# Structures répétitives

## Principes

Les structures répétitives, également appelées **boucles**, permettent de répéter automatiquement un bloc d'instructions tant qu'une condition est vraie. Elles sont essentielles en programmation pour automatiser les tâches répétitives et rendre le code plus concis et efficace.

**Pourquoi utiliser les structures répétitives ?** - Éviter les redondances dans le code. - Automatiser les calculs ou actions répétitives. - Faciliter la gestion de grandes quantités de données.

Les boucles se basent sur une **condition logique** (exemple : `x < 10`) qui est évaluée à chaque itération : - Si la condition est **vraie**, le bloc d'instructions est exécuté. - Si elle est **fausse**, la boucle s'arrête.

Voici un schéma simplifié illustrant le fonctionnement d'une boucle :

**Exemple pratique** : Un programme qui affiche les nombres de 1 à 5 :

En **Python** : ```python i = 1 while i ⇐ 5:

```
  print(i)
  i += 1
  for (int i = 1; i <= 5; i++) {
  System.out.println(i);
```

}

## Condition

Voir Structures conditionnelles Les structures répétitives reposent sur une **condition logique** qui détermine si la boucle doit continuer ou s'arrêter. Cette condition est évaluée à chaque itération.

**Principe** : - Si la condition est **vraie**, le bloc d'instructions s'exécute. - Si elle est **fausse**, la boucle s'arrête.

Exemple en **Python** (affiche les nombres inférieurs à 5) : ```python x = 0 while x < 5:

```
  print(x)
  x += 1
  int x = 0;
```

while (x < 5) {

```
  System.out.println(x);
  x++;
```

### Forme Pour

La boucle **Pour** (ou `for`) est utilisée lorsque le nombre d’itérations est connu à l’avance. Elle suit généralement ce schéma :

1. **Initialisation** : Définir une variable de contrôle (ex. : `i = 0`). 2. **Condition** : Vérifier si la boucle doit continuer (ex. : `i < 10`). 3. **Incrémentation/Décrémentation** : Modifier la variable à chaque itération (ex. : `i++`).

**Syntaxe générale** : ```plaintext Pour (initialisation; condition; incrémentation) {

```
  instructions;
```

}

for i in range(5):

```
  print(i)
  for (int i = 0; i < 5; i++) {
  System.out.println(i);
```

}

# Forme Pour

### Forme TantQue

La boucle **TantQue** (ou `while`) est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle répète un bloc d'instructions tant qu'une **condition logique** est vraie.

**Syntaxe générale** : ```plaintext TantQue (condition) {

```
  instructions;
```

}

x = 0 while x < 5:

```
  print(x)
  x += 1
  int x = 0;
```

while (x < 5) {

```
  System.out.println(x);
  x++;
```

}

# Forme TantQue

# Forme Répéter-Jqa

La boucle **Répéter-Jusqu'à** (ou `do-while` dans certains langages) exécute un bloc d'instructions **au moins une fois**, puis vérifie une condition pour décider si elle doit continuer.

**Syntaxe générale** : ```plaintext Répéter {

```
  instructions;
```

} Jusqu'à (condition);

x = 0 while True:

```
  print(x)
  x += 1
  if x >= 5:
      break
      int x = 0;
```

do {

```
  System.out.println(x);
  x++;
```

} while (x < 5);

# Forme généralisée

Les boucles peuvent être combinées et imbriquées pour résoudre des problèmes plus complexes. Cela permet de parcourir plusieurs dimensions de données ou d'exécuter des tâches répétitives liées.

**Principe** : Une boucle généralisée peut être représentée par : - Une boucle simple : effectue une seule tâche répétitive. - Des **boucles imbriquées** : une boucle à l'intérieur d'une autre.

**Exemple : Boucle imbriquée pour une table de multiplication (en Python)** : ```python for i in range(1, 6):

```
  for j in range(1, 6):
      print(f"{i} x {j} = {i * j}")
  print("---")
  for (int i = 1; i <= 5; i++) {
  for (int j = 1; j <= 5; j++) {
      System.out.println(i + " x " + j + " = " + (i * j));
  }
```

```
  System.out.println("---");

}
```

From:
https://wiki.siochaptalqper.fr/ - **Wiki SIO Chaptal**

Permanent link:
**https://wiki.siochaptalqper.fr/doku.php?id=bloc1:prog:boucles&rev=1736096138**

Last update: **2025/01/05 17:55**