Les boucles en Python

Résumé

Un des grands atouts de l'ordinateur est sa capacité à **répéter** une action des milliers de fois sans se fatiguer ni se tromper. Les **boucles** sont la structure qui permet cette répétition : au lieu d'écrire cent lignes identiques, on écrit une seule instruction et on demande à Python de la répéter autant de fois que nécessaire. Il existe deux types de boucles : `for` quand on sait combien de fois répéter, et `while` quand on répète tant qu'une condition est vraie. Maîtriser les boucles, c'est passer du stade « je recopie » au stade « je programme ».

Prérequis : Structures conditionnelles.

① Mémo

Définition 1 (La boucle `for` : parcours d'une séquence)

```
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`.
- `range(a, b, p)` : de `a` à `b-1` avec un pas de `p`.

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

Définition 2 (La boucle `while` : tant que…)

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

- On utilise `while` quand le nombre d'itérations est inconnu à l'avance.
- Le bloc doit modifier une variable pour que la condition finisse par devenir fausse (sinon boucle infinie).
- `break` sort immédiatement de la boucle.
- `continue` passe directement à l'itération suivante.

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(x):
        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.
- Modifier une liste en la parcourant avec `for` (comportement imprévisible).
- Confondre `for i in range(len(L))` (parcours par indice) et `for x in L` (parcours par valeur).

Erreurs classiques

Code erroné	Code correct	Explication
`for i in range(1, 10):` pour avoir 1 à 10	`for i in range(1, 11):`	La borne supérieure de `range` est exclue : `range(1, 10)` s'arrête à 9.
`while x != 0:` sans modifier `x`	Ajouter `x = x - 1` (ou autre) dans la boucle	Sans modification de la variable de contrôle, la boucle `while` tourne indéfiniment.
Modifier une liste en la parcourant : `for x in L: L.remove(x)`	Parcourir une copie : `for x in L[:]: L.remove(x)`	Supprimer des éléments pendant le parcours saute des éléments ou lève une erreur.
Confondre `break` et `return`	`break` sort de la boucle, `return` sort de la fonction	Dans une fonction, `break` ne quitte que la boucle ; `return` quitte la fonction entière.

② Exemples

Programme 1 · Somme des entiers de 1 à n (accumulateur)

```python
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 2 · Table de multiplication

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

Programme 3 · Compter les voyelles d'un mot (compteur)

```python
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

Programme 4 · 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 !")
```

Programme 5 · Approfondissement NSI : suite de Syracuse

```
def syracuse(n):
    etapes = 0
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
        etapes += 1
    return etapes
```

Tableau d'état pour `syracuse(6)` :

Étape	`n` (début)	Pair ?	Calcul
1	6	oui	$6 \div 2 = 3$
2	3	non	$3 \times 3 + 1 = 10$
3	10	oui	$10 \div 2 = 5$
4	5	non	$3 \times 5 + 1 = 16$
5	16	oui	$16 \div 2 = 8$
6	8	oui	$8 \div 2 = 4$
7	4	oui	$4 \div 2 = 2$
8	2	oui	$2 \div 2 = 1$

La fonction renvoie `8` : il a fallu huit étapes pour atteindre 1. La conjecture de Syracuse affirme que cette suite atteint toujours 1, quel que soit l'entier de départ. C'est un problème ouvert en mathématiques.

Programme 6 · Boucles imbriquées

```python
for i in range(1, 6):
    for j in range(1, 6):
        print(f"{i*j:4}", end="")
    print()   # retour à la ligne après chaque ligne du tableau
```

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 *Compter et filtrer*

1. Écrire une fonction `nb_diviseurs(n)` qui renvoie le nombre de diviseurs d'un entier positif `n`.
2. En déduire une fonction `est_premier(n)` qui teste si `n` est premier.
3. Écrire une fonction `premiers_jusqu_a(n)` qui renvoie la liste des nombres premiers inférieurs ou égaux à `n`.

Exercice 3 *Boucle while*

1. Écrire une fonction `nb_chiffres(n)` qui renvoie le nombre de chiffres d'un entier positif `n` (sans utiliser `str`).
2. Écrire une fonction `somme_chiffres(n)` qui renvoie la somme des chiffres de `n`.
3. Écrire une fonction `miroir(n)` qui renvoie l'entier obtenu en inversant les chiffres de `n`. Par exemple, `miroir(1234)` renvoie `4321`.

Exercice 4 *Boucles imbriquées* Écrire une fonction `triangle(n)` qui affiche un triangle d'étoiles :

```
*
**
***
****
*****
```

pour `n = 5`.

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)` $= 120$.

On peut vérifier avec la formule : $\sum_{k=1}^{n} k^2 = \dfrac{n(n+1)(2n+1)}{6}$, pour $n = 3$: $\dfrac{3 \times 4 \times 7}{6} = 14$. ✓

Corrigé de l'exercice 2.

```python
def nb_diviseurs(n):
    compteur = 0
    for d in range(1, n + 1):
        if n % d == 0:
            compteur += 1
    return compteur

def est_premier(n):
    if n < 2:
        return False
    return nb_diviseurs(n) == 2

def premiers_jusqu_a(n):
    resultat = []
    for k in range(2, n + 1):
        if est_premier(k):
            resultat.append(k)
    return resultat
```

Vérification :

- `nb_diviseurs(12)` renvoie 6 (diviseurs : 1, 2, 3, 4, 6, 12) ;
- `est_premier(7)` renvoie `True` ;
- `premiers_jusqu_a(20)` renvoie `[2, 3, 5, 7, 11, 13, 17, 19]`.

Remarque : cette version est correcte mais peu efficace. On pourrait optimiser `est_premier` en ne testant les diviseurs que jusqu'à $\sqrt{n}$.

Corrigé de l'exercice 3.

```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

def miroir(n):
    resultat = 0
    while n > 0:
        resultat = resultat * 10 + n % 10
        n = n // 10
    return resultat
```

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

Tableau d'état pour `miroir(1234)` :

Étape	`n % 10`	`resultat`	`n`
init	–	0	1234
1	4	$0 \times 10 + 4 = 4$	123
2	3	$4 \times 10 + 3 = 43$	12
3	2	$43 \times 10 + 2 = 432$	1
4	1	$432 \times 10 + 1 = 4321$	0

Corrigé de l'exercice 4.

```python
def triangle(n):
    for i in range(1, n + 1):
        print("*" * i)
```

Remarque : en Python, `"*" * i` répète la chaîne `i` fois, ce qui évite une boucle interne. La version avec boucle imbriquée serait :

```python
def triangle(n):
    for i in range(1, n + 1):
        for j in range(i):
            print("*", end="")
        print()
```