Série 8: Checkpoint 2

Sujet : Checkpoint 2

Consignes : Cette série est une série notée. Elle est à rendre sur Moodle (un rendu par groupe). Les exercices doivent être rendus sous la forme d'une **archive zip contenant tous les exercices**. Chaque exercice doit être nommé `exo-N.py` avec N numéro de l'exercice.
Les notions abordées sont :

1. Variables et types
2. Boucles (`while` et `for`)
3. Utiliser les collections (listes, tuples, dictionnaires)
4. Parcourir les éléments dans des collections
5. Chaines de caractères

Dans cette série, on utilisera pas la fonction `input`. L'utilisation des fonctions n'est pas obligatoire, mais peut considérablement simplifier votre code. Vous êtes libres de les utiliser ou non.

Exercice 1 : Séquence d'ADN

L'ADN est souvent représenté sous forme simplifiée comme une séquences de lettre. Ces lettres symbolisent les nucléotides, c'est à dire les 4 composants de base de l'ADN.

— $A \rightarrow$ Adénine
— $T \rightarrow$ Thymine
— $C \rightarrow$ Cytosine
— $G \rightarrow$ Guanine

Dans le fichier `adn.txt` vous trouverez une séquence d'ADN. Utilisez cette séquence pour les prochains exercices.

Astuce : Utilisez directement la séquence dans le code, on n'utilisera pas `input` dans cette série.

Le séquenceur que nous avons utilisé n'est pas tout neuf, et il fait parfois des petites erreurs. On vous demande donc de vérifier que la séquence donnée est bien de l'ADN (composée de A, T, C, et G uniquement). Si vous trouvez une erreur, donnez l'index de l'erreur dans la chaine et corrigez la lettre en la remplaçant par un T. Puis, affichez à l'écran la séquence d'ADN ainsi corrigée. Ouvrez un bloc notes et sauvegardez le résultat dans un deuxième fichier adn-2.txt.

Contrainte : Vous devez utiliser un set `nucleotides` qui contient les 4 nucléotides valides.

Astuce : Attention ! Les chaines de caractères sont immutables, on ne peut donc pas modifier une lettre au milieu d'une chaine existante ! Vous devez contourner le problème en créant par exemple une nouvelle chaine de caractères `adn2` que vous concaténez au fur et à mesure, ou en convertissant la séquence d'ADN en liste le temps de la modifier.

```
1
2  # Pour cet exemple, nous avons fourni la séquence suivante :
3  # ATGAGCTTCGATGWAAAACCTGT2ACGCCATTCATGAA
4
5  Erreur a l'index 13: W n'est pas un nucléotide valide. Remplacement par T.
6  Erreur a l'index 23: 2 n'est pas un nucléotide valide. Remplacement par T.
7  Nouvelle séquence:
8  ATGAGCTTCGATGTAAAACCTGTTACGGCCATTCATGAA
```

Code Listing 1 – exo-1.py

Exercice 2 : Deuxième brin d'ADN

L'exercice 1 vous montre comment on représente l'ADN avec une simple chaine de caractères. Cette chaine représente la composition d'un des brins d'ADN. Cependant, vous savez sûrement que l'ADN se compose de deux brins, agencés dans une forme hélicoidale.

Pour obtenir le brin d'ADN complémentaire, il faut suivre la règle simple suivante. Chaque A devient un T (et inversement), et chaque C devient un G (et inversement).

Par exemple, la séquence $ATGAC$ devient $TACTG$ (chaque lettre est remplacée par son complémentaire).

Écrivez un programme qui donne le brin d'ADN complémentaire à celui que vous avez obtenu dans l'exercice 1 (que vous avez normalement sauvegardé dans `adn2.txt`).

```
1
2  # Pour cet exemple, nous avons fourni la séquence suivante :
3  # ATGAGCTTCGATGTAAAACCTGTTACGGCCATTCATGAA
4
5  Séquence complémentaire :
6  TACTCGAAGCTACATTTTGGACAATGCCGGTAAGTACTT
```

Code Listing 2 – exo-2.py

Exercice 3 : ARN messager

Contrairement à l'ADN, l'ARN messager n'est composé que d'un seul brin, et il ne contient pas de thymine (T). À la place, on trouve de l'uracile (U).

À cette différence près, obtenir une séquence d'ARN messager revient à peu près à faire le brin d'ADN complémentaire.

Copiez-collez le code de `exo-2.py` dans un nouveau fichier `exo-3.py`, puis adaptez le de manière à ce que la séquence affichée contienne des U à la place des T.

Puis, ouvrez un bloc note et sauvegardez la séquence d'ARN dans un fichier `arn.txt`.

```
1
2  # Pour cet exemple, nous avons fourni la séquence suivante :
3  # ATGAGCTTCGATGTAAAACCTGTTACGCCATTCATGAA
4
5  Séquence d'ARN messager :
6  UACUCGAAGCUACAUUUUGGACAAUGCCGGUAAGUACUU
```

Code Listing 3 – exo-3.py

Exercice 4 : Recherche de STOP

Dans les cellules de l'organisme, l'ARN messager est ensuite lu et interprété par les ribosomes. En fonction des séquences lues, les ribosomes peuvent ensuite en déduire quelles séquences d'acides aminés générer pour former des protéines.

En pratique, les ribosomes lisent l'ARN messager 3 nucléotides par 3 nucléotides. On appelle un groupe de 3 nucléotides un codon. Certains codons signifient "STOP", c'est à dire qu'il s'agit de la fin de la séquence à lire. Ce qui suit un codon "STOP" est ignoré.

Les codons signifiant "STOP" sont les suivantes :
- UAA
- UAG
- UGA

Dans la séquence obtenue dans l'exercice 3, cherchez si il y a un codon "STOP". Affichez ensuite à quel index vous trouvez ce codon, et affichez-le. Puis affichez la nouvelle séquence complète d'ARN, c'est à dire le début de la séquence jusqu'au premièr codon "STOP" (compris).

Puis, ouvrez un bloc notes et enregistrez la nouvelle séquence dans un fichier `arn2.txt`.

Contrainte : Vous devez utiliser un tuple `stop` qui contient les 3 codons signifiant "STOP".

Astuce : Pour la première fois, je vous autorise à utiliser le mot clé `break` dans cet exercice. Pour rappel, un `break` sert à interrompre une boucle **prématurément**, lorsqu'on a pas besoin de terminer l'execution de la dite boucle. Mais attention : un grand pouvoir implique de grandes responsabilités...

```
1
2  # Pour cet exemple, nous avons fourni la séquence suivante :
3  # UACUCGAAGCUACUUUUGGACAUGCGGUAAGUACUU
4
5  Codon STOP trouvée àl'index 30 (UAA)
6  Nouvelle séquence d'ARN :
7  UACUCGAAGCUACAUUUUGGACAAUGCCGGUAA
```

Code Listing 4 – exo-4.py

Exercice 5 : Acides aminés

En fonction de la séquence de codons dans l'ARN messager, les ribosomes produisent des chaines d'acides aminés, ou autrement dit, des protéines. Nous souhaitons maintenant traduire l'ARN messager en séquence d'acides aminés. Pour cela, il existe des règles de traductions qui associent à chaque codon un acide aminé. Plusieurs codons peuvent donner le même acide aminé.

Pour vous éviter ce fastidieux travail, nous vous fournissons un dictionnaire (`acides-amines.py`) permettant de traduire les codons en acides aminés (leurs noms simplifiés). Procédez à la traduction de l'ARN messager obtenu dans l'exercice 4 (`arn2.txt`).
Pour chaque acide aminé traduit, ajoutez le dans une liste. À la fin du programme, affichez le contenu de la liste.
Astuce : Vérifiez bien que la chaine d'acides aminés affichée ne contient qu'un seul "STOP" en dernière position. Sinon, il y a une erreur dans votre exercice 4.

```
1
2  # Pour cet exemple, nous avons fourni la séquence suivante :
3  # UACUCGAAGCUACUUUUGGACAUGCGGUAA
4
5  Séquence d'acides aminés :
6  ['Tyr', 'Ser', 'Lys', 'Leu', 'His', 'Phe', 'Gly', 'Gln', 'Cys', 'Arg', 'STOP']
```

Code Listing 5 – exo-5.py

Exercice 6 : Assemblage final

Vous avez programmé différentes étapes permettant de passer d'une séquence d'ADN à la chaine d'acides aminés générée par le ribosome.

Maintenant, assemblez le contenu des exercices 1, 3, 4 et 5 pour faire un seul et même programme. Le programme commencera par la déclaration de la séquence d'ADN, et affichera chaque étape de la transformation, jusqu'à la chaine d'acides aminés. Adaptez les noms de variables si nécessaire.

Contrainte : Mettez des commentaires pour bien montrer la séparation entre les différentes étapes.

```
1
2  # Pour cet exemple, nous avons fourni la séquence suivante :
3  # ATGAGCTTCGATGWAAAACCTGT2ACGCCATTCATGAA
4
5  Erreur a l'index 13: W n'est pas un nucléotide valide. Remplacement par T.
6  Erreur a l'index 23: 2 n'est pas un nucléotide valide. Remplacement par T.
7  Nouvelle séquence:
8  ATGAGCTTCGATGTAAAACCTGTTACGGCCATTCATGAA
9  Séquence d'ARN messager :
10 UACUCGAAGCUACAUUUUGGACAAUGCCGGUAAGUACUU
11 Codon STOP trouvée àl'index 30 (UAA)
12 Nouvelle séquence d'ARN :
13 UACUCGAAGCUACAUUUUGGACAAUGCCGGUAA
14 Séquence d'acides aminés :
15 ['Tyr', 'Ser', 'Lys', 'Leu', 'His', 'Phe', 'Gly', 'Gln', 'Cys', 'Arg', 'STOP']
```

Code Listing 6 – exo-6.py

Exercice 7 : Mutations (BONUS)

Attention : Ce bonus est **difficile**. Alors même si vous arrivez seulement à détecter si il y a une mutation ou pas, vous aurez la majorité du bonus (sans trouver le nucléotide problématique). Ne passez pas trop de temps dessus si vous bloquez, ou posez nous des questions.

Dans les fichiers `adn-mute-1.txt` et `adn-mute-2.txt`, vous trouverez deux séquences d'ADN proches de `adn.txt`, mais légèrement différentes.

Lorsque la séquence d'ADN est modifiée, la modification peut ne pas changer l'acide aminé crée (puisque plusieurs codons peuvent donner le même acide aminé). La protéine finale est donc la même. Mais si la modification entraine un changement d'acide aminé, la protéine générée est différente.

Écrivez un programme qui vérifie si les séquences modifiées produisent la même protéine que `adn.txt`. Si c'est bien la même protéine, tout va bien. Sinon, c'est qu'il y a eu une mutation.

En cas de mutation, vous afficherez le ou les nucléotides responsables de ce changement.

Astuce : Pour cet exercice, il est recommandé (voire indispensable) de créer une ou plusieurs fonctions. La fonction principale prend en paramètre la séquence d'ADN, et renvoie la liste des acides aminés. Si vous le souhaitez, vous pouvez également séparer les différentes étapes de traduction en plusieurs fonctions (une par exercice).

```
1
2  Comparaison de adn et adn_mute_1 :
3  Pas de mutation détectée.
4
5  Comparaison de adn et adn_mute_2 :
6  Mutation détectée !
7  Au rang 19, Trp est devenu Cys.
8  Le nucléotide de rang 59 est passé de C àA.
```

Code Listing 7 – exo-7.py

Si vous arrivez ici, bravo !

Questions ? remi.dulong@unine.ch, pasquale.derosa@unine.ch, victor.villin@unine.ch