TP n°4 : Fonctions

Instruction générale : hormis pour les exercices corrigés collectivement, vous ferez valider votre travail par l'enseignant.

1 Fonctions

Une fonction est une forme de sous-programme qui est utilisée au sein d'un programme qui soit la contient, soit l'importe depuis d'autres fichiers ou de bibliothèques de fonctions. Les fonctions sont aisément reconnaissables par la forme de leur appel qui implique des parenthèses : `fonction()` ; on peut reconnaître par exemple dans `print()`, `input()`, `range()` des fonctions. Elles permettent de factoriser le code en évitant les répétitions de séquences identiques et le rendent plus lisible, plus structuré.

Une fonction se définit par la commande `def nomFonction(`$\texttt{arg}_1$`,...,`$\texttt{arg}_n$`):` et comme pour les boucles, tout le contenu de celle-ci sera indenté. Un nom de fonction ne peut pas commencer par un chiffre : `1fonction()` n'est pas un nom de fonction valide mais `fonction1()` l'est.

Exemples :

1. La fonction ci-dessous une fois appelée par la commande `soleil()` affichera `1, 2, 3, soleil !`

```
def soleil():
    for i in range(1,4):
        print(i,end=", ")
    print("soleil !")

soleil()
```

2. La fonction ci-dessous appelée par la commande `compteur(10,13)` affichera `10, 11, 12, 13.`

```
def compteur(debut,fin):
    for i in range(debut,fin+1):
        print(i,end=", ")
    print()

compteur(10,13)
```

Exercice 1. [Bonne année !] Écrire une fonction `nouvelAn()` qui affiche le décompte de la nouvelle année : 10, 9,..., 1, bonne année !

2 Arguments et retours

Une fonction informatique peut prendre en entrée plusieurs arguments et donner en retour plusieurs sorties. Toutefois, contrairement à une fonction mathématique, elle n'est ni obligée de prendre un argument en entrée comme on peut le voir dans l'exemple 1 ci-dessus, ni obligée de renvoyer un argument en sortie comme on peut le voir dans les deux exemples ci-dessus (ne pas confondre avec l'affichage donné par `print()`). Les fonctions ne renvoyant pas d'arguments en retours sont appelées **procédures**. Les fonctions peuvent avoir des arguments en entrée et en sortie de plusieurs types : entiers, listes, etc, mais aussi faire appel à d'autres fonctions.

Pour qu'une fonction donne des arguments en retour, il faut utiliser la commande `return`.

Exemple : la fonction ci-dessous permet de définir la fonction mathématique $f(x) = x^2 - 2x + 1$ et son appel à l'intérieur du `print()` renverra `1`.

```
def polynome(x):
    return x**2-2*x+1

print(polynome(0))
```

Exercice 2.

1. Écrire une fonction permettant de calculer l'image d'un réel par le polynôme $3x^2 + \dfrac{x}{2} - 4$. Créer une liste contenant les images des entiers de 0 à 9 de cette fonction et l'afficher.
2. Écrire une fonction permettant de calculer l'image d'un réel par le polynôme $-x^2 + \dfrac{2x}{3} + \dfrac{1}{2}$. Créer une liste contenant les images des entiers de 0 à 9 de cette fonction et l'afficher.

Il est possible d'avoir plusieurs arguments en retours, dans ce cas ils sont séparés par des virgules. Ils peuvent être affectés à des variables dans le reste du programme comme on peut le voir dans l'exemple ci-dessous.

Exemple : le programme ci-dessous calcule l'aire et le volume d'un cube. Les variables `S` et `V` reçoivent respectivement les valeurs retournées par la fonction `cube` dans l'ordre donné par celle-ci : la surface sera affectée à `S` et le volume à `V`.

```
def cube(arete):
    surface=6*arete**2
    volume=arete**3
    return surface,volume

S,V=carre(1)
print(S,V)
```

Exercice 3. [Rectangle] Écrire une fonction prenant en entrée la longueur et la largeur d'un rectangle calculant son périmètre et son aire. Affecter le résultat de cette fonction pour une largeur égale à 2 et une longueur égale à 5 à deux variables `P` et `A` dont affichera le contenu par la suite.

Exercice 4. [Permutation] Python contient une commande permettant de permuter le contenu de deux variables : `a,b=b,a` ; si on avait `a=3` et `b=1.1`, après permutation, on aurait `a=1.1` et `b=3`. Écrire une fonction prenant en entrée deux variables et permutant leurs contenus sans utiliser la commande ci-dessus (le but est de comprendre le fonctionnement d'une permutation informatique, pas d'utiliser bêtement une commande).

Exercice 5. [Somme et produit]

1. Écrire une fonction `somme()` prenant en entrée deux entiers et sommant tous les entiers compris entre ces deux là. Par exemple si on fait `somme(30,40)`, la fonction doit nous renvoyer le résultat de la somme $30+31+32+\cdots+39+40$.
2. Écrire une fonction `produit()` prenant en entrée deux entiers et multipliant tous les entiers compris entre ces deux là. Par exemple si on fait `produit(30,40)`, la fonction doit nous renvoyer le résultat du produit $30\times 31\times 32\times\cdots\times 39\times 40$.

3 Valeur par défaut pour les arguments, étiquette

3.1 Valeur par défaut

Il possible d'affecter aux arguments d'une fonction une valeur par défaut qui permet un appel de la fonction sans avoir à donner une valeur à tous les arguments de celle-ci.

Exemple : Si on effectue l'appel `cube2()`, la fonction ci-dessous prendra la valeur par défaut de `arete`, i.e. 1 et affichera donc `Un cube d'arête 1 a pour surface 6 et volume 1`. Si on effectue l'appel `cube2(2)`, alors on aura en sortie `Un cube d'arête 2 a pour surface 24 et volume 8`.

On remarquera l'utilisation de la fonction `format()` qui permet d'insérer des valeurs dans une chaîne de caractères.

```
def cube2(arete=1):
    surface=6*arete**2
    volume=arete**3
    print("Un cube d'arête {} a une surface de {} et un volume de {}".format(arete,
    surface,volume))
```

Exercice 6. Que donne la fonction ci-dessous pour chacun des appels suivants ? Que constatez-vous ?

1. `fonction();` 2. `fonction(2,3);` 3. `fonction(3);` 4. `fonction(4).`

```
def fonction(a=1,b=2):
    print("a vaut {} et b vaut {}".format(a,b))
```

Exercice 7.

1. Écrire une fonction `rectangle2()` ayant pour valeurs par défaut 1 pour la longueur et la largeur et affichant le texte suivant lors de on appel (toujours avec pour valeurs par défaut 1) : `Un rectangle de largeur 1 et longueur 1 a pour surface 1`.
2. Écrire une fonction `somme2()` comme dans l'exercice 5 avec pour valeur par défaut 1 et 100 et affichant lors de son appel le texte : `La somme de entiers de 1 à 100 vaut 5050`.
3. Écrire une fonction `produit2()` comme dans l'exercice 5 avec pour valeur par défaut 1 et 10 et affichant lors de son appel le texte : `Le produit de entiers de 1 à 10 vaut 3628800`.

3.2 Étiquette

On peut constater dans l'exercice 6 que lorsqu'on donne une valeur en argument à une fonction qui possède plusieurs arguments par défaut, cette valeur est toujours attribuée au premier argument ; ce que l'on ne souhaite pas forcément. Par ailleurs, lors d'un appel d'une fonction, les arguments doivent être donnés dans l'ordre exact de la définition de la fonction. Toutefois, Python est très souple et il est possible de contourner ces problèmes en donnant à la fonction l'étiquette (ou plus simplement le nom) de l'argument en plus de sa valeur. Ceci permet d'appeler une fonction en donnant les noms des variables à l'intérieur de celle-ci, et ce dans n'importe quel ordre.

Exemple : On reprend la fonction de l'exercice 6.
- L'appel `fonction(b=3)` donne `a vaut 1 et b vaut 3`.
- L'appel `fonction(b=3,a=2)` donne `a vaut 2 et b vaut 3`.

Exercice 8. Donner des appels possibles de la fonction ci-dessous donnant les sorties suivantes.

1. `1 1 1 2`; 2. `1 0 0 1`; 3. `0 1 1 1`; 4. `2 0 1 1`.

```
def fonction(a=1,b=1,c=1,d=1):
    print(a,b,c,d)
```

4 Documentations et assertions

Les fonctions et le code pouvant s'avérer complexes, il est généralement nécessaire de mettre des commentaires expliquant les algorithmes, les variables, objectifs d'une séquence. Cela rend les programmes plus clairs et lisibles, à la fois pour soi et pour d'autres développeurs. On peut commenter une ligne grâce à la commande `#` ou sur plusieurs lignes avec les triples guillemets `""" commentaires..."""`. On prendra soin maintenant à bien commenter les fonctions et codes lorsque ceux-ci s'avèrent complexes.

Par ailleurs, il est possible de préciser le type des arguments attendus en entrée et donnés en sortie dans la définition de la fonction comme dans l'exemple ci-dessous. Cela permet de bien documenter sa fonction et donc de faciliter son utilisation par une autre personne.

Exemple :

```
def cube(arete : int or float) -> (int or float, int or float) :

    """ Calcul de la surface et du volume d'un cube. Tous les variables sont des
     entiers ou flottants positifs héritant du type de arete."""

    surface=6*arete**2
    volume=arete**3

    return surface,volume
```

Les commentaires sont utiles aux développeurs mais pas forcément aux utilisateurs (sauf s'ils vont lire le code) et n'empêchent pas les erreurs. Par exemple, dans l'exercice 5, l'utilisateur peut se tromper en entrant autre chose que des entiers, ce qui générera un résultat inattendu ou une erreur. La commande `assert` permet d'éviter cela en générant une erreur précise et prévue par le programmeur afin d'aider l'utilisateur.

Exemple : Dans le code ci-dessous, la commande assert permet d'effectuer le test `((type(arete) is int) or (type(arete) is float)) and (arete>=0)`. Si celui-ci retourne Vrai, alors la fonction s'exécute, sinon le texte après la virgule : `"Il faut entrer un nombre positif."` est affiché en message d'erreur.

```
def cube(arete : int or float) -> (int or float, int or float) :

    """ Calcul de la surface et du volume d'un cube. Tous les variables sont des
     entiers ou flottants positifs héritant du type de arete."""

    assert ((type(arete) is int) or (type(arete) is float)) and (arete>=0), "Il
    faut entrer un nombre positif."

    surface=6*arete**2
    volume=arete**3

    return surface,volume
```

Exercice 9. Reprendre les exercices 3 et 5 et ajouter aux fonctions qui y ont été définies des assertions testant bien que les arguments pris en entrée sont valides. Tester les assertions avec des arguments valides et invalides.

Exercice 10. [Distance] Écrire une fonction calculant la distance entre deux points en dimension deux dont on donnera les coordonnées sous forme de liste : par exemple, pour $(3\,;1)$, on entrera `[3,1]`.

Exercice 11. [Distance*] Écrire une fonction calculant la distance entre deux points en n'importe quelle dimension. On pensera à tester que les points ont bien la même dimension à l'aide d'une assertion.

5 Variables locales et globales

Les variables `surface` et `volume` dans l'exemple précédent sont ce qu'on appelle des variables locales. Locales dans le sens où elles sont propres à la fonction et n'ont pas d'existences en dehors de celle-ci. Python alloue à la fonction une mémoire propre où les variables sont gérées localement indépendamment du reste du programme. En opposition, on trouve les variables globales qui sont valables pour le programme en entier. Elles peuvent aussi être appelées à l'intérieur des fonctions mais dans ce cas il vaut les mettre comme argument.

Exercice 12. Recopier et exécuter les programmes ci-dessous. Que se passe-t-il ? La variable `a` est-elle globale ou locale ?

1.
```
def fonction():
    a=0
    print(a)

fonction()
```
2.
```
def fonction():
    a=0
    print(a)

print(a)
```
3.
```
def fonction(a):
    print(a)

a=0
fonction(a)
```
4.
```
a=0
def fonction():
    print(a)

fonction()
```

Une variable locale peut avoir le même nom qu'une variable globale mais les deux restent indépendantes comme on peut le voir dans l'exemple suivant.

Exemple : Le programme ci-dessous lors de son exécution affiche en sortie `2 5` car le premier `print()` est celui de la fonction donc y on prend la valeur de la variable locale `b` puis affiche `2 7` car cette fois-ci on prend la valeur de la variable globale `b`.

```
def test():
    b = 5
    print(a, b)

a = 2
b = 7
test()
print(a, b)
```

Exercice 13. Que donne les programmes ci-dessous en sorties lors de leurs exécutions ?

1.
```
def fonction():
    a=0
    print(a)

a=1
print(a)
fonction()
```

2.
```
def fonction():
    b=1
    print(b)

fonction()
b=0
print(b)
```

Il peut être parfois nécessaire qu'une fonction affecte une variable globale, ce qu'elle ne fait pas normalement. Pour cela il existe la commande `global` qui transforme une variable locale en globale.

Exemple : la fonction ci-dessous globalise `b`. Lors de l'appel de la fonction, `b` est globalisée avec sa valeur locale. Le programme affiche en sortie `2 5` puis à nouveau `2 5`.

```
def test():
    global b
    b = 5
    print(a, b)

a = 2
b = 7
test()
print(a, b)
```

Exercice 14. Que donne les programmes ci-dessous en sorties lors de leurs exécutions ?

1.
```
def fonction():
    global a
    a=0
    print(a)

a=1
print(a)
fonction()
```

2.
```
def fonction():
    global b
    b=1
    print(b)

b=0
fonction()
print(b)
```

6 Ressources supplémentaires

— Les cours sur Python d'OpenClassRooms
— Le cours du W3 Schools
— Vidéo sur les fonctions