SIO1 Algorithmique : Fonctions et procédures

I - Introduction

1) Le problème: Il arrive souvent qu'une même séquence d'instructions doive être utilisée à plusieurs reprises dans un programme, et on souhaite éviter de la recopier systématiquement.

On utilise pour cela des **fonctions**, qui seront appelées quand c'est nécessaire. Les fonctions sont des sous-programmes.

2) Syntaxe Python pour la définition d'une fonction :

```
def nom_de_la_fonction (liste,de,parametres):        #les : sont obligatoires
    # les paramètres sont séparés par des virgules

    bloc_d_instructions                    # indentation obligatoire

#la fonction est terminée quand il n'y a plus d'indentation
```

- nom_de_la_fonction: Utiliser de préference des caractères minuscules, sans accent ni caractères spéciaux. Ne pas utiliser les mots réservés du langage.

- La ligne contenant l'instruction def se termine obligatoirement par : , lequel introduit un bloc d'instructions à indenter.

- La liste de paramètres (appelée aussi liste d'**arguments**) spécifie quelles informations il faudra fournir lorsque l'on voudra utiliser cette fonction (les parenthèses peuvent rester vides si la fonction ne nécessite pas d'arguments).

- Une fois la fonction définie (de préférence au début du programme), elle sera appelé par son nom dans la suite du programme, chaque fois que nécessaire.

II- Procédures

Ce sont des sous programmes qui ne retournent pas de valeurs (simple suite d'instructions, qui effectuent une action, mais ne gardent rien en mémoire).

1. Sans paramètre

Exemple 1 : Saisir dans la console :

```
>>> def table7():
        n=1
        while n<11:
            print(n*7,end=" ")
            n=n+1
```

Que se passe-t-il après avoir fait « entrer »?
Saisissez alors

```
>>> table7()
```

Créer une procédure offre l'opportunité de donner un nom à tout un ensemble d'instructions. De cette manière, on peut simplifier le corps principal d'un programme.

Ceci va servir à raccourcir un programme, par élimination des portions de code qui se répètent. Par exemple, si on doit afficher la table de 7 plusieurs fois dans un même programme, on n'a pas à réécrire chaque fois l'algorithme qui accomplit ce travail.

2. Avec paramètres

Dans l'exemple 1, on a défini et utilisé une procédure qui affiche les termes de la table de multiplication par 7, mais on pourrait avoir besoin d'afficher la table de 9 ou de 6.

On utilise une fonction **avec paramètre**, en appelant cette fonction, on lui indiquera quelle table on souhaite afficher.

Cette information que l'on transmet à la fonction en l'appelant s'appelle un **argument**.

Exemple 2 : Saisir dans la console :

```
>>> def table(p):
        n=1
        while n<11:
            print(n*p,end=" ")
            n=n+1
```

puis afficher la table de 6 et de 9.

Remarque : L'argument utilisé dans l'appel d'une fonction peut être une variable lui aussi.

Exemple 3 : En utilisant la fonction table(p), afficher les 20 premières tables de multiplication.

3. Avec plusieurs paramètres

La fonction table() définie ci-dessus n'affiche que les dix premiers termes de la table de multiplication, on pourrait souhaiter qu'elle en affiche d'autres, on peut l'améliorer en lui ajoutant des paramètres supplémentaires.

Exemple 4 : Écrire une fonction appelée tableMulti(p,debut,fin) qui prend en paramètres : p (table à afficher), debut (indique le début de la table) et fin (indique la fin).

Tester tableMulti(7,5,15): on doit obtenir

```
...
35 42 49 56 63 70 77 84 91 98 105
```

- Lors de l'appel de la fonction, les arguments utilisés doivent être fournis dans le même ordre que celui des paramètres correspondants (en les séparant eux aussi à l'aide de virgules).
- Le premier argument sera affecté au premier paramètre, le second argument sera affecté au second paramètre, et ainsi de suite.

Exercice 1 : Décrire et expliquer le résultat obtenu :

```
>>> t=11
>>> d=5
>>> f=10
>>> while t<21:
        tableMulti(t,d,f)
        t=t+1
        d=d+3
        f=f+5
```

III- Variables locales, variables globales

Une **variable locale** est déclarée à l'intérieur d'un sous-programme et elle n'est utilisable que dans le sous-programme où elle a été déclarée. Ceci est aussi valable pour le programme principal : une variable déclarée dans le programme principal n'est utilisable que dans le programme principal et pas dans les sous-programmes.

Une **variable globale** est déclarée à l'extérieur du programme principal et des sous-programmes : elle est commune à l'ensemble des sous-programmes et du programme principal, Elle est utilisable partout.

Exercice 2 : Les variables p, debut, fin et n dans l'exemple 4 sont-elles locales ou globales ?
Les variables t, d et f dans l'exercice 1 sont-elles locales ou globales ?

Exemple 5 :

```
>>> def mask():
        p=20
        print(p,q)


>>> p,q=15,38
>>> mask()
20 38
>>> print(p,q)
15 38
```

Expliquez le code ci-contre: p est-il égal à 20 ou à 15 ?

Cela signifie qu'on peut utiliser plusieurs fonctions sans se préoccuper des noms de variables qui y sont utilisées : ces variables ne pourront en effet jamais interférer avec celles définies par ailleurs.

L'instruction global: On peut avoir à définir une fonction qui soit capable de modifier une variable globale. Pour cela, on utilise l'instruction **global**. Cette instruction permet d'indiquer - à l'intérieur de la définition d'une fonction - quelles sont les variables à traiter globalement.

```
>>> def monter():
        a=a+1
        print(a)


>>> a=15
>>> monter()
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in
    monter()
  File "<pyshell#29>", line 2, in
    a=a+1
UnboundLocalError: local variable
```

Expliquer le message d'erreur:

```
>>> def monter():
        global a
        a=a+1
        print(a)


>>> a=15
>>> monter()
16
```

Le même programme avec **global**

```
>>> def monter(a):
        a=a+1
        print (a)


>>> a=15
>>> monter(a)
16
```

Le même programme avec un paramètre (solution plus "propre")

IV- Fonctions

Ce sont des sous programmes qui renvoient une valeur lorsqu'ils se terminent (cette valeur est gardée en mémoire).
L'instruction **return** définit ce que doit être la valeur renvoyée par la fonction.
Exemple 6 : On considère une fonction aire(longueur, largeur) qui renvoie l'aire du rectangle

```
def aire(longueur, largeur):
    a=longueur * largeur
    return a
```

Calculer: aire(5,2)-aire (3,1)

Exercice 3 : 1) Modifier la fonction table(p) de l'exemple 2 pour qu'elle renvoie la liste des dix premiers termes de la table de multiplication choisie, puis l'utiliser pour:

2) Afficher la table de 9,

3) Afficher les 3è, 4è et 5è résultats de la table de 9,

4) Afficher le résultat de 8×9.

Exercices : Algorithmique : Fonctions et procédures

Exercice 1 :

1. Qu'obtient-on comme résultat ? **Ne pas le tester sur Python**, **réfléchissez** !!!

```
x = 7
def f(b):
    x = 3 * b
    print(x)
    return x + 1
print(f(x))
print(x)
```

2. Qu'obtient-on comme résultat ? **Ne pas le tester sur Python**, **réfléchissez** !!!

```
def f(a):
    return 2 * a
def g(a):
    return f(2 * a)
a = 2
print(f(a + 1))
print(g(a + 1))
```

Exercice 2 :

1) Écrire une fonction qui retourne le maximum de deux nombres entiers.

2) Écrire une fonction somme qui prend en paramètres 2 entiers m et n et qui retourne la somme des entiers de m à n (on supposera que $m < n$). Par exemple, somme(5,8) donne 26 ($= 5 + 6 + 7 + 8$).

Exercice 3 :

1) Écrire une fonction echange(a,b) qui échange les valeurs de deux variables.

2) Écrire un algorithme qui utilise la fonction echange pour:

(a) Classer deux variables par ordre croissant

(b) Classer trois variables par ordre croissant.

Exercice 4 :

Écrire une fonction **premier** qui teste si un nombre est premier et retourne un booléen.

Utiliser cette fonction dans un algorithme qui saisit un nombre entier N et affiche tous les nombres premiers inférieurs à N.

Exercice 5 :

Rappel : Avec python on peut générer des nombres aléatoires avec le module random (from random import *), alors la commande randint(1,8) génère un entier aléatoire compris entre 1 et 8.

a) Créer une fonction qui génère 4 nombres aléatoires compris entre 1 et 20. et les met dans une liste.

b) Créer une deuxième fonction qui cherche si les nombres d'une liste sont rangés par ordre croissant.

c) Créer une troisième fonction qui cherche si une liste est constituée de nombres consécutifs (nombres qui se suivent: 5,6,7,8 sont consécutifs mais pas 5,5,7,8 ni 10,12,14,15).

d) Utiliser ces trois fonctions pour tester le nombre d'essais nécessaires pour obtenir 4 nombres rangés dans l'ordre croissant consécutifs .

Exercice 6 :

On emprunte un capital C à un taux annuel t.

Au bout d'un an , on doit rembourser C(1+t)

Au bout de deux ans, on doit rembourser C(1+t)(1+t)

Au bout de trois ans, on doit rembourser C(1+t)(1+t)(1+t)

etc..,

t étant exprimé en pourcentage décimal (0,1 pour 10% par exemple).

1) Écrire une fonction qui renvoie le capital à rembourser et le montant de chaque mensualité à partir des données : capital emprunté, durée et taux.

2) Écrire une fonction qui renvoie le capital restant dû à partir du capital à rembourser, du montant de chaque mensualité et du nombre de mensualités déjà versées.

3) Écrire le programme principal qui saisit le capital emprunté, la durée de l'emprunt et le taux, qui affiche le capital à rembourser, le montant de chaque mensualité, et qui propose de calculer le capital restant dû à partir de la saisie du nombre de mensualités déjà versées.

Exercice 7 :

On veut écrire un programme qui gère un tableau d'entiers, de taille inférieure à 100.

On écrira plusieurs fonctions:

- une fonction Initialise qui remplit le tableau par saisie (la taille du tableau sera passée en paramètre).
- une fonction Maxi et une fonction Mini qui retourneront respectivement le maximum et le minimum.
- une fonction Moyenne qui retourne la moyenne des éléments du tableau.
- Le programme principal qui saisira la taille du tableau , initialisera le tableau, et affichera le maximum, le minimum et la moyenne.

Fonctions récursives (Fonction qui s'appelle elle-même).

Exercice 8 :

Quel résultat a-t-on si l'on calcule f(10)? **Ne pas le tester sur Python**, **réfléchissez** !!!

```
def f(x):
    if x == 0:
        return 0
    return x + f(x - 1)
```

Exercice 9 :

Écrire un programme qui renvoie le calcul des premières factorielles jusqu'à un entier rentré par l'utilisateur. On utilisera une fonction récursive.

Exercice 10 :

1) Écrire une fonction récursive qui renvoie la somme des N premiers multiples de 3.

2) Écrire une fonction récursive qui renvoie la somme des carrés des nombres entiers de 1 à N.

Exercice 11 :

Écrire une fonction récursive PGCD qui renvoie le PGCD de deux nombres A et B

1) à l'aide de divisions successives.

2) à l'aide de soustractions successives.