Chapitre : Les Fonctions

- *Author* : **Ibrahima SY**
- *Email* : syibrahima31@gmail.com
- *Github* : Cliquez
- *Linkdin* : Cliquez
- *School* : Institut Supérieur Informatique (**ISI**)
- *Spécialité* : Licence 2 **GL**

PLAN

1. Introduction

2. Définir une fonction

3. Appel et utilisation des fonctions

4. Modification de la portée

5. Passage des arguments et appel de fonctions

Introduction

En Python, il est possible de définir ses propres fonctions. Les fonctions permettent de :

- Simplifier l'écriture du programme
- Le rendre plus facile à corriger
- Décomposer un programme complexe en plusieurs sous-programmes Un programme décomposé en fonctions est un programme modulaire

Définir une fonction

- Les fonctions et les classes d’objets sont différentes structures de sous-programmes qui ont été imaginées pour faciliter la factorisation du code .
- Nous avons déjà rencontré diverses fonctions pré-programmées telque :
  - `print` : pour afficher
  - `len` : calculer la taille d'un objet

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nomFonction(Param1, Param2, ..., Paramk) :
    ...
    blocInstructions
    ...
```

où :

- `nomFonction` : mêmes contraintes que les caractères pour variables
- `Param<1–k>` : les paramétres de cette fonction
- `blocInstructions` : suite d'instructions exécutées par la fonction ; il s'appelle le corps de la fonction

- Si la fonction doit renvoyer une valeur, on peut renvoyer avec l'instruction return. Ainsi, écrire `return` expression termine l'exécution et renvoie la valeur calculée de l'expression.
- Les instructions éventuelles qui suivent le `return` ne seront jamais exécutées.

Remarque :

> on peut trouver plusieurs instructions `return` dans le corps d'une fonction, mais toutes sauf éventuellement une, devront se trouver dans les instructions conditionnelles.

Une fonction peut ne rien renvoyer:

- soit un l y'a aucune instruction return dans le corps de la fonction et ce dernier s'exécute jusqu'à la fin
- soit on met `return` sans expression derrière
  Dans ces deux cas, l'interpréteur Python considère que la fonction renvoie à un mot clé spécial appelé `none`

Une fonction peut renvoyer plusieurs valeurs

```
>>> def resteEtQuotient(num, denum) :
        return num%denom, num//denom Quotient=num//denom

>>> R, q=resteEtQuotient(9,4)
```

Appel et utilisation des fonctions

Syntaxe de l'appel d'une fonction dans le programme principal. `nomFonction(arg1, arg2, ..., argk)`

- Il faut que le nombre d'`arguments` soit égal au nombre de `paramétres` de la fonction. Cette expression déclenche l'exécution du corps de la fonction dans un contexte où : `Param1=arg1`, `Param2=exp2`, ..., `Paramk=argk`

Portée des variables - Règle LEGB

La portée d'une variable détermine de quel endroit du code on peut accéder à cette variable.
Python utilise la **portée lexicale**. Cela veut dire que la portée d'une variable est déterminée en fonction de l'endroit dans le code où cette variable est définie

Il y'a deux types de variables :

Variables locales

Une variable locale au bloc de code d'une fonction est une **variable dite locale**. Lorsque la fonction retourne, toutes les variables locales de la fonction sont détruites.

Variables globales

Une variable définie en-dehors de toute fonction est une **variable globale**.

La règle LEGB

la règle `LEGB` signifie lorsque l'on référence une variable, on commence à le chercher

1. à l'endroit où elle a été référencée, dans le cas d'une on regarde si elle a été définie `localement`
2. Si elle n'a pas été définie localement à cette fonction, on va aller la chercher dans les `fonctions englobantes`.
   Donc on va remonter de la fonction la plus proche
   jusqu'à la fonction la plus `externe`.
3. Si on ne trouve pas cette variable définie dans les fonctions englobantes, alors on va la chercher `globalement`, c'est-à-dire au niveau des variables `g`
4. Si on ne la trouve toujours pas, on la cherchera dans le module `builtins`.

Exemple : portée de variable

```
def g( ):
    b, c = 2, 4
    b = b + 10
    def h( ):
        c = 5
        print(a, b, c)
    h( )
```

Modification de la portée

la portée des variables peut être modifié avec les instructions `global` et `nonlocal`.

global permet de rend

Le fonctionnement de global

```
## pas de modification de a
>>> a = "a globale"
>>> def f():
        a = "a dans f"
        print(a)
>>> f()
a dans f
```

```
## cette fois la valeur de a change
>>> a = "a globale"
>>> def f():
    global a
    a = "a dans f"
    print(a)
>>> f()
a dans f
```

Le fonctionnement de nonlocal

nonlocal sert à modifier la portée d'une variable locale pour accéder à une variable également
locale mais dans une fonction englobante.

```
a = 'a global'
def f():
    a = 'a de f'
    def g():
        a = 'a de g'
        print(a)
    g()
    print(a)
```

Passage des arguments et appel de fonctions

Les formes étoiles * et ** pour les paramètres

Il existe en tout 4 manières de définir les paramètres d'une fonction et 4 manières de passer les arguments d'une fonction.

La forme étoile `*` :
L'intérêt de cette notation étoile, c'est qu'elle permet de passer une liste quelconque d'arguments, les arguments vont être mis dans un `tuple` par la variable qui suit l'étoile, à la fonction

```
>>> def f(*t):
        print(t)
>>> f(1,2,3)
(1,2,3)
```

La forme double étoile `**` :

La forme double étoile permet de passer n'importe quel argument nommé à la fonction , elle permet aussi d'appeler la fonction sans aucun argument