TP n° 6 : Structures de données

✔ On prendra bien soin de tester toutes les fonctions !

1 Applications directes du cours

Question 1 (*Produit des éléments d'une liste*) Écrire une fonction calculant le produit des éléments d'une liste d'entiers.

Question 2 (*Somme des éléments vérifiant un prédicat*) Écrire une fonction calculant la somme des éléments divisibles par 5 d'une liste d'entiers positifs.

Question 3 (*Nombre d'éléments vérifiant un prédicat*) Écrire une fonction calculant le nombre de caractères en majuscule d'une chaîne de caractères. On pourra utiliser la méthode `isupper`.

Question 4 (*Calculs de sommes*) Écrire des fonctions calculant explicitement :

$$(S_n)^2 = \sum_{k=0}^{n} k^3 \qquad H_n = \sum_{k=1}^{n} \frac{1}{k} \qquad T_n = \sum_{k=1}^{n-1} \frac{1}{k(k+1)}$$

2 Méthodes à connaître

Méthode 1

Lorsque l'on souhaite parcourir une collection indexable, on se pose la question : « *est-il utile d'accéder aux indices ?* »

- Oui : on itère *par indices* : `for i in range(len(iterable))`
- Non : on itère *par éléments* : `for element in iterable`

☞ **Remarque 1**

On peut toujours itérer par indices, même lorsque l'on a besoin uniquement des éléments, mais c'est quand même moins élégant.

Question 5 Modifier la fonction suivante pour utiliser un parcours par élements au lieu d'un parcours par indices :

Python
```
def maximum(liste):
    """Maximum d'une liste non vide d'entiers positifs."""
    maxi = 0
    for i in range(len(liste)):
        if liste[i] > maxi:
            maxi = liste[i]
    return maxi
```

Que fait cette fonction ? Pourquoi faut-il supposer que les entiers sont positifs ? Adapter la fonction dans le cas où l'on ne suppose plus que les entiers sont positifs (en supposant toujours la liste non vide).

Méthode 2

Si l'on a besoin d'accéder à la fois aux indices et aux éléments, on utilise une itération par indices [a].

a. Les plus avancés pourront utiliser `enumerate`.

Question 6 Écrire une fonction qui parcourt une liste et qui affiche chaque élément en indiquant sa position dans la liste. On utilisera la méthode `format` des chaînes de caractères. Par exemple l'appel `affiche_elements([4, 2, 2, 4])` affiche :

```
L'élément d'indice 0 est 4
L'élément d'indice 1 est 2
L'élément d'indice 2 est 2
L'élément d'indice 3 est 4
```

Méthode 3

Lorsque l'on souhaite utiliser simultanément deux éléments successifs d'une liste, le parcours par indices est indiqué.

Question 7 Écrire une fonction `est_triee(liste)` qui vérifie si une liste d'entiers est triée par ordre croissant. Attention à ne pas accéder à des éléments de la liste qui n'existeraient pas !

Méthode 4

Pour compter le nombre d'éléments dans une collection qui vérifient un prédicat donné :

1. on créé un compteur qui joue le rôle d'*accumulateur* (le plus souvent on l'initialise à 0). On choisit un nom de variable adapté à ce que l'on compte ;
2. on parcourt la collection par éléments ou par indices ;
3. pour chaque élément on incrémente le compteur s'il le faut ;
4. on renvoie la valeur du compteur.

On peut avoir besoin de plus d'un compteur ou de renvoyer plutôt le résultat d'un calcul faisant intervenir le compteur.

Question 8 Écrire une fonction qui vérifie s'il y a plus d'occurrences de la lettre `"a"` que de la lettre `"e"` dans une chaîne de caractères, c'est-à-dire qui renvoie `True` si c'est le cas et `False` sinon.

Méthode 5

Pour calculer une somme d'éléments dans une collection qui vérifient un prédicat donné, on procède comme dans la méthode précédente mais l'accumulateur joue le rôle de somme partielle.

Question 9 (♥) Écrire une fonction qui calcule la moyenne d'une liste de nombres.

Méthode 6

Pour créer une liste dont on connaît une expression du terme général :

1. on initialise une liste vide qui joue le rôle d'*accumulateur* ;
2. on itère sur un `range` ;
3. on ajoute le terme général à la liste avec la méthode `append` ;
4. on renvoie la liste ainsi construite.

Cf. question 4

Méthode 7

Pour créer une ou plusieurs listes à partir d'une autre liste, on fait comme pour la méthode précédente, mais en itérant sur la liste, éventuellement avec plusieurs accumulateurs.

Question 10 Écrire une fonction `separe_pairs_impairs(liste)` qui renvoie un couple de listes `(pairs, impairs)` contenant respectivement, dans le même ordre relatif, les éléments pairs et impairs d'une listes d'entiers.

Question 11 Écrire une fonction `separe_petits_grands(liste, seuil)` qui renvoie un couple de listes `(petits, grands)` contenant, dans le même ordre relatif, les éléments plus petits, respectivement strictement plus grands, que la valeur `seuil` dans une liste de nombres donnée en argument.

Méthode 8

Pour manipuler des chaînes de caractères, on use et on abuse des méthodes `format`, `split` et `join`.

Question 12 Que fait le programme suivant ?

Python

```python
def mystere(x, y, z):
    """Docstring et noms de variables à modifier."""
    return z.join(x.split(y))
```

3 Des exercices pour s'entraîner

Exercice 1 (*Factorielle*) Écrire une fonction `factorielle(n)` qui calcule $n!$. *Indication : réutiliser une fonction précédemment écrite.*

Exercice 2 (*Quelques listes à produire*) Créer les listes (il s'agit de remplir les `...`, et de préférence pas à la main...) :

1. `L1` constituée de 12 zéros
2. `L2 = [42, 41, 40, 39, ..., 26, 25, 24]`. On remarquera que `range` accepte un pas négatif en troisième argument
3. `L3 = [0, 0, 0, 2, 2, 2, 4, 4, 4, ..., 10, 10, 10]`
4. `L4 = [1, 2, 2, 3, 3, 3, ..., 7, 7, 7, 7, 7, 7, 7, ..., 10, 10]`
5. `L5 = [1, 2, 1, 3, 2, 1, 4, 3, 2, 1, ..., 9, 8, 7, 6, 5, 4, 3, 2, 1]`

Exercice 3 Écrire une fonction `table_multiplication(m, n)` qui prend en arguments deux entiers n et m et qui imprime à l'écran la table de multiplication de m pour les n premiers entiers. Par exemple, `table_multiplication(9, 6)` affiche :

```
0 fois 9 = 0
1 fois 9 = 9
2 fois 9 = 18
3 fois 9 = 27
4 fois 9 = 36
5 fois 9 = 45
```

Modifier la fonction pour renvoyer en sortie une chaîne de caractères contenant cette table *au lieu* de l'afficher. On rappelle que le caractère spécial `'\n'` correspond au caractère indiquant un passage à la ligne dans une chaîne de caractère. L'appel `table_multiplication(15, 11)` n'affichera donc rien, en revanche l'appel `print(table_multiplication(15, 11))` doit afficher exactement la même chose que ci-dessus.

Exercice 4

1. Écrire une fonction `somme_chiffres(n)` qui renvoie la somme des chiffres de l'écriture décimale de l'entier n. Par exemple `somme_chiffre(139)` doit renvoyer 13 car $1+3+9=13$. *Indication : passer par les chaînes de caractères.*
2. Résoudre les problèmes du Project Euler :
   - http://projecteuler.net/problem=16
   - http://projecteuler.net/problem=20

Exercice 5 Écrire des fonctions qui permettent d'afficher des triangles à `n` lignes de la forme :

```
*          *          *            *            *
**         --         **           ***          * *
***        ***        * *          *****        *   *
****       ----       *  *         *******      *     *
*****      *****      *   *        *********    *       *
******     ------     ******       **********   **********
```

Exercice 6 (*Suite de* Conway) Les premiers termes de la suite de Conway sont : 1, 11, 21, 1211, 111221. Chaque terme étant obtenu en lisant à haute voix le terme précédent, c'est pourquoi Conway avait baptisé cette suite *Look and say*. Par exemple, le terme 1211 se lit « un 1, un 2, deux 1 » donc le terme suivant est 111221.

1. Écrire une fonction `look_and_say(n)` qui retourne le terme qui suit l'entier n dans cette énumération. On représentera les entiers sous forme de chaînes de caractères de leurs chiffres.
2. Afficher les 20 premiers termes de la suite de Conway.

Si on note u_n le nombre de chiffres du n^e terme de cette suite alors on peut montrer que le rapport $\frac{u_{n+1}}{u_n}$ tend vers une constante, appelée constante de Conway.

3. Calculer une valeur approchée de la constante de Conway.

Une propriété remarquable de cette constante est qu'elle ne dépend pas de la valeur initiale (à l'exception de 22 qui produit une suite constante).

4. Le vérifier expérimentalement en testant différentes valeurs initiales.
5. Montrer que dans la suite de Conway ne peuvent apparaître que les chiffres 1, 2 et 3.

https://xkcd.com/859/