Numérique et sciences informatiques
Classe de première

A Langages et programmation
4. Les fonctions

I Quelques fonctions prédéfinies déjà rencontrées

I.1 La fonction `input(...)`

• Un exemple d'appel de la fonction **`input(...)`** :

```
message=input("Entrez votre texte")
```

• L'**argument** (ou le **paramètre**) de la fonction : ici, l'argument de la fonction est l'objet de type **`str`** `"Entrez votre texte"`.
La fonction **`input(...)`** peut prendre un argument ou aucun argument mais elle ne peut pas prendre plus d'un seul argument.
Remarque : La fonction peut prendre en argument des objets d'autres types que **`str`**.

• La **valeur renvoyée** par la fonction **`input(...)`** est un objet de type **`str`**.

Lors de l'appel de la fonction, il faut une instruction d'affectation pour récupérer la valeur renvoyée par la fonction. Sinon, cette valeur renvoyée sera perdue.

I.2 La fonction `print(...)`

• Un exemple d'appel de la fonction **`print(...)`** :

```
A=5
B=8.9
C= "bonjour"
D=B<5
print(A, B, C, D)
```

• **Les arguments** de la fonction : la fonction **`print(...)`** peut prendre un nombre quelconque d'arguments de tous les types.

Attention ! Dans l'appel d'une fonction qui comporte plusieurs paramètres, il faut les séparer par des virgules : `,` (et non pas des points-virgules).

• La **valeur renvoyée** par la fonction : La fonction **`print(...)`** ne renvoie rien. Il s'agit d'afficher des résultats à l'écran et non de faire un calcul et de fournir le résultat à la suite du programme. En informatique, une fonction qui ne renvoie pas de valeurs de sortie s'appelle **une procédure**.

I.3 Un exemple de fonction sans arguments : `random`

• Un exemple d'appel de la fonction **`random()`** :

```
from random import *
nbAleat=random()
print(nbAleat)
```

• **Arguments** de la fonction : la fonction **`random()`** n'utilise aucun argument.

Attention ! Même dans le cas où il n'y a pas d'arguments, il ne faut pas oublier les parenthèses lors de l'appel de la fonction.

• La **valeur renvoyée** par la fonction : La fonction **`random()`** renvoie un objet de type **`float`** compris entre 0 inclus et 1 exclus.

I.4 Appel de procédure et appel des autres fonctions

a) Les procédures

• Si la fonction ne renvoie aucune valeur, par exemple si son rôle est d'effectuer un affichage à l'écran, alors l'appel de la fonction constitue en soi une instruction. Par exemple :

```
print("La variable A contient", A)
```

b) Les fonctions qui renvoie une valeur

• Si la fonction renvoie une valeur à l'issue du traitement, l'appel de la fonction doit s'intégrer dans une instruction qui permet d'exploiter le résultat de la fonction.
Par exemple :

```
nb = input("Entrez la valeur du nombre : ")
nb = float(nb)
```

II Définir ses propres fonctions

II.1 La syntaxe générale de définition d'une fonction en Python

a) En tête et corps de la fonction

• Pour créer une nouvelle fonction, il faut respecter la syntaxe suivante :

```
def nomFonction (arg1, arg2, ...) :
    bloc d'instructions
```

Il ne faut pas oublier :
* le mot-clé **`def`** ;
* les *parenthèses* après le nom de la fonction ;
* les arguments doivent être séparés par une virgule (et non un point-virgule) ;
* le signe '**:**' ;
* *l'indentation* du bloc d'instructions.

• La définition d'une fonction peut utiliser un nombre quelconques d'arguments. Il peut aussi ne pas y en avoir du tout.

• Du vocabulaire :
* la première ligne est appelée l'**en-tête de la fonction**. Elle permet de connaître :
  - le nom de la fonction ;
  - le nom et le nombre des **arguments** (ou **paramètres**) utilisés.
* le bloc d'instructions indenté après le signe '**:**' est appelé le **corps de la fonction**. Il décrit les opérations efectuées par la fonction.

Fonction sans argument

Si la fonction ne comporte pas d'arguments, on n'écrit rien entre les deux parenthèses de l'en-tête.

b) L'instruction `return ...`

• Pour qu'une fonction renvoie une valeur à l'extérieur du programme, il faut que le corps de la fonction comporte au moins une instruction de la forme

```
    return expression
```

où *expression* peut aussi bien un nom de variable qu'une expression à évaluer (comportant ou non un nom des variables).

• Lors de l'execution d'un appel de fonction, l'interpréteur va :
  * créer les variables portant le nom des arguments et leur affecter les valeurs choisies lors de l'appel de la fonction ;
  * exécuter une à une les instructions du corps qui forment le corps de la fonction ;
  * dès que l'interpréteur rencontre une instruction de la forme **return** *expression, il sort de la fonction* et renvoie le résultat de l'évaluation de *expression* pour la suite du programme.

II.2 Un premier exemple

Nous voulons créer une fonction `triple(...)` qui prend pour argument un nombre (de type **`int`** ou **`float`**) et qui renvoie le produit de ce nombre par 3. Nous avons donc écrit le programme suivant qui crée la fonction triple(...) et qui la teste :

```
def triple(nb) :
 return 3*nb

N=487
print("Le triple de", N, "est", triple(N))
```

• Dans ce programme, il faut distinguer :
  * la définition de la fonction `triple(...)` avec son en-tête et son corps
  * les instructions qui suivent cette définition qui sont destinées à tester la fonction. Ce bloc est appelé le **programme principal**.

• Execution du programme par l'interpréteur Python :
  * après vérification que le programme ne comporte pas d'erreurs de syntaxe majeures (par exemple, oubli du mot-clé `def`, ou du signe ' :'), l'interpréteur passe les lignes de définition de la fonction et execute la première instruction du programme qui suit cette définition en ligne 4. Il affecte donc la valeur 487 à la variable `N` ;
  * il évalue un à un les arguments du `print` et rencontre donc l'appel de fonction `triple(N)`
  * il commence l'exécution du corps de la fonction `triple(...)` :
  - il crée la variable `nb` et lui affecte la valeur de la variable `N`, c'est-à-dire 487 ;
  - il évalue l'expression `3*nb` ;
  - il renvoie le résultat trouvé au programme principal, c'est-à-dire 1461, pour finir l'execution du print
  * il exécute l'instruction `print(...)` et affiche donc à l'écran le message suivant

```
"Le triple de 487 est 1461"
```

<u>Nécessité d'un appel de fonction pour la tester</u>

Pour tester une fonction que vous avez créée, il vous faut aussi écrire un programme principal qui appelle cette fonction. Sans cela, la fonction définie ne sera pas exécutée et vous ne pourrez pas savoir si elle fonctionne.

Position de la définition des fonctions et du programme principal

L'appel d'une fonction ne peut apparaître dans un programme qu'une fois que cette fonction a été créée. Il est donc nécessaire que le programme principal qui utilise les fonctions soit situé *après* la définition des fonctions utilisées.

• Essayez vous-même :

```
N=487
print("Le triple de", N, "est", triple(N))

def triple(nombre) :
    return 3*nb
```

Que dit le message d'erreur ?

II.3 A propos de l'instruction `return ...`

• On peut placer une instruction de la forme `return ...` ailleurs qu'à la dernière ligne du corps de la fonction. Par exemple dans un « `if ...` » pour traiter un cas particulier. Dans tous les cas, dès que l'interpréteur rencontre un `return` *exp*, il sort de l'execution de la fonction et renvoie le résultat de l'évaluation de *exp*.

• Un exemple : on veut programmer la fonction `factorielle` qui prend en argument un entier positif n et qui renvoie ;

* 1 si ce nombre n est égal à 0 ;
* le produit de tous les entiers compris entre 1 et n (compris) sinon.

On peut alors écrire le programme suivant :

```
def factorielle(n) :
    if n == 0 :
        return 1
    f = 1
    for C in range(2, n+1) :
        f = C*f
    return f
```

• On voit qu'il n'est pas utile d'écrire un `else` ... après le `if ...` car si la condition `n == 0` est vraie alors l'instruction `return 1` sera exécutée et le reste du corps de la fonction ne sera donc pas executé.