Les dictionnaires

Méthodes de programmation - Initiation à la bioinformatique

1 Exposé de la problématique

1.1 À la découverte de notre ADN

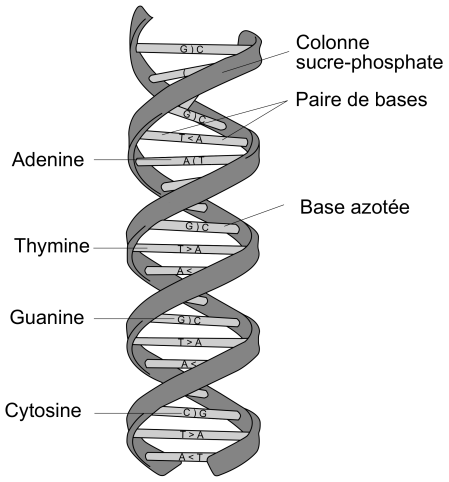
L'objectif de ce TP est de mettre en pratique le langage python pour construire et analyser un brin d'ADN. L'idée est, dans un premier temps, de reprendre les éléments de base du langage (condition, boucles ...) pour créer des fonctions qui construise un brin d'ADN, le lisent, et l'analysent. Je rappelle quelques éléments de base sur l'ADN, et son utilisation pour la synthèse de protéines.

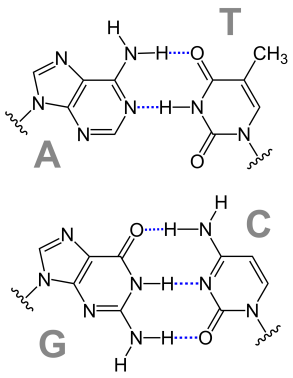
1.2 L'ADN

L'ADN, pour Acide DésoxyriboNucléique, est une macromolécule constituée de deux brins qui forme une double hélice maintenue par des liaisons hydrogène. Ces brins sont formés par un enchainement de maillons appelés, nucléotides qui contiennent les bases de l'ADN :

- A pour Adénine
- T pour Thymine
- G pour Guanine
- C pour Cytosine

Les bases de l'ADN fonctionnent par paire, une sur chaque brin : adénine avec thymine et guanine avec cytosine.





1.3 Traduction et transcription

La transcription est un mécanisme qui permet de "recopier" l'ADN dans le noyau de la cellule pour former un ARN (acide ribonucléique) qui sera utilisé dans la cellule notamment lors de la traduction. L'ARN présente la même structure que l'ADN mais lors de la transcription, la thymine (T) est remplacé par I'uracile (U). Ainsi, un brin d'ADN de type -ATGCACGTGC- est transmis sous forme d'ARN messager (ARNm, parfois noté abusivement ARN) dont la structure est -AUGCACGUGC-.

La traduction de I'ADN consiste à lire l'ARNm issue de la transcription pour synthétiser une protéine avec l'aide de la machinerie cellulaire. L'ARNm est découpé en codons qui sont constitués de 3 bases et correspondent à un acide aminé, c'est le code génétique. Les codons sont lus les uns à la suite des autres et la protéines est assemblée comme une chaîne (peptidique) d'acides aminés.

1.4 Correspondance codons - acides aminés

Le schéma ci-dessous vous donne la correspondance entre un codon, composé de trois bases de l'ARN et un acide aminé.

ARN messager Codon : première base azotée	ARN messager Codon : deuxième base azotée U	C	A	G	ARN messager Codon : troisième base azotée
U	Phe	Ser	Tyr	Cys	U
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	STOP	STOP	A
	Leu	Ser	STOP	Trp	G
C	Leu	Pro	His	Arg	U
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	U
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	U
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

Par exemple, GUA est un codon qui code pour l'acide aminé Val, c'est à dire la Valine. On remarquera que plusieurs codons peuvent coder pour le même acide aminé ce qui limite la portée des erreurs de copies ou des mutations. On note également la présence de codons STOP indiquant la fin de la partie "codante" et qui stoppe la synthèse de la protéine.

Pour prendre un exemple concret, un ARNm contenant la suite suivante -ACGCGUAGCUAA- permet de générer la protéine -Thr-Arg-Ser- (le codon UAA est un codon d'arrêt de croissance pour la protéine).

1.5 Les dictionnaires python

Pour entrer le code génétique dans python on pourra utiliser les dictionnaires. Ces objets python sont un peu comme des listes ils contiennent plusieurs autres objets. À la différence des listes, les éléments d'un dictionnaire sont repérés par une clef et non par un indice. On peut utiliser tout type de clef : des nombres, des chaînes de caractères ou même des tuples.

2 Implémentation informatique

2.1 Synthèse d'une protéine

Dans le fichier qui vous a été envoyé se trouve le code suivant :

```
Colonne_Gauche=["U","U","U","U","C","C","C","C","A","A","A","A","G","G","G","G"]
Horizontal=["U","C","A","G"]
Colonne_Droite=["U","C","A","G","U","C","A","G","U","C","A","G","U","C","A","G"]
Matrice=[  ["Phe","Ser","Tyr","Cys"] , ["Phe","Ser","Tyr","Cys"] , ["Leu","Set","STOP","STOP"] , ["Leu","Set","STOP","Trp"] ,
["Leu","Pro","His","Arg"] , ["Leu","Pro","His","Arg"] , ["Leu","Pro","Gln","Arg"] , ["Leu","Pro","Gln","Arg"] ,
["Ile","Thr","Asn","Ser"] , ["Ile","Thr","Asn","Ser"] , ["Ile","Thr","Lys","Arg"] , ["Met","Thr","Lys","Arg"] ,
["Val","Ala","Asp","Gly"] , ["Val","Ala","Asp","Gly"] , ["Val","Ala","Glu","Gly"] , ["Val","Ala","Glu","Gly"] ]
BASE_ADN = ["A", "T", "C", "G"]
STOP_ADN = ["TAA", "TAG", "TGA"]
BASE_ARN = ["A", "U", "C", "G"]
STOP_ARN = ["UAA", "UAG", "UGA"]
```

Activité 1 **reconnaissance des variables**

Donner le type des différentes variables globales utilisées, et dire à quoi elles correspondent par rapport à la problématique abordée.

Activité 2 **fabrication d'un dictionnaire**

En utilisant, entre autre variable, `Matrice`, fabriquer le dictionnaire `codage_ARNm_Proteine`, ayant pour clé les suites de 3 nucléotides (parmi A, U, C et G), et pour valeur l'acide aminé associé dans la future protéine.

Activité 3 **fabrication d'un dictionnaire**

Fabriquer le dictionnaire `codage_ADN_ARNm`, ayant pour clé un nucléotide de l'ADN, et pour valeur le nucléotide correspondant dans l'ARNm.

Vous allez maintenant écrire une suite de fonctions, dont l'emploi va permettre de générer des chaînes de nucléotides pouvant correspondre à un brin d'ADN ou d'ARNm.

Activité 4 **écriture d'une fonction**

nom : `ADN_3_nucleo`

arguments : aucun

effet : doit renvoyer une chaîne de caractères contenant une suite aléatoire de 3 nucléotides pouvant exister dans l'ADN. Cette suite ne doit pas correspondre à un STOP. Pour rappel, `randint(i,j)` renvoie un entier aléatoire entre i et j au sens large pour i et j.

Activité 5 **écriture d'une fonction**

nom : `ARN_3_nucleo`

arguments : aucun

effet : doit renvoyer une chaîne de caractères contenant une suite aléatoire de 3 nucléotides pouvant exister dans l'ARNm. Cette suite ne doit pas correspondre à un STOP. Pour rappel, `randint(i,j)` renvoie un entier aléatoire entre i et j au sens large pour i et j.

Activité 6 **écriture d'une fonction**

nom : `genese_brin_ADN`

arguments : n (int)

effet : doit renvoyer une chaîne de caractères contenant un enchaînement de nucléotides pouvant exister dans l'ADN, et qui devront générer une suite de n acides aminés dans la future protéine. Cette suite doit, de plus, se terminer par un STOP (aléatoire dans les 3 possibilités).

Activité 7 **écriture d'une fonction**

nom : `genese_brin_ARN`

arguments : n (int)

effet : doit renvoyer une chaîne de caractères contenant un enchaînement de nucléotides pouvant exister dans l'ARN, et qui devront générer une suite de n acides aminés dans la future protéine. Cette suite doit, de plus, se terminer par un STOP (aléatoire dans les 3 possibilités).

Si les deux fonctions précédentes ont été bien écrites, les chaînes de caractère générées doivent nécessairement correspondre à des brins d'ADN ou d'ARN. Cependant, on peut imaginer que l'opérateur fabrique lui-même ces suites « à la main », ou les lit à partir d'un fichier externe. Il est donc nécessaire, avant de générer la protéine, de vérifier que l'opérateur n'a pas fait d'erreur.

Activité 8 **écriture d'une fonction**

nom : `Test_ADN`

arguments : ch (str)

effet : doit renvoyer `True` si la chaîne de caractère est compatible avec un brin d'ADN, `False` sinon. Pour simplifier, on vérifiera que la longueur de la chaîne est bien un multiple de 3, qu'elle se termine bien par un STOP, et que chaque nucléotide est compatible avec `BASE_ADN`.

Activité 9 **écriture d'une fonction**

nom : `Test_ARN`

arguments : ch (str)

effet : doit renvoyer `True` si la chaîne de caractère est compatible avec un brin d'ARN, `False` sinon. Pour simplifier, on vérifiera que la longueur de la chaîne est bien un multiple de 3, qu'elle se termine bien par un STOP, et que chaque nucléotide est compatible avec `BASE_ARN`.

Vous allez écrire maintenant des fonctions permettant de transformer un brin d'ADN en un brin d'ARNm, puis permettant de transformer un brin d'ARNm en la protéine associée.

Activité 10 **écriture d'une fonction**

nom : `ADN_vers_ARNm`

arguments : brin (str)

effet : brin contient les nucléotides (A, T, C ou G) qui constituent l'ADN. La fonction doit renvoyer la chaîne de caractère contenant les nucléotides contenus dans l'ARNm généré à partir de l'ADN. Il est nécessaire de vérifier que la chaîne brin est compatible avec de l'ADN ; dans le cas contraire, la fonction doit renvoyer un signal d'erreur.

Activité 11 **écriture d'une fonction**

nom : `ARNm_vers_proteine`

arguments : brin (str)

effet : brin contient les nucléotides (A, U, C ou G) qui constituent l'ARNm. La fonction doit renvoyer la chaîne de caractère contenant les acides aminés contenus dans la protéine synthétisée à partir de l'ARNm. Il est nécessaire de vérifier que la chaîne brin est compatible avec de l'ARNm ; dans le cas contraire, la fonction doit renvoyer un signal d'erreur. La protéine devra apparaître, par exemple, sous la forme suivante (cas de 4 acides aminés enchaînés) : Pro-Leu-Arg-Ala

Vous allez maintenant utiliser les fonctions utilisées, pour fabriquer une chaîne d'ADN au hasard, afficher l'ARNm associée, et la protéine synthétisée.

Activité 12 **execution d'un script**

1. Vérifiez que les fonctions `Test_ADN` et `Test_ARN` sont efficaces.
2. Exécutez le script ci-dessous, qui permet de générer une protéine aléatoire contenant 20 acides aminés.

```
N=20 # nombre d'acides aminés voulus dans la protéine
ADN=genese_brin_ADN(N) ; print(ADN)
ARN=ADN_vers_ARNm(ADN) ; print(ARN)
Proteine=ARNm_vers_proteine(ARN) ; print(Proteine)
```

2.2 Analyse de la protéine

Cette partie va vous permettre d'utiliser le dictionnaire `AcideAmine`, déjà présent dans le fichier en votre possession. Ce dictionnaire a la particularité d'être sous la forme de deux dictionnaires imbriqués.

```
AcideAmine = {
    "Ala": {
        "nom": "Alanine",
        "masse": 89.09404,
        "polaire": False
    },

     ...

     "Val": {
        "nom": "Valine",
        "masse": 117.14784,
        "polaire": False
    }
}
```

Ce dictionnaire contient en clés les abréviations des différent acides aminés que l'on peut rencontrer dans la protéine (par exemple `Ala`).

La valeur associée à `Ala` est elle-même un dictionnaire secondaire. Il y a plusieurs clés dans ce dictionnaire secondaire : `nom` (valeur : le nom explicite de l'acide aminé (str)), `masse` (valeur : masse molaire en g.mol^{-1} (float)), `polaire` (valeur : True ou False, selon que l'acide aminé est polaire ou pas (Bool)).

Dans un premier temps, vous allez générer un dictionnaire qui va permettre de comptabiliser les acides aminés présents dans une protéine. On suppose que l'on a effectué la transformation suivante sur la chaîne correspondant à la protéine :

```
>>> A="Pro-Leu-Arg-Ala"
>>> B=A.split("-")
>>> B
['Pro', 'Leu', 'Arg', 'Ala']
```

Activité 13 **écriture d'une fonction**

nom : `genese_dico_AA_proteine`

arguments : L (list)

effet : L est une liste de chaînes de caractères, contenant la structure de la protéine sous forme de liste (cas de 4 acides aminés enchaînés) : ['Pro', 'Leu', 'Arg', 'Ala']. La fonction doit retourner un dictionnaire, ayant pour clé le symbole abrégé de l'acide aminé de l'ADN, et pour valeur le nombre de fois où l'acide aminé est rencontré dans la protéine.

L'application de la fonction précédente à l'exemple choisi devrait vous retourner ce résultat :

```
>>> C=genese_dico_AA_proteine(B)
>>> C
{'Ala': '1', 'Arg': '1', 'Leu': '1', 'Pro': '1'}
```

Écrire un script permettant d'extraire quelques renseignements sur la protéine, et ce grâce au dictionnaire généré.

Activité 14 **execution d'un script**

1. Extraire du dictionnaire la liste `L_AA` contenant la liste des acides aminés, ainsi que la liste `N_AA` contenant le nombre de ces acides aminés.
2. Tracer un graphe permettant de visualiser le nombre d'acides aminés en fonction de son abréviation.
3. Calculer la masse molaire de la protéine.
4. Extraire la liste des acides aminés polaires présents.