ALGORITHMIQUE · CHAPITRE 22 · FONCTIONS

Comprendre un appel récursif

Découvrez la récursivité avec une somme de petits entiers. Comprenez comment un appel plus simple produit le résultat attendu en Python et MATLAB.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Identifier le même problème sur une entrée plus petite.
- Distinguer appel récursif et répétition d'une instruction.
- Suivre un premier calcul jusqu'au cas de base.

Prérequis. Savoir lire une condition et une fonction avec paramètre et résultat.

Une somme à construire

Nous cherchons la somme des entiers de 1 à n, avec n entier non négatif et petit pour nos expériences. Pour n = 3, elle vaut `1 + 2 + 3 = 6`. Nous connaissons une solution avec une boucle. Peut-on exprimer ce calcul à partir du même problème posé pour n − 1 ?

Réduire le problème

La somme jusqu'à 3 vaut 3 plus la somme jusqu'à 2. Celle jusqu'à 2 vaut 2 plus la somme jusqu'à 1. On retrouve le **même problème** sur une entrée plus petite. Pour zéro, il n'y a aucun terme : nous choisissons directement le résultat zéro. Cette réponse immédiate est le **cas de base**.

Définir une fonction récursive sur un petit domaine

Une fonction est **récursive** lorsqu'elle s'appelle elle-même, directement ou à travers d'autres appels. Nous utilisons ici un appel direct : `sum_to(n)` calcule la somme des entiers de 1 à n pour un petit entier non négatif. Le contrat donne 0 pour n égal à zéro : aucun entier positif n'est alors ajouté.

Le **cas de base** est un cas dont le résultat est obtenu sans nouvel appel récursif. Ici, `sum_to(0)` renvoie 0. Le **cas récursif** réduit le problème : pour n positif, le résultat est n ajouté à `sum_to(n - 1)`. La **progression** est le passage d'un entier positif à un entier plus petit qui rejoint zéro.

Pour calculer `sum_to(3)`, il faut connaître `sum_to(2)`, qui dépend de `sum_to(1)`, puis de `sum_to(0)`. Les résultats reviennent ensuite dans l'ordre inverse : 0, 1, 3 et 6. L'appel initial ne reçoit pas 0 comme résultat final ; il utilise les retours intermédiaires pour compléter ses additions.

Ce premier exemple suppose le domaine annoncé et ne contrôle pas lui-même toutes les entrées. Un nombre négatif ou fractionnaire n'appartient pas au contrat. Les environnements ont aussi des limites de profondeur d'appel et de représentation des nombres. Nous gardons de petites entrées pour comprendre le mécanisme ; la récursivité n'autorise pas une profondeur arbitraire.

Une fonction qui s’appelle elle-même

Pseudo-code

```
// example.pseudo
// sum_to
FUNCTION sum_to(n)
    IF n == 0 THEN
        RETURN 0
    END IF
    RETURN n + sum_to(n - 1)
END FUNCTION
```

Python

```python
# sum_to.py
def sum_to(n):
    if n == 0:
        return 0
    return n + sum_to(n - 1)

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

MATLAB

```matlab
% sum_to.m
function result = sum_to(n)
    if n == 0
        result = 0;
        return;
    end
    result = n + sum_to(n - 1);
end
```

Le nom de la fonction apparaît dans son propre corps. L'appel `sum_to(n - 1)` doit fournir une valeur avant que l'addition avec n puisse être terminée. En MATLAB, conservez la fonction dans `sum_to.m` et lancez l'appel séparément.

MATLAB

```matlab
% use_sum_to.m
result = sum_to(3);
disp(result); % 6
```

Ne pas imaginer une réponse simultanée

Descendre vers le cas de base

Appel	Ce qu’il attend
sum_to(3)	3 + le résultat pour 2
sum_to(2)	2 + le résultat pour 1
sum_to(1)	1 + le résultat pour 0
sum_to(0)	Rien : il renvoie 0

Quand le résultat zéro est disponible, l'appel pour 1 peut renvoyer 1. Celui pour 2 renvoie alors 3, puis celui pour 3 renvoie 6. Les résultats remontent dans l'ordre inverse de la descente des appels.

Prévoir une petite entrée

Commencez par n = 0, puis 1, puis 3. Pour 0, le corps ne déclenche aucun appel récursif. Pour 1, un appel supplémentaire pour 0 suffit. Gardez de petites entrées : une exécution récursive dispose d'une profondeur d'appels limitée et n'est pas un moyen de répéter sans limite.

Relier expression et retour

▼ Pourquoi retourner seulement sum_to(n - 1) ne calcule-t-il pas la somme demandée ?

Il manquerait l'ajout du terme n à chaque niveau. Avec le cas de base zéro, cette version finirait par renvoyer zéro à tous les appels au lieu de reconstruire la somme.

▼ Quel résultat attend-on pour sum_to(4) ?

La fonction doit obtenir 6 pour l'entrée 3, puis ajouter 4 : le résultat vaut 10. Cette prévision permet de relier un appel au contrat du sous-problème plutôt que de mémoriser une sortie.

Deux obligations complémentaires

Une définition récursive doit expliquer comment obtenir un résultat à partir d'un problème plus petit et comment atteindre une réponse immédiate. Le chapitre suivant examine ces deux obligations et les entrées sur lesquelles elles tiennent.

À retenir - fiche mémo

Appel récursif

La fonction demande une nouvelle exécution d'elle-même sur une entrée plus simple.

Cas de base

Un résultat obtenu sans nouvel appel récursif.

Somme

sum_to(3) attend les résultats des appels pour 2, 1 puis 0 et renvoie 6.

Pseudo-code

```
// example.pseudo
// sum_to
FUNCTION sum_to(n)
    IF n == 0 THEN
        RETURN 0
    END IF
    RETURN n + sum_to(n - 1)
END FUNCTION
```

Point de vigilance. Le domaine de cet exemple est celui des petits entiers non négatifs.

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

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