Chapitre 2. Listes et boucles `for`

2.1 Listes et boucles `for`

python permet de manipuler les listes d'éléments, par exemple `[2,8,5]` est une liste d'entiers. La fonction `len` (abréviation de *length*) retourne la *longueur* d'une liste (on dit aussi sa *taille*), par exemple `len([2,8,5])` vaut 3.

La boucle `for` permet de *parcourir* les éléments d'une liste (on dit que la liste est *itérable*) ; en voici la syntaxe :

```
for variable in liste :
    corps de la boucle
```

Remarquez bien les deux points à la fin de la ligne et le fait que les instructions appartenant au corps de la boucle doivent être indentées (décalées) par rapport au mot clef `for`. L'évaluation par python d'une boucle `for` correspond à l'algorithme suivant :

1. calculer la valeur de la liste et la mémoriser ;
2. sortir de la boucle si tous les éléments de la liste ont été traités ;
3. affecter la valeur du premier élément non traité à la variable de boucle ;
4. exécuter le corps de la boucle ;
5. recommencer le traitement à partir de l'étape 2.

Exercice 2.1.1 Dans l'environnement Python Tutor, entrer l'instruction suivante et analyser ce qui est affiché :

```
for elt in [2, 8, 5]:
    print(elt, elt * elt)
```

Même question pour :

```
u = [2, 8, 5]
for elt in u:
    print(elt, elt * elt)
```

La variable `elt` est-elle définie après exécution de ces instructions ? Si oui, quelle est sa valeur ?

Exercice 2.1.2 On considère la définition suivante :

```
def sommeListe(L):
    s = 0
    for elt in L:
        s = s + elt
    return s
```

Que retournent les appels `sommeListe([1,3,13])`, `sommeListe([1])` et `sommeListe([])` ? Pour vous aider, remplissez le tableau ci-dessous.

	étape 1	*étape 2*	*étape 3*	*étape 4*	*étape 5*	*étape 6*	*étape 7*	*étape 8*	*étape 9*
`L`	[1,3,13]								
`elt`	-								
`s`	0								

L'instruction `return` termine l'exécution de la fonction qui la contient. Que calcule la fonction `somme` si par malheur on indente trop la dernière ligne, comme ci-dessous ?

```
def sommeListe(L):
    s = 0
    for elt in L:
        s = s + elt
        return s    # gros bug !
```

Exercice 2.1.3 Écrire les fonctions suivantes prenant en paramètre une liste de nombres `L`, et **testez** ces fonctions :

— une fonction `moyenneListe(L)` qui calcule et retourne la moyenne de ces nombres,

— une fonction `nbPairsListe(L)` qui compte et retourne combien de ces nombres sont pairs.

— une fonction `maximumListe(L)` qui calcule et retourne le maximum de ces nombres (supposés positifs).

2.2 Utilisation de range dans des boucles for

La fonction `range(debut, fin, pas)` permet de définir une suite arithmétique finie d'entiers de raison `pas` commençant par l'entier `debut` et bornée par `fin` (qui ne fait pas partie de la suite). Les syntaxes possibles de `range` sont :

```
range(debut, fin, pas)
range(debut, fin)       # pas vaut 1 par défaut
range(fin)              # debut vaut 0 par défaut
```

L'argument `pas` est donc optionnel, et vaut 1 par défaut. L'argument `debut` est également optionnel, et vaut 0 par défaut. La suite se termine **juste avant** d'atteindre `fin`.

Exercice 2.2.1 TP Entrer les instructions suivantes et analyser les réponses de python :

```
for j in range(10):
    print(j, j * j)
```

```
for k in range(3, 8):
    print(k, 2 * k + 1)
```

```
for k in range(3, 9, 2):
    print(k)
```

```
for i in range(10):
    print("Bonjour")
```

Exercice 2.2.2 TP Dans l'environnement Python Tutor, tester chacun des groupes d'instructions suivantes et analyser les résultats comme précédemment :

```
for a in [10,11,12]:
    for b in [1,2,3]:
        print(a, b)
```

```
u = [2,6,1,10,6]
v = [2,5,6,4]
for x in u:
    for y in v:
        print (x, y, x + y, x == y)
```

Exercice 2.2.3 On considère la définition suivante :

```
def mystere(n):
    s = 0
    for i in range(1, n+1):
        s = s + i
    return s
```

Que retournent les appels `mystere(2)`, `mystere(3)`, et `mystere(n)` en général ? Connaissez-vous une formule qui permet de calculer le même résultat ?

Exercice 2.2.4 Écrire une fonction `sommeCarres(n)` qui calcule et retourne $\sum_{i=1}^{n} i^2$.

2.3 Exercices de révisions et compléments

Exercice 2.3.1 Multiplication à la main

Écrivez `produit(x,y)` qui calcule le produit de deux nombres x et y, mais sans utiliser l'opérateur `*` : à la place, utilisez une boucle qui réalise des additions successives.

Exercice 2.3.2 Écart-type

1. En vous inspirant de la fonction `sommeListe` de l'exercice 2.1.2, écrivez une fonction `sommeCarresListe(L)` calculant la somme des carrés des éléments de la liste `L`.
2. L'écart-type d'une liste de nombres L permet d'estimer dans quelle mesure les éléments de L s'éloignent de la moyenne des éléments de L. Par exemple l'écart-type de la liste `[8, 8, 8, 12, 12, 12]` est de 2 (puisque tous les éléments sont à distance 2 de la moyenne 10).

   On peut le calculer en utilisant la somme des carrés des éléments de L et la somme des éléments de L (en notant n le nombre d'éléments de L obtenu avec la fonction `len`, et L_i les éléments de la liste L) :

$$EcartType(L) = \sqrt{\frac{\sum_i (L_i{}^2)}{n} - \left(\frac{\sum_i L_i}{n}\right)^2}$$

   Pour calculer une racine carrée, il faut ajouter `from math import sqrt` au début du fichier pour avoir accès à la fonction `sqrt` . En appelant les fonctions `sommeListe` et `sommeCarresListe` écrites précédemment, écrivez une fonction `ecartTypeListe(L)` qui calcule l'écart-type de la liste `L`.

2.4 L'essentiel du chapitre

```
for variable in une_liste:
    instructions exécutées avec variable
    contenant successivement les valeurs
    de une_liste
```

Exemple :

```
s = 0
for a in range(1,10):
    s = s + a
```

La fonction `range` permet de générer des listes d'entiers utilisables par la primitive `for` :

```
list(range(10))                    [0,1,2,3,4,5,6,7,8,9]
list(range(3,7))                             [3,4,5,6]
list(range(1,20,4))                    [1,5,9,13,17]
```

Note : Lorsque l'on a une fonction qui prend en paramètre une liste `L`, on n'utilise pas `range` puisque l'on a déjà une liste pour `for` :

```
def f(L):
  for x in L:
    ...
```

Inversement, si la fonction prend en paramètre un entier `n`, on a besoin d'utiliser `range` pour fabriquer une liste pour `for` :

```
def f(n):
  for i in range(n):
    ...
```