

Travaux pratiques d'introduction 3

Les boucles «for»

Informatique tronc commun MPSI et PCSI

I Présentation

I.1 Introduction

Définition 1 : Boucles

Une boucle est une séquence d'instructions qui se répète. On distingue deux types de boucles :

- Les boucles inconditionnelles : les boucles «for» sont adaptées lorsqu'on connaît à l'avance le nombre de fois qu'une séquence devra se répéter (on parle également de boucles bornées).
- Les boucles conditionnelles : les boucles «while» feront l'objet d'un autre TP, sachez néanmoins que ces boucles présentent l'avantage d'effectuer une séquence d'instructions sans en connaître à l'avance le nombre de répétitions (on parle également de boucles non bornées).

I.2 Syntaxe

Les instructions Python pour répéter n fois un ensemble d'instructions sont

```
for i in range(n):  
    instruction 1 à répéter  
    instruction 2 à répéter  
    ...  
    instruction p à répéter  
suite des instructions après répétition
```

Cette syntaxe se traduit par «pour i allant de 0 à $n - 1$ faire les instructions 1 à p ». L'indice de boucle i prend successivement les valeurs $0, 1, 2, \dots, n - 2, n - 1$. Donc i prend bien n valeurs différentes.

Les instructions à répéter forment un bloc qui est repéré par l'indentation supplémentaire (décalage de quatre espaces ou d'une tabulation). Notons que ce bloc est précédé du symbole de ponctuation « : » comme lors de la définition d'une fonction (pour rappel, dans la syntaxe python, toute instruction qui nécessite un bloc d'instructions est suivie du deux-points « : »).

Le script suivant permet d'afficher le carré des nombres de 0 à 9 :

```
for i in range(10):  
    print(i*i)
```

L'indice « i » prend bien les valeurs $0, 1, 2, \dots, 8, 9$ soit 10 valeurs.

Autre exemple, la fonction suivante calcule la somme des n premiers carrés :

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

Jusqu'ici, on a utilisé l'opération `range` avec un unique paramètre décrivant le nombre de répétitions à effectuer. La syntaxe complète de cette opération est en réalité plus riche et permet de créer une suite finie d'entiers en lui donnant 3 paramètres :

`range(début, fin, pas)`

- qui commence par la valeur `début` (0 par défaut),
- qui progresse de la valeur `pas` à chaque étape (1 par défaut),
- qui s'arrête dès qu'elle dépasse (ou atteint) la valeur `fin`.

L'opération `range(n)` est donc équivalente à `range(0, n, 1)` ou encore à `range(0, n)`.

L'opération `range(1, n+1, 1)` permet d'obtenir le même nombre de répétitions mais avec un indice qui va de 1 à n inclus.

L'opération `range(1, n, 2)` permet d'obtenir des indices impairs qui vont de 1 à n exclus.

L'opération `range(n, 0, -1)` permet d'obtenir des indices décroissants qui vont de n à 1 inclus.

Remarques :

- En python, les chaînes de caractères sont considérées comme un certain type de collections d'objets, en l'occurrence des caractères. On peut donc accéder aux éléments de cette collection à l'aide d'une boucle `for`.

```
mot='Bonjour'
for c in mot :
    print(c)
```

La variable `c` prend bien les caractères 'B', 'o', 'n', 'j', 'o', 'u' et 'r'.

- Si l'indice de boucle n'est pas utilisé dans le bloc d'instructions, on peut le remplacer par le caractère «`_`» qui représente une variable sans nom.

```
for _ in range(4) :
    print('Bonjour')
```

Ce script affiche 4 fois la chaîne de caractères 'Bonjour'.

II Exercices

Exercice 1

Quelles sont les suites d'entiers générées par les commandes suivantes ?

1. `range(5, 17, 3)`
2. `range(5, 9)`
3. `range(5)`
4. `range(6, 0, -1)`
5. `range(0)`
6. `range(4, -1, -1)`
7. `range(4, 4)`
8. `range(10, 5, -2)`

Exercice 2 - Puniton

Écrire une fonction `ecrirePuniton(texte, n)` qui reçoit deux paramètres :

- `texte` est une chaîne de caractères,
- `n` est un entier positif,

et qui écrit n fois le texte à l'écran. Exceptionnellement, il n'y a pas de `return` dans cette fonction.

Exercice 3 - Suite de Héron

Écrire une fonction `heron(n, u0)` qui renvoie le terme u_n de la suite (u_p) définie par u_0 prend la valeur `u0` et $u_{p+1} = \frac{1}{2} \left(u_p + \frac{2}{u_p} \right)$.

On doit trouver `heron(5, 1) = 1.414213562373095`.

Exercice 4 - Suite de Fibonacci

On considère la suite de Fibonacci définie par

$$F_0 = 0, F_1 = 1 \text{ et } F_{n+2} = F_{n+1} + F_n \text{ pour } n \geq 0$$

Écrire une fonction `fibonacci(n)` qui renvoie la valeur F_n . (*Aide : on pourra maintenir deux variables qui correspondent à 2 valeurs consécutives de la suite*).

Affichez les valeurs de F_0 à F_{12} .

Exercice 5 - Suite harmonique

Écrire une fonction `harmonic(n)` qui renvoie $\sum_{k=1}^n \frac{1}{k}$ pour $n \geq 1$.

Exercice 6 - Factorielle

Écrire une fonction `facto(n)` qui renvoie $n!$.

En utilisant la fonction `facto`, écrire une fonction `binomial(n, p)` qui renvoie $\binom{n}{p}$.

Exercice 7 - Vers e

La suite $(S_n)_{n \geq 0}$ définie par $S_n = \sum_{k=0}^n \frac{1}{k!}$ converge vers e lorsque n tend vers l'infini.

À l'aide de la fonction `facto` écrite à l'exercice 6, écrire une fonction `s(n)` qui renvoie la valeur de la suite au rang n .

Vérifier l'hypothèse proposée pour $n = 2$, $n = 3$, $n = 20$, $n = 200$, $n = 2000$ et $n = 3000$.

Exercice 8 - Vers e plus vite

L'interpréteur Python met plusieurs secondes pour calculer `s(3000)` avec la fonction précédente. En effet à chaque répétition de la boucle `for`, la factorielle est calculée complètement.

Écrire une fonction `s_bis(n)` qui renvoie la valeur de la suite au rang n en minimisant le nombre de calculs (donc sans appel à la fonction `facto`).

III Compléments

Exercice 9 - Somme de puissances

Écrire une fonction `sommePuissances(n, p)` qui renvoie $\sum_{k=1}^n k^p$.

Calculer, pour quelques valeurs de n ,

`sommePuissances(n, 3) - (sommePuissances(n, 1))**2`.

Exercice 10 - Suite de Muller (1993)

La suite de Muller est définie par $u_0 = \frac{5}{2}$, $u_1 = \frac{17}{5}$ et, pour $n \geq 0$ $u_n = 30 - \frac{129}{u_{n+1}} + \frac{100}{u_{n+1} \cdot u_{n+2}}$.

Écrire une fonction `muller(n)` qui calcule le terme u_n de cette suite.

Calculer les 50 premières valeurs de u_n ; quelle semble être la limite ?

On prouve, par récurrence, que $u_n = \frac{4^{n+1} + 1}{4^n + 1}$. Cette suite converge vers 4.

On peut utiliser des fractions; elles sont utilisables comme les nombres.

Exemple de code Python :

```
from fractions import Fraction
a = Fraction(2, 6)
print(a, a*2, float(a))
```

Exercice 11 - Usage des fractions dans la suite de Muller

Écrire une fonction `muller_frac(n)` qui calcule le terme u_n de la suite de Muller en utilisant les fractions.

Calculer les 50 premières valeurs de u_n ; quelle semble être la limite ?