Python: Les boucles for

Quand un programme commence à devoir répéter une action plusieurs fois, copier-coller la même ligne devient vite absurde. La boucle `for` est la réponse propre à ce problème : elle décrit une fois ce qu'il faut faire, et Python s'occupe de le refaire autant de fois qu'il faut.

5GMS 5TTR 6TTR 3TTR Exploration

*Tu vas voir ici **pourquoi** on a inventé les boucles `for`, **comment** elles se construisent, et tu termineras en écrivant ta première table de multiplication automatique.*

🎯 Objectifs

À la fin de ce cours, tu seras capable de :

1. Reconnaître les situations où une boucle `for` remplace utilement du copier-coller.
2. Écrire une boucle `for` qui exécute une action **un nombre précis de fois**.
3. Utiliser `range()` avec un, deux ou trois paramètres.
4. **Parcourir** une chaîne de caractères, lettre par lettre.
5. Résoudre seul un petit problème classique (table de multiplication) avec une boucle.

Partie 1 — Le problème : copier-coller, ça ne tient pas

Imagine qu'on te demande d'afficher 5 fois **le même message** — par exemple « Joyeux anniversaire, Léa ! » pour faire une carte un peu folle. Sans boucle, ça donne :

```
print("Joyeux anniversaire, Léa !")
print("Joyeux anniversaire, Léa !")
print("Joyeux anniversaire, Léa !")
print("Joyeux anniversaire, Léa !")
print("Joyeux anniversaire, Léa !")
```

Cinq lignes identiques. Ça marche, mais maintenant :

- **Et s'il en fallait 200 fois ?** Tu vas vraiment écrire 200 `print` ?

- **Et si on veut changer le texte** (mettre « Bon anniversaire » à la place) ? Il faudrait modifier les 200 lignes.
- **Et si on ne sait qu'au moment de l'exécution combien de fois répéter** (l'utilisateur entre le nombre) ? Tu ne peux même pas écrire les `print` à l'avance.

> 🧠 **LA RÉPÉTITION MANUELLE NE PASSE PAS À L'ÉCHELLE.** Dès qu'on commence à copier-coller plus de deux ou trois lignes presque identiques, c'est le signe qu'il faut sortir une boucle.

La boucle `for` répond exactement à ça : tu décris **une seule fois** ce qu'il faut faire, et Python s'occupe de le refaire autant de fois que tu veux.

> 💡 **ET SI CHAQUE LIGNE ÉTAIT DIFFÉRENTE ?** Par exemple, afficher 5 prénoms d'élèves : `Alice`, `Bob`, `Charlie` … Ça se résout aussi avec une boucle, mais on a besoin d'une **liste** pour stocker les prénoms. Les listes arrivent dans un cours suivant — pour l'instant, on se concentre sur le cas « même action, répétée N fois ».

Partie 2 — Première boucle : répéter N fois

Pour démarrer simple, on combine `for` avec la fonction `range()` pour répéter l'action **un nombre précis de fois** :

```
1  for i in range(5):
2      print("Joyeux anniversaire, Léa !")
```

Résultat à l'écran :

```
1  Joyeux anniversaire, Léa !
2  Joyeux anniversaire, Léa !
3  Joyeux anniversaire, Léa !
4  Joyeux anniversaire, Léa !
5  Joyeux anniversaire, Léa !
```

Cinq fois. C'est exactement le programme du début, mais en **deux lignes** au lieu de cinq. Et si tu remplaces `range(5)` par `range(200)`, tu obtiens 200 lignes — sans rien changer d'autre.

Décortiquons cette boucle

```
1  for      i      in     range(5)     :          ← en-tête (« la promesse »)
2  └─┬─┘  └─┬─┘  └─┬─┘  └─────┬─────┘
3    mot-clé  variable          séquence à parcourir
4
5      print("Bonjour")                          ← corps (indenté de 4 espaces)
```

Élément	Rôle
`for`	Mot-clé qui démarre la boucle
`i`	Variable qui prend une nouvelle valeur à chaque tour
`in`	Mot-clé qui sépare la variable de la séquence
`range(5)`	Génère la séquence `0, 1, 2, 3, 4`
`:`	Termine l'en-tête (obligatoire en Python)
Corps indenté	Le code à répéter à chaque tour

> ⚠️ **L'INDENTATION EST OBLIGATOIRE.** Le corps de la boucle DOIT être indenté (4 espaces, ou une tabulation, mais reste cohérent). Sans indentation, Python ne sait pas où finit la boucle.

Partie 3 — Anatomie d'une exécution

Reprends ce code, mais cette fois on **utilise** la variable `i` :

```
for i in range(5):
    print("Tour numéro", i)
```

Résultat :

```
Tour numéro 0
Tour numéro 1
Tour numéro 2
Tour numéro 3
Tour numéro 4
```

À chaque tour, `i` prend une valeur différente :

Tour	Valeur de `i`	Ligne affichée
1	0	`Tour numéro 0`
2	1	`Tour numéro 1`
3	2	`Tour numéro 2`
4	3	`Tour numéro 3`
5	4	`Tour numéro 4`

💡 **POURQUOI ÇA COMMENCE À 0 ?** Parce que `range(5)` produit `0, 1, 2, 3, 4` — cinq valeurs en partant de 0. C'est une convention en programmation : on compte à partir de 0. On verra dans la partie 5 comment forcer un autre point de départ.

Partie 4 — Parcourir une chaîne de caractères

Il existe une autre situation où une boucle `for` est très utile : quand on veut **traiter chaque élément** d'une suite, sans forcément savoir combien il y en a.

Une chaîne de caractères, c'est exactement ça : une suite de lettres.

```
mot = "Python"

for lettre in mot:
    print(lettre)
```

Résultat :

```
P
y
t
h
o
n
```

Ici, on n'a pas écrit `range(6)`. On a écrit `for lettre in mot`. Python sait **tout seul** parcourir la chaîne lettre par lettre, peu importe sa longueur. Si le mot fait 6 lettres, la boucle s'exécute 6 fois. S'il en fait 50, elle s'exécute 50 fois.

🧠 **À RETENIR** : `for` ne sert pas seulement à compter. Sa vraie nature est de dire *« pour chaque élément de cette suite, fais ceci »*. Une chaîne, une liste, un fichier ouvert… tout ça se parcourt avec `for` .

Exemple — Lister les lettres d'un mot

Tu pourrais essayer ça :

```
mot = "ornithorynque"

for lettre in mot:
    print("Une lettre :", lettre)
```

Sans rien savoir d'autre, ton programme va afficher 13 lignes. Tu n'as pas eu à écrire « 13 » dans ton code — c'est Python qui s'adapte.

Partie 5 — La fonction `range()` en détail

Tu as vu `range(5)`. Mais `range()` peut prendre **un, deux ou trois** paramètres. Chaque forme est utile dans des contextes différents.

Forme 1 — `range(fin)` : de 0 à fin − 1

```
1  for i in range(5):
2      print(i)
```

→ `0, 1, 2, 3, 4`

Forme 2 — `range(début, fin)` : de début à fin − 1

```
1  for i in range(1, 6):
2      print(i)
```

→ `1, 2, 3, 4, 5`

> 💡 **TRÈS PRATIQUE** pour compter de 1 à 10 « comme un humain », sans commencer à 0.

Forme 3 — `range(début, fin, pas)` : avec un incrément personnalisé

```
1  for i in range(0, 20, 2):
2      print(i)
```

→ `0, 2, 4, 6, 8, 10, 12, 14, 16, 18`

Le pas peut être **négatif** pour aller à reculons :

```
1  for i in range(10, 0, -1):
2      print(i)
```

→ `10, 9, 8, 7, 6, 5, 4, 3, 2, 1` (parfait pour un compte à rebours)

Récapitulatif

Forme	Génère	Exemple typique
`range(n)`	`0, 1, …, n - 1`	Répéter n fois
`range(a, b)`	`a, a+1, …, b - 1`	Compter de 1 à 10
`range(a, b, pas)`	`a, a+pas, …` (jusqu'à < `b`)	Pairs, impairs, à rebours

> ⚠️ **LA VALEUR DE FIN N'EST JAMAIS INCLUSE.** `range(1, 11)` va de 1 à 10, pas jusqu'à 11. Ça surprend au début, mais c'est ce qui permet d'écrire `range(len(mot))` sans erreur.

🧪 Exercices

Exercice guidé — La table de 7

Énoncé. Écris un programme qui affiche la **table de multiplication par 7**, de 7 × 1 jusqu'à 7 × 10, dans ce format :

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
...
7 x 10 = 70
```

Avant de coder, prends 30 secondes pour réfléchir. **Ne lis pas la suite tout de suite** — essaie de réfléchir à ces trois questions :

1. Combien de lignes vais-je afficher au total ?
2. Qu'est-ce qui **change** d'une ligne à l'autre ?
3. Qu'est-ce qui **reste pareil** ?

Étape 1 — Identifier la séquence

Dix lignes à afficher : pour `i` qui va de **1 à 10**. C'est exactement ce que produit `range(1, 11)` (rappel : la valeur de fin n'est pas incluse, donc 11 pour aller jusqu'à 10).

Premier brouillon, juste pour vérifier que la boucle tourne bien :

```python
for i in range(1, 11):
    print(i)
```

Exécute. Tu dois voir afficher les nombres de 1 à 10. Si oui, **la boucle est bonne** — il ne reste plus qu'à changer ce qu'on affiche.

Étape 2 — Calculer le produit

À chaque tour, on veut afficher `7 × i`. En Python, le `×` s'écrit `*` :

```
for i in range(1, 11):
    print(7 * i)
```

Tu obtiens :

```
7
14
21
28
35
42
49
56
63
70
```

Les bons résultats sont là. Mais le format n'est pas celui demandé.

Étape 3 — Composer la ligne complète

On veut afficher `7 x 1 = 7`, `7 x 2 = 14`, etc. La fonction `print` accepte plusieurs arguments séparés par des virgules : elle les affiche tous séparés par un espace.

```
for i in range(1, 11):
    print(7, "x", i, "=", 7 * i)
```

Résultat :

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

C'est exactement ce qui était demandé.

Étape 4 (bonus) — Plus propre avec une f-string

Quand la ligne à afficher devient longue, les `print(a, "x", b, ...)` deviennent illisibles. Python a une syntaxe plus élégante : les **f-strings** (chaînes formatées).

```
for i in range(1, 11):
    print(f"7 x {i} = {7 * i}")
```

Le préfixe `f` devant les guillemets active la magie : tout ce qui est entre `{ }` est évalué et inséré dans la chaîne. C'est exactement la même sortie qu'avant, mais beaucoup plus lisible.

> 🧠 **HABITUE-TOI AUX F-STRINGS DÈS MAINTENANT.** C'est la façon moderne d'afficher du texte mêlé à des variables en Python. Tu vas en croiser dans presque tous les exercices à partir d'ici.

Pour aller plus loin

Quelques variantes à tester par toi-même :

- **Table choisie** : remplace `7` par une variable `n = 7` en début de programme. Tu pourras changer la table en modifiant une seule ligne.
- **Table interactive** : remplace `n = 7` par `n = int(input("Quelle table ? "))`. Le programme demande à l'utilisateur quelle table afficher.
- **Toutes les tables de 1 à 10** : imbrique deux boucles `for` (mais ça, c'est pour un cours suivant).

Exercice 2 — Prédiction

Avant de lancer chacun de ces programmes, **prévois sur papier** ce qu'il va afficher. Puis vérifie.

```
for i in range(5):
    print(i)
```

```
for i in range(1, 5):
    print(i)
```

```
for i in range(1, 11, 2):
    print(i)
```

```
for i in range(10, 0, -1):
    print(i)
print("BOUM !")
```

Exercice 3 — Initiales en chaîne

Écris un programme qui demande ton prénom à l'utilisateur, puis affiche chaque lettre en majuscule, **une par ligne**. Indice : la méthode `.upper()` sur une lettre la met en majuscule.

```
Ton prénom ? Ludovic
L
U
D
O
V
I
C
```

✍️ À retenir

- Une boucle `for` répète un bloc de code, **soit un nombre précis de fois**, **soit pour chaque élément d'une séquence**.
- La syntaxe générale est :

```
for variable in sequence:
    instructions
```

- `range(n)` produit `0, 1, …, n - 1`. `range(a, b)` part de `a`. `range(a, b, pas)` ajoute un incrément (positif ou négatif).
- La **valeur de fin n'est jamais incluse** : `range(1, 11)` s'arrête à 10.
- Une chaîne de caractères est **itérable** : `for lettre in mot` parcourt les lettres une par une.
- L'indentation du corps de la boucle est **obligatoire**.
- Les **f-strings** (`f"… {variable} …"`) sont la façon moderne d'insérer des variables dans du texte.

Suite

Tu sais maintenant *répéter*. La prochaine étape : faire **évoluer une variable** d'un tour à l'autre. Ça permet de **compter** (combien de voyelles dans un mot ?) et **cumuler** (quelle est la somme des notes ?). C'est ce qu'on découvre dans le prochain cours.

Vidéo à la une à regarder sur Youtube: http://youtu.be/BrknhzrHm8w