

BOUCLES

I. Boucle conditionnelle

1) Syntaxe

Une boucle conditionnelle permet d'exécuter un bloc d'instructions tant qu'une condition est vérifiée.

```
while <condition> :  
    <instruction 1>  
    <instruction 2>  
    ...  
    <instruction n>  
<suite> #ne fait pas partie du bloc
```

Le bloc d'instruction est appelé *corps de la boucle*; chaque exécution du corps de boucle est appelée une *itération*.

Si le booléen renvoyé par la condition vaut `True`, la totalité du corps de la boucle est exécutée, et ce tant que la condition est vérifiée.

Par exemple, le programme :

```
c = 5  
while c > 0:  
    c -= 1  
    print("c vaut ",c)
```

affiche :

```
c vaut 4  
c vaut 3  
c vaut 2  
c vaut 1  
c vaut 0
```

2) Terminaison

Pour s'assurer qu'un programme contenant une boucle `while` termine bien, il suffit de trouver une quantité qui vérifie deux propriétés :

- être un entier positif,
- et décroître strictement à chaque itération.

En effet, il n'existe pas de suite infinie strictement décroissante d'entiers naturels, donc le nombre d'itérations sera nécessairement fini.

Par exemple, pour le programme suivant :

```
c = 5  
while c > 0:  
    c -= 1  
    print("c vaut ",c)
```

la variable `c` est (et reste) un entier, reste positive, et décroît strictement à chaque itération. On peut donc affirmer que le programme termine.

La quantité qui décroît peut ne pas dépendre d'une seule variable mais s'exprimer en fonction de plusieurs variables.

Par exemple, dans le programme suivant :

```
a, b = 20, 10  
while a > 0 and b > 0:  
    if b % 2 == 0:  
        a = a - 2  
        b = b + 1  
    else:  
        a = a + 1  
        b = b - 3
```

ni `a` ni `b` ne décroît strictement à chaque itération ; en revanche, `a + b` est bien un entier positif qui décroît strictement à chaque itération.

II. Boucle inconditionnelle

Il s'agit de boucles énumératives : la machine répète une série d'instructions pour chaque valeur prise dans un objet « itérable ».

```
for <valeur> in <iterable>:
    <instruction 1>
    <instruction 2>
    ...
    <instruction n>
<suite> #ne fait pas partie du bloc
```

Comme précédemment, c'est le niveau d'indentation qui définit le corps de boucle ; le retour au niveau d'indentation de `for` marque la fin de la boucle.

La fonction `range` permet de faire varier un indice d'une valeur initiale (incluse) à une valeur finale (exclue) ; dans le cas où on n'indique qu'une seule valeur, la valeur initiale est 0 :

<pre>>>> for i in range(5): ... print(i) ... 0 1 2 3 4</pre>	<pre>>>> for i in range(8,12): ... print(i) ... 8 9 10 11</pre>
---	--

Il est possible d'ajouter un troisième argument pour préciser le pas :

<pre>>>> for i in range(15,9,-2): ... print(i) ... 15 13 11</pre>	<pre>>>> for i in range(15,8,-2): ... print(i) ... 15 13 11 9</pre>
--	--

On peut aussi utiliser d'autres objets itérables, comme par exemple les listes :

```
>>> for i in [1,5,1,13]:
...     print(2*i)
...
2
10
2
26
```

III. Invariant de boucle

Lorsqu'on écrit un programme (ou une fonction), on préférerait être sûr qu'il est correct, c'est-à-dire qu'il calcule bien ce qu'on souhaite.

Pour démontrer qu'un programme contenant une boucle est correct, on utilise un invariant de boucle, c'est-à-dire une propriété qui :

- est vérifiée avant d'entrer dans la boucle ;
- si elle est vérifiée avant une itération, est vérifiée après celle-ci ;
- permet de montrer que le programme est correct si elle est vérifiée en sortie de boucle.

Exemple

On considère le programme suivant :

```

n = int(input("Entrer un entier positif :"))
a = int(input("Entrer un nombre :"))
r = 1
for i in range(n):
    r = r*a
print(r)

```

Notons r_k la valeur de la variable r après l'exécution de la k -ième itération, et montrons que $r_k = a^k$.

- Avant l'entrée dans la boucle, $r_0 = 1 = a^0$.
- Si $r_k = a^k$, en effectuant une itération supplémentaire, on a

$$r_{k+1} = r_k \times a = a^k \times a = a^{k+1}$$

La boucle effectue n itérations, donc en sortie de boucle, $r = a^n$.

Par conséquent, le programme affiche la valeur de a^n .

Exemple

On suppose que la variable n est un entier positif.

```

r = 1
while n > 0:
    r = 2*r
    n = n - 1

```

Que vaut r à la sortie de la boucle ?

Notons r_k et n_k les valeurs respectives de des variables r et n après k itérations.

- Avant l'entrée dans la boucle, $r_0 = 1$ et $n_0 = n$.
- $r_{k+1} = 2r_k$ et $n_{k+1} = n_k - 1$.

Montrons qu'après k itérations, $r_k = 2^{n-n_k}$ et n_k est un entier positif :

- Cette propriété est vérifiée pour $k = 0$: $n_0 = n$, qui est un entier positif, et $r_0 = 1 = 2^{n-n_0}$;
- Supposons que $r_k = 2^{n-n_k}$ et que n_k est un entier positif. En effectuant une itération supplémentaire, on a :

$$r_{k+1} = 2r_k = 2^{n-n_k+1} = 2^{n-(n_k-1)} = 2^{n-n_{k+1}}$$

De plus, puisqu'on a effectué une itération supplémentaire, la condition d'entrée de boucle était vérifiée, donc $n_k > 0$. Par conséquent, n_k est un entier supérieur ou égal à 1, donc $n_{k+1} = n_k - 1$ est un entier positif.

En sortie de boucle, $n_k > 0$ n'est plus vérifiée : on a donc $n_k \leq 0$. Comme n_k est un entier positif $n_k = 0$, donc $r_k = 2^{n-n_k} = 2^n$.

Exercices

EXERCICE 1

Écrire un programme qui demande à l'utilisateur un entier naturel n et qui affiche $n!$

EXERCICE 2

Les programmes qui suivent terminent-ils pour toutes les valeurs entières de a ?

```

c = 0
if a != 0:
    while a > 0:
        c = c + 1
        a = a - 2

```

```

c = 0
if a != 0:
    while a != 0:
        c = c + 1
        a = a - 2

```

EXERCICE 3

Écrire un programme qui détermine le nombre de bits de l'écriture binaire d'un entier naturel non nul n donné (c'est-à-dire l'entier b tel que $2^{b-1} \leq n < 2^b$)

EXERCICE 4

Écrire un programme demandant deux entiers naturels a et b et calculant le quotient et le reste de la division euclidienne de a par b en utilisant l'algorithme par soustractions.

Montrer que ce programme termine.

EXERCICE 5

n désigne un entier positif.

1. Montrer que le programme suivant termine :

```
x = n
y = n
while y > 0:
    x = x + 2
    y = y - 1
print(x)
```

2. Que calcule-t-il ? Le prouver !

EXERCICE 6

On suppose que n est un entier positif et a un entier. On exécute le programme suivant :

```
r = 1
while n > 0:
    if n % 2 == 1:
        r = r * a
    a = a * a
    n = n // 2
```

On note r_k , n_k et a_k les valeurs respectives de $\mathbf{r}$, $\mathbf{n}$, $\mathbf{a}$, après k itérations.

1. Montrer que $r_k a_k^{n_k} = a_0^{n_0}$. Que vaut $\mathbf{r}$ en sortie de boucle ?
2. Montrer que ce programme termine.