

Cours 5

RÉPÉTITIVE

Retour sur le dernier cours

- Alternative généralisée
 - Évaluation d'une condition (type booléen)
 - Autant de bloc d'instructions que nécessaire
 - Cas par défaut avec un `else` à la fin
 - Indentation du code pour délimiter les blocs d'instructions
- Imbrication d'alternatives

Répétitive

LES BOUCLES

Définition

En programmation, une **boucle** est une **structure de contrôle** qui permet de répéter l'exécution d'un bloc de code tant qu'une certaine condition est vérifiée ou pour un nombre déterminé d'itérations.

Les boucles sont essentielles pour **automatiser** des **tâches répétitives** et **manipuler des collections de données**.

En Python, il existe deux types principaux de boucles : la boucle **for** et la boucle **while**. Chaque type a ses propres avantages et est utilisé dans des contextes différents.

Pourquoi utiliser des répétitives?

Automatisation

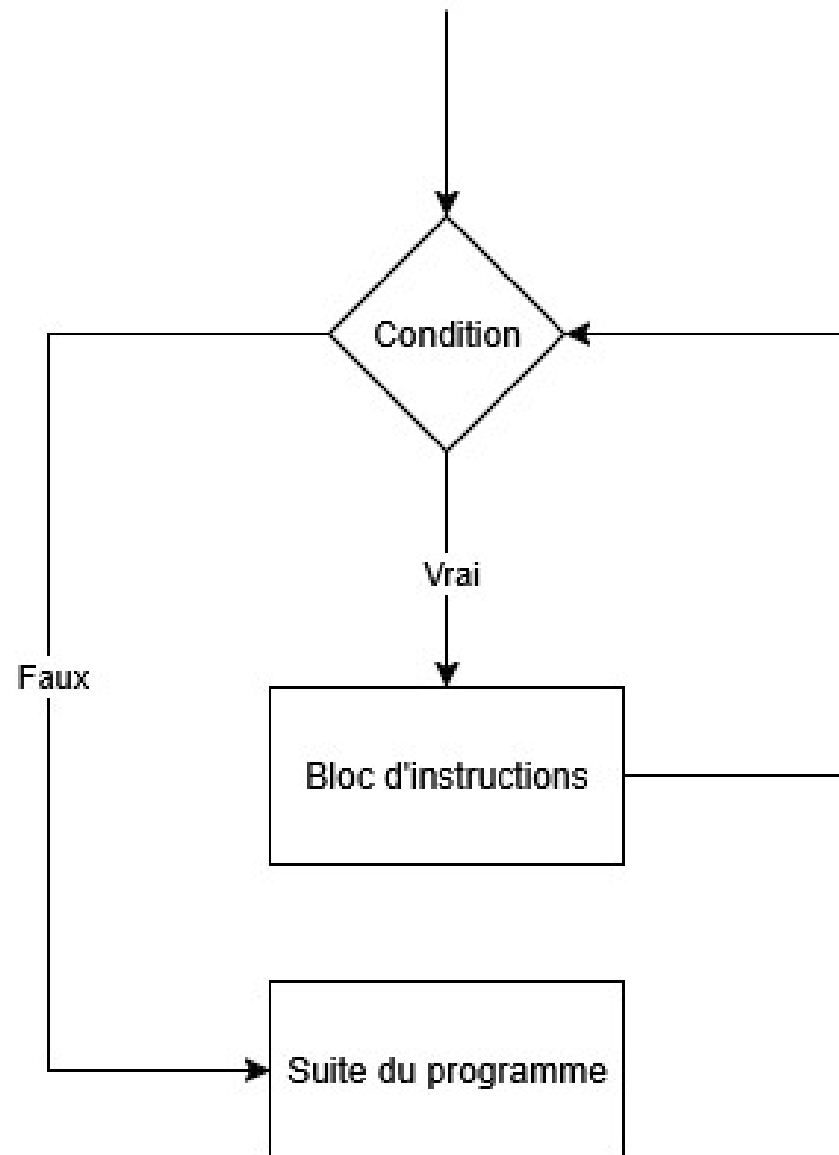
Elles permettent de répéter des actions sans avoir à dupliquer le code.

Traitement de données

Idéal pour parcourir des structures de données comme des listes, des tableaux ou des dictionnaires.

Efficacité

Permet d'effectuer en quelques lignes des tâches répétitives!



Schéma

La boucle `for`

La boucle `for` en Python est souvent utilisée pour **itérer sur une séquence**, comme une plage de nombres (générée par `range(nombre)`).

`range` ici nous permet d'itérer sur les valeurs numériques de **0 à n-1** (donc ici de 0 à 4)

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

```
0  
1  
2  
3  
4
```

La boucle `for`

Il est à noter qu'il n'y a aucune obligation de toujours avoir la valeur 0 comme valeur initiale pour le compteur.

Par exemple, pour écrire une boucle `for` dont le compteur varie **entre 5 et 10**, on écrit :

```
for i in range(5, 11) :  
    print(i)
```

```
5  
6  
7  
8  
9  
10
```

La boucle `while`

La boucle `while` continue d'exécuter un bloc d'instructions **tant qu'une condition est vraie**. Elle permet d'exécuter une séquence d'actions plusieurs fois sans avoir à répéter le code.

Cette boucle est idéale lorsqu'on ne sait pas exactement combien de fois on doit répéter une séquence d'instructions.

```
i : int = 0 # Initialisation du compteur
while i < 5: # Condition d'arrêt
    print(i)
    i += 1 # incrémentation
```

```
0
1
2
3
4
```

Exemple – La sommation

```
n : int = int(input("Veuillez entrer le nombre maximum à additionner: "))

i : int = 1
somme : int = 0

while i <= n:
    somme = somme + i
    i = i + 1

print(f"La somme des entiers de 1 à {n} est {somme}")
```

[Veuillez entrer le nombre maximum à additionner: 5]
La somme des entiers de 1 à 5 est 15

La boucle while

```
mot_de_passe : str = "SikritCode123!«  
  
entree_MDP : str = input("Entrez le mot de passe: ")  
  
while entree_MDP != mot_de_passe:  
    print("Mot de passe erroné! Essayez à nouveau!")  
    entree_MDP = input("Entrez le mot de passe: ")  
  
print("Accès autorisé!")
```

```
[Entrez le mot de passe: chocolat123!]  
Mot de passe erroné! Essayez à nouveau!  
[Entrez le mot de passe: SikritCode123!]  
Accès autorisé!
```

Boucle infinie

Il est important de s'assurer que les conditions des boucles sont bien définies. Une boucle sans fin peut se produire si la condition ne devient jamais fausse. Les exemples suivants créeront une boucle infinie car les conditions sont toujours vraies.

```
while True:  
    print("Cette boucle est infinie!")
```

```
Cette boucle est infinie!  
Cette boucle est infinie!  
Cette boucle est infinie!  
Cette boucle est infinie!  
Cette boucle est infinie!  
...
```

Boucle infinie

Une erreur courante causant une boucle infinie est l'oubli de faire l'incrémentation dans un while!

```
i : int = 0
n : int = 15

while i < n:
    print(i)
```

```
0
0
0
0
0
...
```

Résumé

En conclusion, comprendre et maîtriser les boucles est essentiel pour écrire des programmes efficaces et robustes. Elles vous permettent d'**automatiser des tâches répétitives**, de gérer des données complexes et d'optimiser le flux d'exécution de votre code.

Exercices

Exercice #1

Créer un programme qui demande à l'utilisateur de saisir un nombre entier et qui affiche la factorielle résultante de ce nombre en présentant la démarche des calculs à l'utilisateur.

Veuillez saisir un nombre: 7

Démarche des calculs:

$$1 \times 1 = 1$$

$$1 \times 2 = 2$$

$$2 \times 3 = 6$$

$$6 \times 4 = 24$$

$$24 \times 5 = 120$$

$$120 \times 6 = 720$$

$$720 \times 7 = 5040$$

Résultat

```
nombre : int = int(input("Veuillez saisir le nombre: "))
resultat : int = 1

print("Démarche des calculs:")

for i in range(1, nombre + 1):
    ancien_resultat : int = resultat
    resultat = resultat * i
    print(f"{ancien_resultat } x {i} = {resultat}")
```

Exercice #2

Créer un programme qui demande à l'utilisateur de saisir un nombre et qui affiche cette quantité de nombres de la suite de Fibonacci en partant du premier nombre.

Notez: La suite de Fibonacci est une suite de nombres entiers où chaque terme après les deux premiers est la somme des deux termes précédents.

```
Veillez saisir un nombre : 8
0 1 1 2 3 5 8 13 21
```

La suite de Fibonacci (F_n) est définie par :

$$F_0 = 0, \quad F_1 = 1$$

et pour $n \geq 2$:

$$F_n = F_{n-1} + F_{n-2}$$

Voici les premiers nombres de Fibonacci en exemple:

Valeur de n	0	1	2	3	4	5	6	7	8	9	10
$F(n)$	0	1	1	2	3	5	8	13	21	34	55

Résultat

```
f0 : int = 0
f1 : int = 1
nombreIterations : int = int(input("Veuillez saisir un nombre: "))
resultat : str = ""
for i in range(nombreIterations + 1) :
    resultat += f"{f0} "
    ftemp : int = f0
    f0 = f1
    f1 = ftemp + f1
print(resultat)
```

Résultat (version while)

```
f0 : int = 0
f1 : int = 1

nombreIterations : int = int(input("Veuillez saisir un nombre: "))

n : int = 0
resultat : str = ""
while n <= nombreIterations :
    resultat += f"{f0} "
    ftemp : int = f0
    f0 = f1
    f1 = ftemp + f1
    n = n + 1

print(resultat)
```

Laboratoire #4