TP N°9: Les fonctions (1/2)

Une *fonction* est un bloc d'instructions qui possède un *nom*, reçoit un certain nombre de *paramètres* et renvoie un *résultat*. L'usage des fonctions permet de décomposer les tâches complexes en tâches simples plus abordables, d'éviter la répétition et de rendre le code réutilisable à travers l'importation de fichiers de fonctions.

Déclaration et syntaxe

Une fonction Python a la syntaxe suivante :

```
def nom_fonction([param1, param2,..]):
    ''' documentation '''
        corps de fonction
        [return valeur]
```

def : sert à déclarer un objet de type `function`.

Les paramètres : on ne précise jamais le type des paramètres (notion de *typage dynamique*).

La documentation : est optionnelle mais fortement recommandée (entre triple côtes/guillemets).

Le corps de la fonction : doit être limité par les *indentations*.

return : permet d'arrêter l'exécution d'une fonction et de renvoyer un résultat.

```
def somme (a,b) :
    ''' calculer la somme de a et b '''
        s = a + b
        return s
```

Une fois déclarée, la fonction peut être appelée par son nom suivi par ses paramètres

```
r = somme (1,2)
```

Remarques :

- Si on n'a pas encore désigné de quoi est chargée notre fonction, on peut écrire l'instruction `pass` au niveau du corps de la fonction, ce qui permet d'avoir un bloc d'instructions vide.
- Quand la fonction ne programme pas d'instructions de type `return`, Python retourne tout de même la valeur `None` (un objet vide) ayant comme type `NoneType`. Le mot clé `return` peut figurer dans plusieurs endroits au sein d'une fonction.

Exercice 1

Écrire les fonctions Python suivantes :

- `fact(n)` : calcul et retourne le factoriel d'un entier n passé en paramètre.
- `somme(L)` : calcul et retourne la somme des éléments d'une liste `L` passée en paramètre.
- `diviseurs(n)` : qui retourne la liste des diviseurs propres d'un entier n passée en paramètre (n n'est pas un diviseur propre de lui-même).
- `amis(n,m)` : teste si deux entiers n et m sont amis (deux entiers sont dits amis si chacun est égal à la somme des diviseurs propres de l'autre).

Passage de paramètres

Lors d'un appel à une fonction, Python fait une substitution (replacement) des paramètres formels par des paramètres effectifs. Python utilise deux modes de passage de paramètres:

1. Les types non mutables (int, float, str, complex, bool, tuple...) sont *passés par valeur*.
2. Les types mutables (listes, ensembles, dictionnaires ...) imitent le mode de *passage par variable*.

```python
# Exemple 1 : passage par valeur
def incrementer(n): # n : paramètre formel
    n = n + 1
k = 0
incrementer(k) # k : paramètre effectif
print(k) # k = 0

# Exemple 2 : passage par variable
def ajouter(L,x): # L : paramètre formel
    ''' ajouter à la fin d'une liste L un objet x '''
    L.append(x)
L = [1,2,3]
ajouter(L,4) # L : objet mutable, donc passé par variable
print(L) # L = [1,2,3,4]
```

Paramètres avec valeurs par défaut

Une fonction peut être déclarée avec des paramètres ayant une valeur de départ. Si l'utilisateur omet l'un des paramètres au moment de l'appel, la valeur par défaut sera prise en considération.

```python
def afficher_point(x=0,y=0):
    print('(x,y)=({},{})'.format(x,y))
afficher_point(); afficher_point(1,2); afficher_point(1); afficher_point(y=2)
```

Variables locales et globales

Les variables locales : Toutes les variables de la liste des paramètres d'une fonction, et toutes les variables créées dans une fonction sont des variables locales de la fonction. Les changements des variables locales faites lorsque la fonction est en cours d'exécution n'ont aucun effet sur toutes les variables en dehors de la fonction.
Les variables locales : Toutes les variables du programme principal, et toutes les variables créées dans une fonction avec le mot clé `global` sont des variables globales de la fonction. Toute variable globale est accessible et modifiable par la fonction où elle a été déclarée, par les autres fonctions qui la déclarent comme global et en dehors de toutes les fonctions.

Remarques:

- La *fonction imbriquée (interne)* est une fonction déclarée dans une autre fonction et ne peut être appelée que par la fonction englobante ou par des fonctions imbriquées dans la même fonction englobante.
- `if __name__ == '__main__'` : permet de séparer le corps du programme pricipal des définitions de fonctions, permettant une utilisation en module.

Exercice 2: Écrire en Python les fonctions suivantes

- `maximum(x,y,z)` : retourne le maximum de trois entiers passés en paramètres.
- `estpremier(n)`: vérifie si un entier n passé en paramètre est premier ou non.

Exercice 3

Deux nombres premiers sont *jumeaux* si leur différence est égale à 2. Par exemple, 5 et 7 sont jumeaux, 11 et 13 aussi. Écrire une fonction Python `jumeaux(n)` qui permet de trouver tous les couples de nombres premiers jumeaux compris entre 1 et n.

Exercice 4

La conjecture de Goldbach s'énonce ainsi : Tout nombre entier pair supérieur 3 peut s'écrire comme la somme de deux nombres premiers. Écrire une fonction Python `verif_Goldbach(n)` qui vérifie la conjecture de Goldbach pour tout entier naturel inférieur 1000.