Chapitre 4

Les boucles

Il est très fréquent que certaines parties d'un algorithme soient répétitives. Pour les implémenter dans un langage de programmation, on a besoin de structures spécifiques : les structures de boucles.

I La boucle `for`

La boucle `for`[1], aussi appelée *boucle bornée* est utilisée lorsque le nombre d'itérations est connu à l'avance. On l'utilise pour parcourir une structure existante, comme un tableau ou une chaîne de caractères, ou encore avec un indice qui compte le nombre de tours de boucles (on parle d'itérations).

I.1 Parcours d'un tableau ou d'une chaîne de caractères

On peut facilement parcourir une chaîne de caractère avec une boucle `for`. Ceci se réalise de la manière suivante.

```
for c in "bonjour":
    print(c)
```

```
b
o
n
j
o
u
r
```

De la même manière, on parcours les éléments d'un tableau.

```
for x in [6, 8, 9, 5, -12]:
    print(x)
```

```
6
8
9
5
-12
```

Ceci est particulièrement intéressant lorsqu'il s'agit de faire un traitement sur les éléments d'un tableau. Par exemple, si l'on souhaite calculer la somme des nombres contenus dans un tableau, on pourra s'y prendre ainsi.

1. « for » signifie « pour » en anglais.

```
tab = [6, 8, 9, 5, -12]
somme = 0

for x in tab:
    somme = somme + x

print("La somme des éléments de", tab, "vaut", somme, ".")
```

```
La somme des éléments de [6, 8, 9, 5, -12] vaut 16 .
```

Attention

Encore une fois, l'indentation aura une importance capitale dans une boucle `for`. Reprenons le programme précédent et supposons que l'on souhaite afficher `somme` à chaque tour de boucle. On peut alors procéder ainsi

```
for x in tab:
    somme = somme + x
    print(somme)
```

Alors que si l'on écrit ceci,

```
for x in tab:
    somme = somme + x
print(somme)
```

`somme` ne sera affichée qu'une fois la boucle terminée.

Astuce

Il est possible d'utiliser une structure conditionnelle dans une boucle `for`. On dit alors qu'on *imbrique* un `if` dans un `for`. L'indentation est encore primordiale dans ce cas. Voici un exemple.

```
for c in "zoubidou":
    if c == 'o':
        print("J'ai trouvé un 'o' !")
```

```
J'ai trouvé un 'o' !
J'ai trouvé un 'o' !
```

Exercice

Exercice 25

Écrire un programme Python qui demande une phrase à l'utilisateur et retourne le nombre de caractères `'e'` qu'elle contient.

I.2 Itération sur les entiers

Une autre manière d'utiliser la boucle `for` est de parcourir tous les entiers d'un intervalle. Pour cela, on utilise la fonction `range(a, b)`, comme ceci.

```
for i in range(4, 13):
    print(i)
```

```
4
5
6
7
8
```

```
9
10
11
12
```

`range` signifie « intervalle » en anglais. `range(a, b)` peut être vu comme un tableau contenant tous les entiers de l'intervalle $[a\,;b[$.

Attention

Dans `range(a, b)`, la valeur `b` est exclue ! Si l'on veut l'inclure, il faut alors écrire `range(a, b + 1)`.

Astuce

Si l'on souhaite commencer à `0`, au lieu d'écrire `range(0, n)`, on peut écrire plus simplement `range(n)`.

Astuce

Il est possible de parcourir les entiers de deux en deux ou de trois en trois voir même de -1 en -1, c'est à dire à l'envers ! Pour cela, on peut passer la valeur en troisième paramètre de la fonction `range`, comme ceci.

```
for k in range(1, 10, 2):
    print(k)
```

```
1
3
5
7
9
```

```
for k in range(5, 0, -1):
    print(k)
```

```
5
4
3
2
1
```

Info

Dans une instruction comme `for i in range(...)`, on dit que la variable `i` est *l'indice de la boucle*.

Exercice

Exercice 26

Écrire un programme qui demande à l'utilisateur un entier `n` supérieur à 1 et qui affiche la somme des entiers de 1 à `n`.

Exercice

Exercice 27

Modifier le programme précédent pour qu'il affiche non plus la somme mais le produit.

Info

Il est tout à fait possible de ne pas utiliser l'indice de la boucle, mais il convient tout de même de lui donner un nom.

```
for k in range(6):
    print("C'est moi ou je me répète ?")
```

```
C'est moi ou je me répète ?
C'est moi ou je me répète ?
C'est moi ou je me répète ?
C'est moi ou je me répète ?
C'est moi ou je me répète ?
C'est moi ou je me répète ?
```

II La boucle `while`

La boucle `while`[2], aussi appelée *boucle non bornée* est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. On l'utilise souvent lorsque le critère d'arrêt de la boucle n'est pas connu à l'avance mais dépend justement de l'exécution de la boucle.

II.1 S'arrêter, oui, mais à une condition !

La boucle `while` utilise une expression booléenne comme critère d'arrêt. Ceci signifie qu'à chaque tour de boucle, on va tester cette condition, si elle est vraie, on repart pour au moins un tour de boucle, si elle est fausse, on sort de la boucle.

Par exemple, pour trouver la première puissance de 2 supérieur ou égale à 1000, on a écrit le programme suivant.

```
puissance = 1
while puissance < 1000:
    puissance = puissance * 2
    print("Valeur intermédiaire de puissance :", puissance)
print("Valeur finale de puissance :", puissance)
```

```
Valeur intermédiaire de puissance : 2
Valeur intermédiaire de puissance : 4
Valeur intermédiaire de puissance : 8
Valeur intermédiaire de puissance : 16
Valeur intermédiaire de puissance : 32
Valeur intermédiaire de puissance : 64
Valeur intermédiaire de puissance : 128
Valeur intermédiaire de puissance : 256
Valeur intermédiaire de puissance : 512
Valeur intermédiaire de puissance : 1024
Valeur finale de puissance : 1024
```

Dans cet exemple, on remarque que, comme pour la boucle `for`, l'indentation est primordiale ! Les deux dernière lignes n'ont pas la même indentation : la première est décalée, elle est exécutée à chaque tour de boucle alors que la seconde n'est exécutée qu'une fois la boucle terminée.

Ici, le critère d'arrêt de la boucle est l'expression « `puissance < 1000` ». À chaque fin de boucle, cette expression est évaluée : tant qu'elle est vraie, on ré-exécute le corps de la boucle.

Exercice 28

Modifier le programme précédent pour trouver le premier entier naturel n tel que la somme des entiers de 0 à n est supérieure à 5000.

2. « while » signifie « tant que » en anglais.

Savoir-faire

Choisir entre une boucle `for` et une boucle `while`

Si on connaît à l'avance le nombre de répétitions à effectuer, ou que l'on veut parcourir une structure dont le nombre d'éléments est fixe (un tableau ou une chaîne de caractères par exemple), c'est la boucle **`for`** qu'il faut choisir.
En revanche, si la décision d'arrêter la boucle ne peut s'exprimer que par un test, c'est la boucle **`while`** qui convient.

Exercice

Exercice 29

Quelle boucle est adaptée à l'écriture de programmes traitant les problèmes suivants :

- le calcul du total à payer à une caisse enregistreuse,
- la recherche du jour le plus pluvieux d'une année,
- le calcul du périmètre d'un polygone,
- le calcul de la durée d'une émission de radio, connaissant ses horaires de début et de fin ?

Info

Un langage de programmation qui ne permettrait que d'écrire :

- des expressions,
- des affectations de variables,
- des séquences d'instructions,
- des structures conditionnelles,
- des boucle non bornées (**`while`**)

est dit *complet*. Ceci signifie que tous les programmes que l'on peut imaginer peuvent être écrits dans ce langage. Autrement dit, tout ce qui n'est pas dans ce langage (comme la boucle **`for`** ou les fonctions que nous étudierons au chapitre suivant) n'est qu'une syntaxes pour faciliter l'écriture et la lecture des programmes [3].
De même que tous les objets qui nous entourent sont formés de trois types de particules : les protons, les neutrons et les électrons, que tous les textes que nous lisons sont formés de vingt-six lettres et de quelques signes de ponctuation, que toutes les musiques que nous entendons peuvent être exprimées à l'aide de douze notes, tous les programmes que nous utilisons peuvent ultimement être exprimés avec ces quatre instructions : l'affectation, la séquence, le test et la boucle.

Astuce

Parfois, il n'est pas possible d'exprimer la condition pour rester dans la boucle de manière simple et il est plus pratique (ou plus lisible ou plus efficace) d'utiliser une autre condition dans le corps de la boucle. Pour cela, on utilise l'instruction **`break`**, souvent dans un **`if`**.
Par exemple, dans l'exemple suivant, on demande à l'utilisateur le prix des articles, tant que leur somme n'a pas atteint les 100 euros. On souhaite aussi laisser la possibilité à l'utilisateur d'intérrompre à tout moment avec la touche `'q'`.

```
somme = 0

while somme < 100:
    reponse = input("Entrer le prix suivant ou 'q' pour quitter :")
    if reponse == 'q':
        break
    else:
        somme = somme + float(reponse)

print("La somme atteinte est ", somme)
```

3. On parle de « sucres syntaxiques ».

```
Entrer le prix de l'article suivant ou 'q' pour quitter : 10
Entrer le prix de l'article suivant ou 'q' pour quitter : 11
Entrer le prix de l'article suivant ou 'q' pour quitter : 23.90
Entrer le prix de l'article suivant ou 'q' pour quitter : q
La somme atteinte est  44.9
```

Astuce dans l'astuce : l'instruction **break** peut aussi être utilisée dans une boucle **for**.

II.2 La boucle `for`, cas particulier de la boucle `while`

Il est toujours possible de remplacer une boucle **for** par une boucle **while**. En effet, par exemple, la séquence

```
for i in range(n):
    {une séquence d'instructions}
```

peut être remplacée par [4]

```
i = 0
while i < n:
    {une séquence d'instructions}
    i = i + 1
```

Astuce

Avec les boucles **while**, on a souvent besoin d'ajouter des valeurs à des variables. Ceci peut se faire de manière compacte avec le symbole « `+=` ». En effet, l'instruction « `x = x + 1` » s'écrit aussi « `x += 1` ». On dispose également des instructions « `-=` », « `*=` » et « `/=` ».

Exercice

Exercice 30

Écrire un programme qui affiche tous les nombres impairs de 0 à 100, l'un avec une boucle **for**, l'autre avec une boucle **while**.

II.3 Attention aux boucles infinies !

Avec une boucle **while**, il est possible que le programme ne s'arrête jamais, on parle de *non-terminaison*. Par exemple, la boucle

```
while i != 0:
    i = i - 1
```

termine seulement si la variable `i` est positive ou nul avant d'entrer dans la boucle. Il est parfois utile de vouloir créer une boucle infinie. Pour cela, on utilise l'instruction « **while True:** ». On sort alors de la boucle avec l'instruction **break**.

Info

On peut aussi créer une boucle infinie qui ne fait rien avec l'instruction suivante.

```
while True:
    pass
```

Exercice

Exercice 31

La boucle ci-dessous se termine-t-elle ?

4. Attention tout de même à la différence de valeur de `i` à la sortie des deux boucles. `i` vaut `n - 1` après la boucle **for** et `n` après le **while**.

```
valeur = 0
while valeur != 17:
    valeur += 2
```

Exercices de fin de chapitre

Exercice

Exercice 32 (tiré du sujet 0 de NSI)

Quelle est la valeur affichée à l'exécution du programme Python suivant ?

```
x = 1
for i in range(10):
    x = x * 2
print(x)
```

(a) 2
(b) 1024
(c) 20000000000
(d) 2048

Exercice

Exercice 33 (tiré du sujet 0 de NSI)

Que peut-on dire du programme Python suivant de calcul sur les nombres flottants ?

```
x = 1.0
while x != 0.0:
    x = x - 0.1
```

(a) L'exécution peut ne pas s'arrêter, si la variable `x` n'est jamais égale à `0.0`.
(b) À la fin de l'exécution, la variable `x` vaut `-0.000001`.
(c) À la fin de l'exécution, la variable `x` vaut `0.000001`.
(d) L'exécution s'arrête sur une erreur `FloatingPointError`.

Exercice

Exercice 34

Reprendre le programme de la bataille navale puis le modifier pour qu'il permettre à l'utilisateur de jouer tant qu'il n'a pas gagné.

Exercice

Exercice 35 Turtle 1

Après avoir lu la section II du chapitre 6 qui concerne l'introduction au module graphique Turtle, écrire un programme qui demande à l'utilisateur un nombre de côté et une longueur et affiche le polygone régulier correspondant.
Avec quels paramètres a-t-on l'impression d'avoir tracé un cercle ?

Exercice

Exercice 36 Méthode de Héron

En mathématiques, la méthode de Héron permet d'approcher la valeur d'une racine carrée. À chaque étape, on calcule une nouvelle valeur, plus proche que la précédente du résultat souhaité. Par exemple, dans le cas de l'approximation de $\sqrt{2}$, si l'on a déjà une valeur approchée, x, la valeur suivante, x', se calcule comme ceci :

$$x' = \frac{1}{2}\left(x + \frac{2}{x}\right).$$

En partant d'une valeur initiale égale à 2, programmer la méthode de Héron pour le calcul de la valeur approchée $\sqrt{2}$. Le programme s'arrêtera lorsque le carré de la valeur trouvée est proche

de 2 à 10^{-9} près.

Exercice

Exercice 37 Turtle 2

Après avoir lu la section II du chapitre 6 qui concerne l'introduction au module graphique Turtle, reproduire la forme ci-dessous.