TP Python 5 : fonctions

L.-C. LEFÈVRE

La notion de **fonction** existe dans tous les langages de programmation et a une certaine analogie avec les mathématiques. Il s'agit d'un morceau de programme que l'on écrit une fois (**déclarer**) et que l'on peut à tout moment réutiliser (**appeler**), éventuellement en changeant des paramètres. Les intérêts d'écrire des fonctions sont :

1. Pouvoir réutiliser le code, sans le copier / coller.
2. Diviser son programme en sections avec un nom et une utilité claire, un peu comme on divise un texte en paragraphes.
3. Se partager le travail. Si quelqu'un écrit une fonction, une autre personne peut l'utiliser sans avoir à regarder le code.

I Premiers essais

Les fonctions Python se déclarent avec le mot-clé `def`, suivi de leur nom et de leur(s) argument(s) entre parenthèses et d'un bloc d'instructions. Essayer par exemple :

```
def saluer(nom):
    print("Bonjour", nom, "et bienvenue")
```

Ce programme tout seul ne fait rien, c'est une déclaration ! Tester **sur la même page** :

```
saluer("Louis")
saluer("Clément")
saluer(...) # insérer votre prénom
```

On voit que le même morceau de programme est réutilisable avec un paramètre qu'on peut faire varier, sans recopier tout.
Une fonction peut aussi prendre **plusieurs** arguments ou paramètres, et utiliser tous les calculs mathématiques que nous connaissons :

```
def somme(x, y):
    print("la somme fait", x+y)
```

Tester :

```
somme(3, 4)
somme(12, 5)
```

Remarque importante. Dans la suite, après avoir déclaré votre fonction et lancé le programme, rien ne s'affichera… C'est à **vous** de la tester avec différents paramètres ! On peut pour cela utiliser le mode interactif : écrire directement les appels à la fonction dans la console en noir, valider avec la touche entrée.

Le bloc d'instruction, décalé à droite, peut contenir… tout ce que nous avons déjà vu jusque là.

Question 1. Écrire une fonction `divise(x,y)` qui teste d'abord si `y` est égal à zéro, auquel cas elle affiche qu'on ne peut pas diviser par zéro, sinon elle affiche le résultat de la division x/y.

```
def divise(x,y):
    if ... :
        print("Il ne faut JAMAIS diviser par zéro !")
    else:
        print(...)
```

Question 2. Écrire une fonction `minimum(x,y)` qui affiche le plus petit des deux nombres passés en argument :

```
def minimum(x,y):
    if ... :
        print(x)
    else:
        print(y)
```

II Valeur de retour

Souvent, une fonction sert à calculer **une** valeur à partir de ses arguments et on veut simplement récupérer cette valeur. Le mot-clé `return` **termine la fonction** et envoie la valeur calculée, que l'on peut récupérer dans n'importe quelle expression.
Essayer par exemple la fonction suivante :

```
def carre(x):
    return x * x
```

On peut la tester simplement en mode interactif : `carre(2)`, `carre(3)`, ... Mais aussi dans un programme :

```
y = carre(5)
print(y)
```

Remarque importante. Dans la suite, on évite que les fonctions utilisent des commandes `input` ou `print` mais on préfère leur passer des paramètres comme des arguments et la valeur qu'elle calcule est passée par `return`.

Question 3. Le programme suivant calcule calcule le produit $1 \times 2 \times \cdots \times 10$ et l'affiche :

```
x = 1
for i in range(1, 11):
    print(i, x) # ligne à enlever plus tard, pour vérifier
    x = x * i
print(x)
```

Remplacer ce programme par une fonction `factoriel(n)` qui prend un paramètre `n`, calcule le produit $1 \times 2 \times \cdots \times n$, et le retourne. Pour $n = 10$ cela doit bien sûr redonner le même résultat...

III Application aux vecteurs

Nous allons écrire plusieurs fonctions qui prennent en argument les coordonnées de points ou de vecteurs et renvoient elles-aussi des coordonnées de points ou vecteurs. Pour renvoyer un vecteur avec deux coordonnées il suffit de mettre les deux coordonnées entre des parenthèses.

Par exemple, la fonction suivante prend en argument les coordonnées de deux vecteurs, qu'on notera $\vec{u} : \begin{pmatrix} x_u \\ y_u \end{pmatrix}$ et $\vec{v} : \begin{pmatrix} x_v \\ y_v \end{pmatrix}$, et retourne les coordonnées du vecteur $\vec{u} + \vec{v} : \begin{pmatrix} x_u + x_v \\ y_u + y_v \end{pmatrix}$:

```
def somme(xu, yu, xv, yv):
    return (xu + xv, yu + yv)
```

Question 4. Écrire la fonction `produit(xu, yu, k)` qui prend en argument les coordonnées d'un vecteur $\vec{u} : \begin{pmatrix} x_u \\ y_u \end{pmatrix}$ et un nombre réel k et retourne les coordonnées du vecteur $k\vec{u}$.

Question 5. Écrire la fonction `vecteur(xA, yA, xB, yB)` qui prend en argument les coordonnées d'un point $A : \begin{pmatrix} x_A \\ y_A \end{pmatrix}$ et d'un point $B : \begin{pmatrix} x_B \\ y_B \end{pmatrix}$ et retourne les coordonnées du vecteur $\overrightarrow{AB}$.

Question 6. Écrire la fonction `colineaires(xu, yu, xv, yv)` qui prend en argument les coordonées d'un vecteur $\vec{u}$: $\begin{pmatrix} x_u \\ y_u \end{pmatrix}$ et d'un vecteur $\vec{v}$: $\begin{pmatrix} x_v \\ y_v \end{pmatrix}$, et retourne `True` si les vecteurs $\vec{u}$ et $\vec{v}$ sont colinéaires, et `False` sinon.

Question 7. Écrire la fonction `norme(xu, yu)` qui prend en argument les coordonnées d'un vecteur $\vec{u}$: $\begin{pmatrix} x_u \\ y_u \end{pmatrix}$ et retourne la norme de $\vec{u}$.
On aura besoin pour cela de la fonction racine carrée qui se nomme `sqrt` et pour l'utiliser il faut l'importer : la ligne doit être **avant** la déclaration de fonction

```
from math import sqrt
```

Question 8. Écrire la fonction `distance(xA, yA, xB, yB)` qui prend en argument les coordonnées d'un point A : $\begin{pmatrix} x_A \\ y_A \end{pmatrix}$ et d'un point B : $\begin{pmatrix} x_B \\ y_B \end{pmatrix}$ et retourne la distance AB.
Éventuellement, on peut se servir de des fonctions `vecteur` et `norme` déjà écrites...

```
def distance(xA, yA, xB, yB):
    (xu, yu) = vecteur(...)
    return norme(...)
```

Question 9. Écrire la fonction `aligne(xA, yA, xB, yB, xC, yC)` qui prend en argument les coordonnées de trois points A, B, C et retourne `True` si les points sont alignés et `False` sinon. On peut se servir des fonctions `vecteur` et `colineaires` déjà écrites.