TP2 - Fonctions, instructions conditionnelles

1. Fonctions

Dans un programme, on peut utiliser des fonctions comme en mathématiques : en appliquant une fonction `f` à une valeur `x`, on obtient un résultat que l'on note `f(x)`.

Ainsi, si `L` est une liste, et on écrit `len(L)`, la fonction `len` est **appelée**. Elle **reçoit en argument** («en entrée») la liste `L` et **renvoie comme résultat** («en sortie») la longueur de cette liste.
On peut aussi citer la fonction `abs` (valeur absolue), ou encore la fonction `round` (arrondi à l'entier le plus proche) : ainsi on peut écrire des expressions comme `abs(4-10)` ou `round(len(L)/3)`.

À partir de maintenant, nous définirons nous-mêmes de nouvelles fonctions. Par exemple, dans le précédent TP nous avons écrit un programme qui peut calculer la somme des éléments d'une liste `L`. Définissons une fonction, qui s'appellera `somme`, qui prend en argument une liste `L` et renvoie comme résultat la somme de ses éléments.

```
def somme(L):  # remarquer les deux points en fin de ligne comme pour une boucle
    s=0        # le reste de la définition est INDENT'E, comme pour une boucle
    n=len(L)
    for k in range(0,n):
        s=s+L[k]
    return s
```

Exécuter ces lignes. Python ne répond pas : il lit la définition et l'enregistre. Plus tard, si on écrit quelque chose comme `somme(  [3,8,1,4]  )`, il va :
- *temporairement* faire l'affectation `L=[3,8,1,4]`
- exécuter le code contenu dans la définition
- renvoyer comme résultat la valeur de `s` indiquée par l'instruction `return s`
- oublier toutes les variables qui ont été définies pendant cet appel : `k`, `s` et `L`.

Exercice 1. Définir les fonctions suivantes, inspirées de programmes du TP précédent :

(1) Une fonction `somme_entiers(n)` qui prend en argument un entier `n` et renvoie la somme des entiers de 1 à `n`.

(2) Une fonction `carres(n)` qui prend en argument un entier n et renvoie la liste des n premiers carrés parfaits.

(3) Une fonction `somme_carres(n)` qui prend en argument un entier `n` et renvoie la somme des carrés des n premiers entiers. **Attention** : cette fonction devra faire appel aux fonctions définies plus haut. Elle ne devra contenir aucune boucle dans sa définition.

(4) Une fonction `reverse(L)` qui prend en argument une liste `L` et renvoie une liste contenant les éléments de `L` dans l'ordre inverse.

2. Instructions conditionnelles

Imaginons que l'on veuille exécuter l'instruction `a=a+1` seulement si la variable `v` est égale à 7. On écrira :

```
if v == 7 :
    a=a+1
```

Il y a plusieurs choses à comprendre ci-dessus :

- le rôle des deux points et de l'indentation est toujours le même que pour les boucles et les définitions de fonctions.
- l'opérateur «égal» est utilisé pour les affectations, et le «double égal» sert pour tester l'égalité entre deux valeurs. Ainsi `v==7` est un test d'égalité, alors que `v=7` est une affectation, ce qui n'est pas du tout la même chose !
- dans la suite on utilisera aussi l'opérateur `!=` qui signifie «≠», et les opérateurs de comparaison `<`,`>`,`>=`,`<=`.
- le résultat de ces opérations est un objet de type **booléen**, un type qui n'a que deux valeurs `True` et `False`. Essayer dans le shell d'évaluer `2+3 != 5` et `2+3 <= 6` pour observer les booléens dans leur milieu naturel.

Exercice 2. Définir les deux fonctions ci-dessous, qui prendront en argument une liste de nombres.

```
def bim(G):
    a=0
    for i in range(0,len(G)):
        if G[i]==7:
            a=a+1
    return a
```

```
def bam(G):
    b=False
    for i in range(0,len(G)):
        if G[i]==7:
            b=True
    return b
```

(1) Prédire le résultat de l'appel `bim([4,7,2,0,7,7,5])` : .....
Si besoin, commencer par étudier le programme équivalent :

```
G=[4,7,2,0,7,7,5]
a=0
i=0
if G[i]==7:
    a=a+1
i=1
if G[i]==7:
    a=a+1
#(etc... quelle sera la valeur de *a* quand la boucle se terminera ?)
```

(2) De façon générale, si `L` est une liste, quel est le résultat de l'appel `bim(L)` ? .....
.....

(3) Prédire le résultat de l'appel `bam([4,7,2,0,7,7,5])` : .....

(4) De façon générale, si `L` est une liste, quel est le résultat de l'appel `bam(L)` ? .....
.....

On voit que dans les deux fonctions précédentes, la variable `a` de type nombre entier, ou `b` de type booléen, sert de «mémoire de travail» et renferme l'information utile concernant la partie de la liste déjà parcourue. Sa valeur est mise à jour au fur et à mesure de l'exécution.

Exercice 3. Utiliser une méthode similaire pour définir les fonctions suivantes :

(1) `dernier_7(L)` qui renvoie comme résultat la position du dernier 7 dans la liste `L`

(2) `tous_les_7(L)` qui renvoie comme résultat la liste des positions où il y a un 7 dans la liste `L`

(3) `que_des_7(L)` qui renvoie comme résultat un booléen : `True` si `L` ne contient que des 7, `False` sinon.

Exercice 4. Que fait la fonction `boum` ci-dessous si on lui donne la liste `[4,7,2,0,7,7,5]` ?

```
def boum(L):
  v=L[0]
  for i in range(1,len(L)):
     if L[i] < v :
         v=L[i]
  return v
```

Exercice 5. Définir une fonction `position_max(L)` qui renvoie la position (l'indice) où se trouve le plus grand élément dans `L`.

Remarque : En informatique, contrairement aux mathématiques, l'utilité d'une fonction peut être *autre* que de calculer un résultat. La fonction peut avoir un **effet**. Par exemple, si on a une liste `L`, et on écrit `L.append(8)`, l'intérêt de la fonction `L.append` n'est pas son résultat, mais l'effet qui se produit quand on appelle cette fonction. Cet effet est qu'un nouvel élément est ajouté à la fin de `L` (la valeur de cet élément est 8, la valeur reçue en argument par la fonction).

Une autre fonction souvent utilisée uniquement pour son effet est la fonction `print`. Elle a pour effet d'afficher dans le shell les valeurs qu'on lui donne en argument. Le résultat renvoyé par cette fonction, en revanche, n'a aucun intérêt.

Certaines fonctions cumulent les deux : par exemple, si on a une liste `L`, non vide, la fonction `L.pop()`, qui ne reçoit aucun argument, a un intérêt double : elle a l'effet de retirer le dernier élément de `L`, et elle renvoie comme résultat la valeur de cet élément.