

# Boucles

## Fiche de synthèse

### ① DEUX BOUCLES

Une **boucle** répète un bloc d'instructions indenté, sans le recopier.

- La boucle `for` parcourt une séquence : on l'utilise quand on connaît le nombre de tours.
- La boucle `while` répète tant qu'une condition est vraie : on l'utilise quand on ne connaît pas le nombre de tours.

### ② LA BOUCLE FOR

```
for lettre in "NSI":
    print(lettre)      # N, puis S, puis I

for i in range(5):
    print(i)           # 0, 1, 2, 3, 4
```

À chaque tour, la variable de boucle prend la valeur suivante de la séquence : les caractères d'une chaîne, ou les entiers fournis par `range`.

| Appel                        | Entiers obtenus |
|------------------------------|-----------------|
| <code>range(5)</code>        | 0, 1, 2, 3, 4   |
| <code>range(2, 7)</code>     | 2, 3, 4, 5, 6   |
| <code>range(0, 20, 5)</code> | 0, 5, 10, 15    |

Dans `range(début, fin, pas)`, la fin est toujours **exclue** : pour aller de 1 à 5, on écrit `range(1, 6)`. Le **pas** est l'écart entre deux valeurs.

### ③ ACCUMULER UN RÉSULTAT

```
total = 0
for i in range(1, 6):
    total += i
print(f"La somme vaut {total}")  # 15
```

|       |   |   |   |    |    |
|-------|---|---|---|----|----|
| i     | 1 | 2 | 3 | 4  | 5  |
| total | 1 | 3 | 6 | 10 | 15 |

La variable `total` est un **accumulateur** : elle reçoit sa valeur de départ **avant** la boucle, puis change à chaque tour. Ce tableau, qui suit la valeur des variables tour après tour, est une **trace d'exécution**. Le même schéma construit une chaîne en partant de la chaîne vide "".

### ④ LA BOUCLE WHILE

```
compteur = 1
while compteur <= 5:
    print(compteur)
    compteur += 1
```

La condition est testée avant chaque tour : ici six fois, la dernière donnant `False`. Le corps de la boucle doit faire évoluer la variable testée ; sinon la condition reste vraie et la boucle ne s'arrête jamais : c'est une **boucle infinie**, qu'on interrompt avec Ctrl+C.

```
mot_de_passe = ""
while mot_de_passe != "nsi2026":
    mot_de_passe = input("Mot de passe : ")
print("Accès autorisé !")
```

Personne ne sait combien d'essais il faudra : c'est un cas pour `while`.

### ⑤ BREAK ET CONTINUE

- `break` sort immédiatement de la boucle : on s'arrête, par exemple, à la première voyelle trouvée.
- `continue` saute la fin du bloc et passe au tour suivant.
- `while True` : suivi d'un `break` exécute le bloc au moins une fois, par exemple pour redemander une saisie incorrecte.

### ⑥ BOUCLES IMBRIQUÉES

```
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i} x {j} = {i * j}")
```

La boucle intérieure s'exécute entièrement à chaque tour de la boucle extérieure :  $3 \times 3 = 9$  produits.

**⚠ Attention.** Erreurs fréquentes : écrire `range(1, 5)` pour aller jusqu'à 5 ; donner sa valeur de départ à l'accumulateur dans la boucle ; oublier de faire évoluer la condition d'un `while` ; croire que modifier `i` dans un `for` change le parcours.

#### 📌 À retenir.

- `for` : nombre de tours connu ; `while` : tant qu'une condition est vraie.
- `range(n)` donne  $0, 1, \dots, n - 1$  ; la fin est toujours exclue.
- Un accumulateur reçoit sa valeur de départ avant la boucle.