suite du programme, dans d'autres instructions.

my $i = 0;
while( $i <= 20 ) {
    print "$i\n";
    $i++;
}

Exemple : parcours d'un fichier et sortie de la boucle à la fin du fichier :

while(<FILE>) {
    instructions;
}

**La boucle jusqu'à (until) :**Les instructions sont exécutées tant que la condition est fausse. Même syntaxe que la boucle while .
until ( condition ) {
        instructions;
}

**Si on veut passer une fois dans le bloc d'instruction avant de tester la condition d'arrêt :**

do {
    instructions;
} while (condition);

**La boucle foreach :**
Cette instruction permet de parcourir une liste

Exemple : A chaque tour de boucle, la variable $genome va prendre une valeur de la liste @liste\_genome qui est parcourue dans l'ordre.

foreach $genome (@liste\_genome) {
    print "nom du génome $genome\n";
    $nb\_genome++;
}

**Quelques opérateurs**

Les opérateurs booléens :

and : &&
or : ||
not : !

Les opérateurs logiques :

|  |  |  |
| --- | --- | --- |
| Comparaison | Nombre | Chaîne |
| Egal à | == | eq |
| Différent de  | != | ne |
| Inférieur à | < | lt |
| Supérieur à | > | gt |
| Inférieur ou égal | <= | le |
| Supérieur ou égal | >= | ge |

quelques opérateurs sur les chaînes de caractères :

soit $x la chaîne de caractères :

length($x) renvoie la longueur de la chaîne
chop($x) supprime le dernier caractère de la chaîne. $x est modifiée.
chomp($x) supprime le dernier caractère de $x si c'est une fin de ligne
substr ($x, start, length) extrait la sous-chaîne de $x commençant à la valeur start et de longeur length (les positions commencent à 0)
        my $v = "bienvenue sur le site";
        $ext = substr($v,4,5) #$ext vaudra "venue";

index($chaîne,$sousChaîne,$position) renvoie la position de la première occurrencede $sousChaîne dans $chaîne.
split(expression régulière, $chaîne) permet de découper une chaîne de caractères en fonction d'un séparateur. Elle renvoie la liste de l'ensemble des éléments de la chaîne délimités par le séparateur.
        my($temp,$pos) = split(/:/, $x); si $x = ">BSseq:[245,270]"  $temp recevra >BSseq et $pos recevra  [245,270]
        my(@colonnes) = split(/\s+/, $x);

Concaténation

$pos\_atg = $start . "\t" . ($start + 2); le . permet de concaténer.

Chaîne délimitée par des simples quotes et doubles quotes

- délimitée par des simples quotes :  le contenu de la chaîne n'est pas interprété.
Exemple:
print 'start at $pos \n'; imprimera à l'écran start at $pos \n

- délimitée par des doubles quotes :  le contenu de la chaîne est  interprété. "\n" sera interprété comme une fin de ligne et "\t" comme une tabulation. Une variable sera remplacée par son contenu.
Exemple :
print "start at $pos \n"; imprimera à l'écran start at 256 avec retour à la ligne (si $pos vaut 256) .

opérateur de recherche d'une expression régulière : l'opérateur =~
Exemple :

$string = "bonjour\n";
$string =~ /^bon/; est vrai
if ($string =~ /^bon/) {
    instructions
}

Substitution d'éléments de la chaîne :

s/ancienne/nouvelle/ on ajoute g à la fin si on veut remplacer toutes les parties trouvées plutôt que de la première uniquement

$x = "bonjour bonjour à vous";
$x =~ s/bonjour/hello/;  $x vaut maintenant "hello bonjour à vous"
$x = "bonjour bonjour à vous";
$x =~ s/bonjour/hello/g; $x vaut maintenant "hello hello à vous"

        $pos =~ s/\n//g; remplace le caractère de fin de ligne par rien
        $pos =~ s/\[//g; remplace [ par rien

quelques opérateurs sur les listes et tableaux :

push() et pop()

ajoute ou retire une valeur dans la liste à partir de l'extrémité droite

push(@liste, $newval); identique à @liste = (@liste,$newval)
$odlval = pop(@liste); retire le dernier élément de la liste et le met dans $oldval

shift() et unshift()

Même action que push et pop mais agissent sur l'extrémité gauche.

unshift (@liste, $newval); identique à @liste = ($newval, @liste)
@liste =(2,3,4,5)
$oldval = shift(@liste) $oldval vaudra 2 et @liste vaudra (3,4,5)

sort(@liste) trie les éléments de la liste dans l'ordre ASCII croissant.
@liste =(1, 2, 4, 8, 16, 32, 64)
@sort\_liste = sort[@liste) @sort\_liste vaut (1, 16, 2, 32, 4, 64, 8) donc pas un tri numérique pour les nombres.