TD 02 Structures itératives

Lycée Louis Thuillier - Informatique - PCSIB -

Une chose pour laquelle un ordinateur est très fort c'est répéter un grand nombre de fois une tâche prédéfinie et répétitive : on crée ce qu'on appelle **des boucles itératives**.

1 Boucles *for*

1.1 Notion de boucle itérative

▶ **Structure d'une boucle** *for*

Dans le cas d'une boucle dit *for*, l'ordinateur répète un certain nombre de fois un bloc d'instructions. Mais, plutôt que de donner le nombre de fois que la boucle doit se répéter, on va introduire **un marqueur de boucle**, noté généralement i ou k.

Définition. Structure d'une boucle for

Pour i allant de a jusqu'à b **faire** :

Instruction 1

Instruction 2

Instruction 3

Plein de remarques ....

▷ pour séparer les instructions de la boucles et les autres, on indente ! Ainsi l'Instruction 3 ne sera exécutée **qu'une fois la boucle terminée**.

▷ A la fin de la dernière instruction de la boucle (*ici l'Instruction 2*, le marqueur de boucle est augmenté de +1.

▷ **Attention ! Le marqueur de boucle s'arrête à b-1 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!**

▷ la première ligne crée la variable marqueur i : il ne faut pas que son nom interfère avec celui d'une autre variable déjà appelée i.

Application 1 : Ecrire les structures de boucles pour :
1. *afficher* 10 *fois "Bonjour !"*
2. *afficher tous les nombres entiers de* 3 *à* 13.

.....

.....

.....

.....

.....

▶ Utiliser le compteur, exemple de calcul de somme

L'intérêt d'avoir introduit un marqueur de boucle est qu'on peut l'utiliser !

> ***Exemple 1 :*** ***Calculer la somme des entiers de*** 1 ***à*** 10^6
> *Somme=0*
> ***Pour*** *i allant de* 1 *jusqu'à* 10^6+1 ***faire*** *:*
> *Somme = Somme+i*
> *affiche(Somme)*

> ***Application 2 :*** *De la même façon que précédemment proposer un algorithme qui permet de calculer :*
> 1. *le produit des entiers de* 1 *à* 100
> 2. *la somme des entiers pairs de* 1 *à* 10^5
> 3. *afficher les mille premiers termes de la suite :* $u_0 = 1$ *et* $u_{n+1} = 2u_n + 3$.

1.2 Structure en Python

♡ **Commande ♡. Boucle** *for*

```
for i in ragne(a,b):
    instruction1
    instruction2

instruction3
```

- ▷ les lignes de codes *instruction*1 et *instruction*2 (💣💣💣 **Attention ! indentation**!!!) s'exécuteront à chaque ligne de code.
- ▷ la ligne *instruction*3 (💣💣💣 **Attention ! indentation**!!!) ne 'exécutera qu'une fois la boucle terminée

💣💣💣 **Attention ! La boucle s'arrête pour $i = b - 1$!!!!!!!!!!!!!!!!!!!!!**
💣💣💣 **Attention ! à la virgule et aux deux points!!!!!!!!!!!!!!!!!!!!**

Remarque :

Un bon moyen de comprendre ce que Python réalise pendant cette boucle est d'observer la commande suivante .

```
>>> for i in range(0,7):
...     print(i)
...
...
0
1
2
3
4
5
6
```

*On verra s'afficher les valeurs que prend la variable i : 2,3,4,5,6 (**ET PAS 7!!!!**)*

| ***Application 3 :*** *Répondre aux questions des applications précédentes avec Python.*

► (Pour aller plus loin)

En réalité, l'instruction *for* permet d'écrire des boucles qui sont définies par le parcours des éléments d'un *itérable* donné. Un itérable peut être

- ▷ une chaine de caractère, délimitée par des guillements "...."
  Ex "Obelix"
- ▷ bien une liste délimitée par des crochets. Les éléments sont séparés par des virgules.
  Ex : liste de nombres [2,4,17,-1]

```
>>> nom = "Obelix"

>>> for lettre in nom :
...     print(lettre)
...
...
O
b
e
l
i
x
```

1.3 Exemples et applications

On donne deux exemples de codes. Dans chaque cas**, avant de taper les lignes de commandes**, donner ce que Python va afficher.

Application 4 :

```
for k in range(3,7):
    if k%2==0 :
        print(k, " est pair")
    elif k%3==0:
        print(k," est multiple de 3")
    else :
        print(k," n'est ni multiple de 2, ni de 3")
```

Application 5 :

```
S=0
for i in range(1,11):
    S=S+1
    print(i)
print(S)
```

▶ Applications : introduire un compteur

Il est souvent très utile en Python de compter le nombre de fois qu'une certaines instructions s'exécute ou lorsqu'une condition se réalise. Pour cela une méthode "classique" est d'introduire un **compteur,** *i.e.* une variable qui va nous permette de compter, qui s'incrémente à l'aide d'une condition if.

Exemple 2 : ***Structure pour compter***

Compt=0
Pour *i allant de a jusqu'à b* ***faire*** *:*
    Instruction 1
    Si Condition 1 :
        Compt=Compt+1

Application 6 :
▷ *Combien y a-t-il de multiple de 7 entre 0 et 5000*
▷ *Combien y a-t-il de carré parfait (∼un entier p tel qu'il existe un entier n et $p = n^2$) entre 0 et 10^3.*

2 Boucle *while*

Une limitation des boucles *for* est qu'il faut connaître et préciser à l'avance le nombre d'itérations (∼tours de boucle) que l'on souhaite effectuer. Or ce qui n'est pas toujours le cas.

Exemple 3 : On cherche le plus grand entier n tel que $2^n < 10^4$. Une idée de résolution serait de tester un à un tous les entiers de façon croissante et de vérifier si 2^n est plus grand ou plus petit que 10^4. Dès qu'on trouve un entier pour qui $2^n > 10^3 n$ on s'arrête.

C'est bien une structure itérative : on répète successivement le même bloc d'instruction (on met 2 à la puissance n et on le compare à 10^4) en augmentant à chaque tour un compteur (à chaque tour $n \to n+1$).
On veut alors effectuer la boucle itérative ***tant que*** $2^n < 10^4$.

2.1 Structure d'une boucle *while*

Une boucle *while* $\iff$ va répéter une liste d'instructions **tant que** une condition est vraie

Définition. Structure d'une boucle while

Tant que Condition est vraie **faire** :

    Instruction 1

    Instruction 2

Instruction 3

▷ La Condition est un booléen (comme dans les structures *if*) : sa valeur est soit True soit False.

▷ **Attention !** un des risques des boucles *whiles* est qu'elle peuvent tourner sans fin !!

Exemple 4 : ***Boucle sans fin ....***
S=0
i=1
Tant que *$i < 5$ est vrai* ***faire*** *:*
    S=S+i

Ici la condition sera toujours vraie : la boucle ne s'arrêtera jamais.

Dans une boucle *for* le marqueur de boucle est automatiquement **créé** et **augmenter**. Dans une boucle *while*, il faut le faire à la main si nécessaire. On peut toujours "traduire" une boucle *for* en boucle *while*, l'inverse n'est pas vrai.

Pour i allant de a à b **faire** :
    Instruction

i=a
Tant que $i < b$ est vrai **faire** :
    Instruction
    i=i+1

Méthode en DS. Utilisation d'une boucle white
Pour ne pas faire n'importe quoi avec les boucles *while* on retiendra deux règles :

1. **Condition d'utilisation** :
   une boucle *while* n'est utile que si on ne sait pas à l'avance combien de "tour de boucles" on va devoir exécuter. On préfèrera toujours utiliser une boucle for.

2. **Condition d'arrêt d'une boucle** *while* **:**
   les variables apparaissant dans la condition d'arrêt d'une boucle *while* **doivent** être modifier au sein de la boucle.

> ***Application 7 :*** *Écrire la structure qui permet de trouver le plus grand entier n tel que $2^n < 10^4$.*
>
> .....
> .....
> .....
> .....
> .....

2.2 Structure en Python

♡ **Commande ♡. Boucle** *while*

```
while condition :
    instruction1
    instruction2

instruction3
```

Pour l'exemple introductif, on a le programme suivant :

```
i=0
while 2**i<10**4:
    i=i+1

print(i)
```

> ***Application 8 :*** *On cherche l'entier n tel que $\sum_{i=1}^{n} \frac{1}{i} > 10$. Compléter le programme suivant.*
>
> ```
> Somme=0
> n=.....
> while Somme  ..........  :
>     .........
>     Somme=Somme+ .......
>
> print(n)
> ```

▶ (pour aller plus loin) Instruction *break*

Au sein des structures itérative, l'instruction *break* permet de terminer automatiquement l'exécution de la boucle. La forme générale est :

```
if condition_arret:
    break
```

Cela peut permettre de sécuriser une boucle *while* pour éviter qu'elle ne tourne indéfiniment.

```
a=2
i=0
while a<5:
    i+=1
    print(i)
    if i>100:
        print("Oula c'est beaucoup trop long")
        break
```

Attention ! il y a trois niveau d'indentation ! !

A vérifier si ça ne marche pas :

1. vérifier qu'il y a bien les " : " à la fin des en-tête
   - ▷ *for* ..... :
   - ▷ *while* ..... :
   - ▷ *if* ..... :
2. vérifier que les instructions possède la bonne indentation
3. vérifier que la boucle se termine bien

3 Exercices d'application

Exercice 1 - Comprendre des programmes :

Que va afficher Python avec les programmes suivants :

1)
```
for k in range(5):
    print(k)
    print("Bonjour")
```

2)
```
for k in range(5):
    print(k)
print("Bonjour")
```

3)
```
for k in range(5):
    print("Bonjour")
print(k)
```

4)
```
k = 0
while k < 10:
    print("Bonjour")
```

5)
```
s = 0
for i in range(4):
    for j in range(i):
        s = s + i*j
print(s)
```

Exercice 2 - Somme des entiers pairs :

On veut calculer la somme des entiers pairs entre 0 et 100. Trouver les 4 fautes.

```
S=0
for k in range(1,100):
    if k%2=0:
    S=S+k
    else :
        S=1
print(S)
```

Exercice 3 - Somme de l'inverse des carré :

Calculer la somme des inverses des carrés des entiers de deux façons différentes : avec une boucle *for* et avec une boucle *while*.

Attention ! Le programme devra demander à l(utilisateur la valeur de N souhaitée !

$$\mathcal{S}_N = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + ... + \frac{1}{N^2} = \sum_{k=1}^{N} \frac{1}{k^2}$$

Exercice 4 - Suite par récurrence :

Soit la suite (u_n) telle que $u_0 = 0$ et, pour tout entier naturel n, $u_{n+1} = 2u_n + 1$.
Considérons le programme suivant :

```
u=0
for i in range(5):
    u=2*u+1
print(u)
```

▷ i) Donner pour chaque tour de boucle les valeurs vers lesquelles pointent les variables u et i.
▷ ii) Modifier le programme pour qu'il affiche toutes les valeurs de u_n et pas seulement u_5
▷ iii) Ecrire le programme qui permet de calculer la somme des u_n entre $n = 0$ et $n = 20$

Exercice 5 - Deux suites croisées :

On veut calculer les premiers termes des suites (u_n) et (v_n) définies par $u_0 = 1$, $v_0 = 2$, et, pour tout entier naturel n, $u_{n+1} = u_n + v_n$ et $v_{n+1} = u_n - v_n$.
▷ i) Calculer à la mains les 3 premiers termes de chaque suite

▷ ii) On considère le programme suivant :

```
u=1
v=2
for i in range(3):
    u=u+v
    v=u-v
print(u,v)
```

Tester le programme et comparer les résultats avec ceux obtenus à la main. Quel est le problème ?

▷ iii) On donne le programme suivant :

```
u=1
v=2
for i in range(3):
    w=u
    u=u+v
    v=w-v
print(u,v)
```

Fonctionne-t-il mieux ? Pourquoi ?

▷ iv) A l'aide d'une affectation multiple, ré-écrire le programme précédent afin qu'il fasse deux lignes de moins.

Exercice 6 - Suite de Fibonacci :

On considère la suite (u_n) définie par $u_0 = 0$, $u_1 = 1$ et $u_{n+2} = u_{n+1} + u_n$.

▷ i) Calculer à la main u_2, u_3 et u_4

▷ ii) A l'aide de Python, calculer u_{20} (on doit trouver 10946).
Astuce : on pourra utiliser dans la boucle deux variables u et v, la première pointant vers u_n la seconde vers u_{n+1}

Exercice 7 - Exercice pèle mêle :

1. Calculer la somme de tous les multiples de 3 ou de 5 inférieurs ou égaux à 1000 (on trouvera 234168).
2. Trouver le plus petit entier naturel n tel que :

$$\sum_{k=1}^{n} \frac{1}{k} > 10$$

Réponse : $n = 12368$

```python
# Exercice_corrige.py

##Ex 2
S=0
for k in range(1,101):
    if k%2==0:
        S=S+k
    else :
        S=S
print(S)

##Ex 3
#boucle for
S=0
N=int(input("Donner une valeur de N "))
for k in range(1,N+1):
    S=S+1/k**2
print(S)

#boucle while
S=0
N=int(input("Donner une valeur de N "))
k=1
while k<N+1:
    S=S+1/k**2
    k+=1
print(S)

##Ex 4
#i)
#tour 1 : i->0 et u->1
#tour 2 : i->1 et u->3
#tour 3 : i->2 et u->7
#tour 4 : i->3 et u->15
#tour 5 : i->4 et u->31


#ii)
u=0
for i in range(5):
    u=2*u+1
    print(u)

#iii)
u=0
S=0
for i in range(21):
    u=2*u+1
    S=S+u
print(S)
```

```
49|
50| ##Ex 5
51| #i)
52| #u_0=1 et v_0=2 ; u_1=3 et v_1=-1 ; u_2=2 et v_2=4
53|
54| #ii)
55| # l'affectation de u ne pose pas de problème MAIS sa valeur
est changée. Ainsi, lors de l'affectation
56| # de v, la valeur de u utilisée est la nouvelle. Ainsi dans
ce programme v_(n+1)=u_(n+1)-v_n
57|
58| #iii)
59| # il fonctionne car on sauvegarde l'ancienne valeur de u à
l'aide de la varible w qui est ensuite utilisée
60| # pour affecter à v sa nouvelle valeur
61|
62| #iv)
63| u,v=1,2
64| for i range(3):
65|     u,v=u+v,u-v
66| print(u,v)
67|
68| ##Ex 6
69| u=1
70| v=1
71| for k in range(1,21):
72|     u,v=v,u+v
73| print(u)
74|
75| ##Ex 7
76| #1)
77| S=0
78| for k in range(0,1001):
79|     if k%3==0:
80|         S=S+k
81|     elif k%5==0:
82|         S=S+k
83| print(S)
84|
85| #2)
86| S=0
87| k=1
88| while S<10:
89|     S=S+1/k
90|     k=k+1
91| print(k)
```