DS N°1 - MODULARITÉ ET MISE AU POINT DES PROGRAMMES

Un module `adn`

Les **molécules d'ADN** présentes dans les cellules vivantes contiennent l'**information génétique** et permettent le développement, le fonctionnement et la reproduction des êtres vivants. L'**ADN** est formé de **deux brins** qui sont une succession de **nucléotides** portant chacun une **base azotée**.

Les quatre **bases azotées** sont l'*adénine (A)*, la *thymine (T)*, la *cytosine (C)* et la *guanine (G)*.
Elles s'associent deux par deux : *A* avec *T* d'une part, *C* avec *G* d'autre part.

Exercice 1

Le but de l'exercice est de définir un **module** `adn` contenant la **constante** et les **fonctions** suivantes :

a. La constante `BASES_ASSOCIEES = {'A':'T', 'T':'A', 'C':'G', 'G':'C'}`.
b. La fonction `sequence_alea` qui prend en paramètre d'entrée un entier `n` positif et renvoie une **séquence d'ADN aléatoire** de longueur `n` sous la forme de **chaîne de caractères**.
c. La fonction `complementaire` qui prend en paramètre d'entrée une **chaîne de caractères** `brin` et renvoie le **brin complémentaire** sous forme de **chaîne de caractères**.
d. La fonction `proportionAT` qui prend en paramètre d'entrée une **chaîne** `brin` et renvoie la **proportion** de **bases** *A/T* présentes dans le **brin**. Le valeur renvoyée est donc un **flottant**.

Implémentez ce module `adn`, en **documentant chaque fonction** à l'aide d'une **docstring** contenant *ce que fait la fonction*, la *liste de ses paramètres d'entrée et leur type* ainsi que le *type de la valeur de retour*.

Aide exercice 1

- Pour la fonction `sequence_alea`, vous pourrez utiliser la fonction `choice` du module `random`, qui renvoie une **valeur aléatoire dans une liste**.
- Pour la fonction `proportionAT`, on calcule la **proportion** en divisant le **nombre de bases** *A* et *T* par la **longueur du brin** d'ADN.

Exercice 2

On souhaite maintenant utiliser ce module `adn` (écrit dans un fichier `adn.py`).

Écrivez un **nouveau script** *Python* (incluant les **imports** nécessaires du module `adn`) qui réalise le programme suivant :

- demande à l'utilisateur de saisir un nombre `n`,
- génère `n` séquences d'ADN aléatoires,
- **affiche** enfin la **plus grande proportion** de bases *A/T* parmi toutes les **séquences** générées, ainsi que la **séquence** associée.
  Exemple d'affichage : `"séquence : ATCTCGCATA - proportion : 0.6"`.

Pour rappel, la fonction `input` permet de demander à l'utilisateur de saisir une valeur, et de la récupérer sous la forme d'une **chaîne de caractères** (il sera donc peut-être nécessaire de convertir la valeur en `int` en utilisant la fonction `int()`).
Il n'est pas nécessaire d'écrire une *docstring* ici.

Mise au point et tests

Exercice 3

Voici un programme Python produisant des **erreurs** :

```
1  réponse = int(input("Est-ce qu'il pleut? "))
2
3  if réponse == oui:
4
5     print("parapluie"[9])
6
7  elif réponse == "non":
8
9  print("pas de parapluie")
10
11 else réponse == "autre":
12
13     print("Veuillez répondre par "oui" ou "non")
```

Indiquez les **types d'erreur** que provoque ce code, ainsi que les **lignes concernées**, parmi les types d'erreur suivants : `SyntaxError`, `IndexError`, `NameError`, `IndentationError` et `TypeError`. Finalement, proposez une **correction de ce programme**.

Exercice 4

On considère la fonction suivante :

```
1  def mystere1(t):
2      x = 0
3      y = 0
4      for z in t:
5          x += z
6          y += 1
7      return x/y
```

a. Que fait cette fonction ?
   La **réécrire** en donnant des **noms expressifs** aux **variables** utilisées, et en écrivant sa **docstring**.
b. Écrire un **test** qui provoque une **erreur d'exécution**.
c. Enfin, **corriger** la **fonction** pour qu'elle **passe ce test**.

Exercice 5

On souhaite définir la fonction `partage(t)` qui **partage** la **liste** `t` en **deux listes** contenant respectivement les éléments de `t` de **rang pair** et de **rang impair**, et **renvoie** ces **deux tableaux** sous la forme d'un **tuple**.

a. À partir de cette spécification, écrire un **jeu de tests « boîte noire »** de cette **fonction**.
b. Un programmeur a écrit l'implémentation suivante de cette fonction :

```
1  def partage(t):
2      assert isinstance(t, list)
3      pairs = []
4      impairs = []
5      i = 0
6      while i != len(t):
7          pairs.append(t[i])
8          impairs.append(t[i+1])
9          i += 2
10     return pairs, impairs
```

Est-ce qu'elle passe **tous les tests** écrits précédemment ?
Si oui, écrire un **test « boîte blanche »** qui échoue.
c. **Corriger la fonction** pour qu'elle passe tous les tests.

Exercice 6 *(bonus)*

Peut-on être sûr que la boucle suivante n'est pas une boucle infinie ?

```
1  while n != 1:
2      if n % 2 == 0:
3          n = n // 2
4      else:
5          n = (3*n + 1) // 2
```

Sinon, comment peut-on **modifier le code** pour **interrompre l'exécution** si la **boucle** dépasse un **nombre d'exécutions** donné ?

Réécrivez ce code en limitant à `1000` le **nombre d'itérations** de la boucle.