Sous-programmes

1) Principe

Un sous-programme est un bloc d'instructions réalisant une certaine tâche. Un script bien structuré contient un programme principal, et des sous-programmes.

Les sous-programmes évitent la duplication de code pour exécuter des tâches identiques, peuvent être réutilisés par d'autres scripts et réduisent la complexité des programmes.

2) Notion de paramètre

Les paramètres sont les données à transmettre au sous-programme par le programme principal.

3) Variables locales ←→ variables globales

Un sous-programme peut avoir besoin d'utiliser des variables qui lui sont propres. On parle alors de **variables locales**. Ce sont par exemple des résultats de calculs intermédiaires, des compteurs de tours dans une structure itérative, etc. Ces variables ne sont accessibles qu'au sein du sous-programme qui les définit et utilise.

Un sous-programme peut également manipuler des variables définies par le programme principal. On parle alors de **variables globales**. Il s'agit d'une possibilité mais également d'une mauvaise pratique : un sous-programme manipulant des variables globales n'est pas réutilisable en l'état dans un autre projet.

4) Fonctions

Une **fonction** est un sous-programme qui retourne un résultat au programme principal. La signification d'une fonction est donc la même qu'en Mathématiques.

Un sous-programme qui ne retourne pas de résultat au programme principal est appelé **procédure**.

5) Les sous-programmes en Python

Dans le langage Python, l'implémentation d'un sous-programme commence par le mot clé **def**, se poursuit par son **nom** et la **liste de ses paramètres**. Un bloc est alors reservé aux instructions du sous-programme, délimité comme d'habitude par un « **:** » et une **indentation**.

Syntaxe générale :

```
def monSousProgramme(para1,para2,...,paraN):
    bloc d'instructions de la procédure
    return valeur
```

Exemple : fonction calculant la somme des n premiers entiers, avec deux variables locales :

```
def sommeEntiers(n):
    somme = 0
    for i in range(n+1):
        somme += i
    return somme
```

Exemple d'utilisation du sous-programme sommeEntiers dans le programme principal :

```
nombre = eval(input("Saisir un nombre entier : "))
s = sommeEntiers(nombre)
print(s)
```

Les variables globales ne sont pas modifiables par des sous-programmes.

Pour que la variable globale *varGlobale* soit modifiable par un sous-programme, il faut le signaler dans le sous-programme à l'aide de l'instruction **global** au début du sous-programme :

```
global varGlobale
```

6) Modules de fonctions

Un module de fonctions est un fichier d'extension ".py" contenant des sous-programmes.

Pour pouvoir utiliser les fonctions d'un module dans un programme, on doit l'importer, ce qui signifie mettre ses fonctions à disposition.

Il y a trois méthodes pour importer un module de fonctions :

Méthode 1 : On importe tout le module

```
from NomDuModule import *
```

Pour utiliser une fonction, on l'appelle alors directement par son nom.

Méthode 2 : On importe une seule fonction

```
from NomDuModule import uneFonction
```

Pour l'utiliser la fonction en question on l'appelle directement par son nom.

Méthode 3 : On importe tout le module

```
import NomDuModule
```

Cette fois-ci pour utiliser une fonction dans le programme on utilse une syntaxe de la forme :

```
NomDuModule.nomDeLaFonction(...)
```