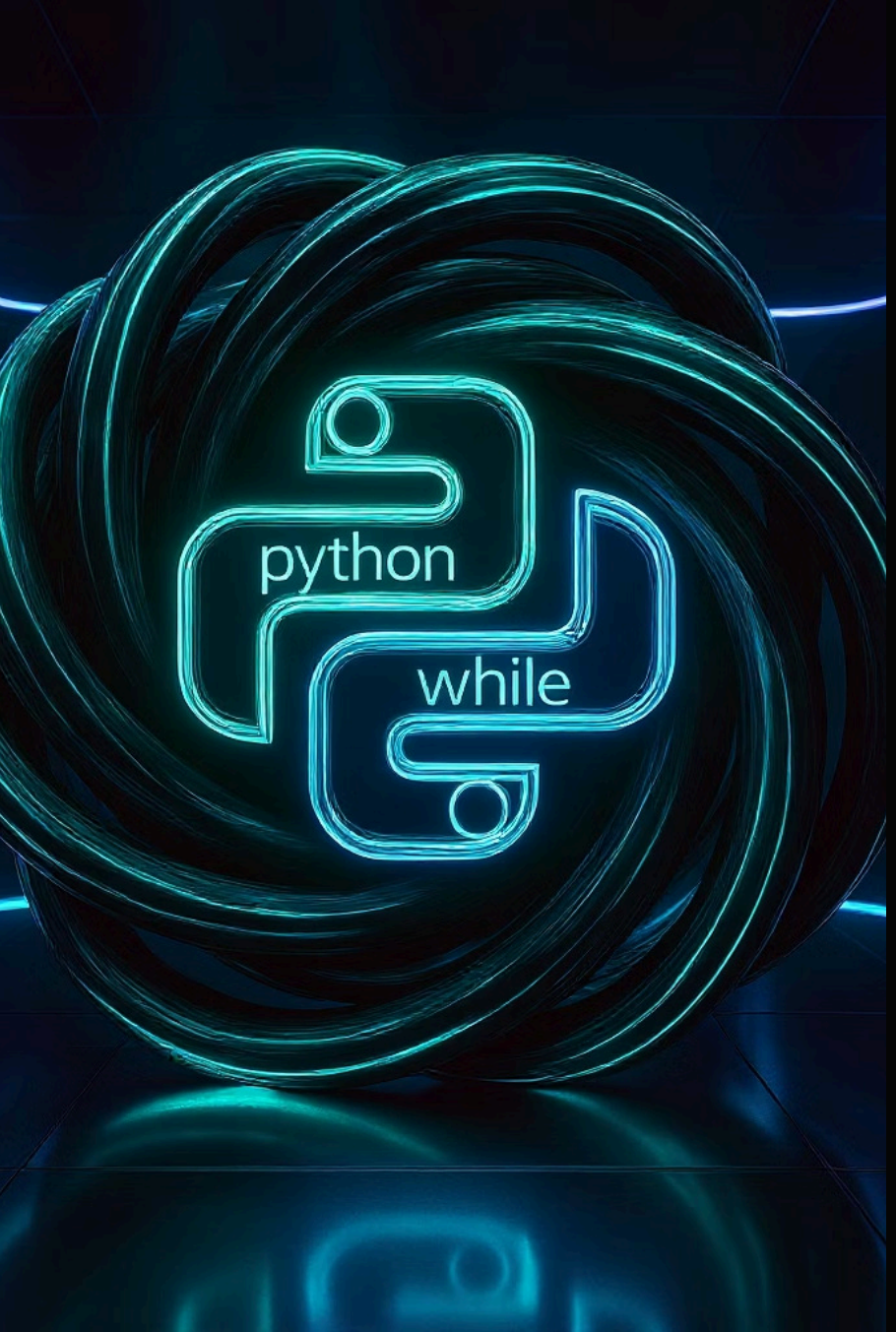


Les Structures Itératives en Python

Les structures itératives sont des outils fondamentaux en programmation, conçus pour automatiser les tâches répétitives et améliorer l'efficacité des programmes. Ce module examine en détail les boucles **for** et **while**, des constructions essentielles dans le langage Python.

Par David DONISA , Enseignant en BTS SIO

Pourquoi utiliser une structure itérative ?

Répétition d'actions

Compter, afficher, calculer ou valider plusieurs fois la même opération sans réécrire le code

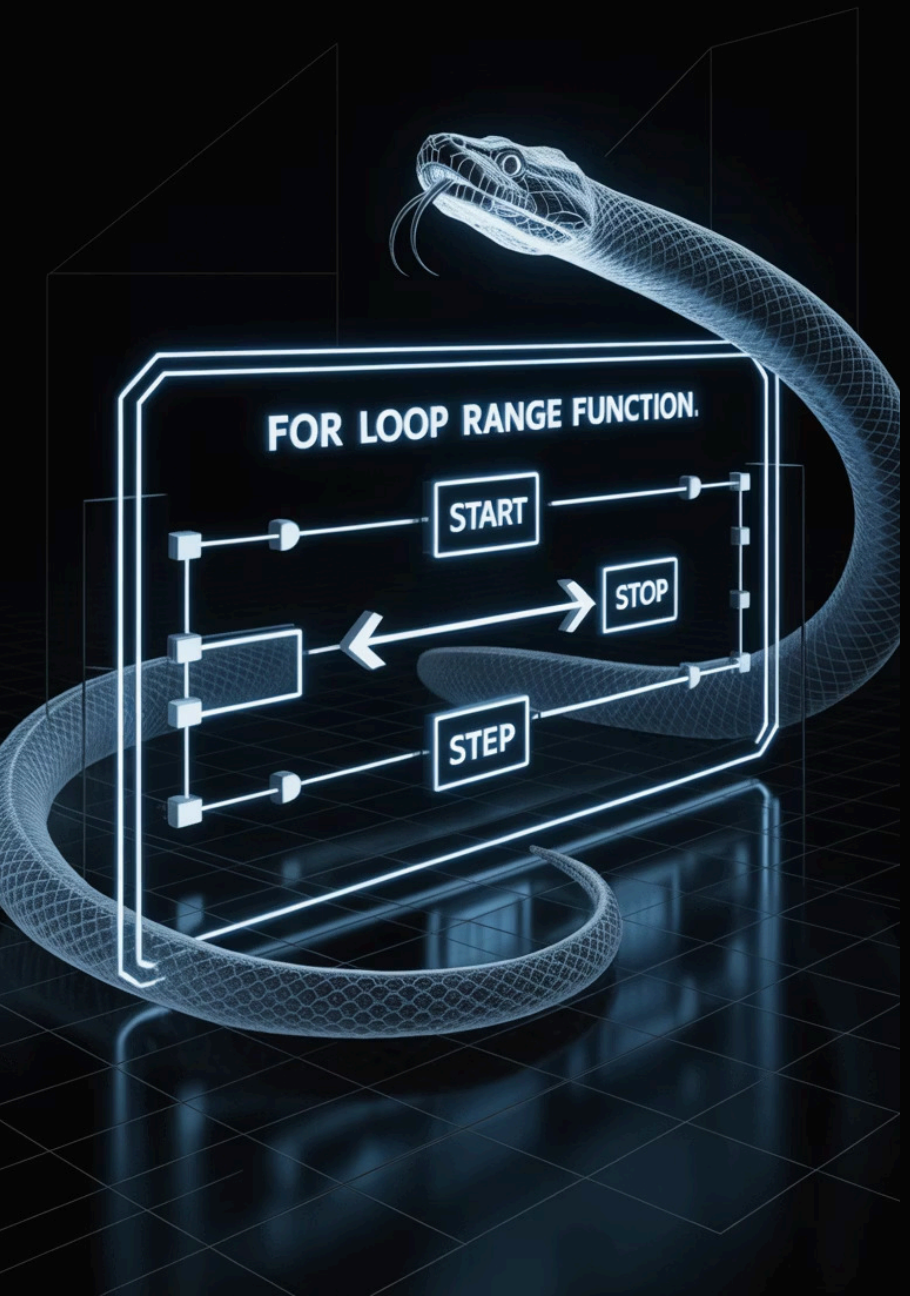
Boucle FOR

Nombre d'itérations connu et contrôlé via `range()`

Boucle WHILE

Répéter "tant que" une condition reste vraie, avec arrêt conditionnel

Les structures itératives évitent la duplication de code et permettent de traiter des volumes de données variables. Dans ce cours, nous nous concentrons sur les variables simples, les conditions et la fonction `range()`.



La boucle FOR avec range()

La fonction `range(début, fin, pas)` génère une séquence d'entiers. Elle commence à 'début', s'arrête **avant** 'fin' et progresse selon le 'pas' défini.

```
for nom_variable in range(0, 11, 2):
    print("La variable vaut", nom_variable)
```

Ce code affiche : 0, 2, 4, 6, 8, 10. La valeur 11 n'est jamais atteinte car `range()` s'arrête avant la limite supérieure.

Important : La borne de fin est toujours exclusive dans `range()`.

Syntaxe abrégée de range()

Version complète

```
range(0, 5, 1)
```

Spécifie début, fin et pas explicitement

Version abrégée

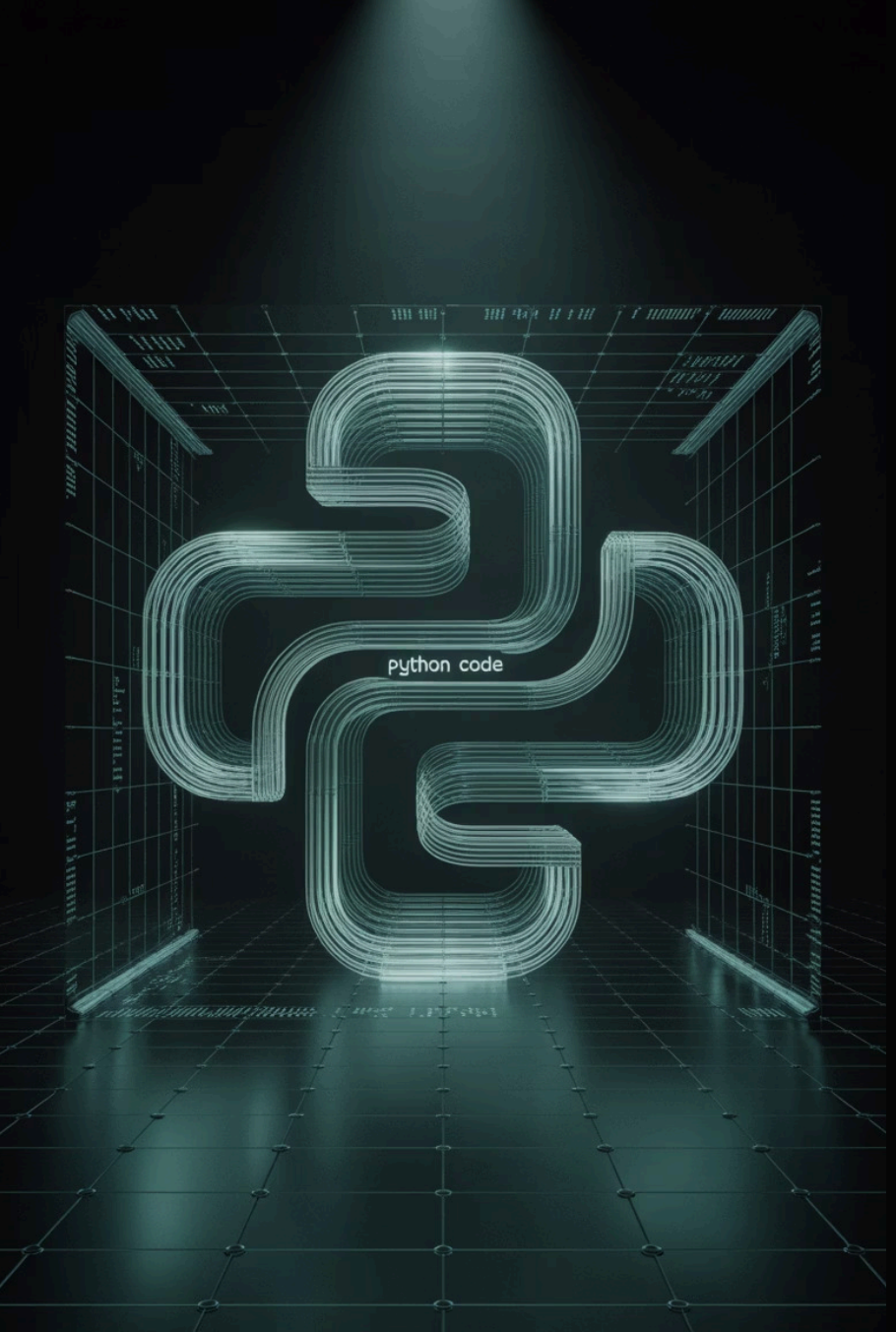
```
range(5)
```

Équivaut à `range(0, 5, 1)`

L'exemple suivant affiche "OK" cinq fois avec le numéro d'itération :

```
for i in range(5):
    print("OK (itération n°", i, ")")
```

Résultat : OK (itération n° 0), OK (itération n° 1)... jusqu'à OK (itération n° 4). Python commence toujours à compter depuis 0 !



Boucles FOR imbriquées

Une boucle à l'intérieur d'une autre permet de créer des motifs complexes. Voici un exemple qui génère un tableau d'étoiles :

```
for ligne in range(1, 4):
    texte = ""
    for colonne in range(1, 5):
        texte = texte + "*"
    print("Ligne", ligne, ":", texte)
```

Ce code produit un tableau de 3 lignes × 4 colonnes. La boucle externe contrôle les lignes, la boucle interne construit chaque ligne en concaténant les étoiles.

Résultat : Ligne 1 : ****, Ligne 2 : ****, Ligne 3 : ****

Mots-clés de contrôle

pass

Instruction "ne rien faire". Sert de placeholder pour du code à venir. Le programme continue normalement.

continue

Saute **directement** à l'itération suivante, ignorant le reste du bloc courant.

break

Arrête **immédiatement** la boucle courante et sort complètement de celle-ci.

Ces mots-clés offrent un contrôle précis sur l'exécution des boucles et permettent d'optimiser les performances de vos programmes.

La boucle WHILE

La boucle `while` s'exécute **tant que** une condition reste vraie. Elle est idéale quand le nombre d'itérations n'est pas connu à l'avance.

```
mon_compteur = 0
while mon_compteur < 10:
    mon_compteur += 1
    if mon_compteur == 3:
        continue
    if (mon_compteur % 5) == 0:
        break
    if (mon_compteur % 2) == 0:
        print("Bonjour (compteur pair)", mon_compteur)
```

Ce code s'arrête au premier multiple de 5 et saute l'affichage quand le compteur vaut 3. Il affiche uniquement les nombres pairs.

Attention : Risque de boucle infinie si la condition ne change jamais !

Éviter la boucle infinie

❌ Exemple d'une mauvaise boucle while

```
x = 0
while x < 3:
    print("Je boucle...")
    # x n'est jamais modifié !
```

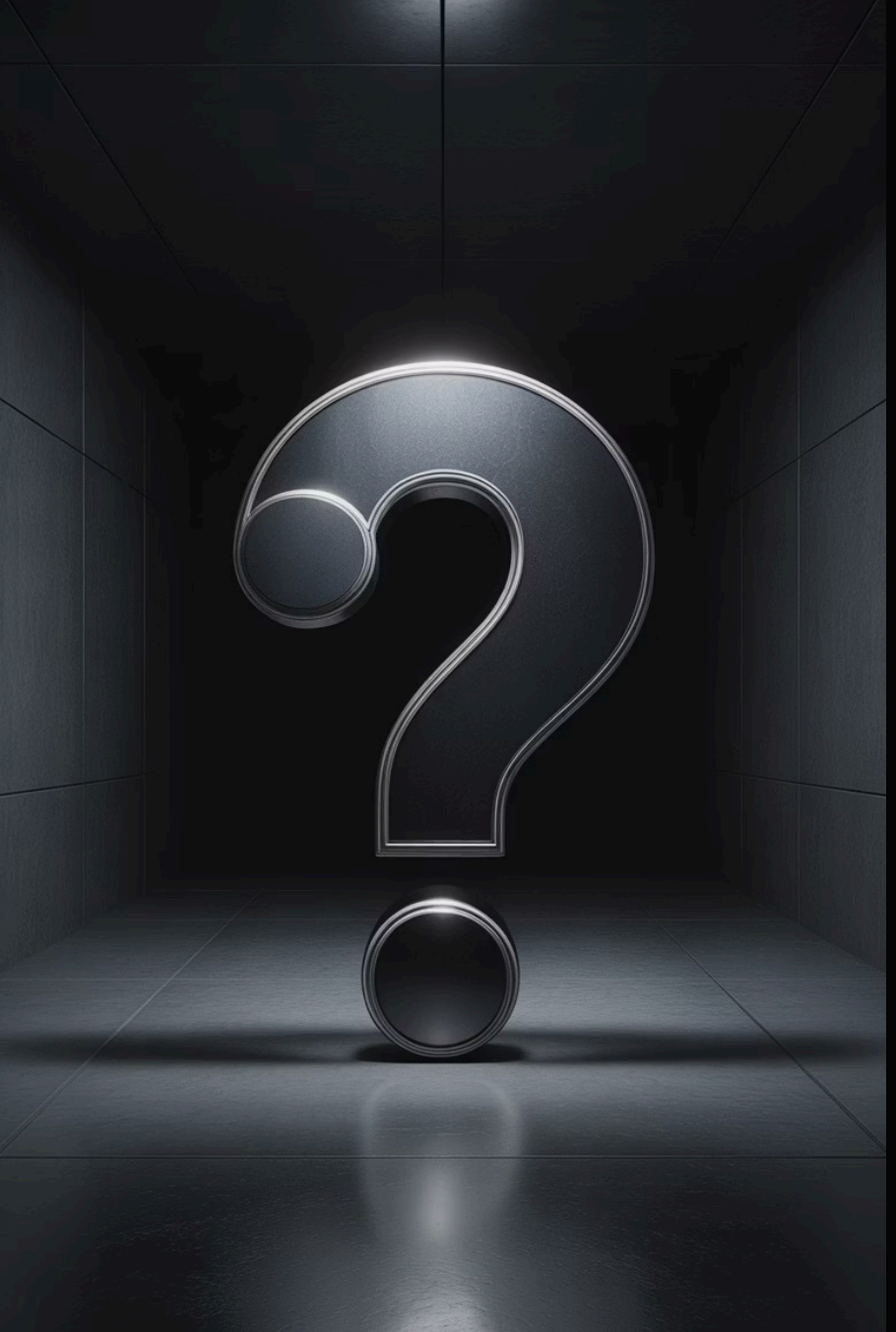
Cette boucle ne s'arrête jamais car `x` reste toujours égal à 0.

✅ Exemple d'une bonne boucle while

```
x = 0
while x < 3:
    print("Tour n°", x)
    x = x + 1
```

La variable `x` est incrémentée à chaque tour, garantissant l'arrêt de la boucle.

Toujours s'assurer qu'une variable liée à la condition évolue à chaque itération ou prévoir un `break` conditionnel.



Quiz : Structures itératives

> **Question :** Que va afficher ce code complexe ?

```
resultat = 0
for i in range(2, 8):
    if i % 3 == 0:
        continue
    if i > 6:
        break
    resultat += i
print("Résultat :", resultat)
```

Analysez attentivement chaque itération, les conditions `continue` et `break`, puis déterminez la valeur finale de `resultat`.

Correction du Quiz

01

i = 2

2 % 3 ≠ 0, 2 ≤ 6 → resultat += 2 → resultat = 2

02

i = 3

3 % 3 = 0 → continue (on passe à l'itération suivante)

03

i = 4

4 % 3 ≠ 0, 4 ≤ 6 → resultat += 4 → resultat = 6

04

i = 5

5 % 3 ≠ 0, 5 ≤ 6 → resultat += 5 → resultat = 11

05

i = 6

6 % 3 = 0 → continue (on passe à l'itération suivante)

06

i = 7

7 % 3 ≠ 0, mais 7 > 6 → break (sortie de boucle)

Résultat final : 11. Seuls les nombres 2, 4 et 5 ont été additionnés, car 3 et 6 sont des multiples de 3 (continue) et la boucle s'arrête avant de traiter 7 (break).