BOUCLES

On a vu que dans le dernier exercice d'application du TP4 (obtention d'une chaîne d'ADN complémentaire) que l'on répétait plusieurs fois la même comparaison.
Un certain type d''instructions va nous permettre de dépasser ce problème : les boucles.

1. Boucles for

(a) Principe

Les deux programmes suivants nous donnent les mêmes résultats

```
L=list(input('saisir la châine ADN :'))

if L[0]=='A':
     L[0]='T'
elif L[0]=='T':
     L[0]='A'
elif L[0]=='G':
     L[0]='C'
elif L[0]=='C':
     L[0]='G'

if L[1]=='A':
     L[1]='T'
elif L[1]=='T':
     L[1]='A'
elif L[1]=='G':
     L[1]='C'
elif L[1]=='C':
     L[1]='G'

if L[2]=='A':
     L[2]='T'
elif L[2]=='T':
     L[2]='A'
elif L[2]=='G':
     L[2]='C'
elif L[2]=='C':
     L[2]='G'

if L[3]=='A':
     L[3]='T'
elif L[3]=='T':
     L[3]='A'
elif L[3]=='G':
     L[3]='C'
elif L[3]=='C':
     L[3]='G'

print('la chaîne complémentaire est',L)
```

```
L=list(input('saisir la châine ADN :'))

for i in range(0,4):
    if L[i]=='A':
        L[i]='T'
    elif L[i]=='T':
        L[i]='A'
    elif L[i]=='G':
        L[i]='C'
    elif L[i]=='C':
        L[i]='G'

print('la chaîne complémentaire est',L)
```

Commentaires

La variable i est appelée **variable d'itération**, elle prend successivement les différentes valeurs de la liste générée par la fonction **range()** (ici [0,1,2,3]) à chaque itération de la boucle. Cette boucle est créée par Python la première fois que la ligne contenant le for est exécutée.

Remarque : une fois la boucle terminée, cette variable d'itération i ne sera pas détruite et contiendra ainsi la dernière valeur de la liste [0,1,2,3].

Attention au caractère deux-points « : » à la fin de la ligne débutant par for.
De la même manière que pour l'instruction **if** Cela signifie que la boucle **for** attend un bloc d'instructions, en l'occurrence toutes les instructions que Python répétera à chaque itération de la boucle.
On appelle ce bloc d'instructions le corps de la boucle.
Il faudra donc bien respecter l'indentation.

(b) Exemples

On donne les programme suivants :

programme 1

```
animaux = ['girafe', 'tigre', 'singe', 'souris
for animal in animaux:
    print(animal)
```

programme 2

```
animaux = ['girafe', 'tigre', 'singe', 'souris']
for animal in animaux[1:3]:
    print(animal)
```

i) Quel est leur rôle ?

ii) Dans chacun des deux cas, quel est le type de la variable d'itération ?

A retenir

Ici, **animaux** est une liste sur laquelle on itère, et animal est une chaîne de caractères car chaque élément de la liste est une chaîne de caractères. La variable d'itération peut être donc être de n'importe quel type selon la liste parcourue.
En Python, une boucle itère toujours sur un objet dit séquentiel (c'est-à-dire un objet constitué d'autres objets) tel qu'une liste.

(c) Nommage de la variable d'itération

i. Si on itère sur les indices :
pour éviter les confusions, on choisira le nom i pour la variable d'itération.

ii. Si, on itère une liste comportant des chaînes de caractères, on mettra un nom explicite pour la variable.
Par exemple :
- for animal in [‘tigre’, ‘singe’, ‘rat’]]
- for prenom in ['Joe', 'Bill', 'John']:

(d) Exemples applications

i. Une erreur à ne pas commettre

```
>>> for animal in animaux:
... print(animal)
  File "<stdin>", line 2
    print(animal)
    ^
IndentationError: expected an indented block
```

Expliquer le message d'erreur et corrigez le code.

ii. Minimum (bis)

On considère une liste L de flottants.

Créer un script retournant le minimum de la liste L.

iii. Triangle

Créez un script qui dessine un triangle comme celui-ci :

```
*
**
***
****
*****
******
*******
********
*********
**********
```