Boucles en Python

Activité – Première NSI

① Pourquoi répéter ?

Exercice 1 *Calculer une somme à la main*

On souhaite calculer la somme $1+2+3+\cdots+10$, c'est-à-dire la somme des dix premiers entiers strictement positifs.

1. Écrire un programme Python qui calcule cette somme **sans aucune boucle**, en additionnant les nombres un à un. Combien de lignes utilisez-vous ?
2. Si l'on devait calculer la somme $1+2+\cdots+1\,000$ avec la même méthode, combien de lignes faudrait-il ?
3. Quelle est la valeur exacte des deux sommes ? *Indication : la formule* $\dfrac{n(n+1)}{2}$ *donne la somme des* n *premiers entiers.*

Remarque 1. Une **boucle** permet de répéter automatiquement un bloc d'instructions. Python en propose deux : `for` (quand le nombre de répétitions est connu à l'avance) et `while` (quand la répétition continue tant qu'une condition est vraie).

② La boucle `for`

Définition 1 **(Boucle `for`)** La boucle `for` parcourt les éléments d'une séquence l'un après l'autre. La séquence la plus courante est produite par `range(n)`, qui génère les entiers $0, 1, 2, \ldots, n-1$.

```
for i in range(n):
    # bloc indenté répété n fois
```

À chaque itération, la variable `i` prend successivement les valeurs 0, 1, ..., $n-1$.

Exercice 2 *Comprendre `range`*

1. Écrire les valeurs successives prises par `i` dans chacune des boucles suivantes :
   a) `for i in range(5):`
   b) `for i in range(2, 7):`
   c) `for i in range(0, 10, 2):`
   d) `for i in range(10, 0, -1):`
2. Combien de tours effectue chacune de ces boucles ?
3. Que produit le programme suivant ?

```python
for i in range(4):
    print("*" * (i + 1))
```

Exercice 3 *Premier programme avec boucle*

Écrire un programme qui calcule la somme $1 + 2 + 3 + \cdots + n$ en utilisant une boucle `for`. Tester avec $n = 10$ (résultat attendu : 55) puis $n = 100$ (résultat attendu : 5050).

Indication : utiliser une variable accumulatrice initialisée à 0.

③ La boucle `while`

Définition 2 **(Boucle `while`)** La boucle `while` répète un bloc d'instructions **tant que** sa condition reste vraie.

```python
while condition:
    # bloc indenté répété tant que condition est vraie
```

Si la condition n'est jamais fausse, la boucle ne s'arrête jamais (boucle infinie).

Exercice 4 *Tracer une boucle `while`*

On considère le programme :

```python
n = 1
while n < 100:
    n = n * 2
print(n)
```

1. Compléter le tableau ci-dessous, indiquant la valeur de `n` avant chaque test de la condition.

Étape	`n`	Condition `n < 100`
Avant boucle	1	Vrai
Après tour 1	2	
Après tour 2		
Après tour 3		
Après tour 4		
Après tour 5		
Après tour 6		
Après tour 7		

2. Que vaut `n` à l'affichage final ?
3. Quelle propriété mathématique caractérise le `n` affiché ?

Exercice 5 *Compléter une boucle `while`*

On souhaite trouver le plus petit entier n tel que $n^2 > 5\,000$. Compléter le programme suivant :

```python
n = 1
while .............:
    n = n + 1
print(n)
```

Quel est le résultat affiché ? Justifier en quelques mots.

④ `for` ou `while` ?

Remarque 2 (Choisir la bonne boucle).

— `for` convient quand on connaît à l'avance **combien de fois** on va répéter le bloc (parcours d'une liste, énumération de `range`).
— `while` convient quand on répète tant qu'une **condition** reste vraie, sans savoir à l'avance combien d'itérations seront nécessaires.

Exercice 6 *Choisir la boucle adaptée*

Pour chacun des problèmes suivants, indiquer si l'on doit utiliser `for` ou `while`, et justifier en une phrase.

1. Calculer la somme des éléments d'une liste donnée.
2. Trouver le premier multiple de 7 supérieur à 1000 (en testant 1001, 1002, ...).
3. Afficher la table de multiplication par 9 (de 9×1 à 9×10).
4. Demander à l'utilisateur de saisir un mot de passe correct, en réessayant tant que la saisie est fausse.
5. Compter le nombre de `'a'` dans une chaîne de caractères.

⑤ Accumulateur, compteur, recherche d'extremum

Propriété 1 (Trois schémas classiques) La plupart des boucles utiles suivent l'un de ces trois schémas :

— **accumulateur** : une variable initialisée à 0 (ou 1 pour un produit) à laquelle on ajoute (ou multiplie) à chaque tour ;
— **compteur** : une variable initialisée à 0 que l'on incrémente lorsqu'une condition est satisfaite ;
— **recherche d'extremum** : une variable initialisée avec le premier élément, comparée à chaque élément suivant.

Exercice 7 *Trois schémas en pratique*

On considère la liste `notes = [12, 8, 15, 7, 18, 9, 14]`.

1. **Accumulateur.** Compléter le programme qui calcule la somme des notes.

```python
notes = [12, 8, 15, 7, 18, 9, 14]
somme = ............
for note in notes:
    somme = ............
print(somme)
```

2. **Compteur.** Compléter le programme qui compte combien de notes sont supérieures ou égales à 10.

```
notes = [12, 8, 15, 7, 18, 9, 14]
nb_bonnes = ............
for note in notes:
    if ............:
        nb_bonnes = ............
print(nb_bonnes)
```

3. **Recherche d'extremum.** Compléter le programme qui trouve la plus haute note.

```
notes = [12, 8, 15, 7, 18, 9, 14]
maxi = ............
for note in notes:
    if note > maxi:
        maxi = ............
print(maxi)
```

⑥ Boucles imbriquées

Exercice 8 *Une table de multiplication*

Le programme suivant utilise deux boucles `for` imbriquées :

```
for i in range(1, 4):
    for j in range(1, 4):
        print(i * j, end=" ")
    print()
```

1. Que produit ce programme à l'écran ? Détailler ligne par ligne.
2. Combien de fois la ligne `print(i * j, end=" ")` est-elle exécutée ?
3. Plus généralement, si l'on remplace `range(1, 4)` par `range(1, n + 1)` dans les deux boucles, combien de multiplications sont effectuées ?

⑦ Ce qu'il faut retenir

Propriété 2 **(`for` avec `range`)** `for i in range(n):` répète n fois le bloc indenté, avec `i` prenant successivement les valeurs $0, 1, \ldots, n-1$. La syntaxe `range(debut, fin, pas)` permet de personnaliser le départ, la borne d'arrêt (exclue) et le pas.

Propriété 3 **(`while` et terminaison)** `while condition:` répète le bloc *tant que* la condition est vraie. Il faut s'assurer que la condition finira par devenir fausse, sinon la boucle est infinie. Pour cela, le bloc doit modifier au moins une variable utilisée dans la condition.

Propriété 4 **(Trois schémas à connaître)**

— **Accumulateur** : `somme = 0` avant la boucle, `somme = somme + x` dans la boucle.

— **Compteur** : `n = 0` avant, `n = n + 1` dans la boucle si une condition est vraie.

— **Extremum** : `m = liste[0]` avant, `m = x` dans la boucle si `x > m` (ou `x < m`).

Remarque 3. Deux boucles imbriquées de n tours chacune effectuent n^2 itérations. Cela peut devenir très long pour de grandes valeurs de n : c'est la notion de **coût quadratique**, que l'on retrouvera notamment dans les tris naïfs.