Paradigme itératif : les boucles

1 Les boucles

Une instruction qui doit être répétée 100 fois n'est pas écrite 100 fois dans un programme. Pour cela, on utilise une boucle.

La structure de boucle est appelée une structure **itérative** et chaque répétition est une **itération**.

La boucle est utilisée pour répéter autant de fois que nécessaire une ou plusieurs instructions.

Il existe deux types de **boucles** en Python (et beaucoup d'autres langages) :

- La boucle **bornée** `for` qui répète **un nombre fini de fois** les instructions contenues dans la boucle.
- La boucle **non bornée** `while` qui répète les instructions contenues dans la boucle **tant que la condition donnée est vraie**.

Remarque

1. Les instructions contenues dans une **boucle** constituent un mini programme qui sera exécuté autant de fois que la boucle le demande.
2. Une **boucle** peut contenir d'autres **boucles**. On parle alors de boucles imbriquées.

2 La boucle for

2.1 Présentation générale

La boucle **for** prélève dans l'ordre les valeurs présentes dans un ensemble de valeurs. La structure est la suivante :

```
for clef in valeurs:
    instruction 1
    instruction 2
    etc
# on désindente, fin de la boucle
```

La **clef** utilisée dans la boucle `for` est une **variable** qui prend une nouvelle valeur à chaque **itération**.

Exemple

Supposons la liste de valeurs "mot", 13, "deux", 6, 125 et "quatre" que l'on souhaite afficher.

```
for c in ["mot",13,"deux",6,125,"quatre"]:
    print(c)
```

On obtient l'affichage :

```
mot
13
deux
6
125
```

```
quatre
```

Remarque

- La clef peut être une lettre, un mot ou même le caractère souligné _.
- Si la liste des valeurs est composée de nombres, alors il est possible d'effectuer des calculs avec la clef.
- Une chaine de caractères constitue un liste de valeurs, chaque valeur étant une lettre. La clef prend alors pour valeur chacune des lettres, y compris les espaces et les symboles de ponctuation).

Exercice

Afficher chaque lettre du mot **boucle** avec une boucle for.

2.2 Itération avec la fonction RANGE

Dans l'exemple précédent, la liste de valeurs est dite **itérable** ce qui signifie que l'on peut parcourir ses éléments un à un.

Si on veut afficher les 6 premiers nombres entiers non nuls, on peut écrire la boucle suivante:

```
for i in [1,2,3,4,5,6]:
    print(i)
```

Seulement, cette méthode sera très longue si on veut afficher les 100 premiers nombres entiers !

La fonction **range** permet de construire cette liste de nombres en lui donnant les bons arguments.

- `range(n)` renvoie les nombres entiers positifs de 0 jusqu'à $n-1$.
- `range(p,n)` renvoie les nombres entiers positifs de p jusqu'à $n-1$.
- `range(p,n,k)` renvoie les nombres entiers positifs de p, $p+k$, $p+2k$, ... jusqu'à $n-1$ au plus.

Exemple

- `range(8)` renvoie les nombres entiers 0, 1, 2, 3, 4, 5, 6, 7.
- `range(3,8)` renvoie les nombres entiers 3, 4, 5, 6, 7.
- `range(3,8,2)` renvoie les nombres entiers 3, 5, 7.

Remarque

L'exécution de `range(3,8,2)` dans l'interpréteur Python n'affiche pas les valeurs. Pour les afficher, il faut une boucle avec un **print**.

Exercice

1. Afficher les 6 premiers nombres entiers non nuls avec une boucle `for`.
2. Afficher les 100 premiers nombres entiers non nuls.

3 La boucle while

Il est parfois utile d'itérer des instructions tant qu'une **condition** est vraie. L'itération s'arrête dès que la condition est fausse. La syntaxe d'une boucle `while` est la suivante:

```
while condition vraie:
    instruction 1
    instruction 2
```

```
    etc
# on désindente, fin de la boucle
```

Exemple

On demande la saisie d'un nombre entier positif et on veut afficher les nombres pairs positifs inférieurs au nombre saisi.

```
n = int(input("saisir un nombre entier:"))
i = 2
while i <= n:
    print(i)
    i = i + 2
```

Si on saisit le nombre 9, on obtient l'affichage:

```
2
4
6
8
```

Remarque

- la condition à vérifier est un test booléen. Tant que celui-ci est vrai, l'itération se poursuit.
- si une condition est toujours vraie, la boucle ne s'arrête pas, elle est infinie.
- dès que la condition est fausse, la boucle s'arrête !
- si la condition devient fausse pendant l'itération, celle-ci se poursuit jusqu'à la fin de la boucle.

```
n = int(input("saisir un nombre entier:"))
i = 2
while i <= n:
    i = i + 2
    print(i)
```

Si on saisit le nombre 9, on obtient l'affichage:

```
4
6
8
10
```

Lorsque la variable `i` prend la valeur 8, dans la boucle la variable `i` prend la valeur 10 et l'affichage se réalise. C'est à l'itération suivante qu'elle s'arrête puisque `i` est alors supérieur à 9.