Les boucles en Python

RÉSUMÉ

Répéter une action est l'une des forces principales d'un ordinateur : il peut faire la même chose des milliers de fois, sans erreur et sans fatigue. Les **boucles** sont l'outil qui permet cette répétition en programmation. Au lieu d'écrire dix fois la même instruction, on l'écrit une seule fois dans une boucle et Python s'occupe de la répéter. Il existe deux types de boucles : `for` (quand on sait combien de fois répéter) et `while` (quand on répète jusqu'à ce qu'une condition soit remplie).

① MÉMO

La boucle `for` : répéter un nombre connu de fois

```
for element in sequence:
    bloc   # exécuté pour chaque élément de la séquence
```

- `range(n)` : entiers de `0` à `n-1`.
- `range(a, b)` : entiers de `a` à `b-1`.

On utilise la boucle `for` quand le nombre de répétitions est connu à l'avance.

La boucle `while` : tant que...

```
while condition:
    bloc   # exécuté tant que la condition est vraie
```

- On utilise `while` quand on ne sait pas à l'avance combien de fois répéter.
- Le bloc doit modifier une variable pour que la condition finisse par devenir fausse (sinon boucle infinie).

Accumulateurs et compteurs

Deux schémas fondamentaux :

```
# Accumulateur : on accumule un résultat
total = 0
for x in liste:
    total = total + x      # ou total += x

# Compteur : on compte les éléments vérifiant une condition
compteur = 0
for x in liste:
    if condition:
        compteur += 1
```

Pièges fréquents

— `range(5)` ne contient pas `5` (la borne supérieure est exclue).
— Oublier d'incrémenter la variable de contrôle dans un `while`, ce qui provoque une boucle infinie.

② EXEMPLES

Programme 4 · Somme des entiers de 1 à n

```
def somme(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
```

Tableau d'état pour `somme(4)` :

Itération	`i`	Calcul	`total`
init	–	–	0
1	1	$0+1$	1
2	2	$1+2$	3
3	3	$3+3$	6
4	4	$6+4$	10

Programme 5 · Table de multiplication

```
def table(n):
    for i in range(1, 11):
        print(f"{n} × {i} = {n * i}")
```

Programme 6 · Compter les voyelles d'un mot

```
def nb_voyelles(mot):
    compteur = 0
    for lettre in mot:
        if lettre.lower() in "aeiouy":
            compteur += 1
    return compteur
```

Tableau d'état pour `nb_voyelles("Python")` :

Itération	`lettre`	Voyelle ?	`compteur`
init	–	–	0
1	`"P"`	non	0
2	`"y"`	oui	1
3	`"t"`	non	1
4	`"h"`	non	1
5	`"o"`	oui	2
6	`"n"`	non	2

`.lower()` convertit chaque lettre en minuscule avant le test, ce qui permet de compter correctement les voyelles majuscules.

Programme 7 · Boucle while : deviner un nombre

```
import random
secret = random.randint(1, 100)
tentative = 0
while tentative != secret:
    tentative = int(input("Proposition : "))
    if tentative < secret:
        print("Plus grand !")
    elif tentative > secret:
        print("Plus petit !")
print("Bravo !")
```

③ EXERCICES

Exercice 1 *Somme et produit*

1. Écrire une fonction `somme_carres(n)` qui renvoie $1^2 + 2^2 + \cdots + n^2$.
2. Écrire une fonction `factorielle(n)` qui renvoie $n! = 1 \times 2 \times \cdots \times n$.

Exercice 2 *Boucle while*

1. Écrire une fonction `nb_chiffres(n)` qui renvoie le nombre de chiffres d'un entier positif `n` (sans utiliser `str`).
   Indice : diviser `n` par 10 avec l'opérateur `//` à chaque tour de boucle et compter combien de fois on peut le faire avant que `n` devienne 0. Chaque division retire un chiffre par la droite.
   Cas particulier : `nb_chiffres(0)` doit renvoyer `1` (le nombre 0 s'écrit avec un seul chiffre), sinon la boucle `while n > 0` ne s'exécute jamais et renvoie 0.
2. Écrire une fonction `somme_chiffres(n)` qui renvoie la somme des chiffres de `n`.

SOLUTIONS DES EXERCICES

Corrigé de l'exercice 1.

```python
def somme_carres(n):
    total = 0
    for i in range(1, n + 1):
        total += i ** 2
    return total

def factorielle(n):
    produit = 1
    for i in range(1, n + 1):
        produit *= i
    return produit
```

Vérification : `somme_carres(3)` = $1 + 4 + 9 = 14$. `factorielle(5)` = 120.

Corrigé de l'exercice 2.

```python
def nb_chiffres(n):
    if n == 0:
        return 1
    compteur = 0
    while n > 0:
        n = n // 10
        compteur += 1
    return compteur

def somme_chiffres(n):
    total = 0
    while n > 0:
        total += n % 10    # dernier chiffre
        n = n // 10         # on retire le dernier chiffre
    return total
```

Principe : `n % 10` extrait le dernier chiffre et `n // 10` le retire.

Tableau d'état pour `somme_chiffres(427)` :

Étape	`n % 10`	`total`	`n` (après `// 10`)
init	–	0	427
1	7	$0 + 7 = 7$	42
2	2	$7 + 2 = 9$	4
3	4	$9 + 4 = 13$	0

La boucle s'arrête car $n = 0$. Résultat : $4 + 2 + 7 = 13$.

Vérification : `nb_chiffres(427)` renvoie `3`. `somme_chiffres(427)` renvoie `13`.