COURS

# Boucles non bornées

Répéter des instructions avec while, suivre l'évolution des variables et comprendre pourquoi une boucle s'arrête.

Nous savons déjà exécuter des instructions et choisir un traitement avec `if`. Nous allons maintenant apprendre à **répéter un traitement tant qu'une condition est vraie**.

Pour ce chapitre, il faut savoir utiliser les variables, les opérateurs, les fonctions et les instructions conditionnelles.

## I. Définitions

> **Définition - Boucle**
>
> **Construction permettant de répéter une séquence d'instructions.**
>
> Une boucle permet de répéter une séquence d'instructions. Cette séquence s'appelle le **corps de la boucle**.

> **Définition - Itération**
>
> **Un tour de la boucle.**
>
> Une itération correspond à une exécution du corps de la boucle. Si le corps est exécuté trois fois, la boucle effectue trois itérations.

> **Définition - Boucle non bornée**
>
> **Boucle dont la répétition est contrôlée par une condition.**
>
> Une boucle non bornée répète son corps tant qu'une condition est vraie. La poursuite de la répétition dépend de cette condition, et non d'un nombre d'itérations fixé dans l'instruction de boucle.

On l'utilise notamment lorsque le nombre de répétitions n'est pas connu à l'avance : redemander une saisie jusqu'à ce qu'elle soit valide, ou poursuivre un calcul jusqu'à atteindre un seuil.

> **Attention**
>
> **Non bornée ne signifie pas infinie.** Une boucle non bornée peut s'arrêter. Pour des exemples simples, on peut même prévoir son nombre d'itérations en étudiant le programme.

- [ ] Je sais définir une boucle, son corps et une itération.

# II. Écrire une boucle avec while

## A. Syntaxe en Python

En Python, le mot-clé `while` se lit « tant que ». Il est suivi d'une condition, puis de deux-points `:`. Le corps est indenté, généralement de quatre espaces.

### Avancer jusqu'à une distance donnée

PYTHON

```
distance = 0
while distance < 12:
    distance = distance + 4
    print(distance)
print("Arrivée")
```

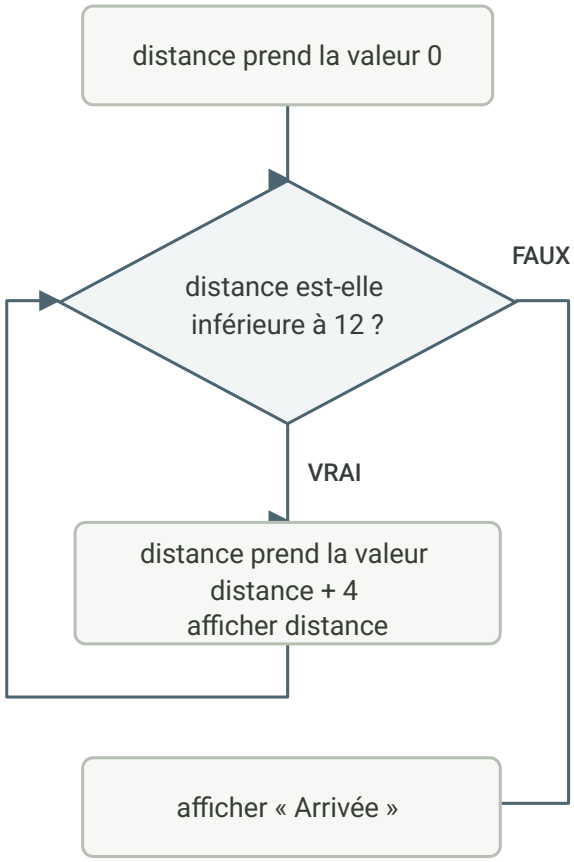
La traduction en français est :

PSEUDO

```
distance prend la valeur 0
Tant que distance est inférieure à 12 :
    distance prend la valeur distance + 4
    afficher distance
afficher "Arrivée"
```

Son schéma de branchement est :

### Avancer jusqu'à une distance donnée




Après le corps, revenir tester la condition.

Affichage obtenu après l'exécution :

```
TEXT
4
8
12
Arrivée
```

**Attention**

`print` sert ici à afficher la valeur de la variable, puis une chaîne de caractères. Cet affichage se fait dans la console.

Les deux instructions indentées sont répétées. L'affichage de `"Arrivée"`, placé après la boucle, n'a lieu qu'une fois.

### B. Déroulement de l'exécution

1. Le programme initialise les variables nécessaires avant la boucle.
2. Il évalue la condition.
3. Si elle est vraie, il exécute **tout le corps**, puis revient tester la condition.
4. Si elle est fausse, il passe à la première instruction située après la boucle.

La condition est donc testée **avant chaque itération**. Elle n'est pas retestée entre les instructions du corps.

La flèche qui revient vers le test représente la répétition.

- □ Je sais reconnaître le corps d'une boucle while et expliquer le retour vers le test.

### C. Si la condition est fausse dès le départ

> **Exemple**
>
> PYTHON
> ```
> nombre = 7
> while nombre < 4:
>     nombre = nombre + 1
> print(nombre)
> ```
>
> Le programme affiche 7. Le premier test est faux : le corps n'est jamais exécuté. Il y a **zéro itération**.

- □ Je sais reconnaître une boucle dont le corps ne sera pas exécuté.

## III. Construire une trace d'exécution

Une trace permet de suivre les valeurs des variables et les résultats des tests. Elle doit aussi montrer **le dernier test, celui qui provoque la sortie**.

Reprenons l'exemple de la distance :

| Test de la condition | Valeur de distance avant le test | Résultat de distance < 12 | Action |
|---|---|---|---|
| 1 | 0 | `True` | distance devient 4 ; afficher 4 |
| 2 | 4 | `True` | distance devient 8 ; afficher 8 |
| 3 | 8 | `True` | distance devient 12 ; afficher 12 |

| Test de la condition | Valeur de distance avant le test | Résultat de distance < 12 | Action |
|---|---|---|---|
| 4 | 12 | `False` | Sortir de la boucle |

Après la sortie, le programme affiche `"Arrivée"`.

> **À retenir**
>
> Dans une boucle qui se termine normalement parce que sa condition devient fausse, le test est effectué une fois de plus que le corps : ici, **quatre tests pour trois itérations**.

> **Application 1**
>
> Pour chacun des programmes suivants :
>
> 1. Dessiner le schéma de branchement, en faisant apparaître le retour vers le test.
> 2. Construire la trace d'exécution, dernier test compris.
> 3. Donner le nombre d'itérations et la valeur affichée.
> 4. Vérifier le résultat dans Thonny.
>
> **Programme 1**
>
> PYTHON
> ```
> n = 2
> while n < 11:
>     n = n + 3
> print(n)
> ```
>
> **Programme 2**
>
> PYTHON
> ```
> n = 9
> while n > 1:
>     n = n - 2
> print(n)
> ```

□ Je sais construire une trace avec les tests, les itérations et les valeurs finales.

# IV. Distinguer if et while

`if` et `while` utilisent tous les deux une condition, mais ils n'ont pas le même rôle.

| Avec if | Avec while |
|---|---|
| Le bloc est exécuté au plus une fois lorsque le programme atteint cette instruction. | Le bloc peut être répété plusieurs fois. |
| Après le bloc, l'exécution continue à la suite. | Après le corps, l'exécution revient vers le test de la boucle. |

**Même condition, exécutions différentes**

Avec `if` :

```python
n = 0
if n < 3:
    n = n + 1
print(n)
```

Le programme affiche `1`.

Avec `while` :

```python
n = 0
while n < 3:
    n = n + 1
print(n)
```

Le programme affiche `3`.

□ Je sais expliquer la différence entre if et while.

# À retenir

**À retenir**

- `while` répète un corps tant que sa condition est vraie.
- Le test précède chaque itération : le corps peut ne jamais s'exécuter.
- L'indentation délimite les instructions répétées.

Poursuivre avec les exercices sur les boucles non bornées.