Informatique VI
Fonctions

Laurent Kaczmarek

PCSI^2 2013-2014
Lycée Louis Le Grand

Mardi 1er octobre 2013

Motivation, exemple introductif

- Calculons la somme des cubes des entiers pairs compris entre 0 et 566.
- On trouve 12919316768 par le programme suivant :

```
somme=0
for i in range(2,567,2):
    somme=somme+i**3
print(somme)
```

- Si dans un projet, on a besoin de calculer la somme des cubes des entiers pairs compris entre 0 et n pour de nombreuses valeurs de n, il sera plus lisible et efficace de créer une portion de code effectuant ce calcul indépendemment du projet, ie un sous-programme auquel on pourra faire appel à loisir.

Motivation, exemple introductif

- Pour cela on définit une *fonction*.
- Voici la définition sous Python d'une fonction `sommeCubes` :

```
def sommeCubes(n):
    somme=0
    for i in range(2,n+1,2):
        somme=somme+i**3
    return somme
```

- On utilise alors cette fonction avec la même syntaxe qu'en Mathématiques :

```
>>> sommeCubes(566)
12919316768
>>> sommeCubes(10000)
1250500050000000
```

Les fonctions en informatique

En informatique, une fonction est une portion de code représentant un sous programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme.

Intérêts

- *factorisation* : éviter la duplication de code en remplaçant les parties dupliquées par un appel à une fonction unique;
- *réutilisation* : une fonction peut être utilisée par plusieurs programmes (bibliothèque) ;
- *lisibilité* : regrouper un ensemble d'instructions dans une fonction, nommée de façon explicite, facilite la relecture du code et cache les détails de codage;
- *structuration* : découper en sous-problèmes ; distribuer de leur programmation à différents développeurs, à différentes étapes de la réalisation d'un projet.

Les fonctions en informatique

Les fonctions en programmation impérative

En programmation impérative, une *fonction* est un algorithme qui prend des *arguments* en entrée, effectue une séquence d'instructions et renvoie un résultat.

Le type function

Une fonction est considérée par Python comme un objet de type *function*.

Définir et appeler

Nous allons à présent décrire les deux principales manipulations sur les fonctions.

- *Définir une fonction* : décrire le corps de la fonction, c'est-à-dire la suite d'instructions qui constitue son calcul.
- *Appeler une fonction* : utiliser une fonction, faire appel à son résultat en fixant les valeurs des arguments.

De nombreuses fonctions sont prédéfinies

- Les fonctions usuelles des Mathématiques, telles que *cos* ou *abs* (la valeur absolue), sont prédéfinies.
- Il faudra le plus souvent charger une *bibliothèque* contenant sa définition avant de pouvoir l'utiliser. Ce mécanisme permet d'économiser de la mémoire en laissant de côté les programmes dont l'utilisateur ne se sert pas.
- Pour charger la bibliothèque `nombibli`, on utilisera l'instruction

```
import nombibli
```

- Pour charger la fonction `nomfonc` de la bibliothèque `nombibli`, on utilisera l'instruction

```
from nombibli import nomfonc
```

Définition d'une fonction

Première syntaxe, utilisation de def

- De manière générale, la syntaxe pour définir une fonction est la suivante :

```
def nom(arg1,...,argN):
    '''Notice de la fonction'''
    Instructions
```

  On n'oubliera pas les : et l'*indentation* qui sont obligatoires en Python.
- On commence par le *mot-clé* def, suivi du nom de la fonction et d'une liste entre parenthèses de *paramètres formels* ; cette première ligne se termine par :.
- Les instructions qui forment le corps de la fonction commencent sur la ligne suivante, indentée par quatre espaces (ou une tabulation).

Définition d'une fonction

Première syntaxe, utilisation de `def`

- La première instruction du corps de la fonction peut être une chaîne de caractères constituant la documentation de la fonction. On peut la visualiser dans un terminal en tapant l'instruction `help(nom)`.
- Le retour à la ligne signale la fin de la fonction.
- La première instruction du corps de la fonction peut être une chaîne de caractères constituant la documentation de la fonction. On peut la visualiser dans l'interpréteur en tapant l'instruction `help(nom)`.
- Le retour à la ligne signale la fin de la fonction.
- Dès que l'instruction `return` est exécutée (si elle est présente), l'exécution de la fonction se termine ; la partie du code écrite après l'instruction `return` n'est jamais exécutée.

Seconde syntaxe, les lambda-fonctions

- Python permet une syntaxe intéressante qui vous laisse définir des mini-fonctions d'une ligne à la volée.
- Empruntées au langage Lisp, ces fonctions dites lambda peuvent être employées partout où une fonction est nécéssaire.
- La syntaxe est la suivante :

  ```
  nomfonction=lambda nomvariable:  expression
  ```

- Exemple :

```
>>> f=lambda x:  x**2+1
>>> f(3)
10
>>> f(10)
101
```

Appel d'une fonction

Utilisation d'une fonction sous Python

- Si `f` est le nom de la fonction, on obtient le résultat de `f` pour un jeu de d'arguments `arg1`, ..., `argN` par l'appel suivant, conforme à l'usage mathématique :

  `f(arg1,...,argN)`

- Exemple :

```
>>> cos(pi/2)
Traceback (most
recent call last):
File "<stdin>", line
1, in <module>
NameError:  name
'cos' is not defined
>>> import math
>>> math.cos(pi/2)
6.123031769111886e-17
```

```
>>> import math
>>> from math import
cos
>>> cos(pi/2)
6.123031769111886e-17
>>> import math
>>> from math import
cos as f
>>> f(pi/2)
6.123031769111886e-17
```

Appel d'une fonction

Utilisation d'une fonction sous Python

- La fonction suivante calcule la partie entière d'un nombre réel x.

```
def pe(x):
    n=0
    if x>0:
        while n<=x:
            n=n+1
        return n-1
    else:
        while x<n:
            n=n-1
        return n
    for k in range(n+1):
```

Appel d'une fonction

Le passage des arguments

- Un paramètre d'entrée est une donnée fournie à la fonction.
- Il peut s'agir d'une donnée brute, d'une variable ou plus généralement d'une expression.
- Cette donnée peut être transmise de deux façons :
  - *passage par copie (ou par valeur)* : le code appelé dispose d'une copie de la valeur qu'il peut modifier sans affecter l'information initiale dans le code appelant ;
  - *passage par adresse (ou par référence)* : le code appelé dispose d'une information lui permettant d'accéder en mémoire à la valeur que le code appelant cherche à lui transmettre. Il peut alors modifier cette valeur là où elle se trouve ; le code appelant aura accès aux modifications faites sur la valeur. Dans ce cas, le paramètre peut aussi être utilisé comme un paramètre de sortie.

Appel d'une fonction

Passage des arguments sous Python

- Sous Python, le passage des arguments d'une fonction se fait par copie sauf pour les listes. Les expressions passées en paramètres sont évaluées avant d'être passées à la fonction.
- Exemples :

```
>>> def f(x):
        x=0
>>> a=1
>>> f(a),a
1
```

```
>>> def g(t):
        t[0]=4
>>> a=[1,1]
>>> g(a),a
[4,1]
```

Un argument peut être une fonction

```
>>> def somme(f,n):
        somme=0
        for k in range(n+1):
            somme=somme+f(k)
```

```
>>> f=lambda x:x
>>> somme(f,100)
>>> 5050
```

Variables locales, globales

Portée d'une variable

- Lors de l'appel d'une fonction, Python crée une table des noms locale qui est supprimée à la fin de l'appel. Ainsi, la portée des variables définies à l'intérieur de la fonction est limitée à la suite d'instructions définie à l'intérieur de celle-ci.
- Au cours de l'exécution, si Python a besoin de la valeur d'une variable `grr`, il va chercher le nom `grr` dans la table *locale* des symboles ;
- si `grr` ne s'y trouve pas, il consultera la table *globale* ;
- si `grr` ne s'y trouve pas non plus, il enverra un message d'erreur.

Exemple

```
>>> n=3
>>> def puiss2(n):
        N=2
        return n**N
>>> puiss2(10)
100
>>> n
3
```

```
>>> N
Traceback (most recent
call last): File
"<pyshell27>", line 1,
in <module> N NameError:
name 'N' is not defined
```

Un petit conseil

- On évitera autant que possible la définition de variables globales telles que la variable `n` de l'exemple précédent. Par exemple, si une variable locale et une variable globale portent le même nom, la variable globale ne sera pas accessible pendant l'exécution de la fonction. De plus, si une fonction utilise une variable globale, sa modification altère le résultat de la fonction.