Chapitre 5 : structure de contrôle, boucle

> **"Objectifs"**
>
> À la fin du chapitre, on doit savoir :
>
> - écrire une boucle `for` sans utilisation de la variable de boucle
> - écrire une boucle `for` avec utilisation de la variable de boucle
> - compter le nombre total d'itérations pour des boucles `for` successives ou imbriquées
> - écrire la condition d'entrée d'une boucle `while` à partir de sa condition de sortie
> - choisir de préférence une boucle `for` si le nombre d'itérations est connu et ne dépend pas d'une condition.

Boucle bornée `for`

> **"Définition 1"**
>
> Soit `n` un entier positif, pour répéter `n` fois le même bloc d'instruction `bloc_boucle`, on utilise l'instruction de contrôle d'une **boucle `for`** avec la syntaxe :
>
> ```
> bloc_avant
> for i in range(n):
>     bloc_boucle
> bloc_apres
> ```
>
> Comme pour une instruction conditionnelle, l'instruction de boucle `for` se termine par le caractère `:` et le bloc qu'elle contrôle est décalé d'une indentation par rapport à l'indentation du `for`.
>
> Une répétition de `bloc_boucle` s'appelle une **itération**.
>
> La variable de boucle `i` peut être ou non utilisée dans `bloc_boucle`, sa valeur est incrémentée de 1 à chaque itération, en commençant à 0. Lors de la dernière itération, la valeur de la variable de boucle est donc `n - 1`.

Le nombre d'itérations d'une boucle `for` est connu à l'avance, on parle de **boucle bornée**.

"Exemple 1"

Programme A

```
for i in range(30):
    print("Bonjour.")
```

Le programme A, affiche 30 fois `"Bonjour."`, la variable de boucle n'est pas utilisée.

Programme B

```
for i in range(30):
    print("Bonjour ", i, " fois.")
```

Le programme **B** affiche :

```
Bonjour 0 fois.
Bonjour 1 fois.
...
Bonjour 29 fois.
```

La variable de boucle est utilisée à chaque itération.

"Exercice 1"

1. Écrire un programme qui affiche trente `'*'` en allant à la ligne pour chaque caractère.

2. Écrire un programme qui demande à l'utilisateur de saisir un entier positif `n`, qui calcule la somme des carrés des entiers successifs de 1 `n` inclus, et qui affiche le résultat

3. Compter le nombre d'étoiles `'*'` affichées par le programme C puis par le programme D.

   Programme C

   ```
   for i in range(300):
       print('*')
   for j in range(100):
       print('*')
   ```

   Programme D

   ```
   for i in range(300):
       for j in range(100):
           print('*')
   ```

"Syntaxe avancée du `range`"

Dans l'instruction de boucle `for i in range(n)`, on peut remplacer `range(n)` par `range(0, n)`.

Plus généralement, la syntaxe complète de `range` est :

```
range(debut, fin, pas)
```

On peut considérer que `range(debut, fin, pas)` produit une séquence d'entiers :

- qui commence à `debut` ;
- qui avance avec un pas de `pas` ;
- qui s'arrête avant d'atteindre `fin`.

La valeur `debut` est incluse, mais la valeur `fin` est exclue.

Les paramètres `debut` et `pas` sont optionnels :

Syntaxe	Signification
`range(fin)`	entiers de `0` inclus à `fin` exclu
`range(debut, fin)`	entiers de `debut` inclus à `fin` exclu
`range(debut, fin, pas)`	entiers de `debut` inclus à `fin` exclu, avec un pas de `pas`

"Attention"

Si le pas est positif, les valeurs augmentent. Si le pas est négatif, les valeurs diminuent. Dans tous les cas, la valeur `fin` n'est jamais atteinte.

"Exercice 2"

Compléter le tableau suivant.

Expression	Séquence d'entiers produite
`range(5)`	_____
`range(0, 5)`	_____
`range(2, 7)`	_____
`range(2, 10, 2)`	_____
`range(1, 10, 3)`	_____
`range(7, 0, -1)`	_____
`range(7, 0, -2)`	_____
`range(5, 5)`	_____
`range(2, 10, -1)`	_____
`range(10, 2, 1)`	_____

Boucle non bornée `while`

"Définition 2"

Une **boucle** `while` permet de répéter un bloc d'instructions tant qu'une condition est vraie.

Sa syntaxe est :

```
bloc_avant
while condition:
    bloc_boucle
bloc_apres
```

Avant chaque itération, Python évalue la condition.

- Si la condition vaut `True`, le bloc de la boucle est exécuté.
- Si la condition vaut `False`, la boucle s'arrête et l'exécution continue après le bloc de boucle.

"Exemple 2"

Le programme suivant affiche les entiers de `0` à `4`.

```
i = 0
while i < 5:
    print(i)
    i = i + 1
print("Fin")
```

La variable `i` est initialisée avant la boucle, puis modifiée à chaque itération.

La boucle s'arrête lorsque la condition `i < 5` devient fausse, c'est-à-dire lorsque `i >= 5`.

On peut obtenir le même affichage avec une boucle `for`. C'est plus simple car la gestion de la variable de boucle est déléguée au `range`.

```
for i in range(5):
    print(i)
```

"Attention"

Dans une boucle `while`, il faut vérifier que la condition finit par devenir fausse. Sinon, la boucle ne s'arrête jamais, c'est une **boucle infinie**.

"Choisir entre `for` et `while`"

On peut simuler n'importer quelle boucle `for` avec une boucle `while` mais les boucles `for` sont plus simples à écrire. On utilisera :

- une boucle `for` lorsque le nombre d'itérations est connu à l'avance ;
- une boucle `while` lorsque le nombre d'itérations n'est pas connu à l'avance et dépend d'une condition.

Par exemple, si on veut demander un mot de passe jusqu'à ce qu'il soit correct, on ne sait pas à l'avance combien d'essais seront nécessaires. Une boucle `while` est nécessaire.

```
mot = ""
while mot != "NSI":
    mot = input("Mot de passe ? ")
print("Accès autorisé")
```

Méthode : trouver la condition d'entrée dans une boucle `while`

"Expression de la condition d'entrée dans une boucle `while`"

Si on veut répéter une action **jusqu'à ce que** la condition `C` soit vraie, alors on continue **tant que** `C` est fausse.
On écrit donc :

```
while not C:
    bloc_boucle
```

La condition `C` est la **condition de sortie** de boucle et `not C` est la **condition d'entrée** de boucle.

Ainsi, la **condition d'entrée** dans une boucle `while` est la **négation de sa condition de sortie**.

"Exercice 3'

Pour chaque situation, écrire la condition d'entrée de la boucle `while`.

1. On veut répéter tant que `x` n'est pas égal à `0`.
   Condition de sortie : `x == 0`
   Condition d'entrée :

---

2. On veut répéter jusqu'à ce que `age` soit supérieur ou égal à `18`.
   Condition de sortie : `age >= 18`
   Condition d'entrée :

---

3. On veut répéter jusqu'à ce que `mot` soit égal à `"oui"`.
   Condition de sortie : `mot == "oui"`
   Condition d'entrée :

---

4. On veut répéter jusqu'à obtenir un entier `n` compris entre `1` et `10`.
   Condition de sortie : `(n >= 1) and (n <= 10)`
   Condition d'entrée :

---

5. On veut répéter jusqu'à obtenir une réponse égale à `"oui"` ou à `"non"`.
   Condition de sortie : `(reponse == "oui") or (reponse == "non")`
   Condition d'entrée :

"Exercice 4"

Pour chaque boucle `while`, indiquer quand la boucle s'arrête.

Programme A

```
x = 0
while x < 10:
    x = x + 1
```

La boucle s'arrête quand :

Programme B

```
x = 20
while x >= 0:
    x = x - 2
```

La boucle s'arrête quand :

Programme C

```
mot = ""
while mot != "stop":
    mot = input("Mot ? ")
```

La boucle s'arrête quand :

Programme D

```
n = int(input("Entier ? "))
while (2 < n) and (n < 15):
    n = int(input("Entier ? "))
```

La boucle s'arrête quand :

Programme E

```
spe = input("Spécialité ? ")
while (spe != "NSI") and (reponse != "MATHS"):
    reponse = input("Réponse ? ")
```

La boucle s'arrête quand :

Erreurs liées aux boucles

"Erreur de boucle infinie"

Une **boucle infinie** est une boucle qui ne s'arrête jamais.
Ce n'est pas une erreur de syntaxe Python : le programme peut être correct du point de vue de Python, mais incorrect du point de vue logique. Cela peut se produire avec une boucle `while` lorsque la condition de sortie de boucle n'est jamais réalisée.

Par exemple, la boucle du programme suivant est infinie, la variable `i` n'atteint jamais 10 car on a oublié l'incrémentation `i = i +1` dans le bloc de la boucle.

```
i = 0
while i < 10:
    print(i)
```

"Erreur `TypeError` avec `range`"

La fonction `range` attend des paramètres entiers. Si l'un des paramètres n'est pas de type `int`, Python provoque une erreur `TypeError`.

Par exemple, l'exécution du code ci-dessous :

```
for i in range(3.5):
    print(i)
```

provoque le message d'erreur suivant :

```
TypeError: 'float' object cannot be interpreted as an integer
```

"Exercice 5"

Pour chaque programme, dire s'il provoque une erreur, une boucle infinie, ou s'il s'exécute normalement.

Programme A

```
for i in range(5):
    print(i)
```

Programme B

```
n = input("Nombre ? ")
for i in range(n):
    print(i)
```

Programme C

```
i = 11
while i != 0:
    i = i - 2
```

Programme D

```
i = 16
while i != 0:
    i = i // 2
```
