ALGORITHMIQUE · CHAPITRE 25 · FONCTIONS

Choisir entre itération et récursivité

Comparez une somme calculée par boucle et par récursivité. Vérifiez les mêmes résultats et expliquez les états conservés et les limites d'exécution.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Exprimer une même somme avec une boucle ou des appels récursifs.
- Comparer les traces plutôt que la seule longueur du code.
- Choisir une formulation adaptée à l'explication et aux limites de la machine.

Prérequis. Savoir tracer une boucle et les appels récursifs de sum_to.

Même résultat, même exécution ?

Pour n = 3, la version récursive renvoie 6. Une version itérative peut produire la même somme en conservant un total et une valeur courante. L'égalité du résultat ne signifie pas que les étapes ou les informations en attente sont identiques.

Construire un total ou attendre un sous-résultat

Deux façons d'obtenir la somme jusqu'à 3

Version itérative	Version récursive
total vaut successivement 0, 1, 3, 6	Les appels descendent de 3 à 0
Chaque tour met à jour le total	Les résultats remontent : 0, 1, 3, 6

Version itérative	Version récursive
Pas d’appel récursif	Des appels attendent le résultat des suivants

Les additions ne sont pas effectuées dans le même ordre de lecture. Pour les petits entiers de cet exercice, les deux méthodes ont le même résultat exact. Nous ne généralisons pas cette observation à toutes les réorganisations de calculs en virgule flottante.

Comparer le même contrat avant de comparer le style

L'**itération** répète des instructions à l'aide d'une boucle. La **récursivité** exprime le résultat à partir d'appels plus petits de la même fonction. Une comparaison utile exige la même entrée et la même sortie : ici, la somme des entiers de 1 à n, avec 0 pour n égal à zéro.

La version itérative conserve un total partiel et ajoute les valeurs dans l'ordre croissant. La version récursive conserve des appels en attente et recompose le résultat après le cas de base. Pour n égal à 3, les deux renvoient 6, mais leurs traces ne sont pas identiques. L'équivalence recherchée concerne le résultat sur le domaine annoncé, pas l'identité de toutes les étapes internes.

Une **limite d'exécution** est une contrainte de l'environnement, comme le nombre d'appels imbriqués autorisé ou la précision des valeurs numériques. Nos exemples utilisent de petits entiers. On ne peut pas conclure qu'une version est plus rapide simplement parce qu'elle comporte moins de lignes ; les appels, les boucles et les affichages ont des coûts différents.

Pour une accumulation linéaire aussi simple, une boucle rend l'état partiel très explicite et évite la profondeur d'appels de cette version récursive. Cela ne signifie pas que toute récursivité est mauvaise : le choix doit correspondre à la structure du problème et à l'objectif d'explication, pas à une règle de préférence sans contexte.

Une fonction itérative

Pseudo-code

```
// example.pseudo
// sum_to_iterative
FUNCTION sum_to_iterative(n)
    total <- 0
    FOR value FROM 1 TO n INCLUSIVE DO
        total <- total + value
    END FOR
    RETURN total
END FUNCTION
```

Python

```python
# sum_to_iterative.py
def sum_to_iterative(n):
    total = 0
    for value in range(1, n + 1):
        total = total + value
    return total

result = sum_to_iterative(3)
print(result)  # 6
```

MATLAB

```
% sum_to_iterative.m
function total = sum_to_iterative(n)
    total = 0;
    for value = 1:n
        total = total + value;
    end
end
```

La fonction conserve la même entrée et le même résultat attendu que `sum_to`, mais remplace les appels récursifs par une boucle. En MATLAB, enregistrez-la dans `sum_to_iterative.m`.

MATLAB

```
% use_sum_to_iterative.m
result = sum_to_iterative(3);
disp(result); % 6
```

Comparer sur un domaine commun

Calculez à la main les résultats attendus pour 0, 1 et 4, puis essayez les deux versions : 0, 1 et 10. Pour n = 3, la boucle effectue trois additions ; la définition récursive effectue aussi trois additions, mais déclenche quatre appels en comptant le cas de base.

Choisir une explication qui reste maîtrisable

Pour cette somme simple, la boucle exprime directement l'accumulation et évite les appels en attente. La définition récursive met en évidence la relation entre n et n − 1. Les deux éclairent le problème sous un angle différent. Une profondeur d'appels importante peut atteindre une limite d'exécution ; il ne faut pas supposer qu'un langage transforme automatiquement cette récursion en boucle.

Préserver le contrat lors d’une réécriture

▼ **Peut-on remplacer le résultat initial zéro par un pour raccourcir un cas ?**

Pas sans changer les autres étapes : pour n = 0, on renverrait un au lieu de zéro. La réécriture doit préserver le comportement sur tout le domaine annoncé, pas seulement sur un exemple positif.

▼ **Une version plus courte est-elle forcément plus facile à comprendre ?**

Non. Il faut pouvoir expliquer ses entrées, ses étapes, son arrêt et son résultat. La concision peut être utile, mais elle ne remplace ni la lisibilité ni la vérification des cas limites.

Comparer avant de préférer

Vous pouvez maintenant comparer une boucle et une récursion sur un petit problème sans étudier les arbres ou les graphes. La suite du parcours rassemble les méthodes de vérification, de justification et de construction d'un programme complet.

À retenir - fiche mémo

Même contrat

Comparez d’abord les résultats attendus sur les mêmes entrées.

Itération

La boucle conserve un total partiel au fil des valeurs.

Récursivité

La version récursive conserve des appels en attente jusqu’au cas de base.

Pseudo-code

```
// example.pseudo
// iterative_sum
total <- 0
FOR value FROM 1 TO n INCLUSIVE DO
    total <- total + value
END FOR
```

Point de vigilance. Moins de lignes ne prouve ni une meilleure vitesse ni une profondeur d’appels illimitée.

Consulter la cheatsheet : Récursivité · **Télécharger la fiche du thème en PDF**

Version web : https://www.clearcode.fr/courses/programming/algorithmic/functions/recursion/iteration-or-recursion