Thème 6 - Langages et programmation

03

Les Boucles (FOR et WHILE) en Python



Les énumérables ou itérables

En mathématiques, on dit qu'un ensemble est *dénombrable* lorsqu'on peut associer à chaque élément de l'ensemble un nombre (traditionnellement 1, 2, 3 ...)

- les voitures qui roulent sur l'autoroute sont dénombrables.
- l'eau qui coule d'un robinet n'est pas dénombrable.

En informatique, il existe un concept similaire qui va désigner les objets que l'on peut **énumérer**, c'est-à-dire les décomposer en une succession ordonnée d'éléments. On les appelle les **énumérables** ou les **itérables** (Python utilise le mot anglais <u>iterable</u>).

- la variable NSI (qui est de type String) est énumérable : on peut la décomposer en N, S, I.
- la variable [4, 3, 17] (qui est de type List) est énumérable : on peut la décomposer en 4, 3, 17.
- la variable 5 (qui est de type Int) n'est PAS énumérable : on ne peut pas la décomposer.

1. • Itérer sur les itérables : la boucle for

Itérer sur une chaîne de caractères

Il existe donc une instruction permettant de faire une (ou plusieurs) action(s) à chaque itération sur un élément énumérable.

Exemple:

Le programme suivant :

```
Script Python

for k in 'NSI':
print(k)
```

va donner ceci:

```
Script Python

N
S
I
```

Analyse grâce à PythonTutor Étudions, grâce à PythonTutor, le détail de cette exécution.

Cliquez sur Next et observez bien l'évolution de la variable k.

```
🗋 Algo
from tutor import tutor
for k in 'NSI':
     print(k)
tutor()
🗋 Algo
N
S
Ι
                                                      Print output (drag lower right corner to resize)
             ▶ 1 for k in 'NSI':
                        print(k)
               2
ine that has just executed
next line to execute
                                                                        Objects
                                                      Frames
    << First
               < Back | Step 1 of 7 | Forward >
                      Last >>
```

La variable k prend donc **successivement** toutes les lettre de la chaîne de caractère "NSI".

Pour chaque valeur de k, la ou les instruction(s) situées de manière indentée sous la ligne du for seront exécutées.

Ici, il y a simplement un print(k), donc chaque lettre de "NSI" s'affiche l'une après l'autre.

```
Exercice 1:
```

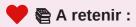
Que donne le script suivant ?

```
Script Python

for m in 'NASA':
    print("bonjour")
```

Dans cet exercice, la variable de boucle m est muette : elle n'apparaît dans les instructions indentées sous le for .

La variable m prend successivement les valeurs 'N, 'A', 'S' et 'A', mais on ne le voit pas.



△ Comment éviter les erreurs classiques

Quand vous écrivez une boucle for , veillez bien à :

- finir la ligne du for par les deux points :
- indenter sous le for les instructions qui doivent être répétées.

Itérer sur une liste

Exemple: Le programme suivant :

```
Script Python

for jour in ["lundi", "mardi", "mercredi", "jeudi", "vendredi"]:
    print(f"je vais au lycée le {jour}")
```

va donner ceci:

```
Script Python

je vais au lycée le lundi
je vais au lycée le mardi
je vais au lycée le mercredi
je vais au lycée le jeudi
je vais au lycée le vendredi
```

```
from tutor import tutor
for jour in ["lundi", "mardi", "mercredi", "jeudi", "vendredi"]:
    print(f"je vais au lycée le {jour}")
tutor()
🗎 Algo
je vais au lycée le lundi
je vais au lycée le mardi
je vais au lycée le mercredi
je vais au lycée le jeudi
je vais au lycée le vendredi
                                                                                         Print output (drag
→ 1 for jour in ["lundi", "mardi", "mercredi", "jeudi", "vendredi"]:
   2
           print(f"je vais au lycée le {jour}")
line that has just executed
next line to execute
                                                                                         Frames
                            < Back | Step 1 of 11 | Forward >
                   << First
                                                            Last >>
```

△ △ Attention:

Très souvent, l'objet itérable que la boucle va parcourir aura été **au préalable** stocké dans une variable :

Exemple: Le programme précédent est équivalent à :

```
semaine = ["lundi", "mardi", "mercredi", "jeudi", "vendredi"]
for jour in semaine:
    print("je vais au lycée le", jour)
```

Notez l'importance d'avoir choisi des noms de variables explicites : ils aident grandement à la lisibilité du code.

Trailer : Dans le cours spécifique sur les listes, nous verrons une toute autre manière de parcourir une liste.

```
je vais au lycée le lundi
je vais au lycée le mardi
je vais au lycée le mercredi
je vais au lycée le jeudi
je vais au lycée le vendredi
```

2. ◆ Comment répéter n fois la même action ?

Comment répéter 10 fois la phrase "Hasta la vista, baby" ?

```
Note:

L'ensemble range:

Le programme suivant:
```

```
for i in range(5):
    print("Hasta la vista, baby")
```

va donner ceci:

```
Hasta la vista, baby
```

Là encore, le i est une variable muette.

```
for i in range(5):
   print("Hasta la vista, baby")
```

```
Hasta la vista, baby
```

Utilisation minimale de l'objet range()

Syntaxe minimale de range() : Le programme suivant :

```
Script Python

for k in range(4):
    print(k)
```

va donner ceci:

```
Script Python

0
1
2
3
```

Interprétation : faire parcourir à une variable k l'ensemble range(n) va faire prendre à k les valeurs 0, 1, 2, ..., n-1.

Utilisation avancée de l'objet range()

Syntaxe complète de range() : Le programme suivant :

```
Script Python

for k in range(5, 15, 2):
    print(k)
```

va donner ceci:

```
Script Python
5
7
9
11
13
```

Interprétation : faire parcourir à k l'ensemble range(start, stop, step) fait :

- démarrer k à la valeur start
- progresser k de step en step
- tant que k est rictement inférieur stop

Remarques:

- si le step est omis, il vaut 1 par défaut.
- l'objet range(5) n'est pas «techniquement» égal à la liste [0, 1, 2, 3, 4], car ce n'est pas un objet de type List:

➢ 및 Exercice 1:

Faire afficher les séries de nombres suivantes.

On utilisera la syntaxe print(k, end = ' ')) pour afficher les nombres horizontalement.

- A. 0 1 2 3 4 5
- B. 10 11 12 13 14 15
- C. 3 6 9 12
- D. 10 9 8 7 6 5 4 3 2 1 0

3. • Une situation classique : la double boucle

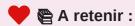
Il est très souvent utile d'imbriquer une boucle dans une autre, notamment lors du parcours de tous les pixels d'une image. Prenons pour l'instant un exemple numérique.

Exercice 2 :

Ecrire un script permettant d'afficher

% Script Python

- 1 * 1 = 1
- 1 * 2 = 2
- 1 * 3 = 3
- 2 * 1 = 2
- 2 * 2 = 4
- 2 * 3 = 6
- 3 * 1 = 3
- 3 * 2 = 6
- 3 * 3 = 9 4 * 1 = 4
- _ _ _ _
- 4 * 2 = 8
- 4 * 3 = 12



- La boucle for s'utilise lorsque :
 - on veut parcourir un à un les éléments d'un objet itérable (une chaîne de caractère, une liste...)
 - on veut répéter une action un nombre de fois **connu à l'avance**. On parle de boucle **bornée**.
- Les instructions répétées peuvent mais ce n'est pas obligatoire faire appel à la variable de boucle, mais il ne faut pas que ces instructions la modifient.
- Ne pas oublier les : et l'indentation !
- range(n) génère une séquence de n nombres entiers: on s'en servira dès qu'on aura besoin de répéter n fois des instructions.

Exercices

Exercice 3 :

Dans l'extrait de code suivant:

- chaine est une variable initialisée avec un str vide : "";
- on veut qu'en sortie de programme cette variable contienne la valeur 'bravo'.

L'idée est d'ajouter une par une les lettres à la variable chaine.

Compléter le code.

Cette variable chaine est appelée un accumulateur.

▶ ■ Exercice 4:

En Python, la fonction ord renvoie le code Unicode d'un caractère et la fonction chr le contraire: elle renvoie le caractère correspondant à un code Unicode.

Par exemple:

& Script Python

```
>>> ord('a')
97
>>> chr(97)
'a'
```

Voici une liste contenant les codes Unicode des lettres d'un mot secret...

À vous d'écrire un programme où en sortie, la variable mot_secret contiendra la chaîne de caractères de ce mot.

% Script Python

```
mystere = [111, 107, 44, 32, 98, 105, 101, 110, 32, 106, 111, 117, 233]
mot_secret = ""
```

ok, bien joué

Exercice 5 :

On souhaite calculer la somme des 1000 premiers nombres entiers naturels, c'est-à-dire:

$$1+2+3+4+5+\cdots+999+1000$$

Écrire un programme avec une variable somme accumulateur (comme à l'exercice 3) qui contiendra la valeur souhaitée en fin de programme.

Exercice 6 :

Ecrire un programme permettant de calculer

$$1 \times 2 \times 3 \times \dots 99 \times 100$$
.



Proposer un code qui écrit la **table de multiplication** de 7, de 8 et de 9.

Boucle WHILE

1. ◆ Premiers exemples

À la différence essentielle des boucles for , dont on peut savoir à l'avance combien de fois elles vont être exécutées, les boucles while sont des boucles dont on ne sort que lorsqu'une condition n'est plus satisfaite.

Avec donc le risque de rester infiniment bloqué à l'intérieur!

```
    Algo

a = 0
while a < 3:
    print("ok")
    a = a + 1
print("fini")

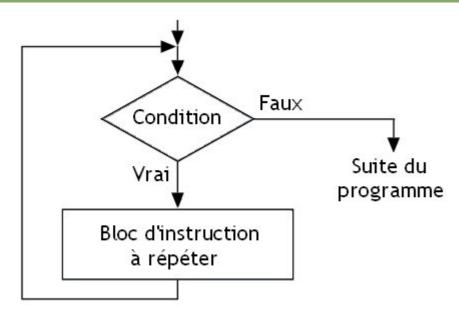
    Algo
</pre>
```

```
Ok
Ok
Ok
Tini
```

2. ◆ Syntaxe générale



Écriture d'une boucle while ```python while condition: instruction1 instruction2 ... instructionN



La condition

La condition est un booléen, c'est-à-dire une expression que Python évaluera à True ou à False.

Exemple de booléens résultant d'une évaluation :

```
Script Python

>>> 1 < 3
True
>>> 5 > 7
False
>>> a = 10
>>> a > 8
True
```

3. ◆ Les pièges ...

piège n°1: ne JAMAIS SORTIR de la boucle

Exemple: Le programme suivant:

```
% Script Python

a = 0
while a < 3:
    print("ok")
    a = a + 1
    a = a * 0
print("ce texte ne s'écrira jamais")</pre>
```

va écrire une suite infinie de ok et ne jamais s'arrêter

piège n°2 : ne JAMAIS ENTRER dans la boucle

Exemple: Le programme suivant:

% Script Python

```
a = 0
while a > 10:
    print("ce texte non plus ne s'écrira jamais")
    a = a + 1
print("fini")
```

va écrire fini et s'arrêter.

```
a = 0
while a > 10:
    print("ce texte non plus ne s'écrira jamais")
    a = a + 1
print("fini")
```

fini

!!! exo " \sqsubseteq Exercice 8 : Trouver le plus petit nombre entier n tel que 2^n soit supérieur à 1 milliard.

La boucle infinie a été présentée comme un danger qu'il faut éviter.

Pourtant, dans quelques situations, il est d'usage d'enfermer volontairement l'utilisateur dans une boucle infinie.

C'est notamment le cas des codes p5 où la fonction draw() est une boucle infinie dont on ne sort que lorsqu'un évènement est intercepté (par exemple, le clic sur la fermeture de la fenêtre d'affichage).

Ou lors de la création d'un jeu....

Observez et exécutez le code suivant :

```
while True :
    reponse = input("tapez sur la lettre S du clavier pour me sortir de cet enfer : ")
    if reponse == 'S' or reponse == 's':
        break

print("merci, j'étais bloqué dans une boucle infinie")
```

```
🗋 Algo
```

```
tapez sur la lettre S du clavier pour me sortir de cet enfer : E
tapez sur la lettre S du clavier pour me sortir de cet enfer : S
merci, j'étais bloqué dans une boucle infinie
```

- le début du code : while True est typique des boucles infinies volontaires. On aurait tout aussi bien pu écrire while 3 > 2 (on rencontre même parfois des while 1)
- vous avez découvert l'expression break qui comme son nom l'indique permet de casser la boucle (cela marche pour while comme pour for) et donc d'en sortir. Son emploi est controversé parmi les puristes de la programmation. Nous dirons juste que c'est une instruction bien pratique.