ALGORITHMIQUE · CHAPITRE 23 · FONCTIONS

Construire le cas de base et la progression

Expliquez pourquoi une fonction récursive s'arrête : entrée autorisée, cas de base atteignable et quantité qui diminue. Testez les cas zéro et négatif.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Nommer les entrées pour lesquelles le raisonnement est valable.
- Relier chaque appel à une diminution vers le cas de base.
- Distinguer validation partielle et contrat complet.

Prérequis. Comprendre la somme récursive et le retour d'une fonction.

Le cas de base suffit-il ?

La fonction précédente s'arrête pour n = 3. Avec n = −1, retirer encore 1 ne rapproche pas de zéro : on obtient −2, puis −3. Le fait d'avoir écrit un cas de base n'assure donc pas qu'il soit atteint pour n'importe quelle entrée.

Annoncer ce qui est autorisé

Notre problème porte sur un **entier non négatif**. Pour n strictement positif, n − 1 est encore un entier non négatif, et il est plus petit. Pour n = 0, la réponse est immédiate. L'appel reste ainsi dans le domaine et se rapproche d'une situation qui ne nécessite plus d'appel.

Contrôle d’entrée, erreur et hypothèse de domaine

Une **entrée invalide** ne respecte pas le contrat. Une **erreur signalée** interrompt le calcul normal et informe l'appelant que le résultat demandé n'est pas fourni. Le pseudo-code écrit `RAISE ERROR`. Python utilise ici `raise ValueError(...)`, et MATLAB `error(...)`, avec un message en anglais.

Le nom `sum_to_checked` indique un contrôle supplémentaire : le programme rejette les valeurs négatives. Il ne vérifie pas à lui seul que l'entrée est un entier scalaire. Le contrat reste donc un entier, et le message d'erreur ne doit pas être interprété comme une validation de tous les types possibles. Une fraction positive pourrait ne jamais atteindre le cas de base égal à zéro.

Pour montrer l'arrêt sur les entrées autorisées, suivons n : il reste entier, diminue de 1 à chaque nouvel appel et ne peut pas rester strictement positif indéfiniment. Le cas de base doit être testé avant de construire l'appel suivant. Une condition d'arrêt présente dans le code ne garantit rien si la suite des arguments ne peut jamais la rejoindre.

Essayez séparément zéro, un petit positif et un négatif. Le premier renvoie immédiatement 0 ; le second descend puis recompose la somme ; le troisième signale une erreur avant toute descente. Ces trois observations répondent à trois questions distinctes et ne constituent pas une preuve pour des valeurs hors du domaine prévu.

Signaler une entrée négative

Pseudo-code

```
// example.pseudo
// sum_to_checked
FUNCTION sum_to_checked(n)
    IF n < 0 THEN
        RAISE ERROR
    END IF
    IF n == 0 THEN
        RETURN 0
    END IF
    RETURN n + sum_to_checked(n - 1)
END FUNCTION
```

Python

```
# sum_to_checked.py
def sum_to_checked(n):
    if n < 0:
        raise ValueError("n must be non-negative")
    if n == 0:
        return 0
    return n + sum_to_checked(n - 1)

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

MATLAB

```
% sum_to_checked.m
function result = sum_to_checked(n)
    if n < 0
        error('n must be non-negative');
    end
    if n == 0
        result = 0;
        return;
    end
    result = n + sum_to_checked(n - 1);
end
```

Cette version signale explicitement un n négatif. Elle suppose toujours que l'entrée est un entier scalaire fini : le test ajouté ne valide pas à lui seul les textes, les tableaux ou les nombres non entiers. Les messages d'erreur ont le même sens, même si les mécanismes d'exception des langages diffèrent.

MATLAB

```
% use_sum_to_checked.m
result = sum_to_checked(3);
disp(result); % 6
```

Suivre la quantité qui diminue

Un domaine et une progression explicites

Entrée	Prochaine entrée	Interprétation
3	2	Plus petit, encore autorisé
1	0	Le prochain appel est le cas de base

Entrée	Prochaine entrée	Interprétation
0	Aucun appel	Réponse immédiate
-1	Aucun appel	Erreur avant la récursion

Pour les entrées autorisées, la quantité n diminue de 1 et ne peut pas descendre indéfiniment en restant un entier non négatif. Ce raisonnement justifie la terminaison mathématique de la définition ; les ressources de la machine restent une autre limite pour une grande entrée.

Comparer trois erreurs de conception

Sans le cas de base, on ne dispose plus d'une réponse immédiate. Avec `n + 1`, on s'éloigne de zéro. Avec `n - 2`, une entrée impaire peut dépasser zéro sans jamais lui être égale. Dans chaque cas, expliquez le problème sur une petite entrée avant d'exécuter quoi que ce soit.

Corriger le raisonnement

▼ Pourquoi 3, 1, −1, −3 ne décrit-il pas une progression correcte vers n = 0 ?

La valeur diminue, mais zéro n'est jamais rencontré. Une diminution ne suffit pas : il faut rester dans un domaine approprié et atteindre le cas de base prévu.

▼ Peut-on remplacer le cas n = 0 par n ≤ 0 sans rien expliquer ?

Cela ferait renvoyer zéro pour des entrées négatives plutôt que signaler qu'elles sont interdites. Ce serait un changement de contrat. Il faut choisir ce comportement et l'annoncer, pas l'introduire pour masquer une récursion incorrecte.

Définir avant de lancer

Avant toute exécution récursive, écrivez le domaine, le ou les cas de base et la relation entre une entrée et l'entrée suivante. Ces trois éléments rendent la terminaison compréhensible et permettent de choisir les cas limites à vérifier.

À retenir - fiche mémo

Domaine

Annoncez les entrées autorisées avant de raisonner sur l’arrêt.

Progression

Sur les entiers positifs, n - 1 diminue et rejoint zéro.

Contrôle

La version présentée rejette les nombres négatifs, mais ne vérifie pas tous les types.

Pseudo-code

```
// example.pseudo
// recursive_contract
// Input: a non-negative integer.
// Base case: n == 0.
// Recursive argument: n - 1.
```

Point de vigilance. Un cas de base présent dans le code ne suffit pas si les appels ne peuvent jamais l’atteindre.

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

Version web : https://www.clearcode.fr/courses/programming/algorithmic/functions/recursion/base-case-and-progress