ALGORITHMIQUE · CHAPITRE 16 · CONDITIONS ET BOUCLES

Répéter POUR

Calculez une somme avec for et un accumulateur. Comprenez les bornes exclues de range en Python et les bornes incluses des séquences MATLAB.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Prévoir les valeurs d'un intervalle de boucle.
- Traduire des bornes incluses entre pseudo-code, Python et MATLAB.
- Initialiser un accumulateur avant la répétition.

Prérequis. Savoir tracer un compteur et comprendre une répétition.

Additionner de 1 à n

Nous cherchons la somme des entiers de 1 à `n`, avec n entier non négatif. Pour n = 4, le résultat attendu est `1 + 2 + 3 + 4 = 10`. Pour n = 0, il n'y a aucun terme à ajouter et nous choisissons la somme zéro.

Conserver un résultat partiel

La variable `total` est un **accumulateur** : elle conserve la somme déjà construite. À chaque tour, `value` prend la valeur du prochain entier et on l'ajoute au total précédent. L'initialisation à zéro doit précéder la boucle, sinon le travail des tours précédents serait perdu.

Définir un accumulateur et les bornes du parcours

Un **accumulateur** conserve un résultat partiel au fil d'un parcours. Pour une somme, on commence à zéro parce que l'ajout de zéro ne modifie pas le premier terme. Après avoir ajouté 1 puis 2, `total` vaut 3 : ce n'est plus la valeur du dernier terme, mais la somme des termes déjà parcourus.

Une **borne** indique où un parcours commence ou se termine. Une borne incluse appartient au parcours ; une borne exclue n'y appartient pas. Le pseudo-code annonce `FROM 1 TO n INCLUSIVE`. Pour n égal à 4, les valeurs sont 1, 2, 3 et 4. Python traduit ce choix par `range(1, n + 1)`, car sa borne supérieure est exclue.

MATLAB utilise `1:n` pour cette progression par pas de 1. Le **pas** est l'écart entre deux valeurs successives. Avec une borne n égale à zéro, `1:0` et `range(1, 1)` produisent ici un parcours vide ; la somme reste zéro. Le programme ne doit pas ajouter artificiellement un terme pour ce cas.

Pour contrôler une somme, notez avant chaque tour la valeur parcourue et le total déjà obtenu. La séquence de totaux attendue pour n égal à 4 est 1, 3, 6, 10. Réinitialiser `total` dans la boucle détruit cette mémoire du calcul et laisse seulement le dernier terme.

Annoncer les bornes

Pseudo-code

```
// example.pseudo
// sum_integers
n <- 4
total <- 0
FOR value FROM 1 TO n INCLUSIVE DO
    total <- total + value
END FOR
PRINT total
```

Python

```python
# sum_integers.py
n = 4
total = 0
for value in range(1, n + 1):
    total = total + value
print(total)  # 10
```

MATLAB

```matlab
% sum_integers.m
n = 4;
total = 0;
for value = 1:n
    total = total + value;
end
disp(total); % 10
```

Dans le pseudo-code, les deux bornes sont incluses. MATLAB parcourt aussi les valeurs de 1 à n avec `1:n`. Python exclut la borne de fin de `range` : écrire `range(1, n + 1)` est nécessaire pour inclure n. La méthode et les valeurs additionnées restent identiques.

Comparer les valeurs et les états

Construire la somme de 1 à 4

value	total avant	total après
1	0	1
2	1	3
3	3	6

value	total avant	total après
4	6	10

Le nombre de tours est quatre, mais le résultat n'est pas quatre : un compteur et une somme ne représentent pas la même chose. Après le tour où value vaut 3, total représente déjà `1 + 2 + 3`. Cette phrase relie l'état au sens de la variable.

Provoquer une erreur de borne

Remplacez en Python `n + 1` par `n`. Pour n = 4, la dernière valeur parcourue devient 3 et le résultat 6. Cette erreur ne se voit pas en lisant seulement le nom de la boucle. Énumérer les valeurs parcourues est un contrôle simple avant toute exécution.

Réutiliser le schéma d’accumulation

▼ Que faut-il initialiser pour calculer un produit plutôt qu'une somme ?

Le produit part de 1, élément neutre de la multiplication, puis multiplie les valeurs successives. Partir de zéro rendrait tous les produits suivants nuls. Il faut aussi définir le résultat attendu quand aucun terme n'est parcouru.

▼ Que produit la version correcte pour n = 0 ?

Aucun tour : range(1, 1) et 1:0 sont vides avec ces pas positifs. total garde zéro, conformément au contrat. Ce raisonnement porte sur les bornes choisies, pas sur toutes les écritures possibles d'un intervalle.

Choisir une répétition connue

POUR convient lorsque les valeurs à parcourir sont définies. TANT QUE exprime plus directement une poursuite dépendant d'un état. Les deux peuvent parfois décrire le même calcul ; le choix doit rendre la règle de répétition facile à comprendre.

À retenir - fiche mémo

Parcours

Le pseudo-code indique explicitement des bornes incluses.

Python

range(1, n + 1) fournit les entiers de 1 à n pour n entier positif.

MATLAB

1:n parcourt ici les mêmes valeurs avec un pas de 1.

Pseudo-code

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

Point de vigilance. L'accumulateur est initialisé avant la boucle ; une somme vide reste zéro.

Consulter la cheatsheet : Conditions et boucles · **Télécharger la fiche du thème en PDF**

Version web : https://www.clearcode.fr/courses/programming/algorithmic/control-flow/loops/for