TP/TD : ADN Mining Perl, Python

Vous trouverez le nécessaire ici : http://www.math-info.univ-paris5.fr/~lomn/Cours/BC/

perl
http://www.shellunix.com/perl.html

Question 1 : Écrivez le programme PERL suivant – *« firstsearch.pl »* - à l'aide de *gedit*, *gvim, emacs* ou autre en essayant de comprendre ligne par ligne ce qu'il fait (en ajoutant des commentaires dans votre propre code à la suite du caractère spécial de commentaires #) puis exécutez-le : $*perl firstsearch.pl*. Ensuite à l'aide de la commande man, étudiez la documentation pour PERL. Éventuellement, surfez sur le WEB.

firstsearch.pl

```
#!/usr/bin/perl -w
# Look for nucleotide string in sequence data

my $target = "ACCCTG";
my $search_string= 'CCAAATTCTTCGGGACCCTGGGGGGGTTAAATTACCCTGACCCTGATG'.
    'CATGGTATGTACAGTAGACTAGGACAACCCTGGGGTAGA';

my @matches;

#Try to find a match in letters 1-6 of $search_string, then look at letters 2-7,
#and so on. Records the starting offset of each match.
foreach my $i (0..length $search_string) {
    if ($target eq substr ( $search_string, $i, length $target)) {
    push @matches, $i;
    }
}

#Make @matches into a comma-separated list of printing
print "My matches occured at the following offsets : @matches.\n";
print "done\n";
```

Question 2 : Avec cette trame, répondez à l'aide d'un code *perl* à ces deux questions en imaginant un test sur une donnée d'intérêt.

1. Recherche de signaux protéine STOP

	.U.	.C.	.A.	.G.
U.	UUU Phe	UCU Ser	UAU Tyr	UGU Cys
	UUC Phe	UCC Ser	UAC Tyr	UGC Cys
	UUA Leu	UCA Ser	UAA Stop	UGA Stop
	UUG Leu	UCG Ser	UAG Stop	UGG Trp
C.	CUU Leu	CCU Pro	CAU His	CGU Arg
	CUC Leu	CCC Pro	CAC His	CGC Arg
	CUA Leu	CCA Pro	CAA Gln	CGA Arg
	CUG Leu	CCG Pro	CAG Gln	CGG Arg
A.	AUU Ile	ACU Thr	AAU Asn	AGU Ser
	AUC Ile	ACC Thr	AAC Asn	AGC Ser
	AUA Ile	ACA Thr	AAA Lys	AGA Arg
	AUG Met	ACG Thr	AAG Lys	AGG Arg
G.	GUU Val	GCU Ala	GAU Asp	GGU Gly
	GUC Val	GCC Ala	GAC Asp	GGC Gly
	GUA Val	GCA Ala	GAA Glu	GGA Gly
	GUG Val	GCG Ala	GAG Glu	GGG Gly

Figure 6.2 : Code génétique standard. Le code est indiqué dans sa version ARN (avec des U et non des T). Il y a trois codons stop UAA («ocre»), UAG («ambre») et UGA («opale»). La traduction démarre en général sur un codon AUG qui permet l'incorporation de la méthionine N-terminale dans les protéines. Chez les bactéries, le codon GUG est aussi parfois utilisé comme codon de démarrage (~30% des gènes chez E. coli), et plus rarement UUG, mais dans ce cas là, ils codent aussi une méthionine et non pas une valine ou une leucine.

2. Calcul de taux de A,G,C,T sur plasmodium et streptomyces.
3.

Simuler de l'ADN de Jurassik Park

La fonction *rand* extrait un nombre aléatoire dans un intervalle donné par la structure passée en paramètre.

int(rand(10)) **:** un entier entre 0 et 10
rand() entre 0 et 1 : un réel entre 0.0 et 1.0
rand (4.5) entre 0 et 4 un réel entre 0 et 4
voir http://www.perlmeme.org/howtos/perlfunc/rand_function.html

```
1.  #!/usr/bin/perl
2.   use strict;
3.   use warnings;
4.   my $random_number = rand();
5. print $random_number . "\n";
```

→ 0.21563085335405

Question 1 : Quelle est l'utilité de ***srand()*** ?

```
1. #!/usr/bin/perl -w
2. # randomposition
3. # A subroutine to randomly select a position in a string.
4. # WARNING: make sure you call srand to seed the
5. # random number generator before you call this function.
6.
7. sub randomposition {
8. my($string) = @_;
9. # This expression returns a random number between 0 and length-1,
10.       # which is how the positions in a string are numbered in Perl.
11.
12.       return int(rand(length($string)));
13.       }
14.
15.
16.
17.       # Test the random position subroutine
18.       my $dna = 'AACCGTTAATGGGCATCGATGCTATGCGAGCT';
19.       srand(time|$$);
20.       for (my $i=0 ; $i < 20 ; ++$i ) {
21.       print randomposition($dna), " ";
22.       }
23.       print "\n";
24.       exit;
```

Question 2 : A partir de tous ces éléments, simuler une mutation SNP (Single-Nucleotide Polymorphism).

Question 3 : Réaliser ce code en Python.

Question 4 : A partir de tous ces éléments, simuler un ADN plausible pour le roman de Michael Crichton *Jurassic Park (p. 132, Section Modélisation de Séquences, du cours de F. Dardel,* « Fabriquer des séances de dinosaure »*)* sachant que
1. Simuler une probabilité p revient à tirer aléatoirement une valeur réelle x entre 0 et 1 et considérer que le test est réussi si x<p.
2. Pour les vertébrés les probabilités conditionnelles de dinucléotides sont données par :

$$P(A|C) = \frac{f_{CA}}{f_C} = \frac{f_{CA}}{f_{CA} + f_{CG} + f_{CC} + f_{CT}}$$

On peut alors calculer l'ensemble de ces probabilités conditionnelles à partir de la table précédente :

5' \ 3'	A	C	G	T
A	34 %	18 %	24 %	25 %
C	36 %	27 %	5 %	32 %
G	28 %	22 %	26 %	23 %
T	22 %	21 %	26 %	31 %

[6] Une recherche avec le programme BLAST sur les bases de données en ligne montre que la séquence de *Jurassic Park* n'est pas une séquence aléatoire que l'auteur aurait généré en tapant au hasard sur son clavier. Il s'agit d'un fragment de la séquence d'un plasmide bactérien appelé pTRC99a et utilisé en biotechnologie pour produire des protéines recombinantes.

Simuler une mutation SNP (Single-Nucleotide Polymorphism).
Faire un BLAST sur votre séquence.

ANNEXE: quelques exemples de code *perl* pour vos archives

Hash Table

```
1. #!/usr/bin/perl -w
2. # various statements
3. my @a = (1, ''4'',9) ;
4. my @names = (''T. Herman'', 18, ''N. Aeschylus'', ''H. Ulysses'', ''Standish'') ;
5. print ''second number : $a[1]\n third name : $name[2]\n'' ;
    #Hash table to be completed
6. my%three_to_one = (
          ALA => A, CYS =>C, ASP =>D, GLU =>E,
          PHE =>F, GLY=> G, HIS =>H, ILE =>I) ;
7. print ''The one-letter code for ARG is $three_to_one{ARG}\n'' ;
```

```
#!/usr/bin/perl
#check for non-DNA characters

my $string= ''CCACAcCaCCcCAaTTG*CCCA'' ;
if  ($string =~ m/([^ATCG])/i) {
   print ''Warning ! Found : $1 in the string'' ;
}

#check for CATs

my $string= ''CCACAtcCaCCcCAaTTG*CCCA'' ;
if  ($string =~ m/CAT/i) {
   print ''Meow.'' ;
}
```

Avec Fonction

```
1. sub greet {
2.        my ($name) = shift ;
3. print ''Hello, $name!\n'' ;
4. }
5.
6. greet(''world'') ; # print hello world
    # to be completed
7. $length = calculate_length($sequence) ;
```

Remarque : les algorithmes populaires d'alignement (BLAST, FASTA) sont des algorithmes heuristiques d'alignement. Ils ne fournissent pas des alignements optimaux comme la programmation dynamique (algorithme NW) au sens de la similarité mais au sens du rapport coût calculatoire / précision d'alignement. Ils utilisent une stratégie à base de *hash-table* et de programmation dynamique. Ils trouvent très vite grâce à un index comme celui calculé ici et stocké pour chaque séquence archivé des amers pour entamer la recherche plus précise de sous-séquences d'alignement par propagation autour de ces amers. Vous êtes donc armé pour tout reprogrammer :-) si vous voulez.